Informatica® Data Engineering Integration
10.5.1

User Guide

Informatica Data Engineering Integration User Guide
10.5.1
September 2021

© Copyright Informatica LLC 2012, 2022

This software and documentation are provided only under a separate license agreement containing restrictions on use and disclosure. No part of this document may be reproduced or transmitted in any form, by any means (electronic, photocopying, recording or otherwise) without prior consent of Informatica LLC.

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

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

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

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

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

Publication Date: 2022-05-10

Table of Contents

Preface 15

Informatica Resources 15

Informatica Network 15

Informatica Knowledge Base 15

Informatica Documentation 15

Informatica Product Availability Matrices 16

Informatica Velocity 16

Informatica Marketplace 16

Informatica Global Customer Support 16

Chapter 1: Introduction to Informatica Data Engineering Integration 17

Informatica Data Engineering Integration Overview 17

Example 17

Data Engineering Integration Component Architecture 18

Clients and Tools 18

Application Services 18

Repositories 19

Hadoop Integration 19

Databricks Integration 21

Data Engineering Integration Engines 21

Run-time Process on the Blaze Engine 22

Run-time Process on the Spark Engine 23

Run-time Process on the Databricks Spark Engine 24

Data Engineering Process 25

Step 1. Collect the Data 25

Step 2. Cleanse the Data 25

Step 3. Transform the Data 26

Step 4. Process the Data 26

Step 5. Monitor Jobs 26

Data Warehouse Optimization Mapping Example 26

Chapter 2: Mappings 29

Overview of Mappings 29

Mapping Run-time Properties 30

Validation Environments 31

Execution Environment 32

Updating Run-time Properties for Multiple Mappings 35

PreSQL and PostSQL Queries for JDBC Sources 35

Sqoop Mappings in a Hadoop Environment 36

Sqoop Mapping-Level Arguments 36

Incremental Data Extraction for Sqoop Mappings. . . . 39
Configuring Sqoop Properties in the Mapping. . . . 41
Configuring Parameters for Sqoop Arguments in the Mapping. . . . 41
Mapping Output Binding. . . . 42
Rules and Guidelines for Mappings in a Non-native Environment. . . . 42
Rules and Guidelines for Mappings on the Blaze Engine. . . . 42
Rules and Guidelines for Mappings on the Spark Engine. . . . 43
Rules and Guidelines for Mappings on the Databricks Spark Engine. . . . 46
Workflows that Run Mappings in a Non-native Environment. . . . 46
Configuring a Mapping to Run in a Non-native Environment. . . . 46
Configure Mappings to Run on Dataproc. . . . 47
Audits. . . . 47
Creating an Audit. . . . 48
Rules and Guidelines for Audits. . . . 52
Example. . . . 53
Mapping Execution Plans. . . . 54
Blaze Engine Execution Plan Details. . . . 55
Spark Engine Execution Plan Details. . . . 56
Databricks Spark Engine Execution Details. . . . 56
Viewing the Execution Plan. . . . 57
Troubleshooting Mappings in a Non-native Environment. . . . 58
Mappings in the Native Environment. . . . 60
Data Processor Mappings. . . . 60
HDFS Mappings. . . . 60
Hive Mappings. . . . 62
Social Media Mappings. . . . 63
Native Environment Optimization. . . . 64

Chapter 3: Mapping Optimization. . . . 67

Mapping Optimization. . . . 67
Mapping Recommendations and Analysis. . . . 68
Recommendations. . . . 69
Insights. . . . 73
Example 74
Enabling and Customizing Access to Sensitive Data. . . . 75
Enabling Data Compression on Temporary Staging Tables. . . . 76
Step 1. Enable Data Compression in the Hadoop Connection. . . . 76
Step 2. Enable Data Compression on the Hadoop Environment. . . . 76
Truncating Partitions in a Hive Target. . . . 77
Hive Warehouse Connector and Hive LLAP. . . . 77
Enabling the Hive Warehouse Connector and Hive LLAP. . . . 78
Scheduling, Queuing, and Node Labeling. . . . 79
Enable Scheduling and Node Labeling. . . . 79

Define YARN Queues. . . . 80
Configure the Blaze Engine to Use Node Labels. . . . 80
Data Engineering Recovery. . . . 81
Spark Engine Optimization for Sqoop Pass-Through Mappings. . . . 81

Chapter 4: Sources. . . . 83

Overview of Sources. . . . 83
PowerExchange Adapter Sources. . . . 83
Sources on Databricks. . . . 85
Complex File Sources on Amazon S3. . . . 85
Complex File Sources on ADLS . . . 85
Complex File Sources on Azure Data Lake Storage Gen2. . . . 86
Complex File Sources on Azure Blob . . . 86
Databricks Delta Lake. . . . 87
Rules and Guidelines for Databricks Sources. . . . 87
File Sources on Hadoop. . . . 88
Complex File Sources on Amazon S3. . . . 88
Complex File Sources on Google Cloud Storage. . . . 89
Complex File Sources on HDFS. . . . 89
Complex File Sources on Hive. . . . 90
Complex File Sources on MapR-FS. . . . 90
Complex File Sources on Azure Blob. . . . 91
Complex File Sources on ADLS. . . . 92
Complex File Sources on Azure Data Lake Storage Gen2. . . . 93
Flat File Sources on Hadoop. . . . 94
Relational Sources on Hadoop. . . . 95
Hive Sources on Hadoop. . . . 95
PreSQL and PostSQL Commands. . . . 95
Rules and Guidelines for Hive Sources on the Blaze Engine. . . . 96
Sqoop Sources on Hadoop. . . . 98
Reading Data from Vertica Sources through Sqoop. . . . 98
Rules and Guidelines for Sqoop Sources. . . . 99
Rules and Guidelines for Sqoop Queries. . . . 99

Chapter 5: Targets. . . . 100

Overview of Targets. . . . 100
PowerExchange Adapter Targets. . . . 100
Targets on Databricks. . . . 102
Complex File Targets on Amazon S3. . . . 102
Complex File Targets on ADLS. . . . 102
Complex File Targets on Azure Data Lake Storage Gen2. . . . 103
Complex File Targets on Azure Blob . . . 103
Databricks Delta Lake . . . 104

Rules and Guidelines for Databricks Targets. . . . 104
File Targets on Hadoop. . . . 105
Complex File Targets on Amazon S3. . . . 105
Complex File Targets on Google Cloud Storage 106
Complex File Targets on ADLS. . . . 106
Complex File Targets on Azure Data Lake Storage Gen2. . . . 107
Complex File Targets on Azure Blob 108
Complex File Targets on MapR-FS. . . . 109
Complex File Targets on HDFS 109
Complex File Targets on Hive. . . . 110
Flat File Targets on Hadoop. . . . 110
Message Targets on Hadoop. . . . 111
Relational Targets on Hadoop. . . . 111
Hive Targets on Hadoop. . . . 112
PreSQL and PostSQL Commands. . . . 112
Truncating Hive Targets. . . . 113
Updating Hive Targets with an Update Strategy Transformation. . . . 113
Rules and Guidelines for Hive Targets on the Blaze Engine. . . . 114
Sqoop Targets on Hadoop. . . . 115
Rules and Guidelines for Sqoop Targets. . . . 116

Chapter 6: Transformations. . . . 117

Overview of Transformations. . . . 117
Address Validator Transformation in a Non-native Environment. . . . 120
Address Validator Transformation on the Blaze Engine. . . . 120
Address Validator Transformation on the Spark Engine. . . . 120
Address Validator Transformation on the Databricks Spark Engine. . . . 120
Aggregator Transformation in a Non-native Environment. . . . 121
Aggregator Transformation on the Blaze Engine. . . . 121
Aggregator Transformation on the Spark Engine. . . . 122
Aggregator Transformation on the Databricks Spark Engine. . . . 122
Case Converter Transformation in a Non-native Environment. . . . 123
Classifier Transformation in a Non-native Environment. . . . 123
Comparison Transformation in a Non-native Environment. . . . 123
Consolidation Transformation in a Non-native Environment. . . . 124
Consolidation Transformation on the Blaze Engine. . . . 124
Consolidation Transformation on the Spark Engine. . . . 124
Consolidation Transformation on the Databricks Spark Engine. . . . 124
Data Masking Transformation in a Non-native Environment. . . . 125
Data Masking Transformation on the Blaze Engine. . . . 125
Data Masking Transformation on the Spark Engine. . . . 125
Data Processor Transformation in a Non-native Environment. . . . 126
Decision Transformation in a Non-native Environment. . . . 127

Decision Transformation on the Spark Engine. . . . 127
Decision Transformation on the Databricks Spark Engine. . . . 127
Expression Transformation in a Non-native Environment. . . . 127
Expression Transformation on the Blaze Engine. . . . 128
Expression Transformation on the Spark Engine. . . . 128
Expression Transformation on the Databricks Spark Engine. . . . 128
Filter Transformation in a Non-native Environment. . . . 128
Filter Transformation on the Blaze Engine. . . . 129
Hierarchical to Relational Transformation in a Non-native Environment. . . . 129
Java Transformation in a Non-native Environment. . . . 129
Java Transformation on the Blaze Engine. . . . 129
Java Transformation on the Spark Engine. . . . 130
Joiner Transformation in a Non-native Environment 131
Joiner Transformation on the Blaze Engine. . . . 131
Joiner Transformation on the Spark Engine. . . . 131
Joiner Transformation on the Databricks Spark Engine. . . . 132
Key Generator Transformation in a Non-native Environment. . . . 132
Key Generator Transformation on the Blaze Engine. . . . 132
Key Generator Transformation on the Spark Engine. . . . 132
Key Generator Transformation on the Databricks Spark Engine. . . . 133
Labeler Transformation in a Non-native Environment. . . . 133
Lookup Transformation in a Non-native Environment. . . . 133
Lookup Transformation on the Blaze Engine. . . . 133
Lookup Transformation on the Spark Engine. . . . 134
Lookup Transformation on the Databricks Spark Engine. . . . 136
Macro Transformation in a Non-native Environment. . . . 137
Match Transformation in a Non-native Environment. . . . 137
Match Transformation on the Blaze Engine. . . . 137
Match Transformation on the Spark Engine. . . . 138
Match Transformation on the Databricks Spark Engine. . . . 138
Merge Transformation in a Non-native Environment. . . . 138
Normalizer Transformation in a Non-native Environment. . . . 138
Parser Transformation in a Non-native Environment. . . . 139
Rank Transformation in a Non-native Environment. . . . 139
Rank Transformation on the Blaze Engine 139
Rank Transformation on the Spark Engine. . . . 139
Rank Transformation on the Databricks Spark Engine. . . . 140
Relational to Hierarchical Transformation in a Non-native Environment. . . . 140
Router Transformation in a Non-native Environment. . . . 141
Rule Specification Transformation in a Non-native Environment. . . . 141
Sequence Generator Transformation in a Non-native Environment. . . . 141
Sequence Generator Transformation on the Blaze Engine. . . . 141

Sequence Generator Transformation on the Spark Engine. . . . 142
Sequence Generator Transformation on the Databricks Spark Engine. . . . 142
Sorter Transformation in a Non-native Environment. . . . 142
Sorter Transformation on the Blaze Engine. . . . 142
Sorter Transformation on the Spark Engine. . . . 143
Sorter Transformation on the Databricks Spark Engine. . . . 144
Standardizer Transformation in a Non-native Environment. . . . 144
Union Transformation in a Non-native Environment. . . . 145
Union Transformation in a Streaming Mapping. . . . 145
Update Strategy Transformation in a Non-native Environment. . . . 145
Update Strategy Transformation on the Blaze Engine. . . . 145
Update Strategy Transformation on the Spark Engine. . . . 146
Update Strategy Transformation on the Databricks Spark Engine. . . . 148
Weighted Average Transformation in a Non-native Environment. . . . 148

Chapter 7: Python Transformation. . . . 149

Python Transformation Overview. . . . 149
Active and Passive Python Transformations. . . . 150
Data Type Conversion. . . . 150
Data Types in Input and Output Ports. . . . 151
Python Transformation Ports. . . . 151
Python Transformation Advanced Properties. . . . 151
Python Transformation Components. . . . 152
Resource File. . . . 152
Python Code. . . . 153
Rules and Guidelines for the Python Transformation. . . . 154
Python Transformation in a Streaming Mapping. . . . 154
Creating a Python Transformation. . . . 154
Creating a Reusable Python Transformation. . . . 155
Creating a Non-Reusable Python Transformation. . . . 155
Example: Add an ID Column to Nonpartitioned Data. . . . 156
Example: Use Partitions to Find the Highest Salary. . . . 157
Use Case: Operationalize a Pre-Trained Model. . . . 159

Chapter 8: Data Preview. . . . 161

Overview of Data Preview. . . . 161
Connections and Cluster Distributions that Support Data Preview. . . . 162
Data Preview Process. . . . 162
Previewing Data. . . . 163
Data Preview Interface for Hierarchical Data. . . . 164
Data Viewer. . . . 165
Exporting Data 165
Hierarchical Type Panel. . . . 166

Data Preview on Transformations. . . . 166
Data Preview Logs. . . . 167
Rules and Guidelines for Data Preview on the Spark Engine. . . . 168

Chapter 9: Cluster Workflows. . . . 169

Cluster Workflows Overview. . . . 169
Cluster Workflows Platform Support. . . . 169
Cluster Workflow Components. . . . 170
Configure Databricks Clusters Using Warm Pools. . . . 171
Cluster Workflows Process 172
Create Cluster Task Properties. . . . 173
Advanced Properties for Amazon EMR. . . . 174
Advanced Properties for Azure HDInsight. . . . 177
Advanced Properties for Azure Databricks. . . . 178
Advanced Properties for AWS Databricks. . . . 180
Advanced Properties for the Blaze Engine. . . . 183
Advanced Properties for a Hive Metastore Database. . . . 184
Mapping Task Properties. . . . 185
Add a Delete Cluster Task. . . . 186
Deploy and Run the Workflow. . . . 186
Monitoring Azure HDInsight Cluster Workflow Jobs 186

Chapter 10: Profiles. . . . 187

Profiles Overview. . . . 187
Native Environment. . . . 187
Hadoop Environment. . . . 188
Column Profiles for Sqoop Data Sources. . . . 188
Sampling Options. . . . 189
Creating a Single Data Object Profile in Informatica Developer. . . . 189
Creating an Enterprise Discovery Profile in Informatica Developer. . . . 190
Creating a Column Profile in Informatica Analyst. . . . 191
Creating an Enterprise Discovery Profile in Informatica Analyst. . . . 193
Creating a Scorecard in Informatica Analyst. . . . 194
Monitoring a Profile. . . . 195
Profiling Functionality Support. . . . 195
Troubleshooting. . . . 196

Chapter 11: Monitoring. . . . 198

Overview of Monitoring. . . . 198
Hadoop Environment Logs. . . . 198
YARN Web User Interface. . . . 199
Accessing the Monitoring URL. . . . 200
Viewing Hadoop Environment Logs in the Administrator Tool. . . . 200

Monitoring a Mapping. . . . 201
Blaze Engine Monitoring. . . . 204
Blaze Job Monitoring Application. . . . 206
Blaze Summary Report. . . . 207
Blaze Engine Logs. . . . 210
Viewing Blaze Logs. . . . 211
Orchestrator Sunset Time. . . . 211
Troubleshooting Blaze Monitoring. . . . 211
Spark Engine Monitoring. . . . 212
Viewing Hive Tasks. . . . 216
Spark Engine Logs. . . . 217
Viewing Spark Logs 217
Troubleshooting Spark Engine Monitoring. . . . 217

Chapter 12: Hierarchical Data Processing. . . . 219

Overview of Hierarchical Data Processing. . . . 219
How to Develop a Mapping to Process Hierarchical Data. . . . 220
Complex Data Types. . . . 222
Array Data Type. . . . 223
Map Data Type. . . . 225
Struct Data Type. . . . 225
Rules and Guidelines for Complex Data Types. . . . 226
Complex Ports. . . . 227
Complex Ports in Transformations. . . . 228
Rules and Guidelines for Complex Ports. . . . 228
Creating a Complex Port. . . . 229
Complex Data Type Definitions. . . . 229
Nested Data Type Definitions. . . . 231
Rules and Guidelines for Complex Data Type Definitions. . . . 231
Creating a Complex Data Type Definition. . . . 232
Importing a Complex Data Type Definition. . . . 232
Type Configuration. . . . 234
Changing the Type Configuration for an Array Port. . . . 234
Changing the Type Configuration for a Map Port. . . . 236
Specifying the Type Configuration for a Struct Port. . . . 237
Complex Operators. . . . 238
Extracting an Array Element Using a Subscript Operator. . . . 239
Extracting a Struct Element Using the Dot Operator. . . . 239
Complex Functions. . . . 240
Rules and Guidelines for Processing Hierarchical Data on the Spark Engine. . . . 242
Midstream Parsing of Hierarchical Data. . . . 242
Midstream Parsing Overview. . . . 242
Midstream Parsing Use Case. . . . 242

How to Use a Midstream Mapping to Parse Hierarchical Data. 243
Rules and Guidelines for Midstream Parsing. 244
Troubleshooting Midstream Parsing. 244

Chapter 13: Hierarchical Data Processing Configuration. 246

Hierarchical Data Conversion. 246
Convert Relational or Hierarchical Data to Struct Data. 247
Creating a Struct Port. 247
Convert Relational or Hierarchical Data to Nested Struct Data. 249
Creating A Nested Complex Port. 250
Extract Elements from Hierarchical Data. 257
Extracting Elements from a Complex Port. 257
Flatten Hierarchical Data. 259
Flattening a Complex Port. 260

Chapter 14: Hierarchical Data Processing with Schema Changes. 262

Overview of Hierarchical Data Processing with Schema Changes. 262
How to Develop a Dynamic Mapping to Process Schema Changes in Hierarchical Data. 263
Flatten Hierarchical Data with Schema Changes. 264
Flatten a Dynamic Struct. 264
Dynamic Complex Ports. 265
Dynamic Ports and Dynamic Complex Ports. 266
Dynamic Complex Ports in Transformations. 267
Input Rules for a Dynamic Complex Port. 267
Input Rule for a Dynamic Array. 268
Input Rules for a Dynamic Map. 269
Input Rules for a Dynamic Struct. 269
Port Selectors for Dynamic Complex Ports. 270
Dynamic Expressions. 270
Example - Dynamic Expression to Construct a Dynamic Struct. 271
Complex Operators. 272
Complex Functions. 272
Rules and Guidelines for Dynamic Complex Ports. 272
Optimized Mappings. 272

Chapter 15: Intelligent Structure Models. 275

Overview of Intelligent Structure Models. 275
Intelligent Structure Discovery Process. 276
Use Case. 277
Using an Intelligent Structure Model in a Mapping. 277
Rules and Guidelines for Intelligent Structure Models. 278
How to Create a Mapping with an Intelligent Structure Model 278
Mapping Example. 280

Create an Intelligent Structure Model in Cloud Data Integration. 281
Before You Begin. 281
Creating an Informatica Intelligent Cloud Services Account. 281
Creating an Intelligent Structure Model. 281
Exporting an Intelligent Structure Model. 282

Chapter 16: Blockchain. 283

Blockchain Overview. 283
Blockchain Process. 284
Blockchain Data Objects. 284
Response Ports. 284
Blockchain Data Object Overview Properties. 285
Creating a Blockchain Data Object. 286
Blockchain Data Object Operations. 288
Blockchain Data Object Read Operation Properties. 288
Blockchain Data Object Write Operation Properties. 290
Creating a Blockchain Data Object Operation. 291
Use Case: Using a Blockchain Source to Improve Services in a Vehicle Lifecycle. 292
Mapping Overview. 292

Chapter 17: Stateful Computing. 295

Overview of Stateful Computing. 295
Windowing Configuration. 296
Frame. 296
Partition and Order Keys. 297
Rules and Guidelines for Windowing Configuration. 299
Window Functions. 299
LEAD. 300
LAG. 300
Aggregate Functions as Window Functions. 300
Rules and Guidelines for Window Functions. 304
Windowing Examples. 304
Financial Plans Example. 304
GPS Pings Example. 306
Aggregate Function as Window Function Example. 308

Appendix A: Connections Reference. 311

Connections Overview. 312
Cloud Provisioning Configuration. 312
AWS Cloud Provisioning Configuration Properties. 313
Azure Cloud Provisioning Configuration Properties. 316
Databricks Cloud Provisioning Configuration Properties. 321
Amazon Redshift Connection Properties. 322

Amazon S3 Connection Properties. . . . 324
Blockchain Connection Properties. . . . 326
Cassandra Connection Properties. . . . 328
Confluent Kafka Connection. . . . 329
General Properties. . . . 330
Confluent Kafka Broker Properties. . . . 330
SSL Properties 331
Creating a Confluent Kafka Connection Using infacmd. . . . 331
Databricks Connection Properties. . . . 331
Google Analytics Connection Properties. . . . 333
Google BigQuery Connection Properties. . . . 334
Google Cloud Spanner Connection Properties. . . . 336
Google Cloud Storage Connection Properties. . . . 337
Google PubSub Connection Properties. . . . 338
Hadoop Connection Properties. . . . 338
Hadoop Cluster Properties. . . . 339
Common Properties. . . . 340
Reject Directory Properties. . . . 342
Blaze Configuration. . . . 342
Spark Configuration. . . . 343
HDFS Connection Properties. . . . 344
HBase Connection Properties. . . . 346
HBase Connection Properties for MapR-DB. . . . 346
Hive Connection Properties. . . . 347
JDBC Connection Properties. . . . 350
JDBC Connection String. . . . 352
Sqoop Connection-Level Arguments. . . . 354
Delta Lake JDBC Connection Properties. . . . 357
JDBC V2 Connection Properties. . . . 357
Kafka Connection Properties. . . . 359
General Properties. . . . 360
Kafka Broker Properties. . . . 361
SSL Properties. . . . 362
Creating a Kafka Connection Using infacmd. . . . 362
Kudu Connection Properties 362
Microsoft Azure Blob Storage Connection Properties. . . . 363
Microsoft Azure Cosmos DB SQL API Connection Properties. . . . 364
Microsoft Azure Data Lake Storage Gen1 Connection Properties. . . . 365
Microsoft Azure Data Lake Storage Gen2 Connection Properties. . . . 366
Microsoft Azure SQL Data Warehouse Connection Properties. . . . 367
Snowflake Connection Properties. . . . 369
Creating a Connection to Access Sources or Targets. . . . 370

Creating a Hadoop Connection. . . . 370
Configuring Hadoop Connection Properties. . . . 371
Cluster Environment Variables. . . . 372
Cluster Library Path. . . . 372
Common Advanced Properties. . . . 372
Blaze Engine Advanced Properties. . . . 373
Spark Advanced Properties. . . . 374

Appendix B: Data Type Reference. . . . 380

Data Type Reference Overview. . . . 380
Transformation Data Type Support in a Non-native Environment. . . . 381
Complex File and Transformation Data Types. . . . 382
Avro Data Types and Transformation Data Types. . . . 382
JSON Data Types and Transformation Data Types. . . . 384
ORC Data Types and Transformation Data Types. . . . 384
Parquet Data Types and Transformation Data Types. . . . 385
Rules and Guidelines for Data Types. . . . 387
Flat File and Transformation Data Types. . . . 388
Hive Data Types and Transformation Data Types. . . . 389
Hive Complex Data Types. . . . 390
Sqoop Data Types. . . . 391
Aurora Data Types. . . . 391
IBM DB2 and DB2 for z/OS Data Types. . . . 392
Greenplum Data Types. . . . 392
Microsoft SQL Server Data Types. . . . 393
Netezza Data Types. . . . 393
Oracle Data Types. . . . 394
Teradata Data Types. . . . 394
Teradata Data Types with TDCH Specialized Connectors for Sqoop. . . . 395
Vertica Data Types. . . . 395

Appendix C: Function Reference. . . . 397

Function Support in a Non-native Environment. . . . 397
Function and Data Type Processing. . . . 399

Index. . . . 400

Preface

Follow the instructions in the Informatica *Data Engineering Integration User Guide* to learn to create Informatica mappings that run in a native, Hadoop, or Databricks environment. Learn about transformation processing behavior in a non-native environment. Learn to configure mappings to run using the Spark engine or the proprietary Blaze engine. Learn to create workflows that deploy several mappings to a cluster.

Informatica Resources

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

Informatica Network

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

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

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

Informatica Knowledge Base

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

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

Informatica Documentation

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

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

Informatica Product Availability Matrices

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

Informatica Velocity

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

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

Informatica Marketplace

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

Informatica Global Customer Support

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

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link:
https://www.informatica.com/services-and-training/customer-success-services/contact-us.html.

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

CHAPTER 1

Introduction to Informatica Data Engineering Integration

This chapter includes the following topics:

- Informatica Data Engineering Integration Overview, 17
- Data Engineering Integration Component Architecture, 18
- Data Engineering Integration Engines, 21
- Data Engineering Process, 25
- Data Warehouse Optimization Mapping Example , 26

Informatica Data Engineering Integration Overview

Informatica Data Engineering Integration enables your organization to process large, diverse, and fast changing data sets so you can get insights into your data. Use Data Engineering Integration to perform data engineering integration and transformation without writing or maintaining external code.

Use Data Engineering Integration to collect diverse data faster, build business logic in a visual environment, and eliminate hand-coding to get insights on your data. Consider implementing a data engineering project in the following situations:

- The volume of the data that you want to process is greater than 10 terabytes.
- You need to analyze or capture data changes in microseconds.
- The data sources are varied and range from unstructured text to social media data.

You can perform run-time processing in the native environment or in a non-native environment. The native environment is the Informatica domain where the Data Integration Service performs all run-time processing. Use the native run-time environment to process data that is less than 10 terabytes. A non-native environment is a distributed cluster outside of the Informatica domain, such as Hadoop or Databricks, where the Data Integration Service can push run-time processing. Use a non-native run-time environment to optimize mapping performance and process data that is greater than 10 terabytes.

Example

You are an investment banker who needs to calculate the popularity and risk of stocks and then match stocks to each customer based on the preferences of the customer. Your CIO wants to automate the process of calculating the popularity and risk of each stock, match stocks to each customer, and then send an email with a list of stock recommendations for all customers.

You consider the following requirements for your project:

- The volume of data generated by each stock is greater than 10 terabytes.
- You need to analyze the changes to the stock in microseconds.
- The stock is included in Twitter feeds and company stock trade websites, so you need to analyze these social media sources.

Based on your requirements, you work with the IT department to create mappings to determine the popularity of a stock. One mapping tracks the number of times the stock is included in Twitter feeds, and another mapping tracks the number of times customers inquire about the stock on the company stock trade website.

Data Engineering Integration Component Architecture

The Data Engineering Integration components include client tools, application services, repositories, and third-party tools that Data Engineering Integration uses for a data engineering project. The specific components involved depend on the task you perform.

Clients and Tools

You can use several Informatica tools and clients to manage data engineering projects.

Based on your product license and use case, you can use the Administrator tool, Developer tool, Analyst tool, and command line interface.

Application Services

Informatica uses application services in the Informatica domain to process data.

Use the Administrator tool to create connections, monitor jobs, and manage application services.

Data Engineering Integration uses the following application services:

Analyst Service

The Analyst Service runs the Analyst tool in the Informatica domain. The Analyst Service manages the connections between service components and the users that have access to the Analyst tool.

Data Integration Service

The Data Integration Service can process mappings in the native environment or push the mapping for processing to a compute cluster in a non-native environment. The Data Integration Service also retrieves metadata from the Model repository when you run a Developer tool mapping or workflow. The Analyst tool and Developer tool connect to the Data Integration Service to run profile jobs and store profile results in the profiling warehouse.

Mass Ingestion Service

The Mass Ingestion Service manages and validates mass ingestion specifications that you create in the Mass Ingestion tool. The Mass Ingestion Service deploys specifications to the Data Integration Service. When a specification runs, the Mass Ingestion Service generates ingestion statistics.

Metadata Access Service

The Metadata Access Service allows the Developer tool to import and preview metadata from a Hadoop cluster.

The Metadata Access Service contains information about the Service Principal Name (SPN) and keytab information if the Hadoop cluster uses Kerberos authentication. You can create one or more Metadata Access Services on a node. Based on your license, the Metadata Access Service can be highly available.

HBase, HDFS, Hive, and MapR-DB connections use the Metadata Access Service when you import an object from a Hadoop cluster. Create and configure a Metadata Access Service before you create HBase, HDFS, Hive, and MapR-DB connections.

Model Repository Service

The Model Repository Service manages the Model repository. The Model Repository Service connects to the Model repository when you run a mapping, mapping specification, profile, or workflow.

REST Operations Hub

The REST Operations Hub Service is an application service in the Informatica domain that exposes Informatica product functionality to external clients through REST APIs.

Repositories

Data Engineering Integration uses repositories and other databases to store data related to connections, source metadata, data domains, data profiling, data masking, and data lineage. Data Engineering Integration uses application services in the Informatica domain to access data in repositories.

Data Engineering Integration uses the following databases:

Model repository

The Model repository stores profiles, data domains, mapping, and workflows that you manage in the Developer tool. The Model repository also stores profiles, data domains, and mapping specifications that you manage in the Analyst tool.

Profiling warehouse

The Data Integration Service runs profiles and stores profile results in the profiling warehouse.

Hadoop Integration

The Informatica domain can connect to clusters that run different Hadoop distributions. Hadoop is an open-source software framework that enables distributed processing of large data sets across clusters of machines. You might also need to use third-party software clients to set up and manage your Hadoop cluster.

The domain can connect to the supported data source in the Hadoop environment, such as HDFS, HBase, or Hive, and push job processing to the Hadoop cluster. To enable high performance access to files across the cluster, you can connect to an HDFS source. You can also connect to a Hive source, which is a data warehouse that connects to HDFS.

It can also connect to NoSQL databases such as HBase, which is a database comprising key-value pairs on Hadoop that performs operations in real-time. The Data Integration Service can push mapping jobs to the Spark or Blaze engine, and it can push profile jobs to the Blaze engine in the Hadoop environment.

Data Engineering Integration supports more than one version of some Hadoop distributions. By default, the cluster configuration wizard populates the latest supported version.

Hadoop Utilities

Data Engineering Integration uses third-party Hadoop utilities such as Sqoop to process data efficiently.

Sqoop is a Hadoop command line program to process data between relational databases and HDFS through MapReduce programs. You can use Sqoop to import and export data. When you use Sqoop, you do not need to install the relational database client and software on any node in the Hadoop cluster.

To use Sqoop, you must configure Sqoop properties in a JDBC connection and run the mapping in the Hadoop environment. You can configure Sqoop connectivity for relational data objects, customized data objects, and logical data objects that are based on a JDBC-compliant database. For example, you can configure Sqoop connectivity for the following databases:

- Aurora
- Greenplum
- IBM DB2
- IBM DB2 for z/OS
- Microsoft SQL Server
- Netezza
- Oracle
- Teradata
- Vertica

The Model Repository Service uses JDBC to import metadata. The Data Integration Service runs the mapping in the Hadoop run-time environment and pushes the job processing to Sqoop. Sqoop then creates map-reduce jobs in the Hadoop cluster, which perform the import and export job in parallel.

Specialized Sqoop Connectors

When you run mappings through Sqoop, you can use the following specialized connectors:

OraOop

You can use OraOop with Sqoop to optimize performance when you read data from or write data to Oracle. OraOop is a specialized Sqoop plug-in for Oracle that uses native protocols to connect to the Oracle database.

You can configure OraOop when you run Sqoop mappings on the Spark engine.

Teradata Connector for Hadoop (TDCH) Specialized Connectors for Sqoop

You can use the following TDCH specialized connectors for Sqoop to read data from or write data to Teradata:

- Cloudera Connector Powered by Teradata
- Hortonworks Connector for Teradata (powered by the Teradata Connector for Hadoop)
- MapR Connector for Teradata

These connectors are specialized Sqoop plug-ins that Cloudera, Hortonworks, and MapR provide for Teradata. They use native protocols to connect to the Teradata database.

Informatica supports Cloudera Connector Powered by Teradata and Hortonworks Connector for Teradata on the Blaze and Spark engines. When you run Sqoop mappings on the Blaze engine, you must configure these connectors. When you run Sqoop mappings on the Spark engine, the Data Integration Service invokes these connectors by default.

Informatica supports MapR Connector for Teradata on the Spark engine. When you run Sqoop mappings on the Spark engine, the Data Integration Service invokes the connector by default.

Note: For information about running native Teradata mappings with Sqoop, see the *Informatica PowerExchange for Teradata Parallel Transporter API User Guide*.

Databricks Integration

The Data Integration Service can push mappings to the Databricks environment. Databricks is an analytics cloud platform that you can use with Microsoft Azure cloud services or Amazon Web Services. Databricks incorporates the open-source Apache Spark cluster technologies and capabilities.

Informatica automatically installs the binaries required to integrate the Informatica domain with the Databricks environment.

The integration requires Informatica connection objects and cluster configurations. A cluster configuration is a domain object that contains configuration parameters that you import from the Databricks cluster. You then associate the cluster configuration with connections to access the Databricks environment.

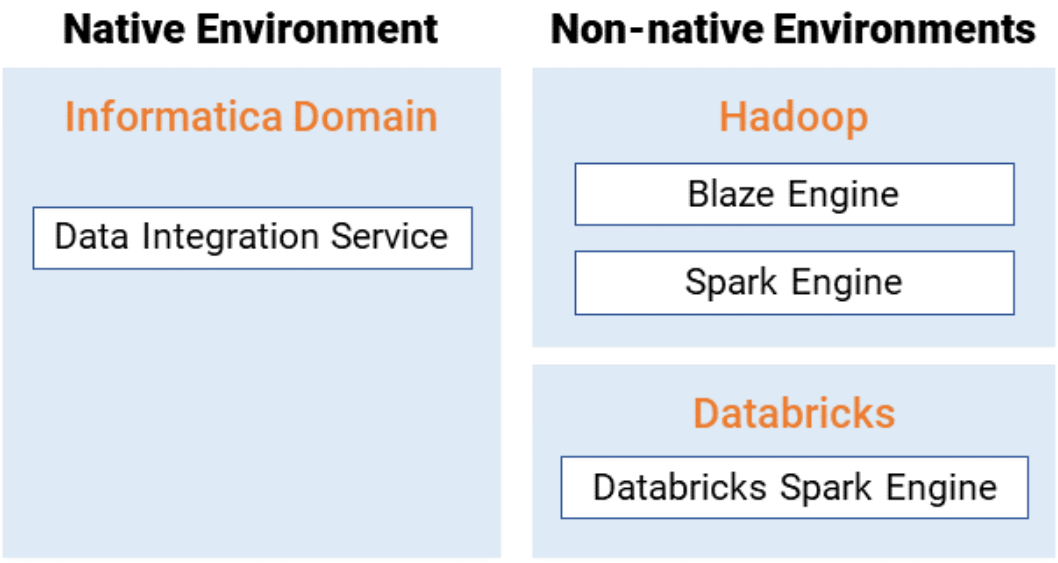
Data Engineering Integration Engines

When you run a mapping, you can choose to run the mapping in the native environment or in a non-native environment, such as Hadoop or Databricks. When you validate a mapping, you can validate it against one or all of the engines. The Developer tool returns validation messages for each engine.

When you run a mapping in the native environment, the Data Integration Service in the Informatica domain runs the mapping. When you run the mapping in a non-native environment, the Data Integration Service pushes the run-time processing to a compute cluster in the non-native environment.

When you run the mapping in a non-native environment, the Data Integration Service uses a proprietary rule-based methodology to determine the best engine to run the mapping. The rule-based methodology evaluates the mapping sources and the mapping logic to determine the engine. The Data Integration Service translates the mapping logic into code that the engine can process, and it transfers the code to the engine.

The following image shows the processing environments and the run-time engines in the environments:



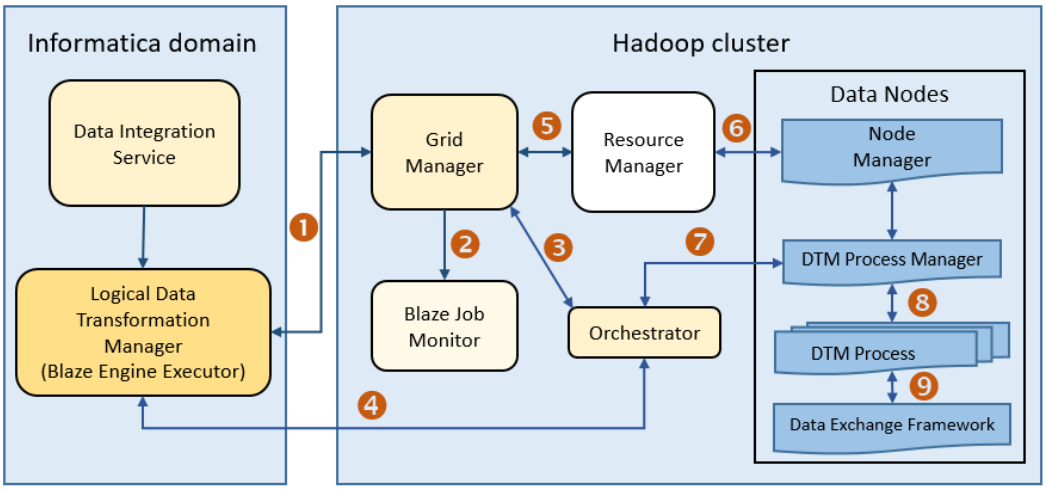
Run-time Process on the Blaze Engine

To run a mapping on the Informatica Blaze engine, the Data Integration Service submits jobs to the Blaze engine executor. The Blaze engine executor is a software component that enables communication between the Data Integration Service and the Blaze engine components on the Hadoop cluster.

The following Blaze engine components appear on the Hadoop cluster:

- Grid Manager. Manages tasks for batch processing.
- Orchestrator. Schedules and processes parallel data processing tasks on a cluster.
- Blaze Job Monitor. Monitors Blaze engine jobs on a cluster.
- DTM Process Manager. Manages the DTM Processes.
- DTM Processes. An operating system process started to run DTM instances.
- Data Exchange Framework. Shuffles data between different processes that process the data on cluster nodes.

The following image shows how a Hadoop cluster processes jobs sent from the Blaze engine executor:



The following events occur when the Data Integration Service submits jobs to the Blaze engine executor:

1. The Blaze Engine Executor communicates with the Grid Manager to initialize Blaze engine components on the Hadoop cluster, and it queries the Grid Manager for an available Orchestrator.
2. The Grid Manager starts the Blaze Job Monitor.
3. The Grid Manager starts the Orchestrator and sends Orchestrator information back to the LDTM.
4. The LDTM communicates with the Orchestrator.
5. The Grid Manager communicates with the Resource Manager for available resources for the Orchestrator.
6. The Resource Manager handles resource allocation on the data nodes through the Node Manager.
7. The Orchestrator sends the tasks to the DTM Processes through the DTM Process Manger.
8. The DTM Process Manager continually communicates with the DTM Processes.
9. The DTM Processes continually communicate with the Data Exchange Framework to send and receive data across processing units that run on the cluster nodes.

Blaze Engine High Availability

The Blaze engine is a highly available engine that determines the best possible recovery strategy for grid tasks and tasklets.

Based on the size of the grid task, the Blaze engine attempts to apply the following recovery strategy:

- No high availability. The Blaze engine does not apply a recovery strategy.
- Full restart. Restarts the grid task.

Application Timeline Server

The Hadoop Application Timeline Server collects basic information about completed application processes. The Timeline Server also provides information about completed and running YARN applications.

The Grid Manager starts the Application Timeline Server in the Yarn configuration by default.

The Blaze engine uses the Application Timeline Server to store the Blaze Job Monitor status. On Hadoop distributions where the Timeline Server is not enabled by default, the Grid Manager attempts to start the Application Timeline Server process on the current node.

The Grid Manager stores Application Timeline Server logs for each user under the logs directory that is configured in the Hadoop connection property infagrid.node.local.root.log.dir.

For example, if infagrid.node.local.root.log.dir is configured with the path `/tmp/infa/logs`, then logs for user User1 are saved in `/tmp/infa/logs/timelineserver_User1`.

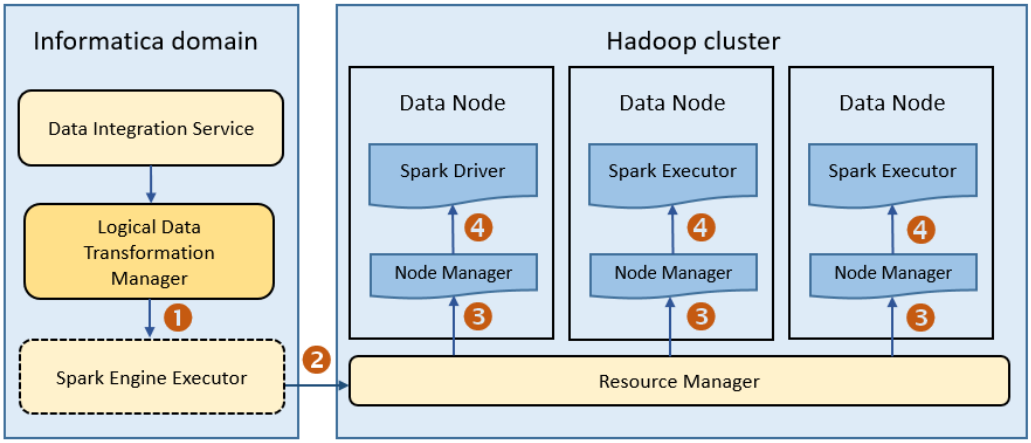
If you do not enable the Application Timeline Server on secured Kerberos clusters, the Grid Manager attempts to start the Application Timeline Server process in HTTP mode.

Run-time Process on the Spark Engine

The Data Integration Service can use the Spark engine on a Hadoop cluster to run Model repository mappings.

To run a mapping on the Spark engine, the Data Integration Service sends a mapping application to the Spark executor. The Spark executor submits the job to the Hadoop cluster to run.

The following image shows how a Hadoop cluster processes jobs sent from the Spark executor:



The following events occur when Data Integration Service runs a mapping on the Spark engine:

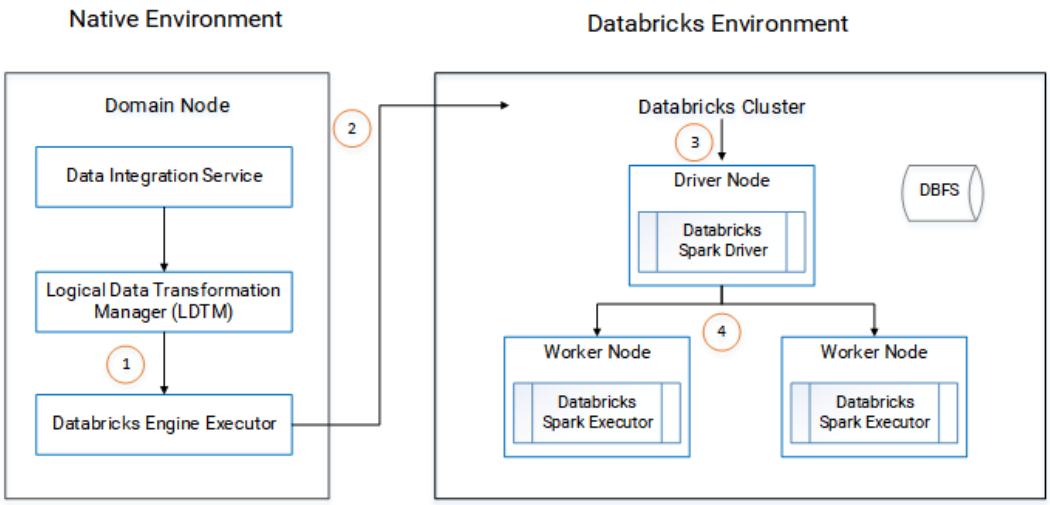
1. The Logical Data Transformation Manager translates the mapping into a Scala program, packages it as an application, and sends it to the Spark executor.
2. The Spark executor submits the application to the Resource Manager in the Hadoop cluster and requests resources to run the application.

   Note: When you run mappings on the HDInsight cluster, the Spark executor launches a spark-submit script. The script requests resources to run the application.
3. The Resource Manager identifies the Node Managers that can provide resources, and it assigns jobs to the data nodes.
4. Driver and Executor processes are launched in data nodes where the Spark application runs.

Run-time Process on the Databricks Spark Engine

When you run a job on the Databricks Spark engine, the Data Integration Service pushes the processing to the Databricks cluster, and the Databricks Spark engine runs the job.

The following image shows the components of the Informatica and the Databricks environments:



1. The Logical Data Transformation Manager translates the mapping into a Scala program, packages it as an application, and sends it to the Databricks Engine Executor on the Data Integration Service machine.
2. The Databricks Engine Executor submits the application through REST API to the Databricks cluster, requests to run the application, and stages files for access during run time.
3. The Databricks cluster passes the request to the Databricks Spark driver on the driver node.
4. The Databricks Spark driver distributes the job to one or more Databricks Spark executors that reside on worker nodes.
5. The executors run the job and stage run-time data to the Databricks File System (DBFS) of the workspace.

Data Engineering Process

As part of a data engineering project, you collect the data from diverse data sources. You can perform profiling, cleansing, and matching for the data. You build the business logic for the data and push the transformed data to the data warehouse. Then you can perform business intelligence on a view of the data.

Based on your data engineering project requirements, you can perform the following high-level tasks:

1. Collect the data.
2. Cleanse the data
3. Transform the data.
4. Process the data.
5. Monitor jobs.

Step 1. Collect the Data

Identify the data sources from which you need to collect the data.

Data Engineering Integration provides several ways to access your data in and out of Hadoop based on the data types, data volumes, and data latencies in the data.

You can use PowerExchange adapters to connect to multiple data engineering sources. You can schedule batch loads to move data from multiple source systems to HDFS without the need to stage the data. You can move changed data from relational and mainframe systems into HDFS or the Hive warehouse. For real-time data feeds, you can move data off message queues and into HDFS.

You can collect the following types of data:

- Transactional
- Interactive
- Log file
- Sensor device
- Document and file
- Industry format

Step 2. Cleanse the Data

Cleanse the data by profiling, cleaning, and matching your data. You can view data lineage for the data.

You can perform data profiling to view missing values and descriptive statistics to identify outliers and anomalies in your data. You can view value and pattern frequencies to isolate inconsistencies or unexpected patterns in your data. You can drill down on the inconsistent data to view results across the entire data set.

You can automate the discovery of data domains and relationships between them. You can discover sensitive data such as social security numbers and credit card numbers so that you can mask the data for compliance.

After you are satisfied with the quality of your data, you can also create a business glossary from your data. You can use the Analyst tool or Developer tool to perform data profiling tasks. Use the Analyst tool to perform data discovery tasks. Use Metadata Manager to perform data lineage tasks.

Step 3. Transform the Data

You can build the business logic to parse data in the Developer tool. Eliminate the need for hand-coding the transformation logic by using pre-built Informatica transformations to transform data.

Step 4. Process the Data

Based on your business logic, you can determine the optimal run-time environment to process your data. If your data is less than 10 terabytes, consider processing your data in the native environment. If your data is greater than 10 terabytes, consider processing your data on a compute cluster.

Step 5. Monitor Jobs

Monitor the status of your processing jobs. You can view monitoring statistics for your processing jobs in the Monitoring tool. After your processing jobs complete you can get business intelligence and analytics from your data.

Data Warehouse Optimization Mapping Example

You can optimize an enterprise data warehouse with the Hadoop system to store more terabytes of data cheaply in the warehouse.

For example, you need to analyze customer portfolios by processing the records that have changed in a 24-hour time period. You can offload the data on Hadoop, find the customer records that have been inserted, deleted, and updated in the last 24 hours, and then update those records in your data warehouse. You can capture these changes even if the number of columns change or if the keys change in the source files.

To capture the changes, you can create the following mappings in the Developer tool:

Mapping_Day1

Create a mapping to read customer data from flat files in a local file system and write to an HDFS target for the first 24-hour period.

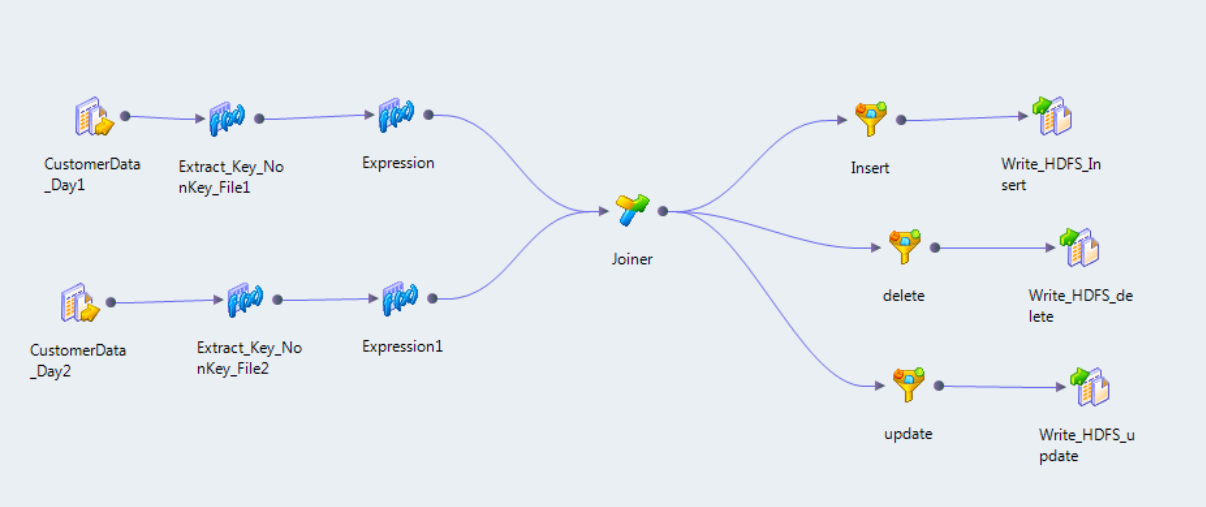
Mapping_Day2

Create a mapping to read customer data from flat files in a local file system and write to an HDFS target for the next 24-hour period.

m_CDC_DWHOptimization

Create a mapping to capture the changed data. The mapping reads data from HDFS and identifies the data that has changed. To increase performance, you configure the mapping to run on Hadoop cluster nodes in a Hadoop environment.

The following image shows the mapping m_CDC_DWHOptimization:



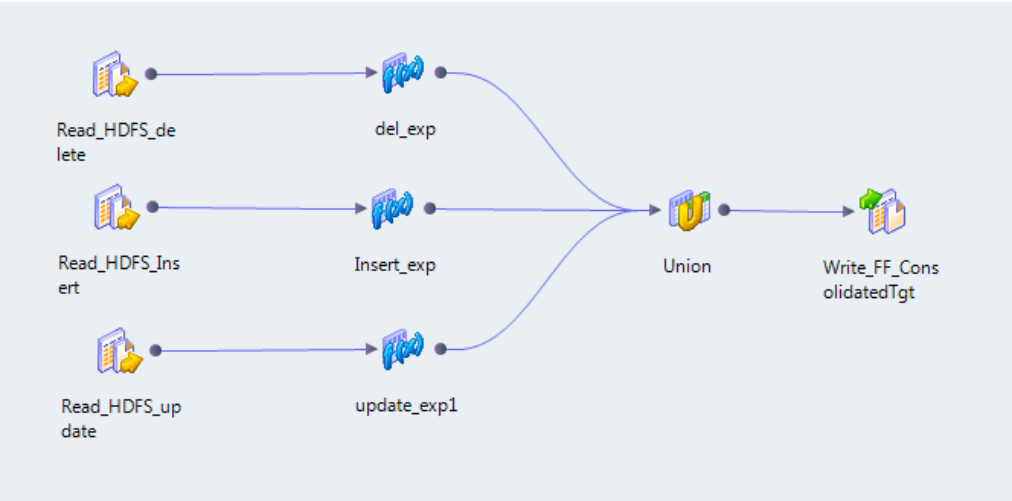
The mapping contains the following objects:

- Read transformations. Transformations that read data from HDFS files that were the targets of Mapping_Day1 and Mapping_Day2. The Data Integration Service reads all of the data as a single column.
- Expression transformations. Extract a key from the non-key values in the data. The expressions use the INSTR function and SUBSTR function to perform the extraction of key values.
- Joiner transformation. Performs a full outer join on the two sources based on the keys generated by the Expression transformations.
- Filter transformations. Use the output of the Joiner transformation to filter rows based on whether or not the rows should be updated, deleted, or inserted.
- Write transformations. Transformations that write the data to three HDFS files based on whether the data is inserted, deleted, or updated.

Consolidated_Mapping

Create a mapping to consolidate the data in the HDFS files and load the data to the data warehouse.

The following figure shows the mapping Consolidated_Mapping:

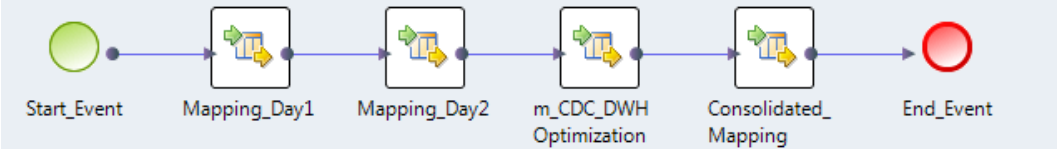


The mapping contains the following objects:

- Read transformations. Transformations that read data from HDFS files that were the target of the previous mapping are the sources of this mapping.
- Expression transformations. Add the deleted, updated, or inserted tags to the data rows.
- Union transformation. Combines the records.
- Write transformation. Transformation that writes data to the flat file that acts as a staging location on the local file system.

You can open each mapping and right-click to run the mapping. To run all mappings in sequence, use a workflow.

The following image shows the example Data Warehouse Optimization workflow:



To run the workflow, use the infacmd wfs startWorkflow command.

CHAPTER 2

Mappings

This chapter includes the following topics:

- Overview of Mappings, 29
- Mapping Run-time Properties, 30
- PreSQL and PostSQL Queries for JDBC Sources, 35
- Sqoop Mappings in a Hadoop Environment, 36
- Mapping Output Binding, 42
- Rules and Guidelines for Mappings in a Non-native Environment, 42
- Workflows that Run Mappings in a Non-native Environment, 46
- Configuring a Mapping to Run in a Non-native Environment, 46
- Audits, 47
- Mapping Execution Plans, 54
- Troubleshooting Mappings in a Non-native Environment, 58
- Mappings in the Native Environment, 60

Overview of Mappings

Configure the run-time environment in the Developer tool to optimize mapping performance and process data that is greater than 10 terabytes. You can choose to run a mapping in the native environment or in a non-native environment. When you run mappings in the native environment, the Data Integration Service processes and runs the mapping. When you run mappings in a non-native environment, the Data Integration Service pushes the processing to a compute cluster, such as Hadoop or Databricks.

You can run standalone mappings and mappings that are a part of a workflow in the non-native environment.

When you select the Hadoop environment, you can also select the engine to push the mapping logic to the Hadoop cluster. Based on the mapping logic, the Data Integration Service can push the mapping logic to one of the following engines in the Hadoop environment:

- Informatica Blaze engine. An Informatica proprietary engine for distributed processing on Hadoop.
- Spark engine. A high performance engine for batch processing that can run on a Hadoop cluster or on a Spark standalone mode cluster.

When you select the Databricks environment, the Integration Service pushes the mapping logic to the Databricks Spark engine, the Apache Spark engine packaged for Databricks.

When you select multiple engines, the Data Integration Service determines the best engine to run the mapping during validation. You can also choose to select which engine the Data Integration Service uses. You might select an engine based on whether an engine supports a particular transformation or based on the format in which the engine returns data.

When you run a mapping in a non-native environment, you must configure a connection to access the environment. You can set the run-time properties for the environment and for the engine that runs the mapping.

To validate the consistency and accuracy of data processed in a mapping, you can create audit rules and conditions for the mapping.

You can view the execution plan for a mapping to run in the non-native environment. View the execution plan for the engine that the Data Integration Service selects to run the mapping.

You can monitor Hive queries and Hadoop jobs in the Monitoring tool. Monitor the jobs on a Hadoop cluster with the YARN Web User Interface or the Blaze Job Monitor web application.

The Data Integration Service logs messages from the Blaze, Spark and Databricks Spark engines and the DTM.

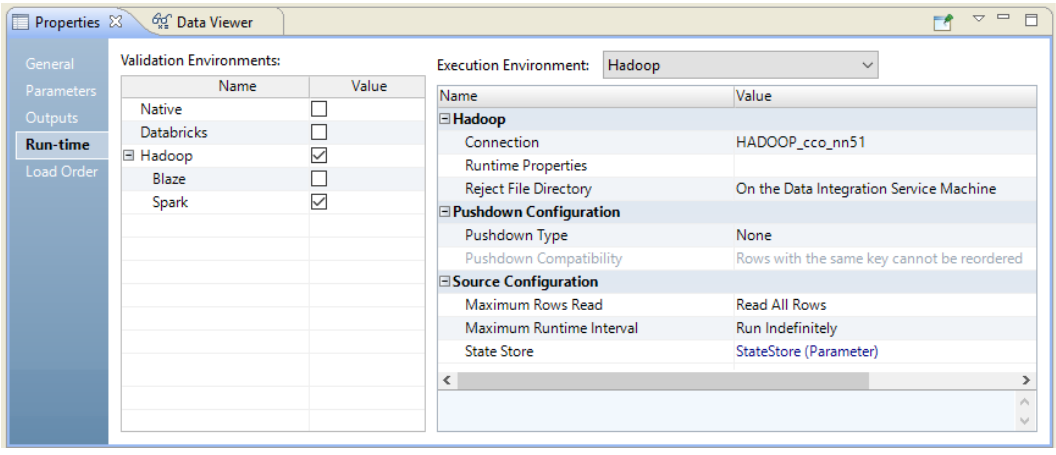
Mapping Run-time Properties

The mapping run-time properties depend on the environment that you select for the mapping.

The mapping properties contain the **Validation Environments** area and an **Execution Environment** area. The properties in the **Validation Environment** indicate whether the Developer tool validates the mapping definition for the native execution environment, the non-native execution environment, or both. When you run a mapping in the native environment, the Data Integration Service processes the mapping.

When you run a mapping in the non-native environment, the Data Integration Service pushes the mapping execution to the compute cluster through a runtime connection. The engine in the compute cluster processes the mapping.

The following image shows the mapping **Run-time** properties in a Hadoop environment:



Validation Environments

The properties in the **Validation Environments** indicate whether the Developer tool validates the mapping definition for the native or non-native execution environment.

You can configure the following properties for the **Validation Environments**:

Native

Default environment. The Data Integration Service runs the mapping in a native environment.

Hadoop

Run the mapping in the Hadoop environment. The Data Integration Service pushes the transformation logic to the Hadoop cluster through a Hadoop connection. Select the engine to process the mapping. You can select the Blaze or Spark engine.

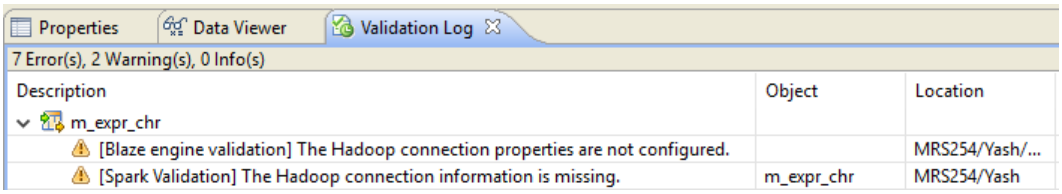
Databricks

Run the mapping in the Databricks environment. The Data Integration Service pushes the transformation logic to the Databricks cluster through a Databricks connection. The Databricks cluster processes the mapping on the Databricks Spark engine.

You can use a mapping parameter to indicate the execution environment for the mapping. When you select the execution environment, click **Assign Parameter**. Configure a string parameter. Set the default value to native, hadoop, or spark-databricks.

When you validate the mapping, validation occurs for each engine that you choose in the **Validation Environments**. The validation log might contain validation errors specific to each engine. If the mapping is valid for at least one mapping, the mapping is valid. The errors for the other engines appear in the validation log as warnings. If the mapping is valid for multiple engines, you can view the execution plan to determine which engine will run the job.

The following image shows a sample validation log:



Execution Environment

Configure non-native properties, pushdown configuration properties, and source configuration properties in the **Execution Environment** area.

The following table describes properties that you can configure for the Hadoop and Databricks environments:

Name	Description
Connection	Configure for the Hadoop and Databricks environments. Defines the connection information that the Data Integration Service requires to push the mapping execution to the compute cluster. Select the non-native connection to run the mapping in the compute cluster. You can assign a user-defined parameter for the non-native connection.
Runtime Properties	Configure for the Hadoop environment. You can configure run-time properties for the Hadoop environment in the Data Integration Service, the Hadoop connection, and in the mapping. You can override a property configured at a high level by setting the value at a lower level. For example, if you configure a property in the Data Integration Service custom properties, you can override it in the Hadoop connection or in the mapping. The Data Integration Service processes property overrides based on the following priorities: 1. Mapping custom properties set using infacmd ms runMapping with the `-cp` option 2. Mapping run-time properties for the Hadoop environment 3. Hadoop connection advanced properties for run-time engines 4. Hadoop connection advanced general properties, environment variables, and classpaths 5. Data Integration Service custom properties
Reject File Directory	Configure for the Hadoop environment. The directory for Hadoop mapping files on HDFS when you run mappings in the Hadoop environment. The Blaze engine can write reject files to the Hadoop environment for flat file, HDFS, and Hive targets. The Spark engine can write reject files to the Hadoop environment for flat file and HDFS targets. Choose one of the following options: - On the Hadoop Cluster. The reject files are moved to the reject directory configured in the Hadoop connection. If the directory is not configured, the mapping will fail. - Defer to the Hadoop Connection. The reject files are moved based on whether the reject directory is enabled in the Hadoop connection properties. If the reject directory is enabled, the reject files are moved to the reject directory configured in the Hadoop connection. Otherwise, the Data Integration Service stores the reject files based on the RejectDir system parameter.

You can configure the following pushdown configuration properties:

Name	Description
Pushdown type	Configure for the Hadoop environment. Choose one of the following options: - None. Select no pushdown type for the mapping. - Source. The Data Integration Service tries to push down transformation logic to the source database. - Full. The Data Integration Service pushes the full transformation logic to the source database.
Pushdown Compatibility	Configure for the Hadoop environment. Optionally, if you choose full pushdown optimization and the mapping contains an Update Strategy transformation, you can choose a pushdown compatibility option or assign a pushdown compatibility parameter. Choose one of the following options: - Multiple rows do not have the same key. The transformation connected to the Update Strategy transformation receives multiple rows without the same key. The Data Integration Service can push the transformation logic to the target. - Multiple rows with the same key can be reordered. The target transformation connected to the Update Strategy transformation receives multiple rows with the same key that can be reordered. The Data Integration Service can push the Update Strategy transformation to the non-native environment. - Multiple rows with the same key cannot be reordered. The target transformation connected to the Update Strategy transformation receives multiple rows with the same key that cannot be reordered. The Data Integration Service cannot push the Update Strategy transformation to the non-native environment.

You can configure the following source properties for the Hadoop and Databricks environments:

Name	Description
Maximum Rows Read	Reserved for future use.
Maximum Runtime Interval	Reserved for future use.
State Store	Reserved for future use.

Parsing JSON Records on the Spark Engines

In the mapping run-time properties, you can configure how the Spark engine parses corrupt records and multiline records when it reads from JSON sources in a mapping.

Configure the following Spark run-time properties:

infaspark.json.parser.mode

Specifies the parser how to handle corrupt JSON records. You can set the value to one of the following modes:

- DROPMALFORMED. The parser ignores all corrupted records. Default mode.
- PERMISSIVE. The parser accepts non-standard fields as nulls in corrupted records.
- FAILFAST. The parser generates an exception when it encounters a corrupted record and the Spark application goes down.

infaspark.json.parser.multiLine

Specifies whether the parser can read a multiline record in a JSON file. You can set the value to true or false. Default is false. Applies only to non-native distributions that use Spark version 2.2.x and above.

Reject File Directory

You can write reject files to the Data Integration Service machine or to the Hadoop cluster. Or, you can defer to the Hadoop connection configuration. The Blaze engine and Spark engine can write reject files to the Hadoop environment for flat file and HDFS targets.

If you configure the mapping run-time properties to defer to the Hadoop connection, the reject files for all mappings with this configuration are moved based on whether you choose to write reject files to Hadoop for the active Hadoop connection. You do not need to change the mapping run-time properties manually to change the reject file directory.

For example, if the reject files are currently moved to the Data Integration Service machine and you want to move them to the directory configured in the Hadoop connection, edit the Hadoop connection properties to write reject files to Hadoop. The reject files of all mappings that are configured to defer to the Hadoop connection are moved to the configured directory.

You might also to choose to defer to the Hadoop connection when the connection is parameterized to alternate between multiple Hadoop connections. For example, the parameter might alternate between one Hadoop connection that is configured to move reject files to the Data Integration Service machine and another Hadoop connection that is configured to move reject files to the directory configured in the Hadoop connection. If you choose to defer to the Hadoop connection, the reject files are moved depending on the active Hadoop connection in the connection parameter.

You cannot create the reject file directory at run time. Depending on the option that you choose, the configured reject file directory must exist on the Data Integration Service machine or in the Hadoop environment before you can store reject files.

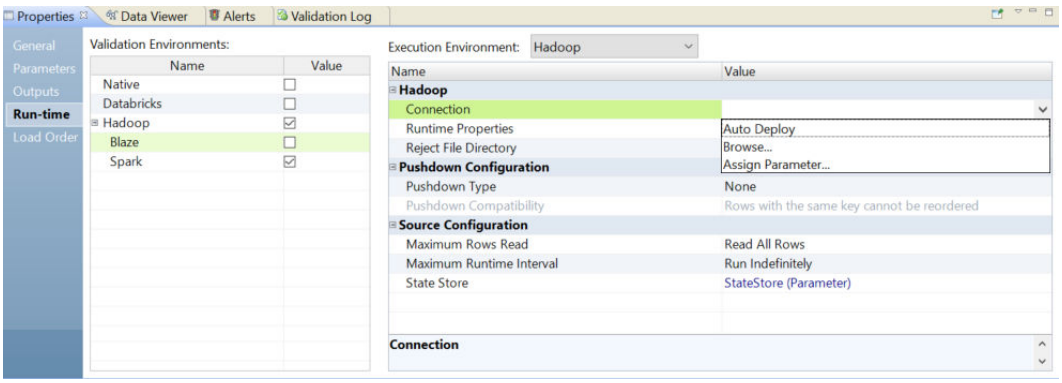
Changing the Compute Cluster for a Mapping Run

You can change the compute cluster that a mapping runs on.

You might want to change the compute cluster to run the mapping on a different compute cluster type or different version.

To change the compute cluster that a mapping runs on, click the connection where it appears in the mapping run-time properties. Then click **Browse**, and select a non-native connection that uses a cluster configuration for the cluster where you want to run the mapping.

The following image shows the **Browse** choice for a non-native connection:



After you change the connection that the mapping uses, you must restart the Data Integration Service for the change to take effect.

Updating Run-time Properties for Multiple Mappings

You can enable or disable the validation environment or set the execution environment for multiple mappings. You can update multiple mappings that you run from the Developer tool or mappings that are deployed to a Data Integration Service. Use the Command Line Interface to perform these updates.

The following table describes the commands to update mapping run-time properties:

Command	Description
dis disableMappingValidationEnvironment	Disables the mapping validation environment for mappings that are deployed to the Data Integration Service.
mrs disableMappingValidationEnvironment	Disables the mapping validation environment for mappings that you run from the Developer tool.
dis enableMappingValidationEnvironment	Enables a mapping validation environment for mappings that are deployed to the Data Integration Service.
mrs enableMappingValidationEnvironment	Enables a mapping validation environment for mappings that you run from the Developer tool.
dis setMappingExecutionEnvironment	Specifies the mapping execution environment for mappings that are deployed to the Data Integration Service.
mrs setMappingExecutionEnvironment	Specifies the mapping execution environment for mappings that you run from the Developer tool.

PreSQL and PostSQL Queries for JDBC Sources

When you create a mapping to access JDBC storage resources, you can use SQL statements to execute SQL commands such as insert, update, and delete on JDBC source tables.

JDBC sources include Hadoop and Databricks sources that use the JDBC connection.

You can create a PreSQL or PostSQL statement that contains one command or a series of commands. To create an SQL statement with more than one command, write the SQL statement with the following delimiter between each command:

```
{INFA_SQL_SPLITTER}
```

PreSQL Queries for JDBC Sources

PreSQL is an SQL command that runs against the JDBC source before the mapping reads from the source. You can input the SQL statement in the Read transformation advanced properties.

For example, the following PreSQL statement contains three commands separated by the `{INFA_SQL_SPLITTER}` delimiter:

```
INSERT INTO 'default'.employee_csv_us_src VALUES
( 201, 'PRE_SQL_AA-INSERT', 1000.021, 'SWE-1',
11,111111.132412341,'2019-11-12','2019-12-01 01:01:01.111'){INFA_SQL_SPLITTER}
INSERT INTO 'default'.employee_csv_us_src VALUES
```

```
( 202, 'PRE_SQL_BB-INSERT', 1000.021, 'SWE-2',
11,111111.132412341,'2019-11-12','2019-12-01 01:01:01.111'){INFA_SQL_SPLITTER}
INSERT INTO 'default'.employee_csv_us_src VALUES
( 203, 'PRE_SQL_CC-INSERT', 1000.021, 'SWE-3',
11,111111.132412341,'2019-11-12','2019-12-01 01:01:01.111'){INFA_SQL_SPLITTER}
```

PostSQL Queries for JDBC Targets

PostSQL is an SQL command that runs against the JDBC target after the mapping writes to the target. You can input the SQL statement in the Write transformation advanced properties.

For example, the following PostSQL statement contains three commands separated by the `{INFA_SQL_SPLITTER}` delimiter:

```
INSERT INTO 'default'.employee_csv_us_src VALUES
( 204, 'POST_SQL_AA-INSERT', 1000.021, 'SWE-1',
11,111111.132412341,'2019-11-12','2019-12-01 01:01:01.111'){INFA_SQL_SPLITTER}
INSERT INTO 'default'.employee_csv_us_src VALUES
( 205, 'POST_SQL_BB-INSERT', 1000.021, 'SWE-2',
11,111111.132412341,'2019-11-12','2019-12-01 01:01:01.111'){INFA_SQL_SPLITTER}
INSERT INTO 'default'.employee_csv_us_src VALUES
( 206, 'POST_SQL_CC-INSERT', 1000.021, 'SWE-3',
11,111111.132412341,'2019-11-12','2019-12-01 01:01:01.111'){INFA_SQL_SPLITTER}
```

Sqoop Mappings in a Hadoop Environment

You can use a JDBC connection that is enabled for Sqoop connectivity to import a Sqoop source or Sqoop target and create a mapping. You can run Sqoop mappings on the Blaze and Spark engines.

If you use Cloudera Connector Powered by Teradata or Hortonworks Connector for Teradata, you can run mappings on the Blaze or Spark engines. If you use MapR Connector for Teradata, you can run mappings on the Spark engine.

In the mapping, you can specify additional Sqoop arguments and disable the Sqoop connector.

Note: If you add or delete a Type 4 JDBC driver .jar file required for Sqoop connectivity from the `externaljdbcjars` directory, changes take effect after you restart the Data Integration Service. If you run the mapping on the Blaze engine, changes take effect after you restart the Data Integration Service and Blaze Grid Manager. When you run the mapping for the first time, you do not need to restart the Data Integration Service and Blaze Grid Manager. You need to restart the Data Integration Service and Blaze Grid Manager only for the subsequent mapping runs.

Sqoop Mapping-Level Arguments

If a data object uses Sqoop, you can click the corresponding **Read** transformation or **Write** transformation in the Sqoop mapping to define the arguments that Sqoop must use to process the data. The Data Integration Service merges the additional Sqoop arguments that you specify in the mapping with the arguments that you specified in the JDBC connection and constructs the Sqoop command.

The Sqoop arguments that you specify in the mapping take precedence over the arguments that you specified in the JDBC connection. However, if you do not enable the Sqoop connector in the JDBC connection but enable the Sqoop connector in the mapping, the Data Integration Service does not run the mapping through Sqoop. The Data Integration Service runs the mapping through JDBC.

You can configure the following Sqoop arguments in a Sqoop mapping:

- m or num-mappers

- split-by
- batch
- infaoptimize
- infaownername
- schema
- verbose

For a complete list of the Sqoop arguments that you can configure, see the Sqoop documentation.

m or num-mappers

The m or num-mappers argument defines the number of map tasks that Sqoop must use to import and export data in parallel.

Use the following syntax:

```
-m <number of map tasks>
```

```
--num-mappers <number of map tasks>
```

If you configure the m argument or num-mappers argument, you must also configure the split-by argument to specify the column based on which Sqoop must split the work units.

Use the m argument or num-mappers argument to increase the degree of parallelism. You might have to test different values for optimal performance.

When you configure the m argument or num-mappers argument and run Sqoop mappings on the Spark or Blaze engines, Sqoop dynamically creates partitions based on the file size.

Note: If you configure the num-mappers argument to export data on the Blaze or Spark engine, Sqoop ignores the argument. Sqoop creates map tasks based on the number of intermediate files that the Blaze or Spark engine creates.

split-by

The split-by argument defines the column based on which Sqoop splits work units.

Use the following syntax:

```
--split-by <column_name>
```

You can configure the split-by argument to improve the performance. If the primary key does not have an even distribution of values between the minimum and maximum range, you can configure the split-by argument to specify another column that has a balanced distribution of data to split the work units.

If you do not define the split-by column, Sqoop splits work units based on the following criteria:

- If the data object contains a single primary key, Sqoop uses the primary key as the split-by column.
- If the data object contains a composite primary key, Sqoop defaults to the behavior of handling composite primary keys without the split-by argument. See the Sqoop documentation for more information.
- If the data object does not contain a primary key, the value of the m argument and num-mappers argument default to 1.

Rules and Guidelines for the split-by Argument

Consider the following restrictions when you configure the split-by argument:

- If you configure the split-by argument and the split-by column contains NULL values, Sqoop does not import the rows that contain NULL values. However, the mapping runs successfully and no error is written in the YARN log.
- If you configure the split-by argument and the split-by column contains special characters, the Sqoop import process fails.
- The split-by argument is required in the following scenarios:
  - You use Cloudera Connector Powered by Teradata or Hortonworks Connector for Teradata, and the Teradata table does not contain a primary key.
  - You create a custom query to override the default query when you import data from a Sqoop source.

batch

The batch argument indicates that Sqoop must export data in batches instead of exporting data row by row.

Use the following syntax:

```
--batch
```

You can configure the batch argument to improve the performance.

The batch argument is required when you configure Sqoop in the following scenarios:

- You write data to a Vertica target.
- You write data to an IBM DB2 for z/OS target.
- You write data to a Teradata target by running the mapping with a generic JDBC connector instead of a TDCH specialized connector for Sqoop. You can invoke the generic JDBC connector by configuring the --driver and --connection-manager Sqoop arguments in the JDBC connection.

infaoptimize

The infaoptimize argument defines whether you want to disable the performance optimization of Sqoop pass-through mappings on the Spark engine.

When you run a Sqoop pass-through mapping on the Spark engine, the Data Integration Service optimizes mapping performance in the following scenarios:

- You read data from a Sqoop source and write data to a Hive target that uses the Text format.
- You read data from a Sqoop source and write data to an HDFS target that uses the Flat, Avro, or Parquet format.

If you want to disable the performance optimization, set the --infaoptimize argument to false. For example, if you see data type issues after you run an optimized Sqoop mapping, you can disable the performance optimization.

Use the following syntax:

```
--infaoptimize false
```

infaownername

The infaownername argument indicates whether Sqoop must honor the owner name for a data object.

To configure Sqoop to honor the owner name for a data object, set the value of the infaownername argument to true.

Use the following syntax:

```
--infaownername true
```

Note: To honor the owner name for a Microsoft SQL Server database, you must configure the -- --schema argument and specify the owner name. Use the following syntax:

```
-- --schema <owner_name>
```

schema

The schema argument indicates whether Sqoop must honor the owner name for a data object that represents a Microsoft SQL Server database.

To honor the owner name for a Microsoft SQL Server source database, specify the owner name in the **Additional Sqoop Import Arguments** field in the Sqoop mapping. Use the following syntax:

```
-- --schema <owner_name>
```

To honor the owner name for a Microsoft SQL Server target database, you must leave the **Owner** field blank in the Sqoop target data object. Specify the owner name in the **Additional Sqoop Export Arguments** field in the Sqoop mapping. Use the following syntax:

```
-- --schema <owner_name>
```

verbose

Configure the --verbose argument when you want detailed information in the logs for debugging purposes.

Use the following syntax:

```
--verbose
```

Note: If you configure the --verbose argument, you might experience significant performance degradation, especially if you run the Sqoop mapping on the Spark engine.

Incremental Data Extraction for Sqoop Mappings

You can configure a Sqoop mapping to perform incremental data extraction based on an ID or timestamp based source column. With incremental data extraction, Sqoop extracts only the data that changed since the last data extraction. Incremental data extraction increases the mapping performance.

To perform incremental data extraction, configure arguments in the **Additional Sqoop Import Arguments** text box of the Read transformation in the Sqoop mapping.

You must configure all of the following arguments for incremental data extraction:

--infa-incremental-type

Indicates whether you want to perform the incremental data extraction based on an ID or timestamp column.

If the source table contains an ID column that acts as a primary key or unique key column, you can configure incremental data extraction based on the ID column. Set the value of the --infa-incremental-type argument as `ID`. Sqoop extracts rows whose IDs are greater than the last extracted ID.

If the source table contains a timestamp column that contains the last updated time for all rows, you can configure incremental data extraction based on the timestamp column. Set the value of the --infa-incremental-type argument as `timestamp`. Sqoop extracts rows whose timestamps are greater than the last read timestamp value or the maximum timestamp value.

Use the following syntax:

```
--infa-incremental-type <ID or timestamp>
```

--infa-incremental-key

Indicates the column name based on which Sqoop must perform the incremental data extraction.

Use the following syntax:

```
--infa-incremental-key <column_name>
```

--infa-incremental-value

Indicates the column value that Sqoop must use as the baseline value to perform the incremental data extraction. Sqoop extracts all rows that have a value greater than the value defined in the --infa-incremental-value argument.

Enclose the column value within double quotes.

Use the following syntax:

```
--infa-incremental-value "<column_value>"
```

--infa-incremental-value-format

Applicable if you configure incremental data extraction based on a timestamp column. Indicates the timestamp format of the column value defined in the --infa-incremental-value argument.

Enclose the format value within double quotes.

Use the following syntax:

```
--infa-incremental-value-format "<format>"
```

Default is "MM/dd/yyyy HH:mm:ss.SSSSSSSSS".

Note: If you want to perform incremental data extraction but you do not configure all the required arguments, the Sqoop mapping fails. The argument values are not case sensitive.

Example

The Sales department in your organization stores information related to new customers in Salesforce. You want to create a Sqoop mapping to read customer records that were created after a certain date and write the data to SAP.

In the Sqoop mapping, you can define the Sqoop import arguments as follows:

```
--infa-incremental-type timestamp --infa-incremental-key CreatedDate --infa-incremental-value
"10/20/2018 00:00:00.000000000" --infa-incremental-value-format "MM/dd/yyyy
HH:mm:ss.SSSSSSSSS"
```

The Data Integration Service reads all customer records whose `CreatedDate` column contains timestamp values greater than `10/20/2018 00:00:00.000000000`. The Sqoop mapping uses the `10/20/2018 00:00:00.000000000` format while reading data.

Configuring Sqoop Properties in the Mapping

You can specify additional Sqoop arguments and disable the Sqoop connector at the mapping level. The Sqoop arguments that you specify at the mapping level take precedence over the arguments that you specified in the JDBC connection.

1. Open the mapping that contains the data object for which you want to configure Sqoop properties.
2. Select the Read or Write transformation that is associated with the data object.
3. Click the **Advanced** tab.
4. To disable the Sqoop connector for the data object, select the **Disable Sqoop Connector** check box.

   A mapping that contains an Update Strategy transformation cannot use a Sqoop-enabled JDBC connection to write to a target. To run the mapping, disable the Sqoop connector in the Write transformation.
5. Perform one of the following steps:
   - To specify additional Sqoop import arguments for the data object, enter the import arguments in the **Additional Sqoop Import Arguments** text box.
   - To specify additional Sqoop export arguments for the data object, enter the export arguments in the **Additional Sqoop Export Arguments** text box.

   The Data Integration Service merges the additional Sqoop arguments that you specified in the mapping with the arguments that you specified in the JDBC connection and constructs the Sqoop command. The Data Integration Service then invokes Sqoop on a Hadoop node.

Configuring Parameters for Sqoop Arguments in the Mapping

In a Sqoop mapping, you can parameterize the Sqoop import and export arguments.

1. Open the Sqoop mapping where you want to parameterize the Sqoop arguments.
2. Click the **Parameters** tab, and define a parameter name and parameter value.

   For example, enter the parameter name as `Param_splitByColumn`
3. Select the Read or Write transformation that is associated with the data object.
4. Click the **Advanced** tab.
5. Perform one of the following steps:
   - To parameterize Sqoop import arguments for the data object, click the **Additional Sqoop Import Arguments** text box and enter the parameter name you defined in the **Parameters** tab.
   - To parameterize Sqoop export arguments for the data object, click the **Additional Sqoop Export Arguments** text box and enter the parameter name you defined in the **Parameters** tab.

   Use the following format to specify the parameter name for a Sqoop import or export argument:

   `--<Sqoop_argument_name> $<parameter_name>`

   For example, enter `--split-by $Param_splitByColumn`

Mapping Output Binding

You can configure mapping outputs for standalone mappings or mappings that are a part of a workflow. You can bind persisted mapping outputs to use them in subsequent runs of the mapping or workflow.

Standalone Deployed Mappings

You can persist a mapping output in the Model repository for mappings that run in the native environment or on the Spark engine. You can bind a mapping output to a mapping parameter to pass the value of the output to the mapping the next time you run the mapping.

For more information about mapping outputs in standalone mappings, see the "Mapping Outputs" chapter in the *Developer Mapping Guide*.

Mapping Tasks in Workflows

If you add a mapping to a workflow that runs on the Spark or Blaze engine, you can configure mapping outputs. You can persist a mapping output in the Model repository to assign the persisted output to a Mapping task input, or you can bind a mapping output to a workflow variable to pass the value of the output to other tasks in the workflow. The mappings that run on these engines provide the total number of target, source, and error rows.

Note: You cannot use the SUM aggregation type in mapping outputs on the Spark engine.

For more information about mapping outputs in workflow Mapping tasks, see the "Mapping Task" chapter in the *Developer Workflow Guide*.

Rules and Guidelines for Mappings in a Non-native Environment

When you run mappings in a non-native environment, some differences in processing and configuration apply. You can run mappings in a non-native environment on the Blaze, Spark, or Databricks Spark engine. Consider the following rules and guidelines when you run mappings in the native and non-native environments.

Rules and Guidelines for Mappings on the Blaze Engine

Consider the following run-time differences on the Blaze engine:

- In a Hadoop environment, sources that have data errors in a column result in a null value for the column. In the native environment, the Data Integration Service does not process the rows that have data errors in a column.
- When you cancel a mapping that reads from a flat file source, the file copy process that copies flat file data to HDFS may continue to run. The Data Integration Service logs the command to kill this process in the Hive session log, and cleans up any data copied to HDFS. Optionally, you can run the command to kill the file copy process.
- If you use Hive 2.0 or higher, the Blaze engine guarantees scale values. For example, when the Blaze engine processes the decimal `1.1234567` with scale 9 using Hive 2.0, the output is `1.123456700`.

Rules and Guidelines for Mappings on the Spark Engine

Consider the following run-time differences on the Spark engine:

Mapping Validation

Consider the following rules and guidelines for mapping validation:

- Mapping validation fails if you configure SYSTIMESTAMP with a variable value, such as a port name. The function can either include no argument or the precision to which you want to retrieve the timestamp value.
- Mapping validation fails if an output port contains a Timestamp with Time Zone data type.

Optimization

- Set the optimizer level to none or minimal if a mapping validates but fails to run. If you set the optimizer level to use cost-based or semi-join optimization methods, the Data Integration Service ignores this at run-time and uses the default.
- The run-time engine does not honor the early projection optimization method in all cases. If the Data Integration Service removes the links between unused ports, the run-time engine might reconnect the ports.
- When you use the auto optimizer level, the early selection optimization method is enabled if the mapping contains any data source that supports pushing filters to the source on Spark or Databricks Spark engines. For more information about optimizer levels, see the *Informatica Developer Mapping Guide*.
- When the Spark engine runs a mapping, it processes jobs on the cluster using HiveServer2 in the following cases:
  - The mapping writes to a target that is a Hive table bucketed on fields of type char or varchar.
  - The mapping reads from or writes to Hive transaction-enabled tables.
  - The mapping reads from or writes to Hive tables where column-level security is enabled.
  - The mapping writes to a Hive target and is configured to create or replace the table at run time.

High Precision

Consider the following rules and guidelines for mappings that run in high precision or low precision mode:

- When you use the TO_DECIMAL or TO_DECIMAL38 function in a mapping that runs in high precision mode, you must specify a scale argument. If the mapping runs in low precision mode, the Spark engine ignores the scale argument and returns a double.
- If you enable high-precision in a streaming mapping, the Spark engine runs the mapping in low-precision mode.
- If the mapping contains a complex port with an element of a decimal data type, the Spark engine runs the mapping in low-precision mode.

Overflow Values

The Spark engine and the Data Integration Service process overflow values differently. The Spark engine processing rules might differ from the rules that the Data Integration Service uses. As a result, mapping results can vary between the native and Hadoop environment when the Spark engine processes an overflow.

Null and Invalid Values and Rejected Rows

Consider the following guidelines when mappings pass null or invalid values for rows:

- The Spark run-time engine drops rejected rows even if you configure the mapping to forward rejected rows. The rejected rows are not written to the session log file.

- If an expression results in numerical errors, such as division by zero or SQRT of a negative number, the Spark engine returns null. In the native environment, the same expression results in a row error.
- The Hadoop environment treats "/n" values as null values. If an aggregate function contains empty or NULL values, the Hadoop environment includes these values while performing an aggregate calculation.
- The Spark engine writes null values for rows when invalid values are passed in the following situations:
  - The *terms* argument in PV, FV, PMT, and RATE finance functions passes a 0 value. The value of *terms* must be an integer greater than 0.
  - The *month* argument in the MAKE_DATE_TIME function passes an invalid value. The value of *month* must be from 1 to 12.

  In the native environment, the Data Integration Service rejects the row and does not write it to the target.
- If data overflow occurs, the Spark engine returns null. For example, if you use the expression `TO_DECIMAL(12.34, 2)` in a port that has a precision of 3 and a scale of 2, the return value is null. The null value will be propagated through the mapping. The mapping might overwrite it using a default value, detect it using the function IS_NULL, and write the null value to the target.

Data Conversions

The Spark engine and the Data Integration Service process data type conversions differently. As a result, mapping results can vary between the native and Hadoop environment when the Spark engine performs a data type conversion. Consider the following processing variations for Spark:

- The results of arithmetic operations on floating point types, such as Decimal, can vary between the native environment and a Hadoop environment. The difference between the results can increase across multiple operations.
- When the number of fractional digits in a double or decimal value exceeds the scale that is configured in a decimal port, the Spark engine trims trailing digits, rounding the value if necessary.
- If you use Hive 2.0 or higher, the Spark engine guarantees scale values. For example, when the Spark engine processes the decimal `1.1234567` with scale 9 using Hive 2.0, the output is `1.123456700`.
- The Spark engine cannot process dates to the nanosecond. It can return a precision for date/time data up to the microsecond.

Scale

The Spark engine and the Data Integration Service process scale differently. The Data Integration Service allows scale to differ between rows of decimal data while the Spark engine uses a fixed scale for each row. Because the scale is fixed, arithmetic operations can result in data overflow.

For example, the arithmetic operation dec(38,0) / dec(10,0) outputs a decimal dec(38,6) on the Spark engine. The operation might result in data overflow based on whether the result can be represented as a decimal dec(38,6).

The following table shows the decimal values of dec(10,0) that result in data overflow for several decimal values of dec(38,0):

Decimal Value of dec(38,0)	Decimal Values of dec(10,0) that Result in Data Overflow
Less than 10^32	None
10^32	Decimals with an absolute value that is less than or equal to 1

Decimal Value of dec(38,0)	Decimal Values of dec(10,0) that Result in Data Overflow
10^33	Decimals with an absolute value that is less than or equal to 10
10^34	Decimals with an absolute value that is less than or equal to 100

Function

Consider the following rules and guidelines for various functions:

- Avoid including single and nested functions in an Aggregator transformation. The Data Integration Service fails the mapping in the native environment. It can push the processing to the Hadoop environment, but you might get unexpected results. Informatica recommends creating multiple transformations to perform the aggregation.
- The Spark METAPHONE function uses phonetic encoders from the `org.apache.commons.codec.language` library. When the Spark engine runs a mapping, the METAPHONE function can produce an output that is different from the output in the native environment. The following table shows some examples:

String	Data Integration Service	Spark Engine
Might	MFT	MT
High	HF	H

- If you use the TO_DATE function on the Spark engine to process a string written in ISO standard format, you must add `*T*` to the date string and `*"T"*` to the format string. The following expression shows an example that uses the TO_DATE function to convert a string written in the ISO standard format YYYY-MM-DDTHH24:MI:SS:

  ```
  TO_DATE('2017-11-03*T*12:45:00','YYYY-MM-DD*"T"*HH24:MI:SS')
  ```

  The following table shows how the function converts the string:

ISO Standard Format	RETURN VALUE
2017-11-03T12:45:00	Nov 03 2017 12:45:00

- The UUID4 function is supported only when used as an argument in UUID_UNPARSE or ENC_BASE64.
- The UUID_UNPARSE function is supported only when the argument is UUID4( ).

Other Rules and Guidelines

You cannot preview data for a transformation that is configured for windowing.

When the Spark engine runs a mapping, it processes jobs on the cluster using HiveServer2 in the following cases:

- The mapping writes to a target that is a Hive table bucketed on fields of type char or varchar.
- The mapping reads from or writes to Hive transaction-enabled tables.
- The mapping reads from or writes to Hive tables where column-level security is enabled.
- The mapping writes to a Hive target and is configured to create or replace the table at run time.

Rules and Guidelines for Mappings on the Databricks Spark Engine

A non-native environment like Databricks Spark uses distributed processing and processes data on different nodes. Each node does not have access to the data that is being processed on other nodes. As a result, the runtime engine might not be able to determine the order in which the data originated. So, when you run a mapping in a non-native environment and you run the same mapping in the native environment, both mappings return correct results, but the results might not be identical. For more information, see the Developer Transformation Guide.

Consider the following run-time differences on the Databricks Spark engine:

- When you use the auto optimizer level, the early selection optimization method is enabled if the mapping contains any data source that supports pushing filters to the source on Spark or Databricks Spark engines. For more information about optimizer levels, see the *Informatica Developer Mapping Guide*.
- Databricks Spark performs auto-optimization on jobs based on configuration settings for the cluster. If you set optimization through the Spark.default.parallelism property, the Databricks Spark engine ignores this setting. As a consequence, it is not possible to configure optimization on the job level.

Workflows that Run Mappings in a Non-native Environment

You can add a mapping that you configured to run in a non-native environment to a Mapping task in a workflow. When you deploy and run the workflow, the Mapping task runs the mapping.

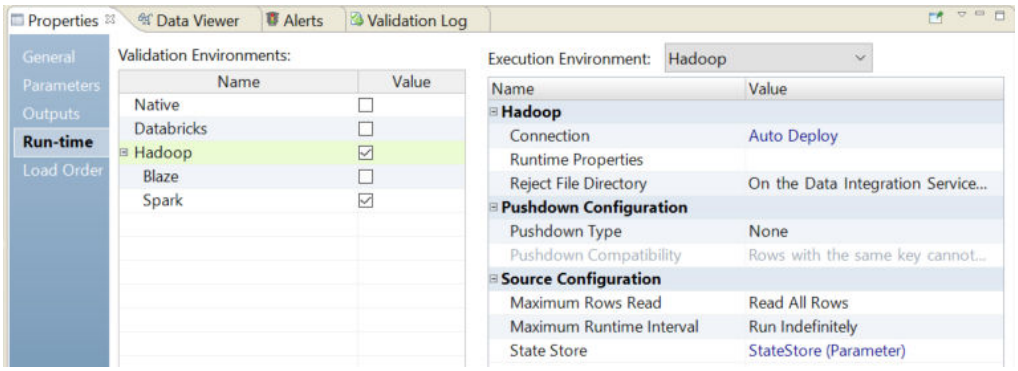
You might decide to run a mapping from a workflow so that you can make decisions during the workflow run. You can configure a workflow to run multiple mappings in sequence or in parallel. You can configure a workflow to send emails that notify users about the status of the Mapping tasks.

Configuring a Mapping to Run in a Non-native Environment

You can configure a mapping to run in a non-native environment. To configure a mapping, you must select a validation environment and an execution environment.

1. Select a mapping from a project or folder from the **Object Explorer** view to open in the editor.
2. In the **Properties** view, select the **Run-time** tab.
3. Select **Hadoop**, **Databricks**, or both as the value for the validation environment.

   When you select the Hadoop environment, the Blaze and Spark engines are selected by default. Disable the engines that you do not want to use.
4. Select **Hadoop** or **Databricks** for the execution environment.



5. Select **Connection** and use the drop down in the value field to browse for a connection or to create a connection parameter:
   - To select a connection, click **Browse** and select a connection.
   - To create a connection parameter, click **Assign Parameter**.
6. Configure the rest of the properties for the execution environment.
7. Right-click an empty area in the editor and click **Validate**.

   The Developer tool validates the mapping.
8. View validation errors on the **Validation Log** tab.
9. To view the execution plan for the mapping, right-click the editor and select **Show Execution Plan**.

Configure Mappings to Run on Dataproc

To run mappings on the Dataproc cluster, configure mappings with the following properties:

1. In the **Parameters** section, create a parameter with the values shown in the following table:

Name	Type	Precision	Scale	Default Value
pushdownConnection	Connection	(blank)	(blank)	Use the dropdown control to select the Hadoop connection that the cluster configuration created.

2. In the **Run-Time** section, choose the following values:
   - Under Validation Environments, select Spark.
   - For Execution Environment, select Hadoop.
   - Under the Hadoop section, for the Connection property, click the right side of the Value cell to see the dropdown menu, and choose the pushdownConnection parameter.

Audits

To validate the consistency and accuracy of data processed in a mapping, you can create an audit for the mapping.

An audit is composed of rules and conditions. Use an audit rule to compute an aggregated value for a single column of data in the source or target. Use an audit condition to compare multiple rules or a rule and constant values. You can run audits in the native and Hadoop environments.

Note: Running an audit on the Blaze engine is available for technical preview only. Technical preview functionality is supported for evaluation purposes but is unwarranted and is not supported in production environments or any environment that you plan to push to production. Informatica intends to include the preview functionality in an upcoming release for production use, but might choose not to in accordance with changing market or technical circumstances. For more information, contact Informatica Global Customer Support.

You can configure audit rules for data sources and targets that use the following connectors:

Connector	Source	Target
Amazon S3	Supported.	Not supported.
HDFS	Supported.	Not supported.
Hive	Supported.	Supported.
JDBC V2	Supported.	Not supported.
Microsoft Azure Synapse	Supported.	Not supported.
Oracle	Supported.	Supported.
Snowflake	Supported.	Not supported.

Example

You are a data engineer for an e-commerce store. You have a data table with clothing sales information, including the sale price. You want to audit the data and verify that there are no outliers. You create a mapping that uses the clothing sales table as the source. You configure audit rules to calculate the maximum and minimum sales prices of the clothing from the Read transformation. After the mapping runs, you verify that the maximum and minimum sales prices are within the range you expect.

Creating an Audit

To configure an audit, perform the following tasks:

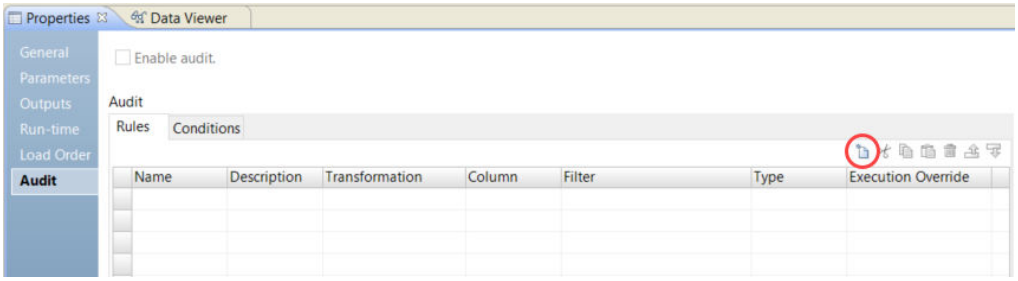
1. Define a rule.
2. Optionally, define conditions.
3. Configure a storage location for the audit results.
4. Run and monitor the mapping.
5. View the audit results.

Step 1. Define Audit Rules

An audit rule calculates an aggregate value for a single field in either a Read transformation or a Write transformation. Each rule returns a single numeric result.

1. In the mapping audit properties, select the **Rules** tab.
2. Select the **New (Insert)** icon.

The following image shows the icon:



A rule appears with default values.

3. Configure the following properties for a rule:

Property	Description
Name	Required. Name of the rule.
Description	Optional. Description of the rule.
Transformation	Required. Select a Read transformation or a Write transformation.
Column	Required. Port from the configured transformation to apply the rule to. You can configure a mapping parameter for the column.
Filter	Optional. Filter expression to apply to the column before calculating the aggregate value. The filter should return a boolean value for each row. Use transformation language to configure the filter. You can configure a mapping parameter for the filter. **Note:** You cannot use aggregate functions in the filter.
Type	Required. Aggregate type to calculate. You can configure a mapping parameter for the type. You can use the following types for any column: - Count - Distinct Count - Null Count You can use the following types for numeric columns: - Average - Max - Median - Min - Percentile - Standard Deviation - Sum - Variance
Execution Override	Optional. Determines whether the Data Integration Service runs a rule before or after it runs the mapping logic. When **Execution Override** is set to *Default*, the Data Integration Service runs rules for the Read transformation before the mapping logic and rules for the Write transformation after the mapping logic. To run a Write transformation rule before the mapping logic executes, set the value to *Pre Mapping*. To run a Read transformation rule after the mapping logic executes, set the value to *Post Mapping*. You can configure a mapping parameter for the execution override.

For example, you can check the average value of a column in the target both before and after the mapping runs. Create two rules with the same transformation, column, and filter. Set **Type** to *Average*. Set **Execution Override** to *Default* for one rule and *Pre Mapping* for the second rule.

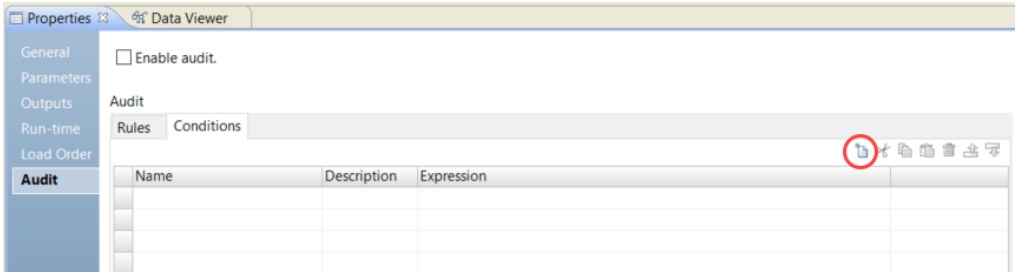
Step 2. Define Audit Conditions

An audit condition compares the results of multiple rules or the result of a rule and constant values. Each condition returns a value of either *Passed* or *Failed*.

You must define at least one rule before you can define a condition.

1. In the mapping audit properties, select the **Conditions** tab.
2. Select the **New (Insert)** icon.

   The following image shows the icon:



   A condition appears with default values.
3. Configure the following properties:

Property	Description
Name	Required. Name of the condition.
Description	Optional. Description of the condition.
Expression	Required. Conditional expression that uses an audit rule and returns a boolean value. Use transformation language to configure the expression. The expression is not case-sensitive.

Step 3. Configure a Storage Location

Set a storage location for the audit results before you run a mapping with audit rules and conditions.

Configure the storage location in the mapping audit properties. You can store the results in a local file, a database that uses a JDBC connection, or both.

Configure local file storage

To store the results in a local file, configure the following properties:

- **File directory.** Local directory to store results file.
- **File name.** Name of the file to store the results. If the file does not exist, it is created at run time.
- **Append.** Select this option to append the results to an existing file. If you do not choose to append the results, the Data Integration Service overwrites any results in the file when the mapping runs.

Configure JDBC connection storage

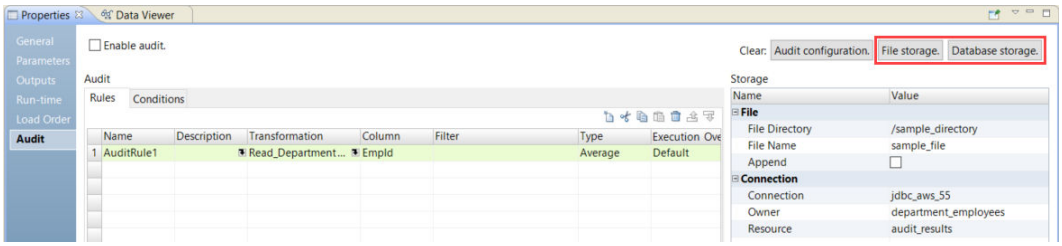
To store the results in a database that uses a JDBC connection, perform the following steps:

1. Verify that you have a JDBC connection defined for the database where you want to store the audit results.
2. Create the resource table to store the results. Use the following schema to create the table:

```
CREATE TABLE <table name>
(
RULE_NAME VARCHAR2 (128),
RESULT VARCHAR2 (128),
EXECUTION_ID VARCHAR2 (128),
DIS_NAME VARCHAR2 (128),
USER_NAME VARCHAR2 (128),
MAPPING_NAME VARCHAR2 (128),
TIMESTAMP VARCHAR2 (29)
)
```

3. Configure the following connection storage properties:
   - **Connection.** Select the JDBC connection.
   - **Owner.** Name of the database that includes the table.
   - **Resource.** Name of the table you created to store the results.

To clear the local file storage configuration, click the **File storage** clear button. To clear the JDBC connection storage configuration, click the **Database storage** clear button. The following image shows these buttons:

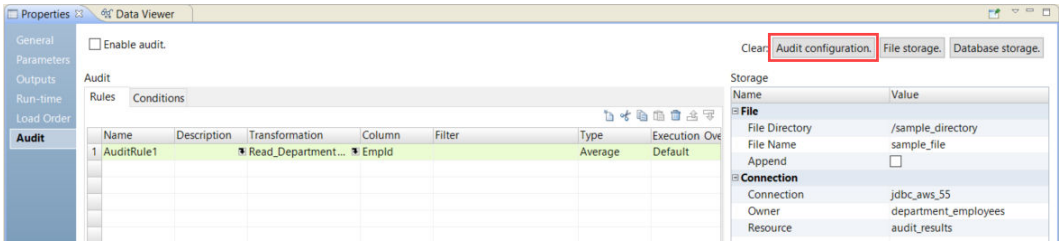


Step 4. Run and Monitor the Mapping

Run the mapping in the Developer tool and monitor the results in the Administrator tool.

1. Select the **Enable Audit** checkbox in the mapping audit properties.
2. Run the mapping.
3. In the Administrator tool, monitor the mapping and audit jobs. For more information, see the *Informatica Administrator Guide*.

If you want to clear the configured audit, including all rules, conditions, and storage configurations, click the **Audit configuration** clear button. The following image shows this button:



Step 5. View Audit Results

You can view the results of the audit rules and conditions in the configured storage location and in the **Audit Viewer** tab. You can view rule and condition results in the **Audit Viewer** and more detailed information in the configured storage location.

Audit results in the Audit Viewer

After the mapping runs, the **Audit Viewer** tab becomes available in the Developer tool. The **Audit Viewer** contains tabs for **Rules** and **Conditions**. On the **Rules** tab, you can see the result value and configured properties for each rule. On the **Conditions** tab, you can see the result value and configured properties for each condition.

Audit results in storage

In the configured storage location, you can view the following information for each rule and condition:

- Rule or condition name
- Result value
- Execution ID
- Data Integration Service name
- User name
- Mapping name
- Timestamp

Rules and Guidelines for Audits

Consider the following rules and guidelines when you configure an audit for a mapping:

- You cannot configure an audit for an optimized mapping.
- If you create an audit rule for a Write transformation that is created from an existing transformation, the ports of the Write transformation must be linked by name.
- If you use a mapplet for the source or target in a mapping, you cannot select the mapplet as the transformation for an audit rule.
- You cannot configure audit rules for a flat file target that has a footer command.
- If you define an audit rule for a Write transformation that creates a target data object at run time and you set the **Execution Override** to *Pre Mapping*, the audit fails the first time the mapping runs. On subsequent runs, the audit results are calculated.
- You cannot run a mapping with an audit when you configure the Data Integration Service to use operating system profiles.

- If you configure JDBC connection storage and the resource table schema is not correct, the mapping and audit run, but the Data Integration Service does not write the results to the table. The audit logs contain the following error message: `SEVERE: [WRT_8425] ERROR: Writer execution failed.`

Example

You are a data architect for an automobile insurance company. Each local branch maintains a database of all their insurance claims. You need to pull collision information from two branches into a single dataset and then verify the accuracy of the data that you moved.

Both input tables contain information about each insurance claim, including a ClaimType. ClaimType is a string value that can be one of a select number of values, including Collision, Property Damage, and Theft. You have a mapping that reads all claim data from each of the branches and writes collision data only to a single target.

Perform the following tasks to audit this data:

1. Define rules.

You first need to calculate the count of collision incidents from each of the sources and the target.

The following table shows the rules that you create:

Name	Transformation	Column	Filter	Type	Execution Override
Branch1_Collisions	Read_Branch1	ClaimType	claim_type = "Collision"	COUNT	Default
Branch2_Collisions	Read_Branch2	ClaimType	claim_type = "Collision"	COUNT	Default
Combined_Collisions	Write_Combined	ClaimType		COUNT	Default

2. Define a condition.

To verify that the combined table contains all collisions from both sources, create a condition to compare the count of the input rows to the count of the output rows.

The following table shows the condition that you create:

Name	Expression
Collision_count_match	Branch1_Collisions + Branch2_Collisions = Combined_Collisions

3. Configure the storage location.

You configure a storage location for the audit results.

4. Run and monitor the mapping.

You select the **Enable Audit** checkbox and then run the mapping.

5. View the results.

In the **Audit Viewer**, you see the following results for the audit rules:

Result	Name	Transformation	Column	Filter	Type	Execution Override
215	Branch1_Collisions	Read_Branch1	ClaimType	claim_type = "Collision"	COUNT	Default
324	Branch2_Collisions	Read_Branch2	ClaimType	claim_type = "Collision"	COUNT	Default
539	Combined_Collisions	Write_Combined	ClaimType		COUNT	Default

You see the following results for the audit condition:

Result	Name	Expression
Passed	Collision_count_match	Branch1_Collisions + Branch2_Collisions = Combined_Collisions

Mapping Execution Plans

The Data Integration Service generates an execution plan to run mappings on a Blaze, Spark, or Databricks Spark engine. The Data Integration Service translates the mapping logic into code that the run-time engine can execute. You can view the plan in the Developer tool before you run the mapping and in the Administrator tool after you run the mapping.

The Data Integration Service generates mapping execution plans to run on the following engines:

Informatica Blaze engine

The Blaze engine execution plan simplifies the mapping into segments. It contains tasks to start the mapping, run the mapping, and clean up the temporary tables and files. It contains multiple tasklets and the task recovery strategy. It also contains pre- and post-grid task preparation commands for each mapping before running the main mapping on a compute cluster. A pre-grid task can include a task such as copying data to HDFS. A post-grid task can include tasks such as cleaning up temporary files or copying data from HDFS.

Spark engine

The Spark execution plan shows the run-time Scala code that runs the mapping logic. A translation engine translates the mapping into an internal representation of the logic. The internal representation is rendered into Scala code that accesses the Spark API. You can view the Scala code in the execution plan to debug the logic.

Databricks Spark engine

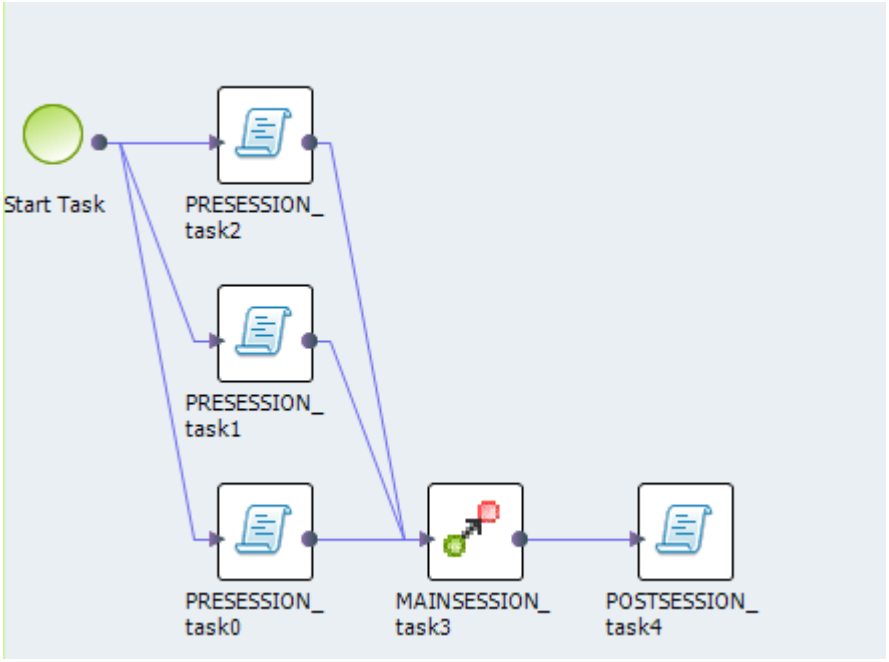
The Databricks Spark execution plan shows the run-time Scala code that runs the mapping logic. A translation engine translates the mapping into an internal representation of the logic. The internal representation is rendered into Scala code that accesses the Spark API. You can view the Scala code in the execution plan to debug the logic.

Blaze Engine Execution Plan Details

You can view details of the Blaze engine execution plan in the Administrator tool and Developer tool.

In the Developer tool, the Blaze engine execution plan appears as a workflow. You can click on each component in the workflow to get the details.

The following image shows the Blaze execution plan in the Developer tool:

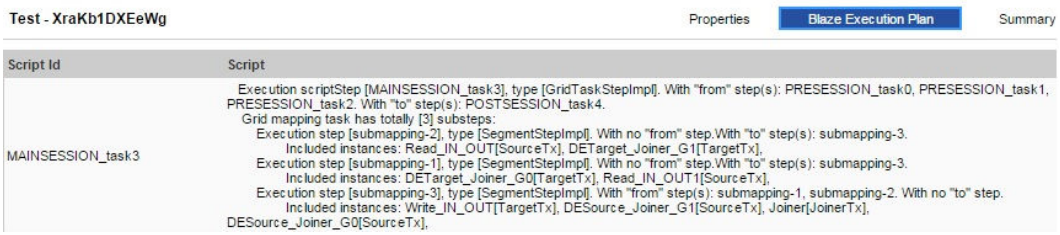


The Blaze engine execution plan workflow contains the following components:

- Start task. The workflow start task.
- Command task. The pre-processing or post-processing task for local data.
- Grid mapping. An Informatica mapping that the Blaze engine compiles and distributes across a cluster of nodes.
- Grid task. A parallel processing job request sent by the Blaze engine executor to the Grid Manager.
- Grid segment. Segment of a grid mapping that is contained in a grid task.
- Tasklet. A partition of a grid segment that runs on a separate DTM.

In the Administrator tool, the Blaze engine execution plan appears as a script.

The following image shows the Blaze execution script:



In the Administrator tool, the Blaze engine execution plan has the following details:

- Script ID. Unique identifier for the Blaze engine script.

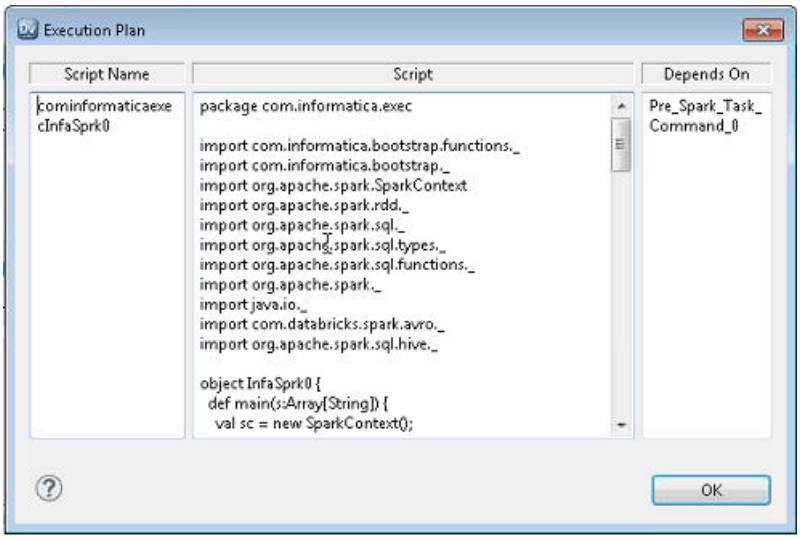
- Script. Blaze engine script that the Data Integration Service generates based on the mapping logic.
- Depends on. Tasks that the script depends on. Tasks include other scripts and Data Integration Service tasks, like the Start task.

Spark Engine Execution Plan Details

You can view the details of a Spark engine execution plan from the Administrator tool or Developer tool.

The Spark engine execution plan shows the Scala code on the Databricks Spark engine.

The following image shows the execution plan for a mapping to run on the Spark engine:



The Spark engine execution plan has the following details:

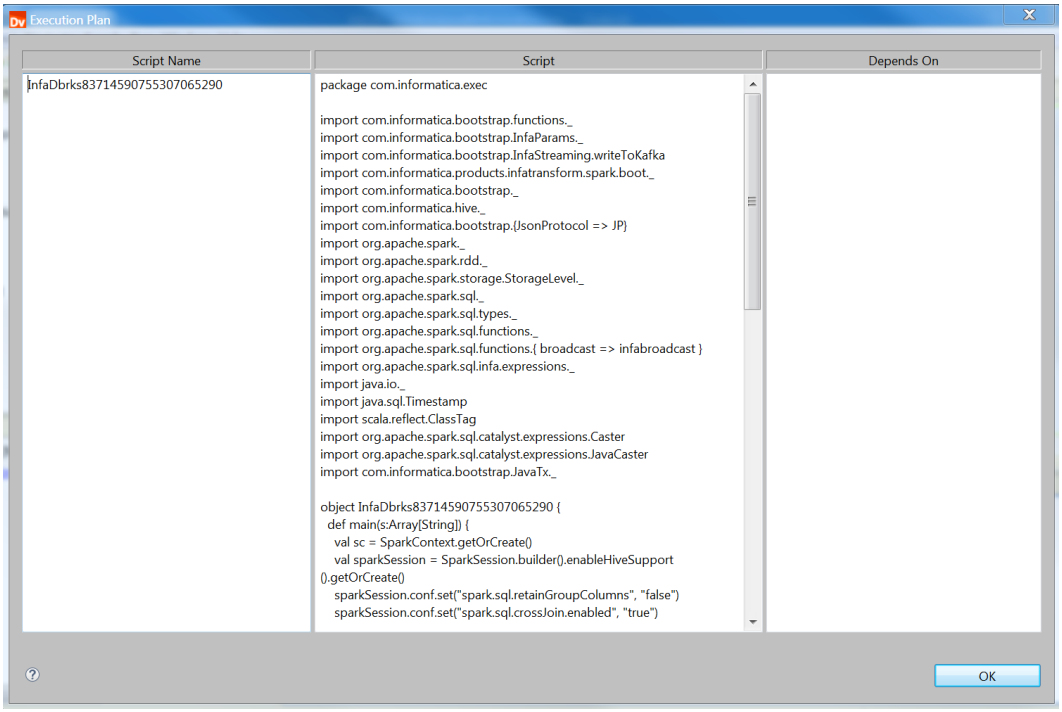
- Script ID. Unique identifier for the Spark engine script.
- Script. Scala code that the Data Integration Service generates based on the mapping logic.
- Depends on. Tasks that the script depends on. Tasks include other scripts and Data Integration Service tasks.

Databricks Spark Engine Execution Details

You can view the details of a Databricks Spark engine execution plan from the Administrator tool or Developer tool.

The Databricks Spark engine execution plan shows the Scala code to run on the Databricks Spark engine.

The following image shows the execution plan for a mapping to run on the Databricks Spark engine:



The Databricks Spark engine execution plan has the following details:

- Script ID. Unique identifier for the Databricks Spark engine script.
- Script. Scala code that the Data Integration Service generates based on the mapping logic.
- Depends on. Tasks that the script depends on. Tasks include other scripts and Data Integration Service tasks.

Viewing the Execution Plan

You can view the engine execution plan for a mapping. You do not have to run the mapping to view the execution plan in the Developer tool.

Note: You can also view the execution plan in the Administrator tool.

1. In the Developer tool, open the mapping.
2. Right-click the editor and select **Show Execution Plan**.

   The **Show Execution Plan** dialog box opens.
3. Select the mapping configuration and select **OK**.

   The Developer tool shows the details for the execution plan.

Troubleshooting Mappings in a Non-native Environment

Consider troubleshooting tips for mappings in a non-native environment.

Hadoop Environment

When I run a mapping with a Hive source or a Hive target on a different cluster, the Data Integration Service fails to push the mapping to Hadoop with the following error: `Failed to execute query [exec0_query_6] with error code [10], error message [FAILED: Error in semantic analysis: Line 1:181 Table not found customer_eur], and SQL state [42000]].`

When you run a mapping in a Hadoop environment, the Hive connection selected for the Hive source or Hive target, and the mapping must be on the same Hive metastore.

When I run a mapping with SQL overrides concurrently, the mapping hangs.

There are not enough available resources because the cluster is being shared across different engines.

Use different YARN scheduler queues for the Blaze and Spark engines to allow HiveServer2 to run SQL overrides through these engines.

Mappings run on the Blaze engine fail with the following preemption error messages:

```
 2018-09-27 11:05:27.208  INFO: Container completion status: id
[container_e135_1537815195064_4755_01_000012]; state [COMPLETE];
diagnostics [Container preempted by scheduler]; exit status [-102]..
2018-09-27 11:05:27.208 SEVERE: Service
[OOP_Container_Manager_Service_2] has stopped running..
```

The Blaze engine does not support YARN preemption on either the Capacity Scheduler or the Fair Scheduler. Ask the Hadoop administrator to disable preemption on the queue allocated to the Blaze engine. For more information, see Mappings Fail with Preemption Errors.

When I configure a mapping to create a partitioned Hive table, the mapping fails with the error "Need to specify partition columns because the destination table is partitioned."

This issue happens because of internal Informatica requirements for a query that is designed to create a Hive partitioned table. For details and a workaround, see Knowledge Base article 516266.

When Spark runs a mapping with a Hive source and target, and uses the Hive Warehouse Connector, the mapping fails with the following error:

```
[[SPARK_1003] Spark task [<task name>] failed with the following error: [User class
threw exception: java.lang.reflect.InvocationTargetException ...
java.sql.SQLException: Cannot create PoolableConnectionFactory (Could not open
client transport for any of the Server URI's in ZooKeeper: Could not establish
connection...)
```

The issue occurs because the Data Integration Service fails to fetch the Hive DT.

Workaround: Add the URL for HiveServer2 Interactive to the advanced properties of the Hadoop connection:

1. In the Ambari web console, browse to **Services** > **Hive** > **Configs** > **Advanced** > **Advanced hive-site** and copy the value of the property hive.server2.authentication.kerberos.principal.
2. Edit the Advanced Properties of the Hadoop connection to add the property spark.sql.hive.hiveserver2.jdbc.url.principal.
3. Paste the value that you copied in step 1 as the value of spark.sql.hive.hiveserver2.jdbc.url.principal.

Time stamp data that is precise to the nanosecond is truncated when a mapping runs on the Spark engine

Spark stores time stamp data to a precision of 1 microsecond (1us) and does not support nanosecond precision. When a mapping that runs on the Spark engine reads datetime data that has nanosecond precision, the data is truncated to the microsecond. For example, `2015-01-02 00:00:00.000456789` is truncated to `2015-01-02 00:00:00.000456`.

The Blaze engine supports nanosecond precision.

Databricks Environment

Mappings fail with the following error: `SEVERE: Run with ID [1857] failed with state [INTERNAL_ERROR] and error message [Library installation timed out after 1800 seconds. Libraries that are not yet installed: jar: "dbfs:/tmp/DATABRICKS/sess6250142538173973565/staticCode.jar"`

This might happen when you run concurrent jobs. When Databricks does not have resources to process a job, it queues the job for a maximum of 1,800 seconds (30 minutes). If resources are not available in 30 minutes, the job fails. Consider the following actions to avoid timeouts:

- Configure preemption environment variables on the Databricks cluster to control the amount of resources that get allocated to each job. For more information about preemption, see the *Data Engineering Integration Guide*.
- Run cluster workflows to create ephemeral clusters. You can configure the workflow to create a cluster, run the job, and then delete the cluster. For more information about ephemeral clusters, see Chapter 9, “Cluster Workflows” on page 169.

When I run mappings on a Dataproc cluster to read or write data to SerDe-backed Hive tables, the mapping fails with the java.lang.ClassNotFoundException: org.apache.hive.hcatalog.data.JsonSerDe error.

Add the following property in the `hive-site.xml` on all the nodes of the Dataproc cluster and restart the Hive server:

```
<property>
  <name>hive.aux.jars.path</name>
  <value>file:///usr/lib/hive/lib/hive-hcatalog-core-<version>.jar</value>
</property>
```

When I run mappings on a non-VPN Dataproc cluster, the mappings fail.

Configure the following properties in the `hdfs-site.xml` on all the nodes of the Dataproc cluster:

```
<property>
  <name>dfs.namenode.rpc-bind-host</name>
  <value>0.0.0.0</value>
</property>
<property>
  <name>dfs.namenode.servicerpc-bind-host</name>
  <value>0.0.0.0</value>
</property>
<property>
  <name>dfs.namenode.http-bind-host</name>
  <value>0.0.0.0</value>
</property>
<property>
  <name>dfs.namenode.https-bind-host</name>
  <value>0.0.0.0</value>
</property>
<property>
  <name>dfs.client.use.datanode.hostname</name>
  <value>true</value>
</property>
```

When I set job-level optimization on a mapping, the session log reflects the setting, but the Databricks Spark engine ignores it.

The Databricks Spark engine performs auto-optimization on jobs based on configuration settings for the cluster. It ignores custom configurations, such as the Spark.default.parallelism property. You cannot configure job-level optimization on a mapping that runs on the Databricks Spark engine.

Important: Informatica integrates with Databricks, supporting standard concurrency clusters. Standard concurrency clusters have a maximum queue time of 30 minutes, and jobs fail when the timeout is reached. The maximum queue time cannot be extended. Setting the preemption threshold allows more jobs to run concurrently, but with a lower percentage of allocated resources, the jobs can take longer to run. Also, configuring the environment for preemption does not ensure that all jobs will run. In addition to configuring preemption, you might choose to run cluster workflows to create ephemeral clusters that create the cluster, run the job, and then delete the cluster. For more information about Databricks concurrency, contact Azure Databricks.

Mappings in the Native Environment

You can run a mapping in the native environment. In the native environment, the Data Integration Service runs the mapping from the Developer tool. You can run standalone mappings or mappings that are a part of a workflow.

In the native environment, you can read and process data from large unstructured and semi-structured files, Hive, or social media web sites. You can include the following objects in the mappings:

- Hive sources
- Flat file sources or targets in the local system or in HDFS
- Complex file sources in the local system or in HDFS
- Data Processor transformations to process unstructured and semi-structured file formats
- Social media sources

Data Processor Mappings

The Data Processor transformation processes unstructured and semi-structured file formats in a mapping. It converts source data to flat CSV records that MapReduce applications can process.

You can configure the Data Processor transformation to process messaging formats, relational data, HTML pages, XML, JSON, AVRO, Parquet, Cobol, Microsoft Excel, Microsoft Word, and PDF documents. You can also configure it to transform structured formats such as ACORD, HIPAA, HL7, EDI-X12, EDIFACT, and SWIFT.

For example, an application produces hundreds of data files per second and writes the files to a directory. You can create a mapping that extracts the files from the directory, passes them to a Data Processor transformation, and writes the data to a target.

HDFS Mappings

Create an HDFS mapping to read or write to HDFS.

You can read and write fixed-width and delimited file formats. You can read or write compressed files. You can read text files and binary file formats such as sequence file from HDFS. You can specify the compression format of the files. You can use the binary stream output of the complex file data object as input to a Data Processor transformation to parse the file.

You can define the following objects in an HDFS mapping:

- Flat file data object or complex file data object operation as the source to read data from HDFS.
- Transformations.
- Flat file data object as the target to write data to HDFS or any target.

Validate and run the mapping. You can deploy the mapping and run it or add the mapping to a Mapping task in a workflow.

HDFS Data Extraction Mapping Example

Your organization needs to analyze purchase order details such as customer ID, item codes, and item quantity. The purchase order details are stored in a semi-structured compressed XML file in HDFS. The hierarchical data includes a purchase order parent hierarchy level and a customer contact details child hierarchy level. Create a mapping that reads all the purchase records from the file in HDFS. The mapping must convert the hierarchical data to relational data and write it to a relational target.

You can use the extracted data for business analytics.

The following figure shows the example mapping:

You can use the following objects in the HDFS mapping:

HDFS Input

The input object, Read_Complex_File, is a Read transformation that represents a compressed XML file stored in HDFS.

Data Processor Transformation

The Data Processor transformation, Data_Processor_XML_to_Relational, parses the XML file and provides a relational output.

Relational Output

The output object, Write_Relational_Data_Object, is a Write transformation that represents a table in an Oracle database.

When you run the mapping, the Data Integration Service reads the file in a binary stream and passes it to the Data Processor transformation. The Data Processor transformation parses the specified file and provides a relational output. The output is written to the relational target.

You can configure the mapping to run in a native or Hadoop run-time environment.

Complete the following tasks to configure the mapping:

1. Create an HDFS connection to read files from the Hadoop cluster.
2. Create a complex file data object read operation. Specify the following parameters:
   - The file as the resource in the data object.
   - The file compression format.
   - The HDFS file location.
3. Optionally, you can specify the input format that the Mapper uses to read the file.

4. Drag and drop the complex file data object read operation into a mapping.
5. Create a Data Processor transformation. Configure the following properties in the Data Processor transformation:
   - An input port set to buffer input and binary data type.
   - Relational output ports depending on the number of columns you want in the relational output. Specify the port size for the ports. Use an XML schema reference that describes the XML hierarchy. Specify the normalized output that you want. For example, you can specify PurchaseOrderNumber_Key as a generated key that relates the Purchase Orders output group to a Customer Details group.
   - Create a Streamer object and specify Streamer as a startup component.
6. Create a relational connection to an Oracle database.
7. Import a relational data object.
8. Create a write transformation for the relational data object and add it to the mapping.

Hive Mappings

Based on the mapping environment, you can read data from or write data to Hive.

In a native environment, you can read data from Hive. To read data from Hive, complete the following steps:

1. Create a Hive connection.
2. Configure the Hive connection mode to access Hive as a source or target.
3. Use the Hive connection to create a data object to read from Hive.
4. Add the data object to a mapping and configure the mapping to run in the native environment.

You can write to Hive in a Hadoop environment. To write data to Hive, complete the following steps:

1. Create a Hive connection.
2. Configure the Hive connection mode to access Hive as a source or target.
3. Use the Hive connection to create a data object to write to Hive.
4. Add the data object to a mapping and configure the mapping to run in the Hadoop environment.

You can define the following types of objects in a Hive mapping:

- A Read Transformation to read data from Hive
- Transformations
- A target or an SQL data service. You can write to Hive if you run the mapping in a Hadoop cluster.

Validate and run the mapping. You can deploy the mapping and run it or add the mapping to a Mapping task in a workflow.

Hive Mapping Example

Your organization, HypoMarket Corporation, needs to analyze customer data. Create a mapping that reads all the customer records. Create an SQL data service to make a virtual database available for end users to query.

You can use the following objects in a Hive mapping:

Hive input

The input file is a Hive table that contains the customer names and contact details.

Create a relational data object. Configure the Hive connection and specify the table that contains the customer data as a resource for the data object. Drag the data object into a mapping as a read data object.

SQL Data Service output

Create an SQL data service in the Developer tool. To make it available to end users, include it in an application, and deploy the application to a Data Integration Service. When the application is running, connect to the SQL data service from a third-party client tool by supplying a connect string.

You can run SQL queries through the client tool to access the customer data.

Social Media Mappings

Create mappings to read social media data from sources such as Facebook and LinkedIn.

You can extract social media data and load them to a target in the native environment only. You can choose to parse this data or use the data for data mining and analysis.

To process or analyze the data in Hadoop, you must first move the data to a relational or flat file target and then run the mapping in the Hadoop cluster.

You can use the following Informatica adapters in the Developer tool:

- PowerExchange for DataSift
- PowerExchange for Facebook
- PowerExchange for LinkedIn
- PowerExchange for Twitter
- PowerExchange for Web Content-Kapow Katalyst

Review the respective PowerExchange adapter documentation for more information.

Twitter Mapping Example

Your organization, Hypomarket Corporation, needs to review all the tweets that mention your product HypoBasket with a positive attitude since the time you released the product in February 2012.

Create a mapping that identifies tweets that contain the word HypoBasket and writes those records to a table.

You can use the following objects in a Twitter mapping:

Twitter input

The mapping source is a Twitter data object that contains the resource Search.

Create a physical data object and add the data object to the mapping. Add the Search resource to the physical data object. Modify the query parameter with the following query:

```
QUERY=HypoBasket:)&since:2012-02-01
```

Sorter transformation

Optionally, sort the data based on the timestamp.

Add a Sorter transformation to the mapping. Specify the timestamp as the sort key with direction as ascending.

Mapping output

Add a relational data object to the mapping as a target.

After you run the mapping, Data Integration Service writes the extracted tweets to the target table. You can use text analytics and sentiment analysis tools to analyze the tweets.

Native Environment Optimization

You can optimize the native environment to increase performance. To increase performance, you can configure the Data Integration Service to run on a grid and to use multiple partitions to process data. You can also enable high availability to ensure that the domain can continue running despite temporary network, hardware, or service failures.

You can run profiles, sessions, and workflows on a grid to increase the processing bandwidth. A grid is an alias assigned to a group of nodes that run profiles, sessions, and workflows. When you enable grid, the Data Integration Service runs a service process on each available node of the grid to increase performance and scalability.

You can also run mapping with partitioning to increase performance. When you run a partitioned session or a partitioned mapping, the Data Integration Service performs the extract, transformation, and load for each partition in parallel.

You can configure high availability for the domain. High availability eliminates a single point of failure in a domain and provides minimal service interruption in the event of failure.

Processing Data on a Grid

You can run an Integration Service on a grid to increase the processing bandwidth. When you enable grid, the Integration Service runs a service process on each available node of the grid to increase performance and scalability.

Large data sets may require additional bandwidth to process large amounts of data. For example, when you run a Model repository profile on a large data set, the Data Integration Service grid splits the profile into multiple mappings and runs the mappings simultaneously on different nodes in the grid.

When you run mappings on a grid, the Data Integration Service distributes the mappings to multiple DTM processes on nodes in the grid. When you run a profile on a grid, the Data Integration Service splits the profile into multiple mappings and distributes the mappings to multiple DTM processes on nodes in the grid. You can optimize the grid to increase performance and scalability of the Data Integration Service.

To optimize the grid, add notes to the grid to increase processing bandwidth of the Data Integration Service.

For more information about the Data Integration Service grid, see the *Informatica Administrator Guide*.

Processing Data on Partitions

You can run a Model repository mapping with partitioning to increase performance. When you run a mapping configured with partitioning, the Data Integration Service performs the extract, transformation, and load for each partition in parallel.

Mappings that process large data sets can take a long time to process and can cause low data throughput. When you configure partitioning, the Data Integration Service uses additional threads to process the session or mapping which can increase performance.

If the nodes where mappings run have multiple CPUs, you can enable the Data Integration Service to maximize parallelism when it runs mappings. When you maximize parallelism, the Data Integration Service dynamically divides the underlying data into partitions and processes all of the partitions concurrently.

Optionally, developers can set a maximum parallelism value for a mapping in the Developer tool. By default, the maximum parallelism for each mapping is set to Auto. Each mapping uses the maximum parallelism value defined for the Data Integration Service. Developers can change the maximum parallelism value in the

mapping run-time properties to define a maximum value for a particular mapping. When maximum parallelism is set to different integer values for the Data Integration Service and the mapping, the Data Integration Service uses the minimum value.

For more information, see the *Informatica Application Services Guide* and the *Informatica Developer Mapping Guide*.

Partition Optimization

You can optimize the partitioning of Model repository mappings to increase performance. You can add more partitions, select the best performing partition types, use more CPUs, and optimize the source or target database for partitioning.

To optimize partitioning, perform the following tasks:

Increase the number of partitions.

When you configure Model repository mappings, you increase the number of partitions when you increase the maximum parallelism value for the Data Integration Service or the mapping.

Increase the number of partitions to enable the Data Integration Service to create multiple connections to sources and process partitions of source data concurrently. Increasing the number of partitions increases the number of threads, which also increases the load on the Data Integration Service nodes. If the Data Integration Service node or nodes contain ample CPU bandwidth, processing rows of data concurrently can increase performance.

Note: If you use a single-node Data Integration Service and the Data Integration Service uses a large number of partitions in a session or mapping that processes large amounts of data, you can overload the system.

Use multiple CPUs.

If you have a symmetric multi-processing (SMP) platform, you can use multiple CPUs to concurrently process partitions of data.

Optimize the source database for partitioning.

You can optimize the source database for partitioning. For example, you can tune the database, enable parallel queries, separate data into different tablespaces, and group sorted data.

Optimize the target database for partitioning.

You can optimize the target database for partitioning. For example, you can enable parallel inserts into the database, separate data into different tablespaces, and increase the maximum number of sessions allowed to the database.

High Availability

High availability eliminates a single point of failure in an Informatica domain and provides minimal service interruption in the event of failure. When you configure high availability for a domain, the domain can continue running despite temporary network, hardware, or service failures. You can configure high availability for the domain, application services, and application clients.

The following high availability components make services highly available in an Informatica domain:

- Resilience. An Informatica domain can tolerate temporary connection failures until either the resilience timeout expires or the failure is fixed.
- Restart and failover. A process can restart on the same node or on a backup node after the process becomes unavailable.

- Recovery. Operations can complete after a service is interrupted. After a service process restarts or fails over, it restores the service state and recovers operations.

When you plan a highly available Informatica environment, consider the differences between internal Informatica components and systems that are external to Informatica. Internal components include the Service Manager, application services, and command line programs. External systems include the network, hardware, database management systems, FTP servers, message queues, and shared storage.

High availability features for the Informatica environment are available based on your license.

CHAPTER 3

Mapping Optimization

This chapter includes the following topics:

- Mapping Optimization, 67
- Mapping Recommendations and Analysis, 68
- Enabling Data Compression on Temporary Staging Tables, 76
- Truncating Partitions in a Hive Target, 77
- Hive Warehouse Connector and Hive LLAP, 77
- Scheduling, Queuing, and Node Labeling, 79
- Data Engineering Recovery, 81
- Spark Engine Optimization for Sqoop Pass-Through Mappings, 81

Mapping Optimization

Optimize mappings and the Hadoop environment to increase performance.

You can optimize mappings and the Hadoop environment in the following ways:

Mapping Recommendations and Analysis

You can use recommendations and analysis generated by the Informatica CLAIRE engine to optimize mappings and projects to reduce development costs and improve application performance. CLAIRE uses machine learning and internal algorithms to analyze mappings individually or in groups. The Developer tool displays the results of this analysis as recommendations and insights.

Configure a highly available Hadoop cluster

You can configure the Data Integration Service and the Developer tool to read from and write to a highly available Hadoop cluster. The steps to configure a highly available Hadoop cluster depend on the type of Hadoop distribution. For more information about configuration steps for a Hadoop distribution, see the *Data Engineering Integration Guide*.

Compress data on temporary staging tables

You can enable data compression on temporary staging tables to increase mapping performance.

Run mappings on the Blaze engine

Run mappings on the highly available Blaze engine. The Blaze engine enables restart and recovery of grid tasks and tasklets by default.

Perform parallel sorts

When you use a Sorter transformation in a mapping, the Data Integration Service enables parallel sorting by default when it pushes the mapping logic to the Hadoop cluster. Parallel sorting improves mapping performance.

Partition Joiner transformations

When you use a Joiner transformation in a Blaze engine mapping, the Data Integration Service can apply map-side join optimization to improve mapping performance. The Data Integration Service applies map-side join optimization if the master table is smaller than the detail table. When the Data Integration Service applies map-side join optimization, it moves the data to the Joiner transformation without the cost of shuffling the data.

Truncate partitions in a Hive target

You can truncate partitions in a Hive target to increase performance. To truncate partitions in a Hive target, you must choose to both truncate the partition in the Hive target and truncate the target table.

Assign resources on Hadoop clusters

You can use schedulers to assign resources on a Hadoop cluster. You can use a capacity scheduler or a fair scheduler depending on the needs of your organization.

Configure YARN queues to share resources on Hadoop clusters

You can configure YARN queues to redirect jobs on the Hadoop cluster to specific queues. The queue where a job is assigned defines the resources that are allocated to perform the job.

Label nodes in a Hadoop cluster

You can label nodes in a Hadoop cluster to divide the cluster into partitions that have specific characteristics.

Optimize Sqoop mappings on the Spark engine

The Data Integration Service can optimize the performance of Sqoop pass-through mappings that run on the Spark engine.

Enable data engineering recovery

You can enable data engineering recovery to recover mapping jobs that the Data Integration Service pushes to the Spark engine for processing.

Mapping Recommendations and Analysis

You can tune and correct mappings as you develop them based on recommendations and analysis generated by the Informatica CLAIRE engine. The recommendations and insights help you optimize mappings and projects to reduce development costs and improve application performance.

CLAIRE uses artificial intelligence and internal algorithms to analyze mappings individually or in groups. The Developer tool displays the results of its analysis of a single mapping as recommendations, and analysis of a group of mappings as insights. By reading and following these tips, you can recognize potential problems and opportunities for improvement.

When you enable recommendations, as you develop a mapping, CLAIRE analyzes your work and raises recommendations for improving the mapping. For example, a recommendation might point out a type mismatch in the mapping flow, or suggest a run-time engine that might improve mapping performance.

If the domain and the Developer tool are enabled for Enterprise Data Catalog or Axon, the Developer tool alerts you to the presence of sensitive data in a mapping. You can customize the types of sensitive data that CLAIRE detects.

You can also run an analysis on a group of mappings, such as all the mappings in a project. The resulting insights identify similarities among a group of mappings so you can find and remove redundancies.

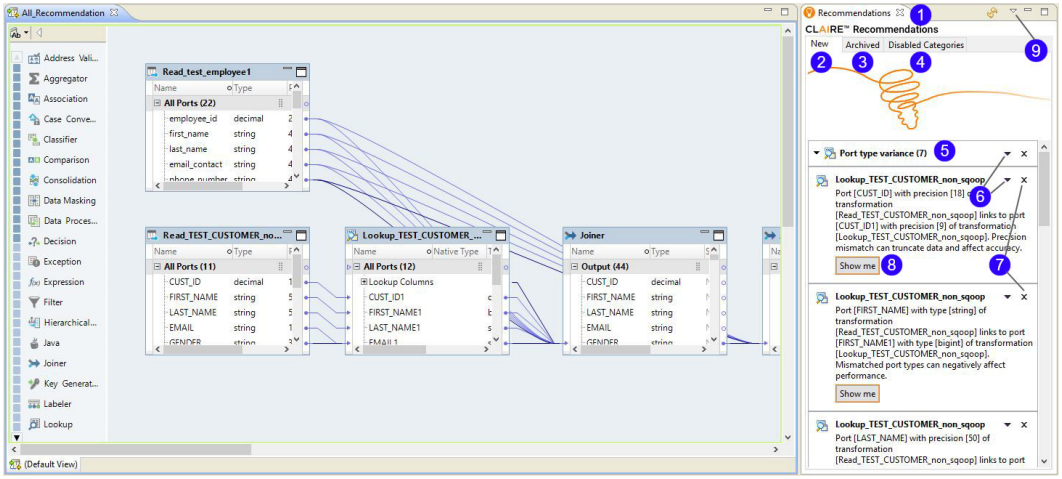
Recommendations

As you develop a mapping, CLAIRE analyzes the mapping for potential errors and opportunities for improvement, and displays recommendations on the **Inbox** tab of the **Recommendations** view.

Each recommendation belongs to a category that identifies the type of the recommendation. For example, if the mapping flow contains a datatype mismatch from one transformation to another, the recommendation to fix the issue appears under the "Port type variance" category.

You can read the recommendation and implement it by manually editing the mapping. When you select the **Show Me** button, the Developer tool displays and highlights the interface where you can make the recommended changes. In addition, you can archive the recommendation or disable the recommendation category.

The following image shows recommendations in the **New** tab of the **Recommendations** view:



The numbers in the image correspond to the following list of elements:

	Element name	Description
1	**Recommendations** view	Shows CLAIRE recommendations, sorted by category.
2	**New** tab	Displays new recommendations as CLAIRE generates them based on changes in the mapping as you develop it.
3	**Archived** tab	Shows recommendations that you have read and moved from the **New** tab using the **X** icon.
4	**Disabled Categories** tab	Displays categories of recommendations that you have read and decided you do not need to see repeatedly.
5	Recommendation category	Recommendations grouped by type.

	Element name	Description
6	Drop-down menu	Commands for disabling recommendation categories.
7	**Archive** icon	Dismisses the recommendation from the **New** tab and moves it to the **Archived** tab.
8	**Show Me** button	Opens the **Properties** view to the property where you can implement the recommendation.
9	**View** menu	Contains option to enable or disable the recommendations engine.

Note: When CLAIRE finds a very large (more than 200) number of recommendations for a mapping, not all of the recommendations are visible in the default size of the Recommendations view even if you scroll to the bottom of the view. To work around this issue, expand the Recommendations view.

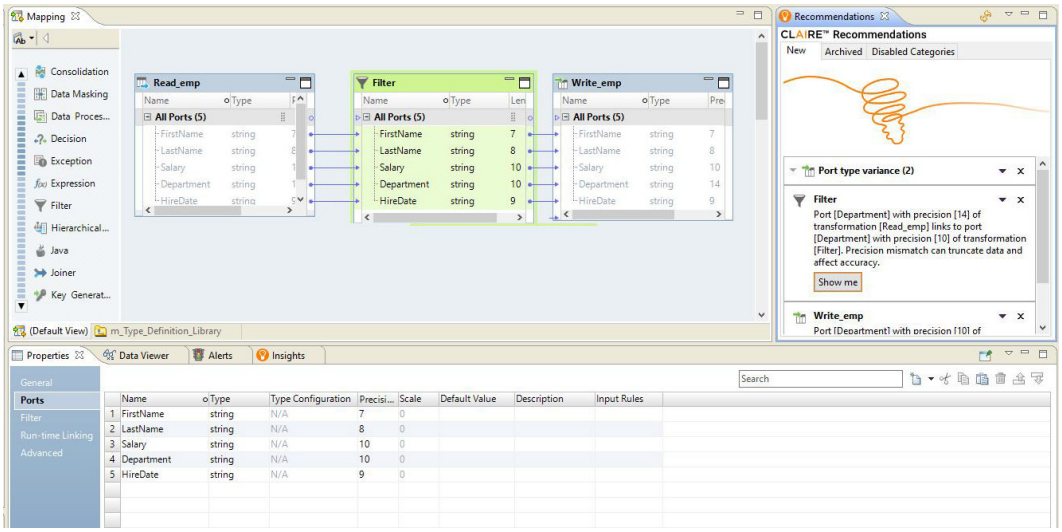
Implementing a Recommendation

As you develop a mapping, the CLAIRE engine analyzes your work and displays tips for fixing or improving the mapping in the **New** tab of the **Recommendations** view.

After you read a recommendation, you can prompt the Developer tool to show you where to implement the recommendation. When you click **Show Me,** the bottom view in the Developer tool opens to the view where you can implement the recommendation.

For example, you might want to implement a recommendation about port consistency. Click the **Show Me** button for the recommendation. The **Properties** view shows where to implement the recommendation by changing the port type.

The following image shows the mapping with a port consistency recommendation for the Filter transformation, and the **Properties** view open to the location where you implement the recommendation:

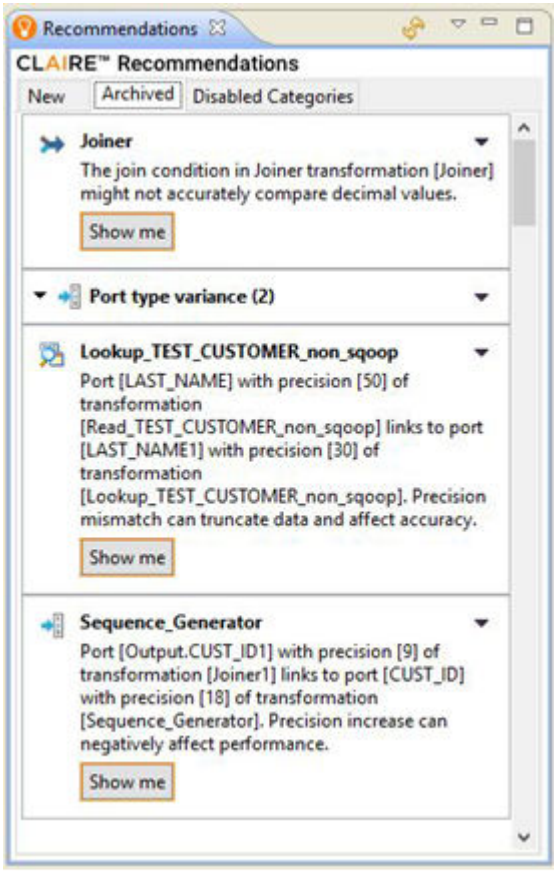


Archiving a Recommendation

After you read a recommendation, you can click the **X** icon to archive it. You can also click **X** to archive a recommendation category.

When you archive a recommendation or category, it moves to the **Archived** tab.

The following image shows recommendations in the **Archived** tab:



Notice that two of the recommendations are under a category, "Port type variance."

Disabling a Recommendation Category

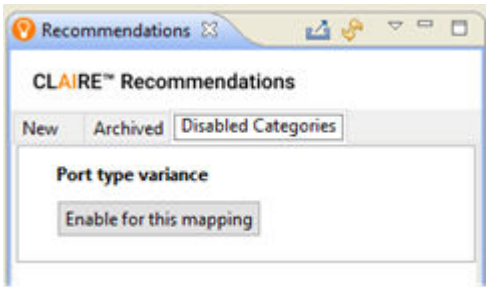
The category at the top of each recommendation identifies the type of the recommendation. After you have read a recommendation, you might want to stop seeing that type of recommendation, so you can disable it. When you disable a recommendation, CLAIRE stops running the rule that produces that category of recommendation.

To disable a category of recommendation, click the drop down menu and choose one of the following options:

- Disable for this mapping. If you choose this option, the Developer tool no longer displays the recommendation category while you develop this mapping.
- Disable for all mappings. If you choose this option, the Developer tool no longer displays the recommendation category for any mappings in the project.

You can view disabled recommendation categories on the **Disabled Categories** tab.

The following image shows a recommendation category on the **Disabled Categories** tab:



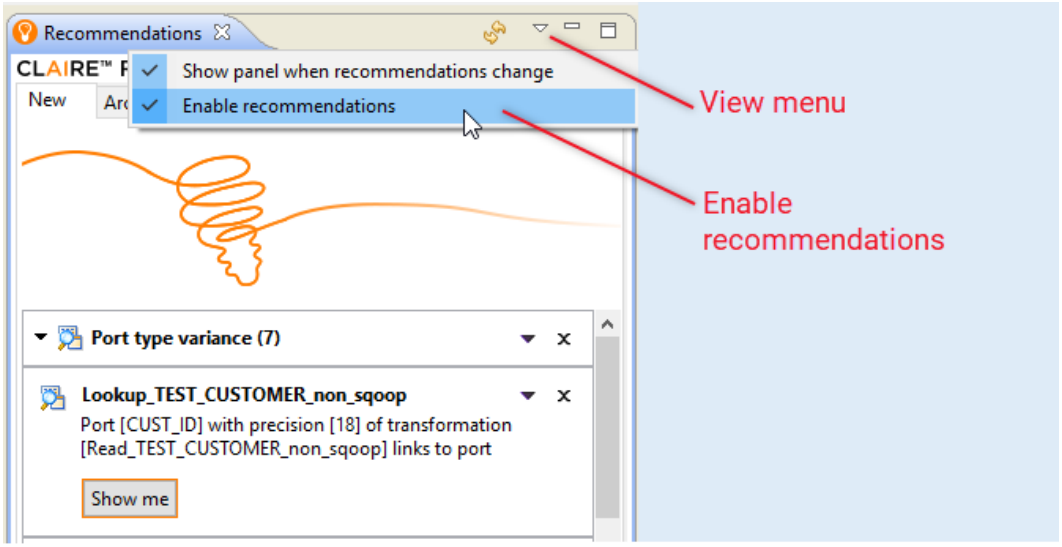
To re-enable a recommendation category, click the **Enable for this mapping** button.

Note: If the recommendation has been disabled for all mappings, the button will read "Enable for all mappings," and will re-enable the recommendation category for all mappings in the project.

Enabling and Disabling Recommendations

You can enable and disable CLAIRE recommendations.

To enable recommendations, click the **View** menu in the **Recommendations** view and select **Enable Recommendations**. The following image shows the **Enable Recommendations** menu item:



You can disable Recommendations. You might want to do this to speed Developer tool performance when working with a very large mapping.

When you disable recommendations, CLAIRE no longer runs rules on mappings, and any recommendations in the **Recommendations** view are lost.

To disable recommendations, click the **View** menu in the **Recommendations** view and deselect **Enable Recommendations**.

To re-enable recommendations, click the **View menu** in the **Recommendations** view and select **Enable Recommendations**.

After you re-enable recommendations, new recommendations appear when you open, save, or refresh a mapping.

Insights

You can run a cross-mapping CLAIRE analysis of a project or project folder. When you run an analysis, CLAIRE examines all of the mappings in the container to find similarities, and displays results in the **Insights** view. By finding mappings with similar structures and functions, you might discover an opportunity to tune mappings or combine them into a smaller number of mappings.

Perform these steps to conduct an analysis:

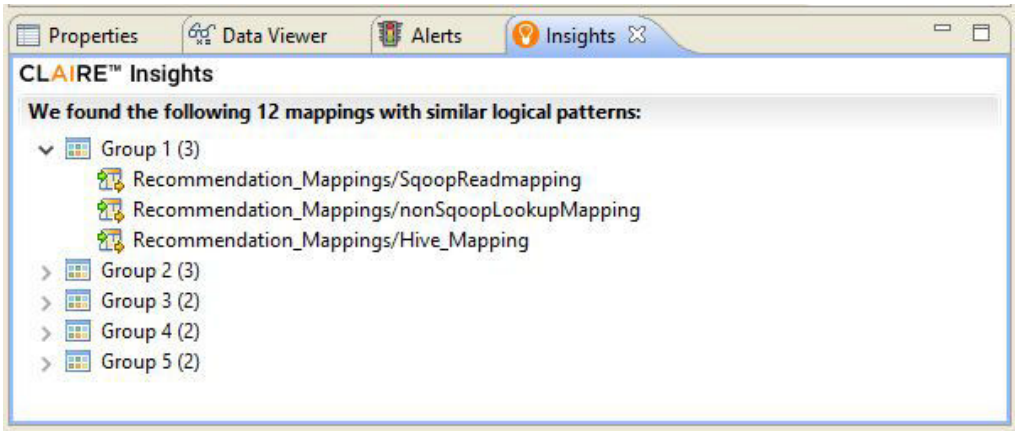
1. Right-click a folder in the repository tree and choose **Analyze**.
   CLAIRE conducts the analysis and displays a message when complete.

   Note: If you cancel the analysis before it finishes, the Developer tool might take a short time to complete cancellation. During this time, the Developer tool does not indicate that cancellation is in progress. Wait for cancellation to complete.

   The following text is an example of a message that you see when analysis is complete:

   ```
   Completed fetching and analyzing [12] mappings in [6] seconds.
   CLAIRE detected [5] groups of mappings with a similar logical pattern.
   See Insights view for details.
   ```

2. Click **OK**.
3. View the results of the analysis in the **Insights** view. The view shows groups of mappings with similar logical patterns.
   The following image shows the **Insights** view with a list of the groups of mappings that the analysis identified:



4. Expand each group to view the mappings in the group.
5. Double-click an individual mapping to open it in the mapping editor.

Analysis Spreadsheet

When you analyze a group of mappings, the analysis produces a report that you can open as a spreadsheet. The report contains detailed information about the mappings that you analyzed.

The spreadsheet contains the following information:

- Mapping. Mapping path and name.
- Transformation list. In the Transformations column, you can view a list of the transformations in the mapping.

For example, the following list shows that the mapping contains a Sorter transformation, two Expression transformations, a Union transformation, and so on:

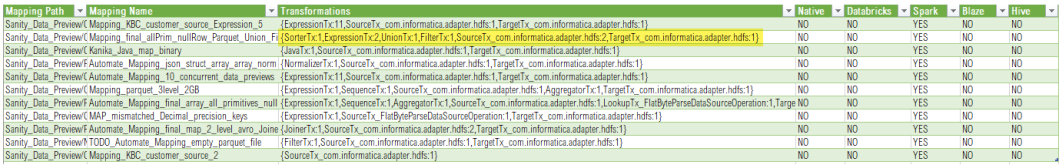
```
{SorterTx:1,ExpressionTx:2,UnionTx:1,FilterTx:1,SourceTx_com.informatica.adapter.hdfs:
2,TargetTx_com.informatica.adapter.hdfs:1}
```

Note that each entry includes the adapter type for Source and Target transformations. In this example, you can see that the two Source transformations and one Target transformations all used the HDFS adapter to connect to resources.

- Run-time engine. The last five columns show which run-time engines are configured for each mapping.

By viewing this information organized as a table, you might recognize similarities in mappings and opportunities for improvement.

The image below shows the entire spreadsheet, using Excel as a viewer, and highlighting the example given above.



Mapping Path	Mapping Name	Transformations	Native	Databricks	Spark	Blaze	Hive
Sanity_Data_Preview\(	Mapping_KBC_customer_source_Expression_5	{ExpressionTx:11,SourceTx_com.informatica.adapter.hdfs:1,TargetTx_com.informatica.adapter.hdfs:1}	NO	NO	YES	NO	NO
Sanity_Data_Preview\(	Mapping_final_allPrim_nullRow_Parquet_Union_F	{SorterTx:1,ExpressionTx:2,UnionTx:1,FilterTx:1,SourceTx_com.informatica.adapter.hdfs:2,TargetTx_com.informatica.adapter.hdfs:1}	NO	NO	YES	NO	NO
Sanity_Data_Preview\(	Kanika_Java_map_binary	{JavaTx:1,SourceTx_com.informatica.adapter.hdfs:1,TargetTx_com.informatica.adapter.hdfs:1}	NO	NO	YES	NO	NO
Sanity_Data_Preview\F	Automate_Mapping_json_struct_array_array_norm	{NormalizerTx:1,SourceTx_com.informatica.adapter.hdfs:1,TargetTx_com.informatica.adapter.hdfs:1}	NO	NO	YES	NO	NO
Sanity_Data_Preview\(	Automate_Mapping_10_concurrent_data_previews	{ExpressionTx:11,SourceTx_com.informatica.adapter.hdfs:1,TargetTx_com.informatica.adapter.hdfs:1}	NO	NO	YES	NO	NO
Sanity_Data_Preview\(	Mapping_parquet_3level_2GB	{ExpressionTx:1,SequenceTx:1,SourceTx_com.informatica.adapter.hdfs:1,AggregatorTx:1,TargetTx_com.informatica.adapter.hdfs:1}	NO	NO	YES	NO	NO
Sanity_Data_Preview\F	Automate_Mapping_final_array_all_primitives_null	{ExpressionTx:1,SequenceTx:1,AggregatorTx:1,SourceTx_com.informatica.adapter.hdfs:1,LookupTx_FlatByteParseDataSourceOperation:1,Targe	NO	NO	YES	NO	NO
Sanity_Data_Preview\(	MAP_mismatched_Decimal_precision_keys	{ExpressionTx:1,SourceTx_FlatByteParseDataSourceOperation:1,TargetTx_com.informatica.adapter.hdfs:1}	NO	NO	YES	NO	NO
Sanity_Data_Preview\F	Automate_Mapping_final_map_2_level_avro_Joine	{JoinerTx:1,SourceTx_com.informatica.adapter.hdfs:2,TargetTx_com.informatica.adapter.hdfs:1}	NO	NO	YES	NO	NO
Sanity_Data_Preview\T	TODO_Automate_Mapping_empty_parquet_file	{FilterTx:1,SourceTx_com.informatica.adapter.hdfs:1,TargetTx_com.informatica.adapter.hdfs:1}	NO	NO	YES	NO	NO
Sanity_Data_Preview\(	Mapping_KBC_customer_source_2	{SourceTx_com.informatica.adapter.hdfs:1}	NO	NO	YES	NO	NO

Accessing the Analysis Spreadsheet

The Developer tool creates the spreadsheet as a text file in the following location: `<Developer tool home>/Clients/DeveloperClient/workspace/`

You can use Excel or another tool to open the file and display its information in columns. Columns are delimited using the pipe (|) symbol.

Example

In this example, you are creating several mappings that ingest and convert customer data.

You create a project with several mappings. Each uses the same relational source and writes to the same target.

As you create one of the mappings, a CLAIRE rule finds that the mapping uses a JDBC connection and is configured to use the Blaze run-time engine. CLAIRE raises the following recommendation, which you view in the **New** tab of the **Recommendations** view:

```
Mappings that use a JDBC connection can run faster when you enable Sqoop on the connection
and run the mapping on the Spark engine.
```

You click **Show Me**, and the **Properties** view opens to settings where you can implement both recommendations.

As development continues, you notice that recommendations of the same type repeatedly appepar in the **New** tab of the **Recommendations** view. You know how you will address the issue, but decide to do it at a later time. To stop seeing the same recommendation, you disable the category of recommendation for the project.

When the mappings in the project are complete, you run an analysis on the entire project. CLAIRE analyzes the mappings and finds that their similar structures indicate that some of the mappings could be combined into a single mapping. You use the analysis spreadsheet to spot similarities between the mappings. After you identify the similar mappings, you edit the mappings in the project to eliminate duplicative logic.

Enabling and Customizing Access to Sensitive Data

When you connect the developer tool to the Enterprise Data Catalog host, you can enable recommendations and insights that alert you when mappings contain sensitive data.

The following types of sensitive data are enabled by default:

- Personal Health Information
- Payment Card Information
- Personally Identifiable Information

Connect the Developer Tool to Enterprise Data Catalog

To view recommendations for sensitive data, connect the Developer tool to the Enterprise Data Catalog host.

1. Choose **Preferences** > **Informatica** > **Enterprise Data Catalog**.
2. Enter the following properties:

Property	Description
Host URL	URL of the EDC host, including the port number. Example: `http://<host name>:<port number>`
Username	Enterprise Data Catalog user name.
Password	Password for the Enterprise Data Catalog user.

3. Click **OK**.

Customize Sensitive Data Groups

You can define additional sensitive data types in developerCore.ini for the CLAIRE engine.

Add the following entry to the `developerCore.ini` file on every Developer tool machine:

```
-additionalSensitiveDataDomainGroups=[column1,column2]
```

The comma-delimited list can contain any column name from the data object that the mapping connects to.

You can find the file in the following path: `<Developer tool installation directory>\clients\DeveloperClient\developerCore.ini`.

Enabling Data Compression on Temporary Staging Tables

To optimize performance when you run a mapping in the Hadoop environment, you can enable data compression on temporary staging tables. When you enable data compression on temporary staging tables, mapping performance might increase.

To enable data compression on temporary staging tables, complete the following steps:

1. Configure the Hadoop connection to use the codec class name that the Hadoop cluster uses to enable compression on temporary staging tables.
2. Configure the Hadoop cluster to enable compression on temporary staging tables.

Hadoop provides following compression libraries for the following compression codec class names:

Compression Library	Codec Class Name	Performance Recommendation
Zlib	org.apache.hadoop.io.compress.DefaultCodec	n/a
Gzip	org.apache.hadoop.io.compress.GzipCodec	n/a
Snappy	org.apache.hadoop.io.compress.SnappyCodec	Recommended for best performance.
Bz2	org.apache.hadoop.io.compress.BZip2Codec	Not recommended. Degrades performance.
LZO	com.hadoop.compression.lzo.LzoCodec	n/a

Step 1. Enable Data Compression in the Hadoop Connection

Use the Administrator tool or the Developer tool to configure the Hadoop connection to enable data compression on temporary staging tables.

1. In the Hadoop connection properties, edit the properties to run mappings in a Hadoop cluster.
2. Select **Temporary Table Compression Codec**.
3. Choose to select a predefined codec class name or enter a custom codec class name.
   - To select a predefined codec class name, select a compression library from the list.
   - To enter a custom codec class name, select custom from the list and enter the codec class name that matches the codec class name in the Hadoop cluster.

Step 2. Enable Data Compression on the Hadoop Environment

To enable compression on temporary staging tables, you must install a compression codec on the Hadoop cluster.

For more information about how to install a compression codec, refer to the Apache Hadoop documentation.

1. Verify that the native libraries for the compression codec class name are installed on every node on the cluster.
2. To include the compression codec class name that you want to use, update the property io.compression.codecs in core-site.xml. The value for this property is a comma separated list of all the codec class names supported on the cluster.

3. Verify that the Hadoop-native libraries for the compression codec class name that you want to use are installed on every node on the cluster.
4. Verify that the LD_LIBRARY_PATH variable on the Hadoop cluster includes the locations of both the native and Hadoop-native libraries where you installed the compression codec.

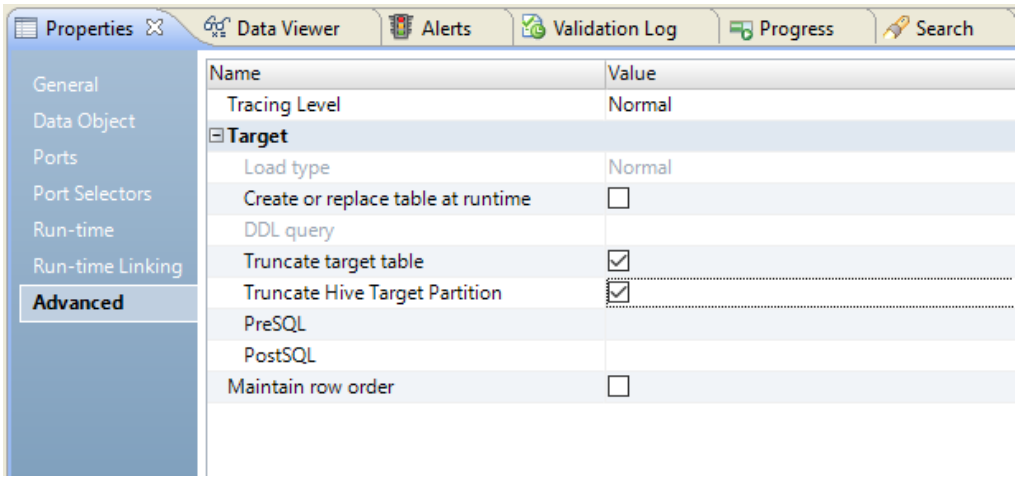
Truncating Partitions in a Hive Target

To truncate partitions in a Hive target, you must edit the write properties for the customized data object that you created for the Hive target in the Developer tool.

You can truncate partitions in a Hive target when you use the Blaze or Spark run-time engines to run the mapping.

1. Open the customized data object in the editor.
2. To edit write properties, select the **Input** transformation in the **Write** view, and then select the **Advanced** properties.

   The following image shows the **Advanced** properties tab:



3. Select **Truncate Hive Target Partition**.
4. Select **Truncate target table**.

Hive Warehouse Connector and Hive LLAP

Enable the Hive Warehouse Connector and Hive LLAP for faster execution of Hive queries when you read from and write to Hive tables. You can use the Hive Warehouse Connector and Hive LLAP with Hortonworks HDP 3.x and Microsoft Azure HDInsight 4.x clusters on the Spark engine.

The Hive Warehouse Connector reads from and writes to Hive tables without using temporary staging tables that require additional storage overhead. Use the Hive Warehouse Connector on the Spark engine to allow Spark code to interact with Hive targets and to use ACID-enabled Hive tables. When you enable the Hive Warehouse Connector, mappings use Hive LLAP to run Hive queries rather than HiveServer2.

Consider the following limitations when you use the Hive Warehouse Connector and Hive LLAP:

- The Hive Warehouse Connector and Hive LLAP are used to run insert queries to ACID-enabled tables that are not bucketed.
- You cannot use the Hive Warehouse Connector and Hive LLAP when you read hierarchical data from a source.
- When you use the Hive Warehouse Connector on Hortonworks HDP clusters, you must use an ORC format target. Data corruption might occur if the target does not use ORC format.
  For more information, see Hortonworks documentation on supported target tables: Apache Hive 3 tables.
- When you use an external table that has compression properties set, the mapping executes using Spark SQL instead of HiveServer2. The mapping fails if the value of the compression property is not one of the following values: LZO, NONE, SNAPPY, ZLIB.

  Note: The property value is case sensitive and must use upper case.

  For more information, refer to the Apache Hive documentation on compression kind: CompressionKind (Hive 2.1.1 API)
- When you use choose RETAIN as the target schema strategy, configure the property *hive.llap.daemon.num.enabled.executors* on the Hadoop cluster. Set the value of this property to the same value as *hive.llap.daemon.num.executors*.
- When you import a mapping with an ACID-enabled source and target, the Summary Statistics view does not reflect any throughput statistics for the mapping job.

Enabling the Hive Warehouse Connector and Hive LLAP

Before you enable the Hive Warehouse Connector, enable Hive LLAP on the Hadoop cluster. To enable the connector, configure the following properties in the Spark advanced properties for the Hadoop connection:

infaspark.useHiveWarehouseAPI

Enables the Hive Warehouse Connector. Set to TRUE.

For example, `infaspark.useHiveWarehouseAPI=true`.

spark.datasource.hive.warehouse.load.staging.dir

Directory for the temporary HDFS files used for batch writes to Hive. Required when you enable the Hive Warehouse Connector.

For example, set to `/tmp`

spark.datasource.hive.warehouse.metastoreUri

URI for the Hive metastore. Required when you enable the Hive Warehouse Connector. Use the value for *hive.metastore.uris* from the hive_site_xml cluster configuration properties.

For example, set the value to `thrift://mycluster-1.com:9083 .`

spark.hadoop.hive.llap.daemon.service.hosts

Application name for the LLAP service. Required when you enable the Hive Warehouse Connector. Use the value for *hive.llap.daemon.service.hosts* from the hive_site_xml cluster configuration properties.

spark.hadoop.hive.zookeeper.quorum

Zookeeper hosts used by Hive LLAP. Required when you enable the Hive Warehouse Connector. Use the value for *hive.zookeeper.quorum* from the hive_site_xml cluster configuration properties.

spark.sql.hive.hiveserver2.jdbc.url

URL for HiveServer2 Interactive. Required to use the Hive Warehouse Connector. Use the value in Ambari for HiveServer2 JDBC URL.

Scheduling, Queuing, and Node Labeling

You can use YARN schedulers, YARN queues, and node labels to optimize performance when you run a mapping in the Hadoop environment.

A YARN scheduler assigns resources to YARN applications on the Hadoop cluster while honoring organizational policies on sharing resources. You can configure YARN to use a fair scheduler or a capacity scheduler. A fair scheduler shares resources evenly among all jobs running on the cluster over time. A capacity scheduler allows multiple organizations to share a large cluster and distributes resources based on capacity allocations. The capacity scheduler guarantees each organization a certain capacity and distributes any excess capacity that is underutilized.

YARN queues are organizing structures for YARN schedulers and allow multiple tenants to share a cluster. The capacity of each queue specifies the percentage of cluster resources that are available for applications submitted to the queue. You can redirect Blaze, Spark, and Sqoop jobs to specific YARN queues.

Node labels allow YARN queues to run on specific nodes in a cluster. You can use node labels to partition a cluster into sub-clusters such that jobs run on nodes with specific characteristics. For example, you might label nodes that process data faster compared to other nodes. Nodes that are not labeled belong to the default partition. You can associate the node labels with capacity scheduler queues.

You can also use the node labels to configure the Blaze engine. When you use node labels to configure the Blaze engine, you can specify the nodes on the Hadoop cluster where you want the Blaze engine to run.

Note: You cannot use node labels on a Cloudera CDH cluster.

Enable Scheduling and Node Labeling

To enable scheduling and node labeling in the Hadoop environment, update the yarn-site.xml properties in the cluster configuration.

Configure the following properties:

yarn.resourcemanager.scheduler.class

Defines the YARN scheduler that the Data Integration Service uses to assign resources on the cluster.

```
<property>
    <name>yarn.resourcemanager.scheduler.class</name>
    <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.[Scheduler Type].
[Scheduler Type]Scheduler</value>
</property>
```

For example:

```
<property>
    <name>yarn.resourcemanager.scheduler.class</name>

<value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacitySched
uler</value>
</property>
```

yarn.node-labels.enabled

Enables node labeling.

```
<property>
    <name>yarn.node-labels.enabled</name>
    <value>TRUE</value>
</property>
```

yarn.node-labels.fs-store.root-dir

The HDFS location to update the node label dynamically.

```
<property>
    <name>yarn.node-labels.fs-store.root-dir</name>
    <value>hdfs://[Node name]:[Port]/[Path to store]/[Node labels]/</value>
</property>
```

Define YARN Queues

You can define multiple YARN queues to redirect jobs to specific queues. You can redirect Blaze, Spark, and Sqoop jobs.

To redirect jobs on the Blaze engine to a specific queue, configure the following Blaze configuration property in the Hadoop connection:

Property	Description
YARN Queue Name	The YARN scheduler queue name used by the Blaze engine that specifies available resources on a cluster.

To redirect jobs on the Spark engine to a specific queue, configure the following Spark configuration property in the Hadoop connection:

Property	Description
YARN Queue Name	The YARN scheduler queue name used by the Spark engine that specifies available resources on a cluster. The name is case sensitive.

To redirect Sqoop jobs to a specific queue, configure the following JDBC connection property:

Property	Description
Sqoop Arguments	The Sqoop connection-level argument to direct a MapReduce job to a specific YARN queue. Use the following format: `-Dmapred.job.queue.name=<YARN queue name>` If you do not direct the job to a specific queue, the Spark engine uses the default queue.

Configure the Blaze Engine to Use Node Labels

If you enable node labeling in the Hadoop environment, you can use node labels to run Blaze components as YARN applications on specific cluster nodes.

To run Blaze components on the labeled cluster nodes, specify the node labels when you configure the Blaze engine.

To specify node labels on the Blaze engine, list the node labels in the following Hadoop connection property:

Property	Description
Blaze YARN Node Label	Node label that determines the node on the Hadoop cluster where the Blaze engine runs. If you do not specify a node label, the Blaze engine runs on the nodes in the default partition. If the Hadoop cluster supports logical operators for node labels, you can specify a list of node labels. To list the node labels, use the operators `&&` (AND), `||` (OR), and `!` (NOT).

Note: When the Blaze engine uses node labels, Blaze components might be redundant on the labeled nodes. If a node contains multiple labels and you specify the labels in different Hadoop connections, multiple Grid Manager, Orchestrator, or Job Monitor instances might run on the same node.

Data Engineering Recovery

An administrator can enable data engineering recovery to recover a job configured to run on the Spark engine when a Data Integration Service node stops unexpectedly.

When a Data Integration Service node fails before a running job is complete, the Data Integration Service sends the job to another node, which resumes processing job tasks from the point at which the node failure occurred. Recovery occurs upon node startup.

To use data engineering recovery, you must configure jobs to run on the Spark engine and submit jobs from the infacmd client.

An administrator configures data engineering recovery in Data Integration Service properties. For more information about data engineering recovery, see the *Data Engineering Administrator Guide*.

Spark Engine Optimization for Sqoop Pass-Through Mappings

When you run a pass-through mapping with a Sqoop source on the Spark engine, the Data Integration Service optimizes mapping performance in the following scenarios:

- You write data to a Hive target that uses the Text format.
- You write data to a Hive target that was created with a custom DDL query.
- You write data to an Hive target that is either partitioned with a custom DDL query or partitioned and bucketed with a custom DDL query.
- You write data to an existing Hive target that is both partitioned and bucketed.
- You write data to an HDFS target that uses the Flat, Avro, or Parquet format.

If you want to disable the performance optimization, set the --infaoptimize argument to false in the JDBC connection or Sqoop mapping. For example, if you see data type issues after you run an optimized Sqoop mapping, you can disable the performance optimization.

Use the following syntax:

```
--infaoptimize false
```

Rules and Guidelines for Sqoop Spark Engine Optimization

Consider the following rules and guidelines when you run Sqoop mappings on the Spark engine:

- The Data Integration Service does not optimize mapping performance in the following scenarios:
  - There are unconnected ports between the source and target in the mapping.
  - The data types of the source and target in the mapping do not match.
  - You write data to an existing Hive target table that is either partitioned or bucketed.
  - You run a mapping on an Azure HDInsight cluster that uses WASB to write data to an HDFS complex file target of the Parquet format.
  - The date or time data type in the Sqoop source is mapped to the timestamp data type in the Hive target.
  - The Sqoop source contains a decimal column and the target is a complex file.
- If you configure Hive-specific Sqoop arguments to write data to a Hive target, Sqoop ignores the arguments.
- If you configure a delimiter for a Hive target table that is different from the default delimiter, Sqoop ignores the delimiter.

CHAPTER 4

Sources

This chapter includes the following topics:

- Overview of Sources, 83
- PowerExchange Adapter Sources, 83
- Sources on Databricks, 85
- File Sources on Hadoop, 88
- Relational Sources on Hadoop, 95
- Hive Sources on Hadoop, 95
- Sqoop Sources on Hadoop, 98

Overview of Sources

You can push a mapping to a non-native environment that includes a source from the native environment or non-native environment. Some sources have restrictions when you reference them in a non-native environment.

Access to sources in a non-native environment require a PowerExchange adapter.

When a mapping runs in the Hadoop environment, an HDFS source cannot reside on a remote cluster.

PowerExchange Adapter Sources

PowerExchange sources can run in the Hadoop or Databricks environment, based on the adapter.

The following table lists PowerExchange sources and the non-native engines that support the sources:

PowerExchange Adapter	Supported Engines
Amazon Redshift	- Blaze - Spark - Databricks Spark
Amazon S3	- Blaze - Spark - Databricks Spark

PowerExchange Adapter	Supported Engines
Google Analytics	- Spark
Google BigQuery	- Spark
Google Cloud Spanner	- Spark
Google Cloud Storage	- Spark
HBase	- Blaze - Spark
HDFS	- Blaze - Spark
Hive	- Blaze - Spark
JDBC V2	- Spark - Databricks Spark
MapR-DB	- Blaze - Spark
Microsoft Azure Blob Storage	- Spark - Databricks Spark
Microsoft Azure Cosmos DB	- Spark - Databricks Spark
Microsoft Azure Data Lake Storage Gen1	- Spark - Databricks Spark
Microsoft Azure Data Lake Storage Gen2	- Spark - Databricks Spark
Microsoft Azure SQL Data Warehouse	- Spark - Databricks Spark
Snowflake	- Spark - Databricks Spark
Teradata Parallel Transporter API	- Blaze - Spark

Sources on Databricks

A mapping that runs in the Databricks environment can include file and database sources.

The following table lists the storage types and sources:

Storage Type	Source
File	- Amazon Simple Storage Service (Amazon S3) - Databricks Delta Lake storage - Microsoft Azure Blob Storage - Microsoft Azure Data Lake Store (ADLS)
Database	- Amazon Redshift - JDBC V2 - Microsoft Azure Cosmos DB - Microsoft Azure SQL Data Warehouse - Snowflake

Complex File Sources on Amazon S3

Use a PowerExchange for Amazon S3 connection to read data from Amazon S3 data objects.

The following table shows the complex files that a mapping can process within Amazon S3 storage on the Databricks Spark engine:

File Type	Supported Formats
Avro	- Flat - Hierarchical
JSON	- Flat - Hierarchical
ORC	- Flat
Parquet	- Flat - Hierarchical

Complex File Sources on ADLS

Use a PowerExchange for Microsoft Azure Data Lake Store connection to read data from ADLS data objects.

The following table shows the complex files that a mapping can process within ADLS storage in the Databricks environment:

File Type	Format
Avro	- Flat - Hierarchical
JSON	- Flat - Hierarchical

File Type	Format
Parquet	- Flat - Hierarchical
Note: The read operation for hierarchical files must be enabled to project columns as complex data type.	

Complex File Sources on Azure Data Lake Storage Gen2

Use a PowerExchange for Microsoft Azure Data Lake Storage Gen2 connection to read data from Azure Data Lake Storage Gen2 data objects.

The following table shows the complex files that a mapping can process within Azure Data Lake Storage Gen2 in the Databricks environment:

File Type	Format
Avro	- Flat - Hierarchical
JSON	- Flat - Hierarchical
ORC	- Flat
Parquet	- Flat - Hierarchical
Note: The read operation for hierarchical files must be enabled to project columns as complex data type.	

Complex File Sources on Azure Blob

Use a PowerExchange for Microsoft Azure Blob Storage connection to read data from Blob Storage data objects. You can read compressed binary files.

The following table shows the complex files that a mapping can process within Azure Blob storage in the Databricks environment:

File Type	Format
Avro	- Flat - Hierarchical
JSON	- Flat - Hierarchical
Parquet	- Flat - Hierarchical
Note: The read operation for hierarchical files must be enabled to project columns as complex data type.	

Databricks Delta Lake

Delta Lake is a data lake resource that stores data in large tables. Databricks uses proprietary Delta software to manage stored data and allow fast access to the data. Delta Lake supports ACID transactions.

Mappings can access Delta Lake resources on the AWS or Azure platforms. You can access Delta Lake tables built on top of any of the following storage types:

- Azure blob storage
- Azure Data Lake Storage (ADLS) Gen1 or Gen2
- Amazon Web Services (AWS) S3.

To use Delta Lake resources in a mapping, create a relational data object and use a JDBC connection to connect to the resource.

Rules and Guidelines for Databricks Sources

File Sources

Consider the following general rules and guidelines for file sources:

- The flat file source must reside on one of the following storage resources:
  - For the Azure platform: Microsoft Azure Blob Storage, Microsoft Azure Data Lake Store (ADLS) Gen1 or Gen2
  - For the AWS platform: S3
- The row delimiter must be /n.
- The file cannot be fixed width.
- Multiple column delimiters are not supported.
- To read multiline data, set the text qualifier to single or double quotes and enclose the data in the quoted qualifier.
- Empty values only are treated as null values.
- When the path to the physical data object source includes spaces in the name of a file or directory, the path that is rendered in the source column of the output has the characters `%20` for spaces. For example, `C:\\testdirectory\flat file.txt` is rendered as `C:\\testdirectory\flat%20file.txt`. This behavior happens for flat file and complex files.

Delta Lake Sources

Consider the following guidelines for using Delta Lake sources:

- Mappings that access Delta Lake tables must use the Databricks run-time engine. If you run a Delta Lake mapping in the native environment with the JDBC connection, the mapping succeeds, but no data is written to the target.

Null Processing

Consider the following rules and guidelines for null processing:

Unexpected values converted to nulls

The Databricks Spark engine generates null values for all fields in the same record if any field contains an unexpected value based on the following scenarios:

- Any type mismatch occurs, such as passing string data to a numeric column.
- Data is out of bounds, such as with bigint or int data types.

Consider using a Filter transformation to filter out null rows.

Date/time values converted to nulls

When the Databricks Spark engine reads date/time values, it uses the format YYYY-MM-DD HH24:MM:SS.US. If the date format read from the source does not match this format, the Databricks Spark engine converts the date values to nulls.

Double and Decimal Conversions

When the Databricks Spark engine reads from an Azure or AWS source, it converts double and decimal data types to scientific notation. When it converts that data back to a double or decimal to write to the target, it drops precision greater than 15 and maintains precision of 15 digits.

File Sources on Hadoop

A mapping that runs in the Hadoop environment can process complex files and flat files.

To read large volumes of data, you can connect a complex file source to read data from a directory of files that have the same format and properties. You can read compressed binary files.

You can read complex files from the following storage types in the Hadoop environment:

- Amazon Simple Storage Service (Amazon S3)
- Hadoop Distributed File System (HDFS)
- Hive
- MapR File System (MapR-FS)
- Microsoft Azure Blob Storage (Azure Blob Storage)
- Microsoft Azure Data Lake Store (ADLS)

Complex File Sources on Amazon S3

Use a PowerExchange for Amazon S3 connection to read data from Amazon S3 data objects.

The following table shows the complex files that a mapping can process within Amazon S3 storage in the Hadoop environment:

Resource Format	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1] [2]	- Blaze - Spark
Intelligent Structure Model or Sample File	- Flat - Hierarchical [1] [2]	- Blaze - Spark
JSON	- Flat - Hierarchical [1] [2]	- Blaze - Spark
ORC	- Flat	- Spark

Resource Format	Supported Formats	Supported Engines
Parquet	- Flat - Hierarchical [1] [2]	- Blaze - Spark

[1] To run on the Blaze engine, the complex file data object must be connected to a Data Processor transformation.

[2] To run on the Spark engine, the complex file read operation must be enabled to project columns as complex data type.

Complex File Sources on Google Cloud Storage

You can use a PowerExchange for Google Cloud Storage connection to read data from Google Cloud Storage data objects.

The following table shows the complex files that a mapping can process within Google Cloud Storage in the Hadoop environment:

Resource Format	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1]	Spark
JSON	- Flat - Hierarchical [1]	Spark
Parquet	- Flat - Hierarchical [1]	Spark

[1] To run on the Spark engine, the complex file read operation must be enabled to project columns as complex data type.

Complex File Sources on HDFS

Use a PowerExchange for HDFS connection to read data from HDFS data objects.

The following table shows the complex files that a mapping can process within HDFS storage in the Hadoop environment:

Resource Format	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1] [2]	- Blaze - Spark
Intelligent Structure Model or Sample File	- Flat - Hierarchical [1] [2]	- Blaze - Spark
JSON	- Flat [1] - Hierarchical [1] [2]	- Blaze - Spark
ORC	- Flat	- Spark

Resource Format	Supported Formats	Supported Engines
Parquet	- Flat - Hierarchical [1] [2]	- Blaze - Spark

[1] To run on the Blaze engine, the complex file data object must be connected to a Data Processor transformation.

[2] To run on the Spark engine, the complex file read operation must be enabled to project columns as complex data type.

Complex File Sources on Hive

Use a PowerExchange for Hive connection to read data from a Hive data warehouse.

The following table shows the complex files that a mapping can process within Hive storage in the Hadoop environment:

Resource Format	Supported Formats	Supported Engines
Avro	- Flat	- Blaze
ORC	- Flat - Hierarchical[1]	- Blaze - Spark
Parquet	- Flat	- Blaze
RCfile	- Flat	- Blaze
Sequence	- Flat	- Blaze
Text	- Flat	- Blaze

[1] Hierarchical data is only supported on the Spark engine.

Complex File Sources on MapR-FS

Use a PowerExchange for HDFS connection to read data from MapR-FS data objects.

The following table shows the complex files that a mapping can process within MapR-FS storage in the Hadoop environment:

File Type	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1] [2]	- Blaze - Spark
JSON	- Flat [1] - Hierarchical [1] [2]	- Blaze - Spark
ORC	- Flat	- Spark

File Type	Supported Formats	Supported Engines
Parquet	- Flat - Hierarchical [1] [2]	- Blaze - Spark

[1] To run on the Blaze engine, the complex file data object must be connected to a Data Processor transformation.

[2] To run on the Spark engine, the complex file read operation must be enabled to project columns as complex data type.

Complex File Sources on Azure Blob

You can use a PowerExchange for HDFS or a PowerExchange for Microsoft Azure Blob Storage connection to read data from Azure Blob data objects.

The following table shows the complex files that a mapping can process within Azure Blob storage in the Hadoop environment:

Resource Format	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1]	Spark
Intelligent Structure Model or Sample File	- Flat - Hierarchical [1]	Spark
JSON	- Flat - Hierarchical [1]	Spark
Parquet	- Flat - Hierarchical [1]	Spark

[1] To run on the Spark engine, the complex file read operation must be enabled to project columns as complex data type.

[2] To run on the Spark engine, the complex file read operation must be enabled to project columns as complex data type.

The following table shows the complex files that a PowerExchange for HDFS connection can process within Azure Blob Storage in the Hadoop environment:

File Type	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1] [2]	- Blaze - Spark
JSON	- Flat [1] - Hierarchical [1] [2]	- Blaze - Spark
ORC	- Flat	- Spark

File Type	Supported Formats	Supported Engines
Parquet	- Flat - Hierarchical [1] [2]	- Blaze - Spark

[1] To run on the Blaze engine, the complex file data object must be connected to a Data Processor transformation.

[2] To run on the Spark engine, the complex file read operation must be enabled to project columns as complex data type.

Complex File Sources on ADLS

You can use a PowerExchange for HDFS or a PowerExchange for Microsoft Azure Data Lake Store connection to read data from ADLS data objects. If you use a PowerExchange for Microsoft Azure Data Lake Store connection, you cannot read compressed binary files from ADLS.

The following table shows the complex files that a PowerExchange for Microsoft Azure Data Lake Store connection can process within ADLS storage in the Hadoop environment:

File Type	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1]	Spark
Intelligent Structure Model or Sample File	- Flat - Hierarchical [1]	Spark
JSON	- Flat - Hierarchical [1]	Spark
Parquet	- Flat - Hierarchical [1]	Spark

[1] To run on the Spark engine, the complex file read operation must be enabled to project columns as complex data type.

The following table shows the complex files that a PowerExchange for HDFS connection can process within ADLS storage in the Hadoop environment:

File Type	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1] [2]	- Blaze - Spark
JSON	- Flat [1] - Hierarchical [1] [2]	- Blaze - Spark
ORC	- Flat	- Spark

File Type	Supported Formats	Supported Engines
Parquet	- Flat - Hierarchical [1] [2]	- Blaze - Spark

[1] To run on the Blaze engine, the complex file data object must be connected to a Data Processor transformation.

[2] To run on the Spark engine, the complex file read operation must be enabled to project columns as complex data type.

Complex File Sources on Azure Data Lake Storage Gen2

You can use a PowerExchange for HDFS or a PowerExchange for Microsoft Azure Data Lake Storage Gen2 connection to read data from Azure Data Lake Storage Gen2 data objects.

The PowerExchange for Microsoft Azure Data Lake Storage Gen2 connection is an Azure Blob File System (ABFS) connection. You can use it if you have a PowerExchange for Microsoft Azure Data Lake Storage Gen2 connection license.

The following table shows the complex files that a PowerExchange for Microsoft Azure Data Lake Storage Gen2 mapping can process within Azure Data Lake Storage Gen2 in the Hadoop environment:

Resource Format	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1]	Spark
Intelligent Structure Model or Sample File	- Flat - Hierarchical [1]	Spark
JSON	- Flat - Hierarchical [1]	Spark
ORC	- Flat	Spark
Parquet	- Flat - Hierarchical [1]	Spark

[1] To run on the Spark engine, the complex file read operation must be enabled to project columns as complex data type.

The following table shows the complex files that a PowerExchange for HDFS connection can process within Azure Data Lake Storage Gen2 in an HDInsight cluster in the Hadoop environment:

File Type	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1] [2]	- Blaze - Spark
JSON	- Flat [1] - Hierarchical [1] [2]	- Blaze - Spark
ORC	- Flat	- Spark

File Type	Supported Formats	Supported Engines
Parquet	- Flat - Hierarchical [1] [2]	- Blaze - Spark

[1] To run on the Blaze engine, the complex file data object must be connected to a Data Processor transformation.

[2] To run on the Spark engine, the complex file read operation must be enabled to project columns as complex data type.

Flat File Sources on Hadoop

A mapping that is running in a Hadoop environment can read a flat file source from a native environment.

Consider the following limitations when you configure the mapping to read a flat file source:

- You cannot use an indirect source type.
- The row size in a flat file source cannot exceed 190 MB.
- You cannot use a command to generate or to transform flat file data and send the output to the flat file reader at run time.

Generate the Source File Name

You can generate the source file name for the flat file data object. The content of the file name column remains consistent across different modes of execution.

When you push processing to the specific engine for the required file types, the file name column returns the path based on the following formats:

Run-time Engine	Source File Location	Returned Path

Blaze	Local system	`<staged path><local file path>`
		For example,
		`hdfs://host name:port/hive/warehouse/home/devbld/Desktop/ff.txt`

Spark	HDFS	`hdfs://<host name>:<port>/<file name path>`
		For example,
		`hdfs://host name:port/hive/warehouse/ff.txt`

Spark	Local system	`<local file path>`
		For example,
		`/home/devbld/Desktop/ff.txt`

The file name column returns the content in the following format for a high availability cluster: `hdfs://<host name>/<file name path>`

For example, `hdfs://irldv:5008/hive/warehouse/ff.txt`

Relational Sources on Hadoop

You can run mappings with relational sources on the Blaze engine.

The Data Integration Service does not run pre-mapping SQL commands or post-mapping SQL commands against relational sources. You cannot validate and run a mapping with PreSQL or PostSQL properties for a relational source in a Hadoop environment.

The Data Integration Service can use multiple partitions to read from the following relational sources:

- IBM DB2
- Oracle

Hive Sources on Hadoop

You can include Hive sources in an Informatica mapping that runs in the Hadoop environment.

Consider the following restrictions when you configure a Hive source in a mapping that runs in the Hadoop environment:

- A mapping fails to run when you have Unicode characters in a Hive source definition.
- The Hive source cannot reside on a remote cluster. A remote cluster is a cluster that is remote from the machine that the Hadoop connection references in the mapping.
- If a Hive source contains hierarchical data, you cannot push transformation logic to the source database.
- The third-party Hive JDBC driver does not return the correct precision and scale values for the Decimal data type. As a result, when you import Hive tables with a Decimal data type into the Developer tool, the Decimal data type precision is set to 38 and the scale is set to 0. Consider the following configuration rules and guidelines based on the version of Hive:
  - Hive 0.11. Accept the default precision and scale for the Decimal data type in the Developer tool.
  - Hive 0.12. Accept the default precision and scale for the Decimal data type in the Developer tool.
  - Hive 0.12 with Cloudera CDH 5.0. You can configure the precision and scale fields for source columns with the Decimal data type in the Developer tool.
  - Hive 0.13 and above. You can configure the precision and scale fields for source columns with the Decimal data type in the Developer tool.
  - Hive 0.14 or above. The precision and scale used for the Decimal data type in the Hive database also appears in the Developer tool.

A mapping that runs on the Spark engine can have partitioned Hive source tables and bucketed sources.

PreSQL and PostSQL Commands

You can create SQL commands for Hive sources. You can execute the SQL commands to execute SQL statements such as insert, update, and delete on the Hive source.

PreSQL is an SQL command that runs against the Hive source before the mapping reads from the source. PostSQL is an SQL command that runs against the Hive source after the mapping writes to the target.

You can use PreSQL and PostSQL on the Spark engine. The Data Integration Service does not validate PreSQL or PostSQL commands for a Hive source.

Note: You can manually validate the SQL by running the following query in a Hive command line utility:

```
CREATE VIEW <table name> (<port list>) AS <SQL>
```

where:

- `<table name>` is a name of your choice
- `<port list>` is the comma-delimited list of ports in the source
- `<SQL>` is the query to validate

Pre-Mapping SQL Commands

PreSQL is an SQL command that runs against a Hive source before the mapping reads from the source.

For example, you might use a Hive source in a mapping. The data stored in the Hive source changes regularly and you must update the data in the Hive source before the mapping reads from the source to make sure that the mapping reads the latest records. To update the Hive source, you can configure a PreSQL command.

Post-Mapping SQL Commands

PostSQL is an SQL command that runs against a Hive source after the mapping writes to the target.

For example, you might use a Hive source in a mapping. After the mapping writes to a target, you might want to delete the stage records stored in the Hive source. You want to run the command only after the mapping writes the data to the target to make sure that the data is not removed prematurely from the Hive source. To delete the records in the Hive source table after the mapping writes to the target, you can configure a PostSQL command.

Rules and Guidelines for Pre- and Post-Mapping SQL Commands

Consider the following restrictions when you run PreSQL and PostSQL commands against Hive sources:

- When you create an SQL override on a Hive source, you must enclose keywords or special characters in backtick (`) characters.
- When you run a mapping with a Hive source in the Hadoop environment, references to a local path in pre-mapping SQL commands are relative to the Data Integration Service node. When you run a mapping with a Hive source in the native environment, references to local path in pre-mapping SQL commands are relative to the Hive server node.

Rules and Guidelines for Hive Sources on the Blaze Engine

You can include Hive sources in an Informatica mapping that runs on the Blaze engine.

Consider the following rules and guidelines when you configure a Hive source in a mapping that runs on the Blaze engine:

- A mapping that runs on the Blaze engine can have bucketed Hive sources and Hive ACID tables.
- Hive ACID tables must be bucketed.
- The Blaze engine supports Hive tables that are enabled for locking.
- Hive sources can contain quoted identifiers in Hive table names, column names, and schema names.
- The TEXT storage format in a Hive source for a Blaze mapping can support ASCII characters as column delimiters and the newline characters as a row separator. You cannot use hex values of ASCII characters. For example, use a semicolon (;) instead of 3B.
- You can define an SQL override in the Hive source for a Blaze mapping.
- The Blaze engine can read from an RCFile as a Hive source. To read from an RCFile table, you must create the table with the `SerDe` clause.

- The Blaze engine can read from Hive tables that are compressed. To read from a compressed Hive table, you must set the `TBLPROPERTIES` clause.

RCFile as Hive Tables

The Blaze engine can read and write to RCFile as Hive tables. However, the Blaze engine supports only the ColumnarSerDe SerDe. In Hortonworks, the default SerDe for an RCFile is LazyBinaryColumnarSerDe. To read and write to an RCFile table, you must create the table by specifying the SerDe as `org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe`.

For example:

```
CREATE TABLE TEST_RCFile
(id int, name string)
ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe' STORED AS RCFILE;
```

You can also set the default RCFile SerDe from the Ambari or Cloudera manager. Set the property `hive.default.rcfile.serde` to `org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe`.

Compressed Hive Tables

The Blaze engine can read and write to Hive tables that are compressed. However, to read from a compressed Hive table or write to a Hive table in compressed format, you must set the `TBLPROPERTIES` clause as follows:

- When you create the table, set the table properties:
  `TBLPROPERTIES ('property_name'='property_value')`
- If the table already exists, alter the table to set the table properties:
  `ALTER TABLE table_name SET TBLPROPERTIES ('property_name' = 'property_value');`

The property name and value are not case sensitive. Depending on the file format, the table property can take different values.

The following table lists the property names and values for different file formats:

File Format	Table Property Name	Table Property Values
Avro	avro.compression	BZIP2, deflate, Snappy
ORC	orc.compress	Snappy, ZLIB
Parquet	parquet.compression	GZIP, Snappy
RCFile	rcfile.compression	Snappy, ZLIB
Sequence	sequencefile.compression	BZIP2, GZIP, LZ4, Snappy
Text	text.compression	BZIP2, GZIP, LZ4, Snappy

Note: The Blaze engine does not write data in the default ZLIB compressed format when it writes to a Hive target stored as ORC format. To write in a compressed format, alter the table to set the TBLPROPERTIES clause to use ZLIB or Snappy compression for the ORC file format.

The following text shows sample commands to create table and alter table:

- Create table:

```
create table CBO_3T_JOINS_CUSTOMER_HIVE_SEQ_GZIP
(C_CUSTKEY DECIMAL(38,0), C_NAME STRING,C_ADDRESS STRING,
C_PHONE STRING,C_ACCTBAL DECIMAL(10,2),
```

```
C_MKTSEGMENT VARCHAR(10),C_COMMENT vARCHAR(117))
partitioned by (C_NATIONKEY DECIMAL(38,0))
TBLPROPERTIES ('sequencefile.compression'='gzip')
stored as SEQUENCEFILE;
```

- Alter table:

```
ALTER TABLE table_name
SET TBLPROPERTIES (avro.compression'='BZIP2');
```

Sqoop Sources on Hadoop

Sqoop sources are valid in mappings in a Hadoop environment.

You can include a JDBC-compliant database as a Sqoop source in an Informatica mapping that runs in a Hadoop environment.

For example, you can include the following sources in a Sqoop mapping:

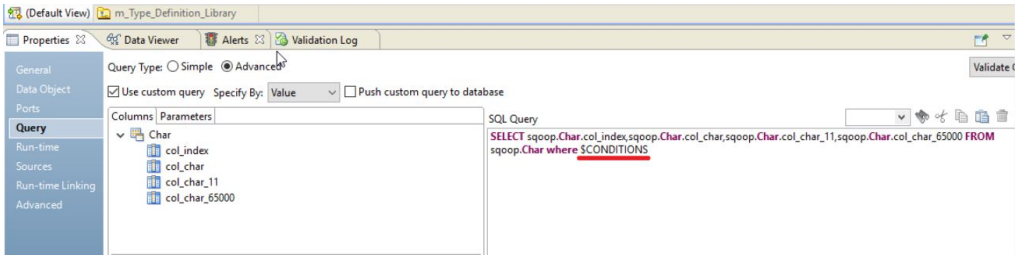
- Aurora
- Greenplum
- IBM DB2
- IBM DB2 for z/OS
- Microsoft SQL Server
- Netezza
- Oracle
- Teradata
- Vertica

Reading Data from Vertica Sources through Sqoop

To read Vertica data through Sqoop connectivity, Sqoop requires a custom query.

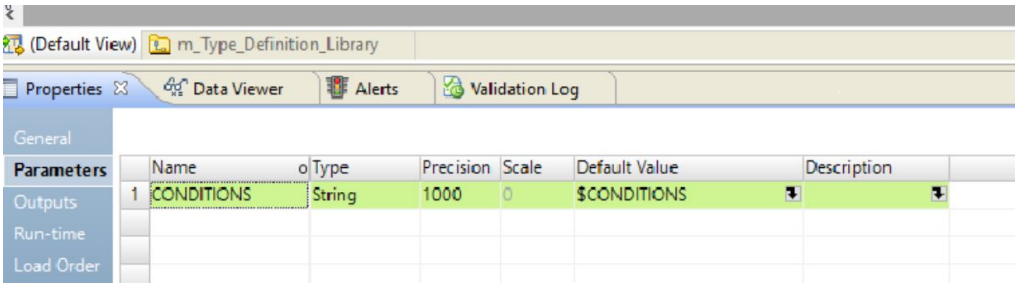
Create the custom query with the `$CONDITIONS` parameter in the WHERE clause. Then, define a mapping parameter with the name as `CONDITIONS` and set its value to `$CONDITIONS`. At run time, Sqoop replaces `$CONDITIONS` with a unique condition expression.

1. Open the Sqoop data object and click the **Query** tab in the **Properties** view.
2. Define a custom query and include `$CONDITIONS` in the WHERE clause of the custom query as shown in the following image:



3. Open the Sqoop mapping and click the **Parameters** tab in the **Properties** view.

4. Define a mapping parameter with the name as `CONDITIONS` and set its value to `$CONDITIONS` as shown in the following image:



Rules and Guidelines for Sqoop Sources

Consider the following rules and guidelines when you configure a Sqoop source in a mapping:

- If you specify a sort condition in a mapping, the Data Integration Service ignores the Order By condition.
- You cannot sort columns in a Sqoop source.
- You cannot read distinct rows from a Sqoop source.
- When you read data from an Oracle source through Sqoop and run the mapping on the Blaze or Spark engine, Sqoop treats the owner name as case sensitive.
- Sqoop uses the values that you configure in the **User Name** and **Password** fields of the JDBC connection. If you configure the --username or --password argument in a JDBC connection or mapping, Sqoop ignores the arguments. If you create a password file to access a database, Sqoop ignores the password file.

Rules and Guidelines for Sqoop Queries

Consider the following rules and guidelines when you configure a Sqoop query in a mapping:

- To override the default query in a mapping with an advanced query, you must define a mapping parameter with the name as `CONDITIONS` and set its value to `$CONDITIONS`. You must then include `$CONDITIONS` in the WHERE clause of the custom query. For more information about configuring the custom query, see the following Informatica Knowledge Base article: KB 564550
- If you define a custom query, you must verify that the metadata of the custom query matches the metadata of the source object. Otherwise, Sqoop might write blank values to the target.
- When you enable OraOop and configure an advanced query to read data from an Oracle source through Sqoop, the mapping fails on the Spark engine.

CHAPTER 5

Targets

This chapter includes the following topics:

- Overview of Targets, 100
- PowerExchange Adapter Targets, 100
- Targets on Databricks, 102
- File Targets on Hadoop, 105
- Message Targets on Hadoop, 111
- Relational Targets on Hadoop, 111
- Hive Targets on Hadoop, 112
- Sqoop Targets on Hadoop, 115

Overview of Targets

You can push a mapping to a non-native environment that includes a target from the native environment or non-native environment. Some targets have restrictions when you reference them in a non-native environment.

Access to targets in a non-native environment might require a PowerExchange adapter.

PowerExchange Adapter Targets

PowerExchange targets can run in the Hadoop or Databricks environment, based on the adapter.

The following table lists PowerExchange targets and the non-native engines that support the targets:

PowerExchange Adapter	Supported Engines
Amazon Redshift	- Blaze - Spark - Databricks Spark
Amazon S3	- Blaze - Spark - Databricks Spark

PowerExchange Adapter	Supported Engines
Google Analytics	- Spark
Google BigQuery	- Spark
Google Cloud Spanner	- Spark
Google Cloud Storage	- Spark
HBase	- Blaze - Spark
HDFS	- Blaze - Spark
JDBC V2	- Spark - Databricks Spark
Kudu	- Spark
Hive	- Blaze - Spark
MapR-DB	- Blaze - Spark
Microsoft Azure Blob Storage	- Spark - Databricks Spark
Microsoft Azure Cosmos DB	- Spark - Databricks Spark
Microsoft Azure Data Lake Storage Gen1	- Spark - Databricks Spark
Microsoft Azure Data Lake Storage Gen2	- Spark - Databricks Spark
Microsoft Azure SQL Data Warehouse	- Spark - Databricks Spark
Snowflake	- Spark - Databricks Spark
Teradata Parallel Transporter API	- Blaze - Spark

Targets on Databricks

A mapping that runs in the Databricks environment can include file and database targets.

The following table lists the storage types and targets:

Storage Type	Target
File	- Amazon Simple Storage Service (Amazon S3) - Databricks Delta Lake storage - Microsoft Azure Blob Storage - Microsoft Azure Data Lake Store (ADLS)
Database	- Amazon Redshift - JDBC V2 - Microsoft Azure Cosmos DB - Microsoft Azure SQL Data Warehouse - Snowflake

Complex File Targets on Amazon S3

Use a PowerExchange for Amazon S3 connection to write data to Amazon S3 data objects.

The following table shows the complex files that a mapping can process within Amazon S3 storage on the Databricks Spark engine:

File Type	Supported Formats
Avro	- Flat - Hierarchical
JSON	- Flat - Hierarchical
ORC	- Flat
Parquet	- Flat - Hierarchical

Complex File Targets on ADLS

Use a PowerExchange for Microsoft Azure Data Lake Store connection to write data to ADLS data objects.

The following table shows the complex files that a mapping can process within ADLS storage in the Databricks environment:

File Type	Format
Avro	- Flat - Hierarchical
JSON	- Flat - Hierarchical

File Type	Format
Parquet	- Flat - Hierarchical
Note: The write operation for hierarchical files must be enabled to project columns as complex data type.	

Complex File Targets on Azure Data Lake Storage Gen2

Use a PowerExchange for Microsoft Azure Data Lake Storage Gen2 connection to write data to Azure Data Lake Storage Gen2 data objects.

The following table shows the complex files that a mapping can process within Azure Data Lake Storage Gen2 in the Databricks environment:

File Type	Format
Avro	- Flat - Hierarchical
JSON	- Flat - Hierarchical
ORC	- Flat
Parquet	- Flat - Hierarchical
Note: The write operation for hierarchical files must be enabled to project columns as complex data type.	

Complex File Targets on Azure Blob

Use a PowerExchange for Microsoft Azure Blob Storage connection to write data to Blob Storage data objects. You can write compressed binary files.

The following table shows the complex files that a mapping can process within Azure Blob storage in the Databricks environment:

File Type	Format
Avro	- Flat - Hierarchical
JSON	- Flat - Hierarchical
Parquet	- Flat - Hierarchical
Note: The write operation for hierarchical files must be enabled to project columns as complex data type.	

Databricks Delta Lake

You can use Databricks Delta Lake as a target in mappings.

Delta Lake is a data lake resource that stores data in large tables. Databricks uses proprietary Delta software to manage stored data and allow fast access to the data. Delta Lake supports ACID transactions.

Mappings can access Delta Lake resources on the AWS or Azure platforms. You can access Delta Lake tables built on top of any of the following storage types:

- Azure blob storage
- Azure Data Lake Storage (ADLS) Gen1 or Gen2
- Amazon Web Services (AWS) S3.

To use Delta Lake resources in a mapping, create a relational data object and use a JDBC connection to connect to the resource.

To perform data ingestion to a Delta Lake target, use a pass-through mapping with a flat file source.

Rules and Guidelines for Databricks Targets

File Targets

Consider the following general rules and guidelines for file targets:

- The file targets must reside on Microsoft Azure Blob Storage or Microsoft Azure Data Lake Store.
- The row delimiter must be /n.
- The file cannot be fixed width.
- Multiple column delimiters are not supported.
- Empty values only are treated as null values.

Delta Lake Targets

Consider the following guidelines for using Delta Lake targets:

- Mappings that access Delta Lake tables must use the Databricks run-time engine. If you run a Delta Lake mapping in the native environment with the JDBC connection, the mapping succeeds, but no data is written to the target.

Null Processing

Consider the following rules and guidelines for null processing:

Null value conversions

When the Databricks Spark engine writes to a target, it converts null values to empty strings (" "). For example, 12, AB,"",23p09udj.

Consider the null processing behavior based on the following Databricks targets:

- File target. The Databricks Spark engine writes all null values as empty strings.
- Azure SQL Data Warehouse. The Databricks Spark engine can write the empty strings to string columns, but when it tries to write an empty string to a non-string column, the mapping fails with a type mismatch.

To allow the Databricks Spark engine to convert the empty strings back to null values and write to the target, configure the following advanced property in the Databricks Spark connection:
infaspark.flatfile.writer.nullValue=true

Unexpected values converted to nulls

The Databricks Spark engine generates null values for all fields in the same record if any field contains an unexpected value based on the following scenarios:

- Any type mismatch occurs, such as passing string data to a numeric column.
- Data is out of bounds, such as with bigint or int data types.

Consider using a Filter transformation to filter out null rows.

Date/time values converted to nulls

When the Databricks Spark engine writes date/time values, it uses the format YYY-MM-DD HH24:MM:SS.US. If the date format that passes to the target does not match this format, the Databricks Spark engine writes null values.

Double and Decimal Conversions

When the Databricks Spark engine reads from an Azure or AWS source, it converts double and decimal data types to scientific notation. When it converts that data back to a double or decimal to write to the target, it drops precision greater than 15 and maintains precision of 15 digits.

File Targets on Hadoop

A mapping that runs in the Hadoop environment can process complex files and flat files.

To write large volumes of data, you can connect a complex file target to write data to a directory of files that have the same format and properties. You can read compressed binary files.

You can write to complex files in the following storage types in the Hadoop environment:

- Amazon Simple Storage Service (Amazon S3)
- Hadoop Distributed File System (HDFS)
- Hive
- MapR File System (MapR-FS)
- Microsoft Azure Blob Storage (Azure Blob Storage)
- Microsoft Azure Data Lake Store (ADLS)

Complex File Targets on Amazon S3

Use a PowerExchange for Amazon S3 connection to write data to Amazon S3 data objects.

The following table shows the complex files that a mapping can process within Amazon S3 storage in the Hadoop environment:

File Type	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1] [2]	- Blaze - Spark
JSON	- Flat - Hierarchical [1] [2]	- Blaze - Spark

File Type	Supported Formats	Supported Engines
ORC	- Flat	- Spark
Parquet	- Flat - Hierarchical [1] [2]	- Blaze - Spark

[1] To run on the Blaze engine, the complex file data object must be connected to a Data Processor transformation.

[2] To run on the Spark engine, the complex file write operation must be enabled to project columns as complex data type.

Complex File Targets on Google Cloud Storage

You can use a PowerExchange for Google Cloud Storage connection to write data to Google Cloud Storage data objects. If you use a PowerExchange for Google Cloud Storage connection, you cannot write compressed binary files to Google Cloud Storage.

The following table shows the complex files that a PowerExchange for Google Cloud Storage connection can process within Google Cloud Storage storage in the Hadoop environment:

File Type	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1]	Spark
JSON	- Flat - Hierarchical [1]	Spark
Parquet	- Flat - Hierarchical [1]	Spark

[1] To run on the Spark engine, the complex file write operation must be enabled to project columns as complex data type.

Complex File Targets on ADLS

You can use a PowerExchange for HDFS or a PowerExchange for Microsoft Azure Data Lake Store connection to write data to ADLS data objects. If you use a PowerExchange for Microsoft Azure Data Lake Store connection, you cannot write compressed binary files to ADLS.

The following table shows the complex files that a PowerExchange for Microsoft Azure Data Lake Store connection can process within ADLS storage in the Hadoop environment:

File Type	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1]	Spark
JSON	- Flat - Hierarchical [1]	Spark

File Type	Supported Formats	Supported Engines
Parquet	- Flat - Hierarchical [1]	Spark
[1] To run on the Spark engine, the complex file write operation must be enabled to project columns as complex data type.		

The following table shows the complex files that a PowerExchange for HDFS connection can process within ADLS storage in the Hadoop environment:

File Type	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1 2]	- Blaze - Spark
JSON	- Flat [1] - Hierarchical [1 2]	- Blaze - Spark
ORC	- Flat	- Spark
Parquet	- Flat - Hierarchical [1 2]	- Blaze - Spark
[1] To run on the Blaze engine, the complex file data object must be connected to a Data Processor transformation. [2] To run on the Spark engine, the complex file write operation must be enabled to project columns as complex data type.		

Complex File Targets on Azure Data Lake Storage Gen2

You can use a PowerExchange for HDFS or a PowerExchange for Microsoft Azure Data Lake Storage Gen2 connection to write data to Azure Data Lake Storage Gen2 data objects.

The following table shows the complex files that a PowerExchange for Microsoft Azure Data Lake Storage Gen2 mapping can process within Azure Data Lake Storage Gen2 in the Hadoop environment:

File Type	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1]	Spark
JSON	- Flat - Hierarchical [1]	Spark
ORC	- Flat	Spark
Parquet	- Flat - Hierarchical [1]	Spark
[1] To run on the Spark engine, the complex file write operation must be enabled to project columns as complex data type.		

The following table shows the complex files that a PowerExchange for HDFS connection can process within Azure Data Lake Storage Gen2 in the Hadoop environment:

File Type	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1] [2]	- Blaze - Spark
JSON	- Flat [1] - Hierarchical [1] [2]	- Blaze - Spark
ORC	- Flat	- Spark
Parquet	- Flat - Hierarchical [1] [2]	- Blaze - Spark

[1] To run on the Blaze engine, the complex file data object must be connected to a Data Processor transformation.
[2] To run on the Spark engine, the complex file write operation must be enabled to project columns as complex data type.

Complex File Targets on Azure Blob

You can use a PowerExchange for HDFS or a PowerExchange for Microsoft Azure Blob Storage connection to write data to Azure Blob data objects.

The following table shows the complex files that a mapping can process within Azure Blob storage in the Hadoop environment:

File Type	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1]	Spark
JSON	- Flat - Hierarchical [1]	Spark
Parquet	- Flat - Hierarchical [1]	Spark

[1] To run on the Spark engine, the complex file write operation must be enabled to project columns as complex data type.

The following table shows the complex files that a PowerExchange for HDFS connection can process within Azure Blob Storage in the Hadoop environment:

File Type	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1] [2]	- Blaze - Spark
JSON	- Flat [1] - Hierarchical [1] [2]	- Blaze - Spark
ORC	- Flat	- Spark

File Type	Supported Formats	Supported Engines
Parquet	- Flat - Hierarchical [1] [2]	- Blaze - Spark

[1] To run on the Blaze engine, the complex file data object must be connected to a Data Processor transformation.

[2] To run on the Spark engine, the complex file write operation must be enabled to project columns as complex data type.

Complex File Targets on MapR-FS

Use a PowerExchange for HDFS connection to read data from MapR-FS data objects.

The following table shows the complex files that a mapping can process within MapR-FS storage in the Hadoop environment:

File Type	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1] [2]	- Blaze - Spark
JSON	- Flat [1] - Hierarchical [1] [2]	- Blaze - Spark
ORC	- Flat	- Spark
Parquet	- Flat - Hierarchical [1] [2]	- Blaze - Spark

[1] To run on the Blaze engine, the complex file data object must be connected to a Data Processor transformation.

[2] To run on the Spark engine, the complex file read operation must be enabled to project columns as complex data type.

Complex File Targets on HDFS

Use a PowerExchange for HDFS connection to write data to HDFS data objects.

The following table shows the complex files that a mapping can process within HDFS storage in the Hadoop environment:

File Type	Supported Formats	Supported Engines
Avro	- Flat - Hierarchical [1] [2]	- Blaze - Spark
JSON	- Flat [1] - Hierarchical [1] [2]	- Blaze - Spark
ORC	- Flat	- Spark

File Type	Supported Formats	Supported Engines
Parquet	- Flat - Hierarchical [1] [2]	- Blaze - Spark
[1] To run on the Blaze engine, the complex file data object must be connected to a Data Processor transformation. [2] To run on the Spark engine, the complex file write operation must be enabled to project columns as complex data type.		

Complex File Targets on Hive

Use a PowerExchange for Hive connection to write data to a Hive data object.

The following table shows the complex files that a mapping can process within Hive storage in the Hadoop environment:

Resource Format	Supported Formats	Supported Engines
Avro	- Flat	- Blaze
ORC	- Flat - Hierarchical[1]	- Blaze - Spark
Parquet	- Flat	- Blaze
RCfile	- Flat	- Blaze
Sequence	- Flat	- Blaze
Text	- Flat	- Blaze
[1] Hierarchical data is only supported on the Spark engine.		

Flat File Targets on Hadoop

Consider the following rules and guidelines for flat file targets in the Hadoop environment:

- A mapping that runs in the Hadoop environment can write to a flat file target in the native environment.
- The Data Integration Service truncates the target files and reject files before writing the data. When you use a flat file target, you cannot append output data to target files and reject files.
- The Data Integration Service can write to a file output for a flat file target. When you have a flat file target in a mapping, you cannot write data to a command.

Consider the following rules and guidelines for HDFS flat file targets:

- The Data Integration Service truncates the target files and reject files before writing the data. To append output data to HDFS target files and reject files, choose to append data if the HDFS target exists.

  Data is appended to reject files only if the reject file directory is on the Data Integration Service machine. If the directory is in the Hadoop environment, rejected rows are overwritten.
- When you choose to append data if the HDFS target exists, the Data Integration Service appends the mapping execution ID to the names of the target files and reject files.

- When you use a HDFS flat file target in a mapping, you must specify the full path that includes the output file directory and file name. The Data Integration Service might generate multiple output files in the output directory when you run the mapping in a Hadoop environment.
- An HDFS target cannot reside on a remote cluster. A remote cluster is a cluster that is remote from the machine that the Hadoop connection references in the mapping.

Message Targets on Hadoop

You can write to Kafka messaging targets.

Apache Kafka is a publish-subscribe messaging system. Kafka runs as a cluster comprised of one or more servers each of which is called a broker. Kafka brokers process data in the form of messages, publishes the messages to a topic, subscribes the messages from a topic, and then writes it to the Kafka target.

You can use the Kafka data object write operation as a target. If you want to configure high availability for the mapping, ensure that the Kafka cluster is highly available.

For more information about Kafka clusters, Kafka brokers, and partitions, see https://kafka.apache.org/documentation.

Technical Preview: Support to write the batch data to a Kafka target in Data Engineering Integration is available for technical preview. Technical preview functionality is supported for evaluation purposes but is unwarranted and is not supported in production environments or any environment that you plan to push to production. Informatica intends to include the preview functionality in an upcoming release for production use, but might choose not to in accordance with changing market or technical circumstances. For more information, contact Informatica Global Customer Support.

Relational Targets on Hadoop

You can run mappings with relational targets on the Blaze engine.

The Data Integration Service does not run pre-mapping SQL commands or post-mapping SQL commands against relational targets in a Hadoop environment. You cannot validate and run a mapping with PreSQL or PostSQL properties for a relational target in a Hadoop environment.

The Data Integration Service can use multiple partitions to write to the following relational targets:

- IBM DB2
- Oracle

Hive Targets on Hadoop

A mapping run in the Hadoop environment can write to a Hive target. When you write to a Hive target, consider the processing differences for functionality such as table types, DDL queries, and bucketing.

Hive Table Types

A Hive target can be an internal table or an external table. Internal Hive tables are managed by Hive and are also known as managed tables. External Hive tables are tables managed by an external source such as HDFS, Amazon S3, or Microsoft Azure Blob Storage.

When a mapping creates or replaces a Hive table, the type of table that the mapping creates depends on the run-time engine that you use to run the mapping:

- On the Blaze engine, mappings create managed tables.
- On the Spark engine, mappings create external tables.

DDL Queries

For mappings that run on the Spark engine or the Blaze engine, you can create a custom DDL query that creates or replaces a Hive table at run time. However, with the Blaze engine, you cannot use a backtick (`) character in the DDL query. The backtick character is required in HiveQL when you include special characters or keywords in a query.

Bucketing

The Spark engine can write to bucketed Hive targets. Bucketing and partitioning of Hive tables can improve performance by reducing data shuffling and sorting.

Hortonworks HDP 3.1

Hortonworks HDP 3.1 uses ACID-enabled ORC targets by default and uses ACID for all managed tables. If you do not want to use ACID-enabled tables, use external tables.

PreSQL and PostSQL Commands

You can create SQL commands for Hive targets. You can execute the SQL commands to execute SQL statements such as insert, update, and delete on the Hive target.

PreSQL is an SQL command that runs against the Hive target before the mapping reads from the source. PostSQL is an SQL command that runs against the Hive target after the mapping writes to the target.

You can use PreSQL and PostSQL on the Spark engine. The Data Integration Service does not validate PreSQL or PostSQL commands for a Hive target.

Pre-Mapping SQL Commands

PreSQL is an SQL command that runs against a Hive target before the mapping reads from a source.

For example, you might use a Hive target in a mapping. The data stored in the Hive target contains old records from the previous day and you must delete the old records in the Hive target before you run the mapping that writes new records to the target. To delete the old records in the Hive target, you can configure a PreSQL command.

Post-Mapping SQL Commands

PostSQL is an SQL command that runs against a Hive target after the mapping writes to the target.

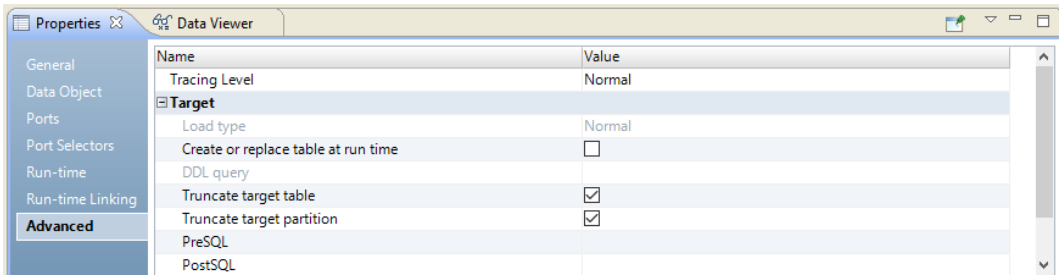
For example, you might use a Hive target in a mapping that runs in a test environment. After the mapping writes to the target, you might want to delete the records stored in the Hive target so that you can test the

same Hive target again in the next mapping run. To delete the records in the Hive target, you can configure a PostSQL command.

Truncating Hive Targets

Truncate Hive target tables to delete the table contents. You can truncate internal and external Hive tables in the Hadoop environment.

Truncate a Hive table in the Hive table properties. The following image shows the Hive table properties:



To truncate the entire Hive table, choose the option to truncate the target table.

To truncate only the partitions in the Hive table for which the transformation received input data, you must choose to truncate the target table and to truncate the partition in the Hive target table.

Note: You must choose both "Truncate target table" and "Truncate target partition" options due to the possibility that input data might affect all partitions and cause the whole table to be truncated.

You can design a mapping to truncate an internal or external Hive table that is bucketed and partitioned.

Consider the following restrictions when you truncate a Hive table in a mapping that runs in the Hadoop environment:

- The Data Integration Service can truncate the partition in the Hive target in which the data is being inserted.
- You must truncate the target table to overwrite data to a Hive table with Hive version 0.7. The Data Integration Service ignores write, update override, delete, insert, and update strategy properties when it writes data to a Hive target.

Updating Hive Targets with an Update Strategy Transformation

For mappings that run on the Spark engine, you can use Hive MERGE statements to perform Update Strategy tasks. When a query uses a MERGE statement instead of INSERT, UPDATE or DELETE statements, processing is more efficient.

To use Hive MERGE, select **true** for the option in the Advanced Properties of the Update Strategy transformation.

The mapping ignores the Hive MERGE option and the Data Integration Service uses INSERT, UPDATE and DELETE to perform the operation under the following scenarios:

- The mapping runs on the Blaze engine.
- Hive MERGE is restricted on the Hadoop distribution.

The mapping log contains results of the operation, including whether restrictions affected results.

When the update affects partitioning or bucketing columns, updates to the columns are omitted.

Note: The Developer tool and the Data Integration Service do not validate against this restriction. If the Update Strategy expression violates these restrictions, the mapping might produce unexpected results.

Rules and Guidelines for Hive Targets on the Blaze Engine

You can include Hive targets in an Informatica mapping that runs on the Blaze engine.

Consider the following rules and guidelines when you configure a Hive target in a mapping that runs on the Blaze engine:

- Mappings that run on the Blaze engine can read and write to sorted targets.
- The Blaze engine supports Hive tables that are enabled for locking.
- The Blaze engine can create or replace Hive target tables.
- If you enable the Write transformation to maintain row order for a Hive target, the Blaze engine does not maintain row order.
- A mapping that runs on the Blaze engine can have partitioned and bucketed Hive tables as targets. However, if you append data to a bucketed table, the Blaze engine overwrites the data in the bucketed target.
- When a mapping writes to a partitioned Hive target, the impersonation user who runs the mapping must have the *ALL* privilege on the target table. The Blaze engine uses the 'MSCK REPAIR TABLE' command to write to a partitioned Hive target and *ALL* privilege is required to run the command.
- To write to a Hive ACID table, the mapping must contain an Update Strategy transformation connected to the Hive target. The update strategy expression must flag each row for insert.
- You cannot use update or delete update strategy properties.

RCFile as Hive Tables

The Blaze engine can read and write to RCFile as Hive tables. However, the Blaze engine supports only the ColumnarSerDe SerDe. In Hortonworks, the default SerDe for an RCFile is LazyBinaryColumnarSerDe. To read and write to an RCFile table, you must create the table by specifying the SerDe as `org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe`.

For example:

```
CREATE TABLE TEST_RCFIle
(id int, name string)
ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe' STORED AS RCFILE;
```

You can also set the default RCFile SerDe from the Ambari or Cloudera manager. Set the property `hive.default.rcfile.serde` to `org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe`.

Compressed Hive Tables

The Blaze engine can read and write to Hive tables that are compressed. However, to read from a compressed Hive table or write to a Hive table in compressed format, you must set the `TBLPROPERTIES` clause as follows:

- When you create the table, set the table properties:
  `TBLPROPERTIES ('property_name'='property_value')`
- If the table already exists, alter the table to set the table properties:
  `ALTER TABLE table_name SET TBLPROPERTIES ('property_name' = 'property_value');`

The property name and value are not case sensitive. Depending on the file format, the table property can take different values.

The following table lists the property names and values for different file formats:

File Format	Table Property Name	Table Property Values
Avro	avro.compression	BZIP2, deflate, Snappy
ORC	orc.compress	Snappy, ZLIB
Parquet	parquet.compression	GZIP, Snappy
RCFile	rcfile.compression	Snappy, ZLIB
Sequence	sequencefile.compression	BZIP2, GZIP, LZ4, Snappy
Text	text.compression	BZIP2, GZIP, LZ4, Snappy

Note: The Blaze engine does not write data in the default ZLIB compressed format when it writes to a Hive target stored as ORC format. To write in a compressed format, alter the table to set the TBLPROPERTIES clause to use ZLIB or Snappy compression for the ORC file format.

The following text shows sample commands to create table and alter table:

- Create table:

```
create table CBO_3T_JOINS_CUSTOMER_HIVE_SEQ_GZIP
(C_CUSTKEY DECIMAL(38,0), C_NAME STRING,C_ADDRESS STRING,
C_PHONE STRING,C_ACCTBAL DECIMAL(10,2),
C_MKTSEGMENT VARCHAR(10),C_COMMENT vARCHAR(117))
partitioned by (C_NATIONKEY DECIMAL(38,0))
TBLPROPERTIES ('sequencefile.compression'='gzip')
stored as SEQUENCEFILE;
```

- Alter table:

```
ALTER TABLE table_name
SET TBLPROPERTIES (avro.compression'='BZIP2');
```

Sqoop Targets on Hadoop

Sqoop targets are valid in mappings in a Hadoop environment.

You can include a JDBC-compliant database as a Sqoop target in an Informatica mapping that runs in a Hadoop environment:

For example, you can include the following targets in a Sqoop mapping:

- Aurora
- Greenplum
- IBM DB2
- IBM DB2 for z/OS
- Microsoft SQL Server
- Netezza
- Oracle
- Teradata
- Vertica

You can insert data. You cannot update or delete data in a target. If you configure update arguments, Sqoop ignores them.

Rules and Guidelines for Sqoop Targets

Consider the following rules and guidelines when you configure a Sqoop target in a mapping:

- If a column name or table name contains a special character, the Sqoop export process fails.
- If you configure the **Maintain Row Order** property for a Sqoop target, the Data Integration Service ignores the property.
- If a mapping contains a Sqoop source, an Aggregator transformation, and a flat file target, you must disable the **Maintain Row Order** property for the target. Otherwise, the mapping fails.
- When you run a Sqoop mapping on the Blaze engine, verify that you have not deleted any target port from the mapping. Otherwise, the mapping fails.
- When you export null data to a Microsoft SQL Server column that is defined as not null, the Data Integration Service fails the Sqoop mapping on the Blaze engine instead of rejecting and writing the null data to the bad file.
- When you write data to an Oracle target through Sqoop and run the mapping on the Blaze or Spark engine, Sqoop treats the owner name as case sensitive.
- A mapping that contains an Update Strategy transformation cannot use a Sqoop-enabled JDBC connection to write to a target. To run the mapping, disable the Sqoop connector in the Write transformation.
- Sqoop uses the values that you configure in the **User Name** and **Password** fields of the JDBC connection. If you configure the --username or --password argument in a JDBC connection or mapping, Sqoop ignores the arguments. If you create a password file to access a database, Sqoop ignores the password file.
- When you write data to a Vertica target through Sqoop, the --batch argument is required.

CHAPTER 6

Transformations

This chapter includes information about transformation support in the non-native environment.

Overview of Transformations

Due to the differences between native environments and non-native environments, only certain transformations are valid or are valid with restrictions in a non-native environment. Some functions, expressions, data types, and variable fields are not valid in a non-native environment.

Consider the following processing differences that can affect whether transformations and transformation behavior are valid or are valid with restrictions in a non-native environment:

- Compute clusters use distributed processing and process data on different nodes. Each node does not have access to the data that is being processed on other nodes. As a result, the run-time engine might not be able to determine the order in which the data originated.
- Each of the run-time engines in the non-native environment can process mapping logic differently.
- Much of the processing behavior for batch mappings on the Spark engine also apply to streaming mappings.
- You can access Delta Lake tables with any transformation that runs on Databricks Spark.

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

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

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

Transformation	Supported Engines
Merge	- Blaze - Spark** - Databricks Spark
Normalizer	- Blaze - Spark* - Databricks Spark
Parser	- Blaze - Spark* - Databricks Spark
Python	- Spark* - Databricks Spark
Rank	- Blaze - Spark* - Databricks Spark
Router	- Blaze - Spark* - Databricks Spark
Rule Specification	- Databricks Spark
Sequence Generator	- Blaze - Spark
Sorter	- Blaze - Spark* - Databricks Spark
Standardizer	- Blaze - Spark* - Databricks Spark
Union	- Blaze - Spark* - Databricks Spark
Update Strategy	- Blaze - Spark - Databricks Spark
Weighted Average	- Blaze - Spark** - Databricks Spark
Window	- Spark***

** Supported for both batch and streaming mappings.*

*** Supported with restrictions in batch mappings. Not supported in streaming mappings. For information about the Data Processor transformation support on the Spark engine, see the KB article.*

**** Supported for streaming mappings only. For more information, see the Data Engineering Streaming User Guide.*

Address Validator Transformation in a Non-native Environment

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

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

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

Address Validator Transformation on the Blaze Engine

The Address Validator transformation is supported with the following restrictions:

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

Address Validator Transformation on the Spark Engine

The Address Validator transformation is supported with the following restrictions:

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

Address Validator Transformation in a Streaming Mapping

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

Address Validator Transformation on the Databricks Spark Engine

The Address Validator transformation is supported with the following restrictions:

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

Aggregator Transformation in a Non-native Environment

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

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

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

Aggregator Transformation on the Blaze Engine

Some processing rules for the Blaze engine differ from the processing rules for the Data Integration Service.

Mapping Validation

Mapping validation fails in the following situations:

- The transformation contains stateful variable ports.
- The transformation contains unsupported functions in an expression.

Aggregate Functions

If you use a port in an expression in the Aggregator transformation but you do not use the port within an aggregate function, the run-time engine might use any row in the port to process the expression.

The row that the run-time engine uses might not be the last row in the port. Processing is distributed, and thus the run-time engine might not be able to determine the last row in the port.

Data Cache Optimization

The data cache for the Aggregator transformation is optimized to use variable length to store binary and string data types that pass through the Aggregator transformation. The optimization is enabled for record sizes up to 8 MB. If the record size is greater than 8 MB, variable length optimization is disabled.

When variable length is used to store data that passes through the Aggregator transformation in the data cache, the Aggregator transformation is optimized to use sorted input and a pass-through Sorter transformation is inserted before the Aggregator transformation in the run-time mapping.

To view the Sorter transformation, view the optimized mapping or view the execution plan in the Blaze validation environment.

During data cache optimization, the data cache and the index cache for the Aggregator transformation are set to Auto. The sorter cache for the Sorter transformation is set to the same size as the data cache for the Aggregator transformation. To configure the sorter cache, you must configure the size of the data cache for the Aggregator transformation.

Aggregator Transformation on the Spark Engine

Some processing rules for the Spark engine differ from the processing rules for the Data Integration Service.

Mapping Validation

Mapping validation fails in the following situations:

- The transformation contains stateful variable ports.
- The transformation contains unsupported functions in an expression.

Aggregate Functions

If you use a port in an expression in the Aggregator transformation but you do not use the port within an aggregate function, the run-time engine might use any row in the port to process the expression.

The row that the run-time engine uses might not be the last row in the port. Processing is distributed, and thus the run-time engine might not be able to determine the last row in the port.

Data Cache Optimization

You cannot optimize the data cache for the transformation to store data using variable length.

Aggregator Transformation in a Streaming Mapping

Streaming mappings have additional processing rules that do not apply to batch mappings.

Mapping Validation

Mapping validation fails in the following situations:

- A streaming pipeline contains more than one Aggregator transformation.
- A streaming pipeline contains an Aggregator transformation and a Rank transformation.
- An Aggregator transformation is upstream from a Lookup transformation.
- An Aggregator transformation is in the same streaming pipeline as a passive Lookup transformation configured with an inequality lookup condition.

Aggregator Transformation on the Databricks Spark Engine

Some processing rules for the Databricks Spark engine differ from the processing rules for the Data Integration Service.

Mapping Validation

Mapping validation fails in the following situations:

- The transformation contains stateful variable ports.
- The transformation contains unsupported functions in an expression.

Aggregate Functions

If you use a port in an expression in the Aggregator transformation but you do not use the port within an aggregate function, the run-time engine might use any row in the port to process the expression.

The row that the run-time engine uses might not be the last row in the port. Processing is distributed, and thus the run-time engine might not be able to determine the last row in the port.

Data Cache Optimization

You cannot optimize the data cache for the transformation to store data using variable length.

Case Converter Transformation in a Non-native Environment

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

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

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported without restrictions.

Classifier Transformation in a Non-native Environment

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

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

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported without restrictions.

Comparison Transformation in a Non-native Environment

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

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

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported without restrictions.

Consolidation Transformation in a Non-native Environment

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

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

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported with restrictions.

Consolidation Transformation on the Blaze Engine

The Consolidation transformation might process data differently in the native environment and in a non-native environment.

The transformation is supported with the following restrictions:

- The transformation might process records in a different order in each environment.
- The transformation might identify a different record as the survivor in each environment.

Consolidation Transformation on the Spark Engine

The Consolidation transformation might process data differently in the native environment and in a non-native environment.

The transformation is supported with the following restrictions:

- The transformation might process records in a different order in each environment.
- The transformation might identify a different record as the survivor in each environment.

Consolidation Transformation on the Databricks Spark Engine

The Consolidation transformation might process data differently in the native environment and in a non-native environment.

The transformation is supported with the following restrictions:

- The transformation might process records in a different order in each environment.
- The transformation might identify a different record as the survivor in each environment.

Data Masking Transformation in a Non-native Environment

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

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

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch and streaming mappings.
- Databricks Spark engine. Not supported.

Data Masking Transformation on the Blaze Engine

The Data Masking transformation is supported with the following restrictions.

Mapping validation fails in the following situations:

- The transformation is configured for repeatable expression masking.
- The transformation is configured for unique repeatable substitution masking.

You can use the following masking techniques on this engine:

Credit Card

Email

Expression

IP Address

Key

Phone

Random

SIN

SSN

Tokenization

URL

Random Substitution

Repeatable Substitution

Dependent with Random Substitution

Dependent with Repeatable Substitution

Data Masking Transformation on the Spark Engine

The Data Masking transformation is supported with the following restrictions.

Mapping validation fails in the following situations:

- The transformation is configured for repeatable expression masking.
- The transformation is configured for unique repeatable substitution masking.

You can use the following masking techniques on this engine:

Credit Card

Email

Expression

IP Address

Key

Phone

Random

SIN

SSN

Tokenization

URL

Random Substitution

Repeatable Substitution

Dependent with Random Substitution

Dependent with Repeatable Substitution

To optimize performance of the Data Masking transformation, configure the following Spark engine configuration properties in the Hadoop connection:

spark.executor.cores

Indicates the number of cores that each executor process uses to run tasklets on the Spark engine.

Set to: `spark.executor.cores=1`

spark.executor.instances

Indicates the number of instances that each executor process uses to run tasklets on the Spark engine.

Set to: `spark.executor.instances=1`

spark.executor.memory

Indicates the amount of memory that each executor process uses to run tasklets on the Spark engine.

Set to: `spark.executor.memory=3G`

Data Masking Transformation in a Streaming Mapping

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

Data Processor Transformation in a Non-native Environment

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

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

- Blaze engine. Supported without restrictions.

- Spark engine. Supported with restrictions in batch mappings. Not supported in streaming mappings.*
- Databricks Spark engine. Not supported.

** For information about the Data Processor transformation support on the Spark engine, see the KB article.*

Decision Transformation in a Non-native Environment

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

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

- Blaze engine. Supported without restrictions.
- Spark engine. Supported with restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported with restrictions.

Decision Transformation on the Spark Engine

The Decision transformation is supported with the following restrictions:

- The transformation properties must specify that the transformation is partitionable.

Decision Transformation on the Databricks Spark Engine

The Decision transformation is supported with the following restrictions:

- The transformation properties must specify that the transformation is partitionable.

Expression Transformation in a Non-native Environment

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

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

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

Expression Transformation on the Blaze Engine

Mapping validation fails in the following situations:

- The transformation contains stateful variable ports.
- The transformation contains unsupported functions in an expression.

An Expression transformation with a user-defined function returns a null value for rows that have an exception error in the function.

Expression Transformation on the Spark Engine

Mapping validation fails in the following situations:

- The transformation contains stateful variable ports.
- The transformation contains unsupported functions in an expression.

Note: If an expression results in numerical errors, such as division by zero or SQRT of a negative number, it returns a null value and rows do not appear in the output. In the native environment, the expression returns an infinite or an NaN value.

Expression Transformation in a Streaming Mapping

Streaming mappings have additional processing rules that do not apply to batch mappings.

You cannot use the EXTRACT_STRUCT function for streaming mappings.

Expression Transformation on the Databricks Spark Engine

Mapping validation fails in the following situations:

- The transformation contains stateful variable ports.
- The transformation contains unsupported functions in an expression.

Note: If an expression results in numerical errors, such as division by zero or SQRT of a negative number, it returns a null value and rows do not appear in the output. In the native environment, the expression returns an infinite or an NaN value.

Filter Transformation in a Non-native Environment

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

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

- Blaze engine. Supported with restrictions.
- Spark engine. Supported without restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported without restrictions.

Filter Transformation on the Blaze Engine

When a mapping contains a Filter transformation on a partitioned column of a Hive source, the Blaze engine can read only the partitions that contain data that satisfies the filter condition. To push the filter to the Hive source, configure the Filter transformation to be the next transformation in the mapping after the source.

Hierarchical to Relational Transformation in a Non-native Environment

The Hierarchical to Relational transformation processing in a non-native environment depends on the engine that runs the transformation.

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

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions.
- Databricks Spark engine. Not supported.

Java Transformation in a Non-native Environment

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

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

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch and streaming mappings.
- Databricks Spark engine. Not supported.

Java Transformation on the Blaze Engine

To use external .jar files in a Java transformation, perform the following steps:

1. Copy external .jar files to the Informatica installation directory in the Data Integration Service machine at the following location: `<Informatic installation directory>/services/shared/jars`. Then recycle the Data Integration Service.
2. On the machine that hosts the Developer tool where you develop and run the mapping that contains the Java transformation:
    a. Copy external .jar files to a directory on the local machine.
    b. Edit the Java transformation to include an import statement pointing to the local .jar files.
    c. Update the classpath in the Java transformation.
    d. Compile the transformation.

Some processing rules for the Blaze engine differ from the processing rules for the Data Integration Service.

Partitioning

- The following restrictions apply to the Transformation Scope property:
  - The value Transaction for transformation scope is not valid.
  - If you enable an input port for partition key, the transformation scope must be set to All Input.
  - Stateless must be enabled if the transformation scope is row.

Java Transformation on the Spark Engine

Some processing rules for the Spark engine differ from the processing rules for the Data Integration Service.

General Restrictions

The Java transformation is supported with the following restrictions on the Spark engine:

- The Java code in the transformation cannot write output to standard output when you push transformation logic to Hadoop. The Java code can write output to standard error which appears in the log files.
- For date/time values, the Spark engine supports the precision of up to microseconds. If a date/time value contains nanoseconds, the trailing digits are truncated.

Partitioning

The Java transformation has the following restrictions when used with partitioning:

- The Partitionable property must be enabled in the Java transformation. The transformation cannot run in one partition.
- The following restrictions apply to the Transformation Scope property:
  - The value Transaction for transformation scope is not valid.
  - If you enable an input port for partition key, the transformation scope must be set to All Input.
  - Stateless must be enabled if the transformation scope is row.

Mapping Validation

Mapping validation fails in the following situations:

- You reference an unconnected Lookup transformation from an expression within a Java transformation.
- You select a port of a complex data type as the partition or sort key.
- You enable nanosecond processing in date/time and the Java transformation contains a port of complex data type with an element of a date/time type. For example, a port of type `array<data/time>` is not valid if you enable nanosecond processing in date/time.

The mapping fails in the following situations:

- The Java transformation and the mapping use different precision modes when the Java transformation contains a decimal port or a complex port with an element of a decimal data type.

  Even if high precision is enabled in the mapping, the mapping processes data in low-precision mode in some situations, such as when the mapping contains a complex port with an element of a decimal data type, or the mapping is a streaming mapping. If high precision is enabled in both the Java transformation and the mapping, but the mapping processes data in low-precision mode, the mapping fails.
- Binary null characters are passed to an output port. To avoid a mapping failure, you can add code to the Java transformation that replaces the binary null characters with an alternative character before writing the data to the output ports.

Using External .jar Files

To use external .jar files in a Java transformation, perform the following steps:

1. Copy external .jar files to the Informatica installation directory in the Data Integration Service machine at the following location:
   `<Informatic installation directory>/services/shared/jars`
2. Recycle the Data Integration Service.
3. On the machine that hosts the Developer tool where you develop and run the mapping that contains the Java transformation:
   a. Copy external .jar files to a directory on the local machine.
   b. Edit the Java transformation to include an import statement pointing to the local .jar files.
   c. Update the classpath in the Java transformation.
   d. Compile the transformation.

Java Transformation in a Streaming Mapping

Streaming mappings have additional processing rules that do not apply to batch mappings.

The Data Integration Service ignores the following properties:

- Partitionable. The Data Integration Service ignores the property and can process the transformation with multiple threads.
- Stateless. The Data Integration Service ignores the property and maintains the row order of the input data.

Joiner Transformation in a Non-native Environment

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

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

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

Joiner Transformation on the Blaze Engine

Mapping validation fails in the following situations:

- The transformation contains an inequality join and map-side join is disabled.
- The Joiner transformation expression references an unconnected Lookup transformation.

Map-side join is disabled when the Joiner transformation is configured for detail outer join or full outer join.

Joiner Transformation on the Spark Engine

Mapping validation fails in the following situation:

- The join condition is of binary data type or contains binary expressions.

Joiner Transformation in a Streaming Mapping

Streaming mappings have additional processing rules that do not apply to batch mappings.

Mapping Validation

Mapping validation fails in the following situations:

- A Joiner transformation is downstream from an Aggregator transformation.
- A Joiner transformation is downstream from a Rank transformation.
- A streaming pipeline contains more than one Joiner transformation.
- A Joiner transformation joins data from streaming and non-streaming pipelines.

General Guidelines

To specify a join condition, select the TIME_RANGE function from the Advanced condition type on the Join tab and enter a join condition expression. The TIME_RANGE function determines the time range for the streaming events to be joined.

Joiner Transformation on the Databricks Spark Engine

Mapping validation fails in the following situation:

- The join condition is of binary data type or contains binary expressions.

Key Generator Transformation in a Non-native Environment

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

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

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported with restrictions.

Key Generator Transformation on the Blaze Engine

The Key Generator transformation is supported with the following restriction:

- When you run a mapping with a Key Generator transformation on more than one node, the sequence generation option on the transformation may not produce unique ID values.

Key Generator Transformation on the Spark Engine

The Key Generator transformation is supported with the following restriction:

- When you run a mapping with a Key Generator transformation on more than one node, the sequence generation option on the transformation may not produce unique ID values.

Key Generator Transformation on the Databricks Spark Engine

The Key Generator transformation is supported with the following restriction:

- When you run a mapping with a Key Generator transformation on more than one node, the sequence generation option on the transformation may not produce unique ID values.

Labeler Transformation in a Non-native Environment

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

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

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported without restrictions.

Lookup Transformation in a Non-native Environment

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

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

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

Lookup Transformation on the Blaze Engine

Some processing rules for the Blaze engine differ from the processing rules for the Data Integration Service.

Multiple Matches

The Lookup transformation finds values based on the condition you configure in the transformation. Choose how to handle multiple matches in the lookup source.

Return First Row; Return Last Row

The transformation returns the first matching result or the last matching result.

The Data Integration Service orders results to identify the first and last row. The following rules determine result order:

- Ordering depends on the lookup ports present in the lookup condition and lookup output ports.
- Numerical value sorting is by ascending order.
- String value sorting is lexographical.

- Date value sorting is by earliest date first.

Return Any Row

The transformation returns any of the rows that match the lookup condition. The transformation creates an index based on the key ports instead of all Lookup transformation ports. When you choose this option, performance can improve because the process of indexing rows is simpler.

Return All Rows

The Lookup transformation returns all rows that match the lookup condition.

Report Error

When the Lookup transformation uses a static cache or no cache, the Data Integration Service marks the row as an error. The Lookup transformation writes the row to the session log by default, and increases the error count by one.

When the Lookup transformation has a dynamic cache, the Data Integration Service fails the session when it encounters multiple matches. The session fails while the Data Integration Service is caching the lookup table or looking up the duplicate key values.

Also, if you configure the Lookup transformation to output old values on updates, the Lookup transformation returns an error when it encounters multiple matches. The transformation creates an index based on the key ports instead of all Lookup transformation ports.

Rules and Guidelines

Mapping validation fails in the following situations:

- Case sensitivity is disabled.
- The lookup condition contains binary data type.
- The lookup condition uses a field with a complex data type.
- The cache is configured to be shared, named, persistent, dynamic, or uncached. The cache must be a static cache.

If you add a data object that uses Sqoop as a Lookup transformation in a mapping, the Data Integration Service does not run the mapping through Sqoop. It runs the mapping through JDBC.

Lookup Transformation on the Spark Engine

Some processing rules for the Spark engine differ from the processing rules for the Data Integration Service.

Multiple Matches

The Lookup transformation finds values based on the condition you configure in the transformation. Choose how to handle multiple matches in the lookup source.

Return First Row; Return Last Row

The transformation returns the first matching result or the last matching result.

The Data Integration Service orders results to identify the first and last row. The following rules determine result order:

- Ordering depends on the lookup ports present in the lookup condition and lookup output ports.
- Numerical value sorting is by ascending order.
- String value sorting is lexographical.
- Date value sorting is by earliest date first.

Return Any Row

The transformation returns any of the rows that match the lookup condition. The transformation creates an index based on the key ports instead of all Lookup transformation ports. When you choose this option, performance can improve because the process of indexing rows is simpler.

Return All Rows

The Lookup transformation returns all rows that match the lookup condition.

Report Error

When the Lookup transformation uses a static cache or no cache, the Data Integration Service marks the row as an error. The Lookup transformation writes the row to the session log by default, and increases the error count by one.

When the Lookup transformation has a dynamic cache, the Data Integration Service fails the session when it encounters multiple matches. The session fails while the Data Integration Service is caching the lookup table or looking up the duplicate key values.

Also, if you configure the Lookup transformation to output old values on updates, the Lookup transformation returns an error when it encounters multiple matches. The transformation creates an index based on the key ports instead of all Lookup transformation ports.

Rules and Guidelines

Mapping validation fails in the following situations:

- Case sensitivity is disabled.
- The lookup condition contains binary data type.
- The lookup condition uses a field with a complex data type.
- The cache is configured to be shared, named, persistent, dynamic, or uncached. The cache must be a static cache.

The mapping fails in the following situation:

- The transformation is unconnected and used with a Joiner or Java transformation.

Note: If an HBase lookup does not result in a match, it generates a row with null values for all columns. You can add a Filter transformation after the Lookup transformation to filter out null rows.

Lookup Transformation in a Streaming Mapping

Streaming mappings have additional processing rules that do not apply to batch mappings.

Mapping Validation

Mapping validation fails in the following situations:

- The lookup is a data object.
- An Aggregator transformation is in the same streaming pipeline as a passive Lookup transformation configured with an inequality lookup condition.
- A Rank transformation is in the same streaming pipeline as a passive Lookup transformation configured with an inequality lookup condition.
- A pipeline contains more than one passive Lookup transformation configured with an inequality condition.

The mapping fails in the following situations:

- The transformation is unconnected.

General Guidelines

Consider the following general guidelines:

- Using a float data type to look up data can return unexpected results.
- Use a Lookup transformation to look up data in a flat file, HDFS, Hive, relational, JDBC V2, and HBase data.
- To avoid cross join of DataFrames, configure the Lookup transformation to ignore null values that match.

HBase Lookups

To use a Lookup transformation on uncached HBase tables, perform the following steps:

1. Create an HBase data object. When you add an HBase table as the resource for a HBase data object, include the ROW ID column.
2. Create a HBase read data operation and import it into the streaming mapping.
3. When you import the data operation to the mapping, select the **Lookup** option.
4. On the Lookup tab, configure the following options:
   - Lookup column. Specify an equality condition on ROW ID
   - Operator. Specify =
5. Verify that format for any date value in the HBase tables is of a valid Java date format. Specify this format in the **Date Time Format** property of the **Advanced Properties** tab of the data object read operation.

Note: If an HBase lookup does not result in a match, it generates a row with null values for all columns. You can add a Filter transformation after the Lookup transformation to filter out null rows.

Mapping validation fails in the following situations:

- The condition does not contain a ROW ID.
- The transformation contains an inequality condition.
- The transformation contains multiple conditions.
- An input column is of a date type.

JDBC V2 Lookups

You can add a JDBC V2 data object read operation as a lookup in a streaming mapping. You can run streaming mappings with JDBC V2 lookup in Azure Databricks service in Microsoft Azure cloud services.

Lookup Transformation on the Databricks Spark Engine

Some processing rules for the Databricks Spark engine differ from the processing rules for the Data Integration Service.

Multiple Matches

When you choose to return the first, last, or any value on multiple matches, the Lookup transformation returns any value.

If you configure the transformation to report an error on multiple matches, the Databricks Spark engine drops the duplicate rows and does not include the rows in the logs.

Note: If an HBase lookup does not result in a match, it generates a row with null values for all columns. You can add a Filter transformation after the Lookup transformation to filter out null rows.

Mapping Validation

Mapping validation fails in the following situations:

- Case sensitivity is disabled.
- The lookup condition contains binary data type.
- The lookup condition uses a field with a complex data type.
- The cache is configured to be shared, named, persistent, dynamic, or uncached. The cache must be a static cache.
- The lookup source is not Microsoft Azure SQL Data Warehouse.

Macro Transformation in a Non-native Environment

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

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

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions for batch and streaming.
- Databricks spark engine. Supported without restrictions.

Match Transformation in a Non-native Environment

The Match transformation processing in the non-native environment depends on the engine that runs the transformation.

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

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported with restrictions.

Match Transformation on the Blaze Engine

The Match transformation is supported with the following restrictions:

- A Match transformation generates cluster ID values differently in native and non-native environments. In a non-native environment, the transformation appends a group ID value to the cluster ID.
- Mapping validation fails when the Match transformation is configured to write identity index data to database tables.

Match Transformation on the Spark Engine

The Match transformation is supported with the following restrictions:

- A Match transformation generates cluster ID values differently in native and non-native environments. In a non-native environment, the transformation appends a group ID value to the cluster ID.
- Mapping validation fails when the Match transformation is configured to write identity index data to database tables.

Match Transformation on the Databricks Spark Engine

The Match transformation is supported with the following restrictions:

- A Match transformation generates cluster ID values differently in native and non-native environments. In a non-native environment, the transformation appends a group ID value to the cluster ID.
- Mapping validation fails when the Match transformation is configured to write identity index data to database tables.

Merge Transformation in a Non-native Environment

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

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

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported without restrictions.

Normalizer Transformation in a Non-native Environment

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

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

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported without restrictions.

Parser Transformation in a Non-native Environment

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

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

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported without restrictions.

Rank Transformation in a Non-native Environment

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

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

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

Rank Transformation on the Blaze Engine

Some processing rules for the Blaze engine differ from the processing rules for the Data Integration Service.

The data cache for the Rank transformation is optimized to use variable length to store binary and string data types that pass through the Rank transformation. The optimization is enabled for record sizes up to 8 MB. If the record size is greater than 8 MB, variable length optimization is disabled.

When variable length is used to store data that passes through the Rank transformation in the data cache, the Rank transformation is optimized to use sorted input and a pass-through Sorter transformation is inserted before the Rank transformation in the run-time mapping. To view the Sorter transformation, view the optimized mapping or view the execution plan in the Blaze validation environment.

During data cache optimization, the data cache and the index cache for the Rank transformation are set to Auto. The sorter cache for the Sorter transformation is set to the same size as the data cache for the Rank transformation. To configure the sorter cache, you must configure the size of the data cache for the Rank transformation.

Rank Transformation on the Spark Engine

Some processing rules for the Spark engine differ from the processing rules for the Data Integration Service.

Mapping Validation

Mapping validation fails in the following situations:

- Case sensitivity is disabled.

Data Cache Optimization

You cannot optimize the data cache for the transformation to store data using variable length.

Rank Transformation in a Streaming Mapping

Streaming mappings have additional processing rules that do not apply to batch mappings.

Mapping Validation

Mapping validation fails in the following situations:

- A Rank transformation is in the same streaming pipeline as a passive Lookup transformation configured with an inequality lookup condition.
- A Rank transformation is upstream from a Joiner transformation.
- A streaming pipeline contains more than one Rank transformation.
- A streaming pipeline contains an Aggregator transformation and a Rank transformation.

Rank Transformation on the Databricks Spark Engine

Some processing rules for the Databricks Spark engine differ from the processing rules for the Data Integration Service.

Mapping Validation

Mapping validation fails in the following situations:

- Case sensitivity is disabled.

Data Cache Optimization

You cannot optimize the data cache for the transformation to store data using variable length.

Relational to Hierarchical Transformation in a Non-native Environment

The Relational to Hierarchical transformation processing in a non-native environment depends on the engine that runs the transformation.

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

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions.
- Databricks Spark engine. Not supported.

Router Transformation in a Non-native Environment

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

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

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported without restrictions.

Rule Specification Transformation in a Non-native Environment

A rule specification is a repository object that a user creates in the Analyst tool. You can add a rule specification to a mapping in the same manner as you add a transformation to a mapping. Rule specification processing in a non-native environment depends on the engine that runs the object.

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

- Blaze engine. Not supported.
- Spark engine. Not supported.
- Databricks Spark engine. Supported without restrictions.

Sequence Generator Transformation in a Non-native Environment

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

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

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported when the domain and the Databricks cluster reside in the same virtual network in AWS or Azure Databricks environments.

Sequence Generator Transformation on the Blaze Engine

A mapping with a Sequence Generator transformation consumes significant resources when the following conditions are true:

- You configure the transformation to maintain row order.
- The mapping runs in a single partition.

Sequence Generator Transformation on the Spark Engine

The Sequence Generator transformation does not maintain row order in output data. If you enable the **Maintain Row Order** property on the transformation, the Data Integration Service ignores the property.

Note: Spark sequence generator with maintain row order works fine if the data set is relatively small and can run or forced to run on a single partition.

Sequence Generator Transformation on the Databricks Spark Engine

The Databricks Spark engine can process the Sequence Generator transformation on Amazon Web Services or Azure Databricks clusters.

You can include a Sequence Generator transformation in a mapping that runs on Databricks Spark when the Databricks cluster and the Informatica domain reside in the same virtual network.

Sorter Transformation in a Non-native Environment

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

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

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

Sorter Transformation on the Blaze Engine

Some processing rules for the Blaze engine differ from the processing rules for the Data Integration Service.

Mapping Validation

Mapping validation fails in the following situations:

- The target is configured to maintain row order and the Sorter transformation is not connected directly to a flat file target.

Parallel Sorting

The Data Integration Service enables parallel sorting with the following restrictions:

- The mapping does not include another transformation between the Sorter transformation and the target.
- The data type of the sort keys does not change between the Sorter transformation and the target.
- Each sort key in the Sorter transformation must be linked to a column in the target.

Global Sort

The Blaze engine can perform global sorts in the following situations:

- The Sorter transformation is connected directly to flat file targets.
- The target is configured to maintain row order.

- The sort key is not a binary data type.

If any of the conditions are not true, the Blaze engine performs a local sort.

Data Cache Optimization

If a Sorter transformation is inserted before an Aggregator or Rank transformation to optimize the Aggregator or Rank data cache, the size of the sorter cache is the same size as the data cache for the Aggregator or Rank transformation. To configure the sorter cache, you must configure the size of the data cache for the Aggregator or Rank transformation.

Sorter Transformation on the Spark Engine

Some processing rules for the Spark engine differ from the processing rules for the Data Integration Service.

Mapping Validation

Mapping validation fails when case sensitivity is disabled.

The Data Integration Service logs a warning and ignores the Sorter transformation in the following situations:

- There is a type mismatch in between the target and the Sorter transformation sort keys.
- The transformation contains sort keys that are not connected to the target.
- The Write transformation is not configured to maintain row order.
- The transformation is not directly upstream from the Write transformation.

Null Values

The Data Integration Service treats null values as low even if you configure the transformation to treat null values as high.

Data Cache Optimization

You cannot optimize the sorter cache to store data using variable length.

Sorter Transformation in a Streaming Mapping

Streaming mappings have additional processing rules that do not apply to batch mappings.

Mapping Validation

Mapping validation fails in the following situation:

- The Sorter transformation in a streaming mapping does not have an upstream Aggregator transformation.

General Guidelines

Consider the following general guidelines:

- The mapping runs in complete output mode if it contains a Sorter transformation.
- To maintain global sort order, ensure that the target is single-partitioned. Source can be single- or multi-partitioned.

Sorter Transformation on the Databricks Spark Engine

Some processing rules for the Databricks Spark engine differ from the processing rules for the Data Integration Service.

Mapping Validation

Mapping validation fails when case sensitivity is disabled.

The Data Integration Service logs a warning and ignores the Sorter transformation in the following situations:

- There is a type mismatch in between the target and the Sorter transformation sort keys.
- The transformation contains sort keys that are not connected to the target.
- The Write transformation is not configured to maintain row order.
- The transformation is not directly upstream from the Write transformation.

Null Values

The Data Integration Service treats null values as low even if you configure the transformation to treat null values as high.

Data Cache Optimization

You cannot optimize the sorter cache to store data using variable length.

Parallel Sorting

The Data Integration Service enables parallel sorting with the following restrictions:

- The mapping does not include another transformation between the Sorter transformation and the target.
- The data type of the sort keys does not change between the Sorter transformation and the target.
- Each sort key in the Sorter transformation must be linked to a column in the target.

Standardizer Transformation in a Non-native Environment

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

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

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch and streaming mappings.
- Databricks Spark engine. Supported without restrictions.

Union Transformation in a Non-native Environment

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

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

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch mappings. Supported with restrictions in streaming mappings.
- Databricks Spark engine. Supported without restrictions.

Union Transformation in a Streaming Mapping

Streaming mappings have mapping validation restrictions.

Mapping Validation

Mapping validation fails in the following situation:

- The transformation is configured to merge data from streaming and non-streaming pipelines.

Update Strategy Transformation in a Non-native Environment

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

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

- Blaze engine. Supported with restrictions.
- Spark engine. Supported with restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported with restrictions.

Update Strategy Transformation on the Blaze Engine

Some processing rules for the Blaze engine differ from the processing rules for the Data Integration Service.

General Restrictions

You can use the Update Strategy transformation on the Hadoop distributions that support Hive ACID.

If the Update Strategy transformation receives multiple update rows for the same primary key value, the transformation selects one random row to update the target.

If multiple Update Strategy transformations write to different instances of the same target, the target data might be unpredictable.

The Blaze engine executes operations in the following order: deletes, updates, inserts. It does not process rows in the same order as the Update Strategy transformation receives them.
Hive targets always perform Update as Update operations. Hive targets do not support Update Else Insert or Update as Insert.

Mapping Validation and Compile Validation

Mapping validation fails in the following situations:

- The Update Strategy transformation is connected to more than one target.
- The Update Strategy transformation is not located immediately before the target.
- The Update Strategy target is not a Hive target.
- The Update Strategy transformation target is an external ACID table.
- The target does not contain a primary key.
- The Hive target property to truncate the target table at run time is enabled.
- The Hive target property to create or replace the target table at run time is enabled.

The mapping fails in the following situation:

- The target is not ORC bucketed.
- The Hive target is modified to have fewer rows than the actual table.

Compile validation errors occur and the mapping execution stops in the following situations:

- The target table is not enabled for transactions.
- The Hive version is earlier than 0.14.

Using Hive Target Tables

To use a Hive target table with an Update Strategy transformation, you must create the Hive target table with the following clause in the Hive Data Definition Language: `TBLPROPERTIES ("transactional"="true").`

To use an Update Strategy transformation with a Hive target, verify that the following properties are configured in the hive-site.xml configuration set associated with the Hadoop connection:

```
hive.support.concurrency         true
hive.enforce.bucketing           true
hive.exec.dynamic.partition.mode nonstrict
hive.txn.manager         org.apache.hadoop.hive.ql.lockmgr.DbTxnManager
hive.compactor.initiator.on      true
hive.compactor.worker.threads    1
```

Update Strategy Transformation on the Spark Engine

Some processing rules for the Spark engine differ from the processing rules for the Data Integration Service.

General Restrictions for Hive Targets

You can use the Update Strategy transformation on the Hadoop distributions that support Hive ACID.

You can also use the Update Strategy transformation to write mapping results to JDBC-compliant relational targets.

The Update Strategy transformation does not forward rejected rows to the next transformation when the target is a Hive table or JDBC-compliant table.

If the Update Strategy transformation receives multiple update rows for the same primary key value, the transformation selects one random row to update the target.

If multiple Update Strategy transformations write to different instances of the same target, the target data might be unpredictable.

If the mapping runs on the Spark engine, you can choose the Use Hive Merge option. The option has the following restrictions:

- A single row for delete or update cannot match multiple rows in the target. When the mapping violates this restriction, the mapping fails with a runtime error.
- If you configure the Update Strategy expression to update partitioning or bucketing columns, the mapping ignores the Hive MERGE option and does not update the columns.

Note: The Developer tool and the Data Integration Service do not validate against these restrictions. If the expression or the mapping violates these restrictions, the mapping might run, but the results will not be as expected.

Hive targets always perform Update as Update operations. Hive targets do not support Update Else Insert or Update as Insert.

Using Hierarchical Data in Update Strategy Expressions

You can configure the expression to use the primitive types in a complex data field.

For example, you have a nested array of struct data types defined by the following statement:

```
addresses array<struct<city:varchar(15), apartment:smallint, pincode:int>>
```

You can use the city column in the update strategy expression since it is a primitive type using the following expression:

```
IIF( (addresses[0].city = 'NEW YORK'), DD_INSERT, DD_UPDATE)
```

Mapping Validation

Update Strategy transformation output fields must exactly match the target input fields.

Mapping validation fails in the following situations.

- The Update Strategy transformation is connected to more than one target.
- The Update Strategy transformation is not located immediately before the target.
- The Update Strategy transformation target is an external ACID table.
- The target does not contain a connected primary key.
- The property to enable truncation of the target table at run time is selected.
- One of the following target strategies for the target table at run time is selected:
  - Create or replace the target table
  - ApplyNewColumns
  - ApplyNewSchema
  - Fail

The mapping fails in the following situations when the target is a Hive target:

- The target table is not enabled for transactions.
- The target is not ORC bucketed.

Using Hive Target Tables

To use a Hive target table with an Update Strategy transformation, you must create the Hive target table with the following clause in the Hive Data Definition Language: `TBLPROPERTIES ("transactional"="true").`

To use an Update Strategy transformation with a Hive target, verify that the following properties are configured in the hive-site.xml configuration set associated with the Hadoop connection:

```
hive.support.concurrency         true
hive.enforce.bucketing           true
hive.exec.dynamic.partition.mode nonstrict
hive.txn.manager         org.apache.hadoop.hive.ql.lockmgr.DbTxnManager
hive.compactor.initiator.on      true
hive.compactor.worker.threads    1
```

Update Strategy Transformation on the Databricks Spark Engine

Some processing rules for the Databricks Spark engine differ from the processing rules for the Data Integration Service.

General Restrictions

You can only use the Update Strategy transformation on the Databricks Spark Engine when you use a Delta Lake table target.

The Update Strategy transformation is supported only on Databricks distributions that support Delta ACID.

Mapping Validation

Mapping validation fails in the following situations.

- The Update Strategy transformation is connected to more than one target.
- The Update Strategy transformation is not located immediately before the target.
- The Update Strategy transformation target is an external ACID table.
- The target does not contain a connected primary key.
- The property to enable truncation of the target table at run time is selected.
- One of the following target strategies for the target table at run time is selected:
  - Create or replace the target table
  - ApplyNewColumns
  - ApplyNewSchema
  - Fail

Weighted Average Transformation in a Non-native Environment

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

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

- Blaze engine. Supported without restrictions.
- Spark engine. Supported without restrictions in batch mappings. Not supported in streaming mappings.
- Databricks Spark engine. Supported without restrictions.

CHAPTER 7

Python Transformation

This chapter includes the following topics:

- Python Transformation Overview, 149
- Python Transformation Ports, 151
- Python Transformation Advanced Properties, 151
- Python Transformation Components, 152
- Rules and Guidelines for the Python Transformation, 154
- Creating a Python Transformation, 154
- Example: Add an ID Column to Nonpartitioned Data, 156
- Example: Use Partitions to Find the Highest Salary, 157
- Use Case: Operationalize a Pre-Trained Model, 159

Python Transformation Overview

Use the Python transformation to execute Python code in a mapping that runs on the Spark or Databricks Spark engine.

The Python transformation provides an interface to define transformation functionality using Python code.

Python is a language that uses simple syntax, dynamic typing, and dynamic binding, making Python an ideal choice to increase productivity or to participate in rapid application development. When you use your Python code in a data engineering mapping, the Python code is embedded into the generated Scala code that the Spark or Databricks Spark engine runs to process large, diverse, and fast-changing data sets.

You can also use the Python transformation for machine learning. In the transformation, you can specify a resource file that contains a pre-trained model and load the pre-trained model in the Python code. For example, you can load a pre-trained model to classify input data or to create predictions.

Before you can use the Python transformation, configure the corresponding Spark advanced properties in the Hadoop connection or Databricks connection properties. Then, ensure that the worker nodes on the cluster contain an installation of Python.

For more information about installing Python, see the *Data Engineering Integration Guide*.

Note: The Data Integration Service and the Blaze engine do not support the Python transformation.

Active and Passive Python Transformations

A Python transformation generates output rows differently based on whether the transformation is active or passive.

After you create the transformation, you cannot change whether the transformation is active or passive.

Active Python Transformation

An active transformation can change the number of rows that pass through it.

To define the number of rows in the output, call the generateRow() method in the code to generate each output row. You might choose to generate multiple output rows from a single input row or generate a single output row from multiple input rows. For example, if the transformation contains two input ports that represent a start date and an end date, you can call the generateRow() method to generate an output row for each date between the start date and the end date.

Passive Python Transformation

A passive transformation cannot change the number of rows that pass through the transformation. The transformation calls the generateRow() method to generate an output row after processing each input row.

Data Type Conversion

A Python transformation converts Developer tool data types to Python data types, based on the Python transformation port type.

When a Python transformation reads input rows, it converts input port data types to Python data types. When a Python transformation writes output rows, it converts Python data types to output port data types.

For example, the following processing occurs for an input port with the double data type in a Python transformation:

- The Python transformation converts the double data type in the input port to the Python float data type.
- The transformation uses the value in the input port as the value for the Python float data type.
- To generate the output row, the Python transformation converts the Python float data type to the double data type.

The following table shows how the Python transformation maps Developer tool data types to Python data types:

Developer Tool Data Type	Python Data Type
Integer	Int
Decimal	Float
Double	Float
Timestamp	Datetime
String	Str
The Python transformation does not support data types that are not listed in this table.	

Deferment Notice: Support for binary ports in the Python transformation is deferred. Support will be reinstated in a future release.

When you write code in the Python transformation, the data types of the output ports in the Python transformation must be compatible with the data types in the Python code. Thus, if you configure an output port in the Python transformation to be a double data type, the corresponding variable in the Python code must be a float.

Data Types in Input and Output Ports

The data types in corresponding input and output ports in the Python transformation must be the same. If the data types are not the same, convert the data type in the Python code.

For example, you create an input port with the integer data type and an output port with the string data type. You define the Python code to process the data in the input port and write the data to the output port. In the Python code, you can use the Python function `str()` to convert the integer data type in the input port and write the output as a string data type in the output port.

Python Transformation Ports

A Python transformation can have input and output ports.

To create and edit ports for a non-reusable Python transformation, use the **Ports** tab in the editor. To create and edit ports for a reusable Python transformation, use the **Overview** view in the editor. After you add ports to the transformation, you can use the port names as variables in the Python code.

Use the following guidelines to create input and output ports:

- The port name can contain only ASCII characters.
- The port name cannot be a Python keyword. For example, do not use port names such as `import, global` or `class`.
- The port name cannot be `resourceJepFile` or `resourceFilesArray`.
- You cannot configure user-defined default values in output ports in the Python transformation. Set user-defined default values in the Python code.

  For example, you can write `output_port = 'value'` to set the default value `'value'` for the output port `output_port`.

You can enable one or more input rows as partition keys. The Data Integration Service uses the partition keys to repartition the data before the code runs. If you do not select an input row as a partition key, the data is processed using its default partitioning scheme.

Python Transformation Advanced Properties

The Python transformation includes advanced properties for the transformation.

You can define the following advanced property for the Python transformation on the Advanced tab:

Tracing Level

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

Is Active

Indicates whether the transformation is an active transformation. An active transformation can change the number of rows that pass through it.

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

Python Transformation Components

The Python transformation uses resource files and the Python code to define transformation functionality.

The Python transformation contains the following components:

Resource File

A file that contains the resources that you access in the Python code.

The file can be a pre-trained model that has been trained on a larger data set outside the Developer tool. You can use the pre-trained model to classify data or make predictions based on the data that you pass to the Python transformation. You can access the pre-trained model in the Python code.

Python Code

The Python code that the Python transformation uses to process data that you pass to the transformation. When you write Python code, you might reconstruct input variables, load a pre-trained model, and define output variables.

Resource File

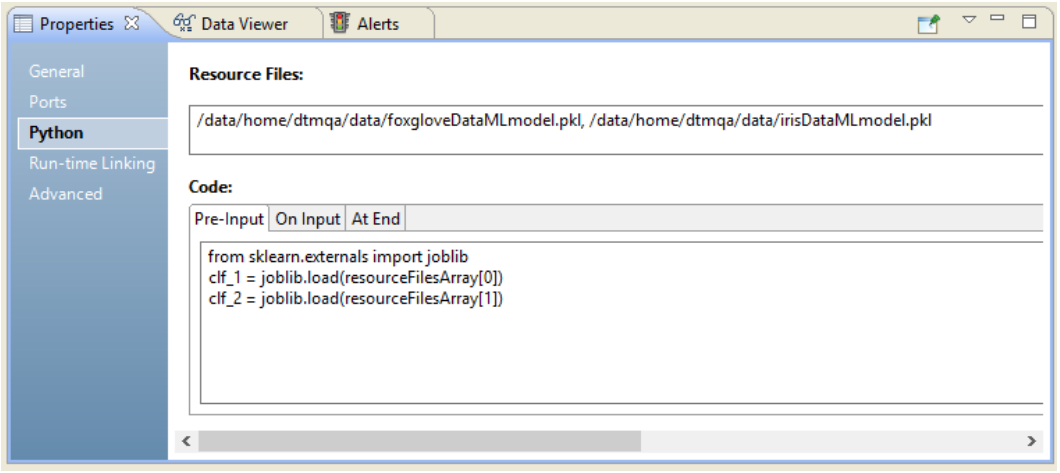
A resource file is a file that contains the resources that you use in the Python code. If you use a pre-trained model, you specify the pre-trained model as a resource file in the Python transformation.

In the Python transformation, list the resource file path on the Data Integration Service machine. Separate multiple resource file paths using a comma.

When you access a resource file in the Python code, you reference the array `resourceFilesArray`. To access a specific resource file, you specify an index to locate the resource file according to the order that the resource file appears in the list of resource file paths.

For example, you specify several resource files. To reference the first resource file in the Python code, use `resourceFilesArray[0]`.

The following image shows how you can specify a resource file in the Python transformation and access the resource file in the Python code:



The resource files `foxgloveDataMLmodel.pkl` and `irisDataMLmodel.pkl` are listed using the resource file paths on the Data Integration Service machine. The Python code accesses the resource files to reconstruct the respective Python objects. To access the first resource file, the Python code references the resource file using `resourceFilesArray[0]`. To access the second resource file `irisDataMLmodel.pkl`, the Python code references the resource file using `resourceFilesArray[1]`.

Python Code

The Python code defines how the Python transformation processes data. When you write Python code, you might reconstruct input variables, load a pre-trained model, define output variables, and define additional transformation functionality.

To write the Python code, use the following tabs:

Pre-Input

On the Pre-Input tab, define code that can be interpreted once and shared between all rows of data.

Use the Pre-Input tab to perform the following tasks:

- Declare import statements.
- Declare variables.
- Initialize variables.
- Define helper methods.

On Input

On the On Input tab, define how the Python transformation behaves when it receives an input row while processing a partition. The Python transformation processes the code on the On Input tab for each partition and each row in the partition.

At End

On the At End tab, define how the Python transformation behaves after it processes all input data in a partition. You can also call the generateRow() method to generate output rows for the partition.

Guidelines for Writing Python Code

Use the following rules to write Python code:

- Define variables before you use them. For example, you cannot reference a variable on the Pre-Input tab if the variable is defined on the On Input tab.
- To access input ports, call the input port name.
- To set output ports, set the output port to a value. You must set the output port to a value for each output port that you define in the Python transformation.
- To define how the transformation writes data from the input ports to output ports, set the output port to the value of the input port.

  For example, write `output_port = input_port` to write the data from the input port `input_port` to the output port `output_port`.
- To access resource files, use the variable `resourceFilesArray`. Specify the resource file using an index such as `resourceFilesArray[0]`.
- When you run the Python transformation, the Data Integration Service does not validate the Python code.

Rules and Guidelines for the Python Transformation

Consider the following rules and guidelines when you use a Python transformation:

- You cannot pass complex ports to the Python transformation.
- Mapping validation fails if a user-defined default value is assigned to an output port.
- The mapping fails in the following situations:
  - An output port is not assigned a value in the Python code.
  - The data types in corresponding input and output ports are not the same, and the Python code does not convert the data type in the input port to the data type in the output port.
  - The Python transformation contains decimal ports and high precision is enabled in the mapping.

Python Transformation in a Streaming Mapping

Streaming mappings have additional processing rules that do not apply to batch mappings.

Consider the following restrictions:

- When you close a Jep instance, you might not be able to call CPython modules.
- Do not use the names `inRowsList` and `outRowsList` in the Python code. These names are reserved for the Scala code that the Spark engine uses to interact with the Python environment.

Creating a Python Transformation

In the Developer tool, you can create a reusable or non-reusable Python transformation.

Creating a Reusable Python Transformation

You can create a reusable Python transformation to use the transformation in multiple mappings. Create a reusable Python transformation in the Developer tool.

1. In the **Object Explorer** view, select a project or a folder.
2. Click **File** > **New** > **Transformation**.

   The **New** dialog box appears.
3. Select the Python transformation.
4. Click **Next**.
5. Enter a name for the transformation.
6. Optionally, select **Create as Active**.
7. Click **Finish**.

   The transformation appears in the editor.
8. In the **Overview** view, click the **New** button to add a port to the transformation.
9. Edit the port to set the name, data type, and precision.

   Use port names as variables in the Python code.

   Array, map, and struct data types are not available for a reusable transformation.
10. In the **Python** view, specify the resource files and write the Python code for the transformation.
11. In the **Advanced** view, edit the advanced properties for the transformation.

Creating a Non-Reusable Python Transformation

You can create a non-reusable Python transformation to use the transformation in a single mapping. Create a non-reusable Python transformation in the Developer tool.

1. In a mapping or a mapplet, drag a Python transformation from the mapping palette to the editor.

   The transformation appears in the editor.
2. In the **General** tab, edit the transformation name and the description.
3. In the **Ports** tab, click the **New** button to add a port to the transformation.
4. Edit the port to set the name, data type, and precision.

   Use port names as variables in the Python code.
5. In the **Python** tab, specify the resource files and compile the Python code for the transformation.
6. In the **Run-time Linking** tab, create run-time links between the Python transformation and an upstream or downstream transformation in the mapping if the ports change at run time.
7. In the **Advanced** tab, edit the advanced properties for the transformation.

Example: Add an ID Column to Nonpartitioned Data

Your organization runs a solar thermal power system that uses sensors to monitor the system's health. Currently, each sensor is identified by its location. Instead, you want to identify each sensor using an ID to simplify future analytics on the data.

You collect the following data on sensor readings:

SensorLocation	LastReadingTime
Area A	7/9/2019 11:36:09
Area B	7/9/2019 16:43:42
Area C	7/9/2019 13:23:53

To add an ID column and assign ID values to each sensor, complete the following steps:

Step 1. Create a Python transformation.

Create a passive Python transformation.

Step 2. Pass data to the Python transformation.

Pass data from upstream transformations in the mapping to the Python transformation.

After you pass the data to the Python transformation, the Python transformation contains the following input ports:

- SensorLocation
- LastReadingTime

Step 3. Create output ports.

Use the Ports tab in the Python transformation to create the output port `SensorID_out` to represent the ID column.

Additionally, create the following output ports to pass incoming data to downstream transformations:

- SensorLocation_out
- LastReadingTime_out

Step 4. Set the ID value for each row.

On the On Input tab, set the ID value for each row that is processed and write the data to the output ports using the following code:

```
SensorID="".join(str(x) for x in map(ord, SensorLocation))

SensorID_out = SensorID
SensorLocation_out = SensorLocation
LastReadingTime_out = LastReadingTime
```

Step 5. Run the mapping.

If the output ports in the Python transformation are linked directly to a Write transformation, the target contains the following data after you run the mapping:

SensorID_out	SensorLocation_out	LastReadingTime_out
65114101973265	Area A	7/9/2019 11:30:00
65114101973266	Area B	7/9/2019 11:35:00
65114101973267	Area C	7/9/2019 11:40:00

Example: Use Partitions to Find the Highest Salary

You are an HR staff member at your organization. You are working on a project to model how employee salaries are associated with aspects of life that employees find important. You want to use the information to better personalize the organization's wellness program.

You can use the Python transformation to complete the first part of your project to determine which employee earns the highest salary in their department.

The following table shows the data that your organization might collect:

DepartmentName	DepartmentID	EmployeeName	SalaryIndex	EmployeeSince
HR	1	Jane Smith	500	2/16/2010
R&D	2	Ellioth Consar	150	3/29/2018
Finance	3	Concor Valashe	230	11/22/2007
Marketing	4	Manchini Voliore	800	5/17/2009
HR	1	Blaze Concave	501	8/25/2016
R&D	2	Janet Encarr	890	1/26/2019
HR	1	Chelsea Blanch	389	9/3/2018
R&D	1	Samuel Coin	10	1/26/2005

To use the Python transformation for the task, complete the following steps:

Step 1. Add a Python transformation to the mapping.

Create an active Python transformation.

Step 2. Pass data to the Python transformation.

Pass the following ports from upstream transformations in the mapping to the Python transformation:

- DepartmentName
- DepartmentID
- EmployeeName

- SalaryIndex
- EmployeeSince

Step 3. Partition the data by department.

Partition the data by department to track the highest salary within each department. To partition the data by department, select the input port `DepartmentID` as the partition key.

Step 4. Create output ports.

On the Ports tab in the Python transformation, create the following output ports to pass data to downstream transformations:

- DepartmentName_out
- DepartmentID_out
- EmployeeName_out
- SalaryIndex_out
- EmployeeSince_out

Step 5. Initialize a map.

Declare a map variable `outputmap` to associate each department ID with the employee in the department who has the highest salary.

Add the following code on the Pre-Input tab:

```
print("Using partitions to find the employee with the highest salary")
outputmap = {}
```

Step 6. Define code to process the data.

For each input row that passes through the Python transformation, define code that checks if the employee's salary is higher than the maximum salary of the previous rows that have been processed. If the employee's salary is higher, update the employee who has the maximum salary in the department.

Add the following code on the On Input tab:

```
DepartmentID_out = DepartmentID
print("Processing rows for department ID " + str(DepartmentID_out))

outputmap.setdefault(DepartmentID, None)
updateMax = False

if outputmap.get(DepartmentID, None) is None:
    updateMax = True
else:
    max_salary = outputmap[DepartmentID]['SalaryIndex']
    if max_salary is None:
        updateMax = True
    if SalaryIndex > max_salary:
        updateMax = True

if updateMax == True:
    employee_data = {'SalaryIndex':SalaryIndex,'EmployeeName':EmployeeName,
                     'EmployeeSince':EmployeeSince,'DepartmentName':DepartmentName}

    outputmap[DepartmentID] = employee_data
```

Step 7. Write the data to the output ports.

On the At End tab, use the data in the map variable `outputmap` to generate a row for the employee that has the highest salary in each department.

Add the following code on the At End tab:

```
for x in outputmap:
    DepartmentID_out = x
    smap = outputmap[x]
    SalaryIndex_out = smap["SalaryIndex"]
    EmployeeName_out = smap["EmployeeName"]
    DepartmentName_out = smap["DepartmentName"]
    EmployeeSince_out = smap["EmployeeSince"]

    ## Generate the output row
    generateRow()
```

Step 8. Run the mapping.

If the output ports in the Python transformation are linked directly to a Write transformation, the target contains the following data after you run the mapping:

DepartmentName	DepartmentID	EmployeeName	SalaryIndex	EmployeeSince
Finance	3	Concor Valashe	230	11/22/2007
Marketing	4	Manchini Voliore	800	5/17/2009
HR	1	Blaze Concave	501	8/25/2016
R&D	2	Janet Encarr	890	1/26/2019

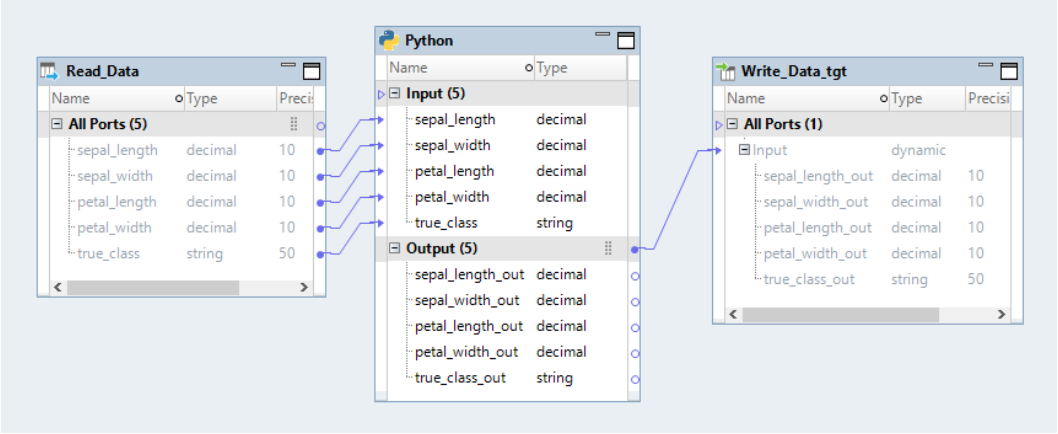
Use Case: Operationalize a Pre-Trained Model

You work for a pharmaceutical company and you are studying data on flower formation in foxglove in your research to provide a better treatment for heart diseases. You want to find out whether the common foxglove *Digitalis purpurea* or the woolly foxglove *Digitalis lanata* can provide a better prognosis for the development of a disease.

To perform your research, you must classify data on the length and width of the flower sepals and petals by flower species. To classify the data, you developed a pre-trained model outside of the Developer tool.

You operationalize the pre-trained model in the Developer tool. In the Developer tool, you create a mapping that contains a passive Python transformation. In the Python transformation, you list the pre-trained model as a resource file. You write a Python script that accesses the pre-trained model. You pass the data on flower sepals and petals to the Python transformation to classify the data by foxglove species.

The following image shows the mapping that you might create:



The passive Python transformation uses the following components:

Resource File

Specify the path of the pre-trained model as the resource file.

For example, you might use a pre-trained model that is stored in the file `foxgloveDataMLmodel.pkl` in the following path:

```
/data/home/dtmqa/data/foxgloveDataMLmodel.pkl
```

Python Code

Specify the Python code on the Pre-Input and On Input tabs.

Use the Pre-Input tab to import libraries, load the resource file, and initialize variables.

For example, you might enter the following code on the Pre-Input tab:

```
from sklearn import svm
from sklearn.externals import joblib
import numpy as np
clf = joblib.load(resourceFileArrays[0])
classes = ['common', 'woolly']
```

On the On Input tab, define how the Python transformation uses the pre-trained model to evaluate each row of data.

For example, you might enter the following code on the On Input tab:

```
input = [sepal_length, sepal_width, petal_length, petal_width]
input = np.array(input).reshape(1,-1)
pred = clf.predict(input)
predicted_class = classes[pred[0]]
sepal_length_out = sepal_length
sepal_width_out = sepal_width
petal_length_out = petal_length
petal_width_out = petal_width
true_class_out = true_class
```

The Python transformation processes the data in the input ports according to the Python code and writes the data to the output ports.

CHAPTER 8

Data Preview

This chapter includes the following topics:

- Overview of Data Preview, 161
- Data Preview Process, 162
- Previewing Data, 163
- Data Preview Interface for Hierarchical Data, 164
- Data Preview on Transformations, 166
- Data Preview Logs, 167
- Rules and Guidelines for Data Preview on the Spark Engine, 168

Overview of Data Preview

You can preview data within a mapping in the Developer tool. You can choose sources and transformations in a mapping as preview points. Previewing data helps to design and debug mappings.

Data preview jobs run on either the Data Integration Service or the Spark engine. The Spark engine runs the job in the following cases:

- The preview point or any upstream transformation contains hierarchical data.
- The preview point or any upstream transformation is a Python transformation.
- The preview point or any upstream transformation is an Expression transformation configured for windowing.
- The mapping contains a combination of transformations that must run on the Spark engine.

When you view the results of data preview on hierarchical data, you can view the complex data types, the schema, and values. You can also navigate through the structure with breadcrumbs. You can export the results to a JSON file.

Before you preview data, verify that Spark is configured as the mapping run-time environment. If you are previewing hierarchical data in a complex file object, add the complex file object to a mapping to preview the data. You cannot preview data in mappings configured for the Databricks Spark engine.

Monitor ad hoc data preview jobs in the Monitor tab of the Administrator tool. From the Monitor tab, you can view job status, view job logs, and cancel jobs.

Connections and Cluster Distributions that Support Data Preview

When you preview data using the Spark engine, configure the mapping with a supported connection and Hadoop distribution.

Connections

You can use the Spark engine to preview data on mappings that use the following connections:

- HBase
- HDFS
- Hive
- JDBC configured for Sqoop

When you configure a mapping that uses a JDBC connection, you can use a generic JDBC connection, or use the specialized drivers for Oracle or Teradata.

Cluster Distributions

You can preview data in mappings configured to run with the following distributions:

- Amazon EMR
- Azure HDInsight
- Cloudera CDH
- Cloudera CDP
- Hortonworks HDP
- MapR

Before you preview data on Amazon EMR, you must configure the `/etc/hosts` file on all nodes in the cluster to include the machine and IP address of the Data Integration Service.

Data Preview Process

When you preview data on the Spark engine, the following process occurs:

1. The Data Integration Service uses the Data Preview Service Module to determine whether to run the preview job in the native environment or on the Spark engine.
2. The Data Preview Service Module pushes the job to the Spark engine and generates a Spark workflow based on the preview point.
3. Based on the cluster distribution that you configure, either the Spark Jobserver or the DTM submits the Spark workflow tasks on the Hadoop cluster to generate the preview data.
4. The run-time engine stages the data based on the configured HDFS staging directory.
5. The Data Integration Service passes the staged data to the Developer tool and then deletes the staged data.
6. The results of the preview appear in the data viewer of the Developer tool.

Note: When you run data preview, the Data Integration Service validates the validation environments selected in the **Run-time** view.

Note: If you enable HTTPS protocol on the Data Integration Service, the Spark Jobserver also uses HTTPS protocol. The Spark Jobserver uses the same HTTPS keystore configuration that you set in the Data Integration Service process properties in the Administrator tool.

Previewing Data

When you run data preview in the Data Viewer view, you can perform tasks, such as viewing the structure and data of a hierarchy field and filtering ports to view. You can also export the results to a JSON file.

Before you preview data, verify that Spark is configured as the mapping run-time environment. If you are previewing hierarchical data in a complex file object, add the complex file object to a mapping to preview the data.

1. Open a mapping in the Developer tool, and select a preview point.
2. To run the data preview, click **Run** in the Data Viewer view.

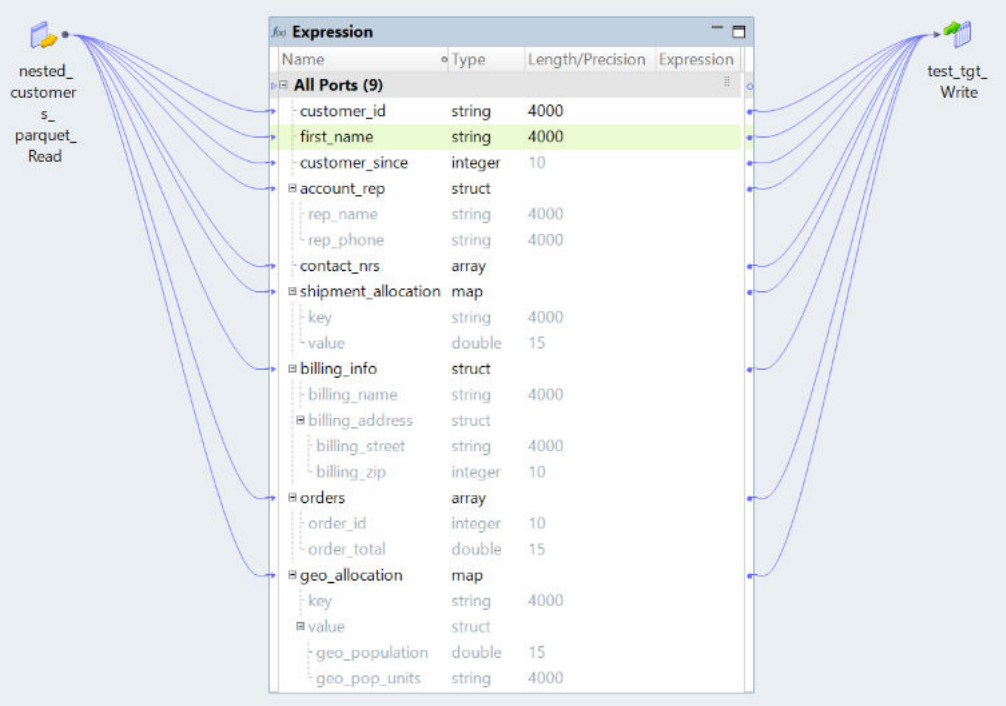
   The output of the data preview can be viewed in the Data Viewer view.
3. To filter ports in the output, choose the ports you want to view.
4. To view the structure and the data in a hierarchy field, select the field.

   The details of the hierarchy field appear on the hierarchical type panel.
5. To export the results to a JSON file, right-click the hierarchy field and select **Export Data**.

Data Preview Interface for Hierarchical Data

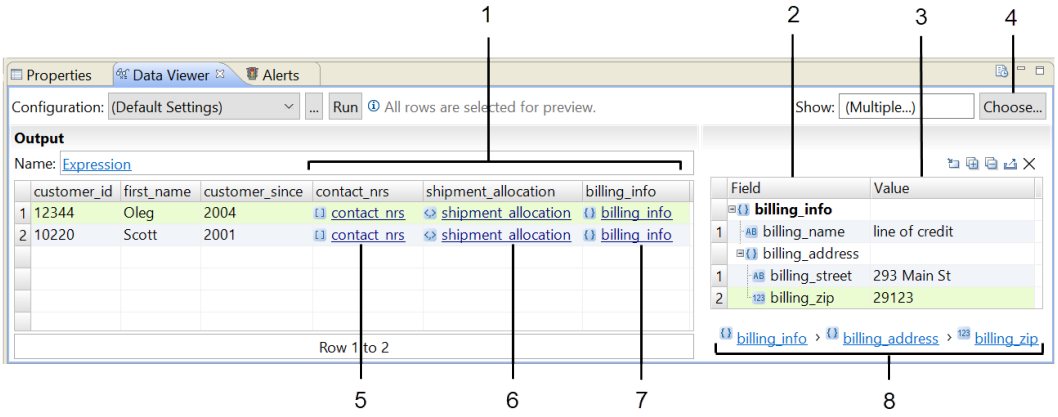
The data preview interface consists of Data Viewer view and hierarchical type panel. Preview data in a mapping based on the preview point that you choose.

The following image shows a mapping which has complex data types:



To view the hierarchical data after expressions have been evaluated, you use the Expression transformation as a preview point.

The following image shows the results of the data preview:



1. Hierarchy keys
2. Schema
3. Value
4. Port filter
5. Array data type
6. Map data type
7. Struct data type
8. Breadcrumbs

Data Viewer

When you preview data, you view ports in the preview point represented as columns in the Data Viewer.

To identify the hierarchical types, the following punctuation icons are used:

- [ ] is used to indicate array data type.
- { } is used to indicate struct data type.
- <,> is used to indicate map data type.

You can see the following information in the Data Viewer:

- View tooltips to see details regarding the type of hierarchical data. If you need to limit the number of ports that appear, you can choose the ports you want to view in the output.
- Highlight a field to see the details in the hierarchical type panel. When you right click any hierarchy field, you can export the data in that field in JSON format.
- If the data includes dates, note that date fields have an additional time stamp component (00:00:00).

Note: If the Data Integration Service node and the cluster node are located in different time zones, the Data Viewer displays a date and time based on the time zone of the cluster instead of the time zone of the Data Integration Service or Data Viewer.

Exporting Data

You can export the data that appears in the **Data Viewer** view to a JSON file. Export data when you want to create a local copy of the data.

1. In the **Data Viewer** view, right-click the results and select **Export Data**.

2. Enter a file name.
3. Select the location where you want to save the file.
4. Click **OK**.

The following rules and guidelines apply when you export data to a JSON file:

- JSON keys must be strings. If the source data is a map that uses structures as keys, the Data Integration Service serializes the keys when you export the data. The exported JSON file might include escape characters before double quotation marks that are added during serialization.
- JSON does not support binary data. When you export data that includes binary keys, the Data Integration Service replaces the key with the string "<binary>" which results in invalid JSON.
- If the source data is a map that contains duplicate keys of a complex data type, the Data Integration Service might produce invalid JSON when you export the preview data.

Hierarchical Type Panel

When you click a hierarchy field, the details of the hierarchy field appears in the hierarchical type panel.

In the hierarchical type panel, you can see the schema and values for the hierarchical data selected. You can navigate through the structure with the breadcrumbs. If the structure is very deep, you can expand and collapse the fields. When you expand the structure, you can view the data within the hierarchy. For example, when you view billing information in the hierarchical type panel, you can see the values for the billing name, street and ZIP Code.

Note: When you expand all rows, the Developer tool expands a maximum of 500 rows. If you have more than 500 rows, you can manually expand them to view more information.

Data Preview on Transformations

When you use a transformation as the preview point, the Spark engine might process the data preview job differently.

Note: When you choose a transformation as the preview point, the Spark engine might not maintain the sorted order when you configure the **Maintain row order** property.

Rank transformation

When you choose a Rank transformation as the preview point for data preview on the Spark engine, row order is maintained.

Sorter transformation

To maintain sorted row order when you preview data with a Sorter transformation before the preview point in a mapping, configure preferences to show all rows.

1. From the **Window** menu, open **Preferences**.
2. Navigate to **Informatica** > **Run Configurations** > **Data Viewer**.
3. In the **Results** tab, find the **Rows** section and select Show all rows.

Transformations with multiple output groups

When you preview data using the Spark engine, you cannot use the following transformations as preview points when the transformations are configured with multiple output groups:

Normalizer

Router

Update Strategy

Data Preview Logs

View logs when you preview data on the Spark engine to monitor jobs and find the cause of error messages.

The Spark Jobserver logs are located in the following directory: `<Informatica installation directory>/services/DataIntegrationService/modules/DataPreviewService/spark-jobserver-0.9.0/<Data Integration Service name>/log`

You can view the following types of logs when you run data preview jobs using the Spark Jobserver:

Spark Jobserver start-up log

Messages logged when the Spark Jobserver starts.

The Spark Jobserver start-up log is persisted until you manually clear it.

Location: `<Spark Jobserver log directory>/server_start.log`

Spark Jobserver log

Messages logged as the Spark Jobserver runs.

The Spark Jobserver log is persisted until you manually clear it.

Location: `<Spark Jobserver log directory>/spark-job-server.log`

Spark context log

Messages logged when a YARN application is created.

The Spark context log is not persisted. The log is created each time a Spark context is created.

Location: `<Spark Jobserver log directory>/jobserver-INFA_DATA_PREVIEW_APP_<Data Integration Service name>8650657650347819144/spark-job-server.out`

Note: When a data preview job fails, use the job ID found in the session log to locate the stacktrace in the Spark context log that relates to the failed job.

Mapping run log

Messages logged when a mapping runs.

The mapping run logs are not persisted. The logs are created each time you run a mapping.

Location: `<Informatica installation directory>/logs/<node name>/services/DataIntegrationService/disLogs/ms`

Note: To view mapping run logs, select the **Show Log** button in the Data Viewer view.

Rules and Guidelines for Data Preview on the Spark Engine

Consider the following rules and guidelines when you work with data preview on the Spark engine:

- You cannot preview data in mappings with dynamic complex data types, such as dynamic arrays, dynamic maps, and dynamic structs.
- If a map data type source with primitive keys includes duplicate keys, the Data Viewer displays only one instance of the duplicate key-value pair. If a map data type source with complex keys includes duplicate keys, the Data Viewer displays all key-value pairs.
- You cannot preview data for a mapping that reads hierarchical data from a Hive source.
- You can run up to 10 concurrent data preview jobs on the Spark engine.
- Previewing data on the Spark engine is memory intensive. Increase the Heap memory size when you run concurrent preview jobs.
- For high volume data preview jobs that use Spark Jobserver, configure the following Spark advanced properties in the Hadoop connection to increase driver and executor memory:

  spark.driver.cores

  spark.driver.memory

  spark.executor.cores

  spark.executor.memory

  For more information, see "Spark Advanced Properties" on page 374.
- Effective in version 10.4.0, previewing hierarchical data when the Data Integration Service runs on a grid is available for technical preview.

Technical preview functionality is supported for evaluation purposes but is unwarranted and is not supported in production environments or any environment that you plan to push to production. Informatica intends to include the preview functionality in an upcoming release for production use, but might choose not to in accordance with changing market or technical circumstances. For more information, contact Informatica Global Customer Support.

CHAPTER 9

Cluster Workflows

This chapter includes the following topics:

- Cluster Workflows Overview, 169
- Cluster Workflow Components, 170
- Configure Databricks Clusters Using Warm Pools, 171
- Cluster Workflows Process , 172
- Create Cluster Task Properties, 173
- Mapping Task Properties, 185
- Add a Delete Cluster Task, 186
- Deploy and Run the Workflow, 186

Cluster Workflows Overview

A cluster workflow is a workflow that automates the creation of a cluster using a Create Cluster task, and then runs one or more mappings on that cluster. You can create a cluster workflow to run Mapping and other tasks on a cloud platform cluster. To use an ephemeral cluster to run mappings in the workflow, include a Delete Cluster task to terminate and delete the cluster after the mappings run.

The cluster workflow uses other elements that enable communication between the Data Integration Service and the cloud platform, such as a cloud provisioning configuration and a cluster connection.

A cluster workflow contains a Create Cluster task that you configure with information about the cluster to create. If you want to create an ephemeral cluster, you can include a Delete Cluster task. An ephemeral cluster is a cloud platform cluster that you create to run mappings and other tasks, and then terminate when tasks are complete. Create ephemeral clusters to save cloud platform resources.

You can use cluster workflows to create Hadoop clusters on the Microsoft Azure or Amazon AWS cloud platforms in a Hadoop environment, or Databricks clusters in a Databricks environment on Azure or AWS.

Cluster Workflows Platform Support

Cluster workflow capabilities vary depending on the environment and the cloud platform.

Hadoop Clusters

On the Azure platform, you can create an ephemeral HDInsight cluster that accesses ADLS Gen2 resources.

On the AWS platform, you can create an ephemeral Amazon EMR cluster to access S3, Redshift, and Snowflake resources.

On Cloudera Altus, you create a workflow with Command tasks that perform the tasks that a cluster workflow automates. For more information about creating a cluster on Cloudera Altus, see the article "How to Create Cloudera Altus Clusters with a Cluster Workflow" on the Informatica Documentation Portal.

Databricks Clusters

On the Azure and AWS Databricks platforms, you can configure Databricks ephemeral clusters.

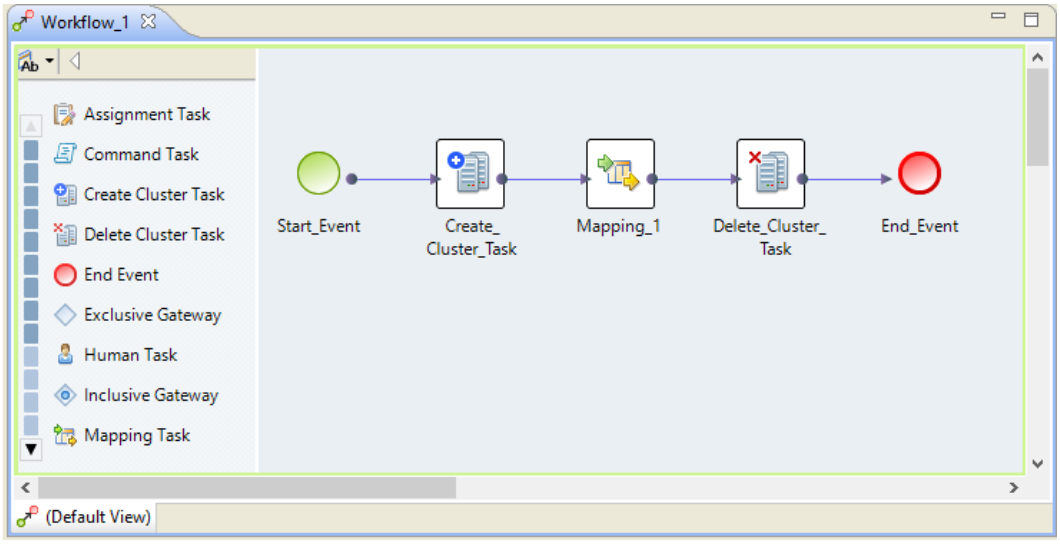
Optionally, you can configure the clusters to start from sets of cached VM instances, called warm pools. These instances wait on standby in a running state for ephemeral cluster creation. You can choose to have the instances remain on standby when the ephemeral clusters are terminated. For more information, see the Databricks documentation.

Enable warm pools for ephemeral Databricks clusters in the advanced properties of the Create Cluster workflow task.

Cluster Workflow Components

The cluster workflow that creates an ephemeral cluster includes a Create Cluster task, at least one Mapping task, and a Delete Cluster task.

The following image shows a sample cluster workflow:



A cluster workflow uses the following components:

Cloud provisioning configuration

The cloud provisioning configuration is associated with the Create Cluster task through the cluster connection.

Cluster connection

The cluster connection to use with a cluster workflow is associated with a cloud provisioning configuration. You can use a Hadoop or Databricks cluster connection. When you run the workflow the Data Integration Service creates a temporary cluster connection.

Create Cluster task

The Create Cluster task contains all the settings that the cloud platforms require to create a cluster with a master node and worker nodes. It also contains a reference to a cloud provisioning configuration. Include one Create Cluster task in a cluster workflow.

Mapping task

Add a data engineering mapping to the Mapping task. A cluster workflow can include more than one Mapping task. You can run some mappings on an existing cluster and you can run some mappings on a cluster that the workflow creates. You configure the mappings and Mapping tasks based on where you want to run the task.

Delete Cluster task

The Delete Cluster task terminates the cluster and deletes the cluster and all resources that the workflow creates.

Configure Databricks Clusters Using Warm Pools

On the Azure and AWS Databricks platforms, you can configure Databricks ephemeral clusters to start on sets of cached VM instances, called warm pools. The VM instances wait on standby in a running state for ephemeral cluster creation. When you enable warm pools, you reduce the time it takes to create clusters if the pool contains enough idle instances.

Prerequisite

Ephemeral clusters from warm pools require the Databricks administrator to create a cluster pool in the Databricks workspace. Choose from any available instance types for the VMs in the pool.

Configure the following properties when you create pools:

Property	Description
Min Idle	Minimum number of instances that will remain running in the pool in an idle state. These instances do not terminate, regardless of the setting specified in the Idle Instance Auto Termination property.
Idle Instance Auto Termination	The number of minutes that the pool will maintain an instance in an idle state after jobs are complete.

Note: The idle instances in the pool do not terminate regardless of the value in the Idle Instance Auto Termination property. Amazon charges for idle instances in a pool.

Configure the Cluster Workflow to Use Warm Pool Instances

Enable warm pools for Databricks ephemeral clusters in the Create Cluster task Advanced Properties.

The following table describes the properties to enable warm pools:

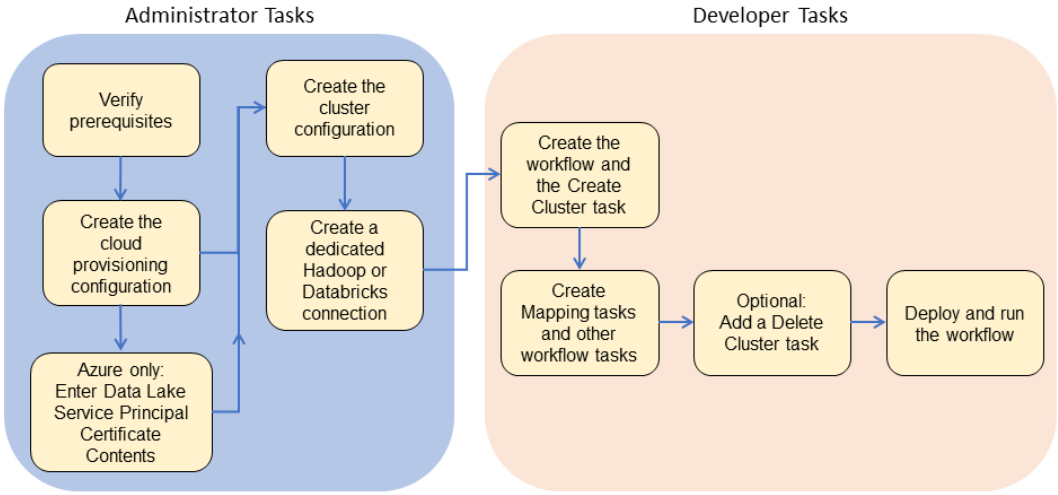
Property	Description
Warm Pools	Check this option to enable creation of the Databricks cluster using warm pools. Default is unchecked (false).
Pool ID	Required when you enable warm pools. Value of the DatabricksInstancePoolId property that identifies the warm pool resource.

Note: When you enable warm pools, some of the advanced properties that allow you to specify cluster instance details are disabled. Set cluster instance details when you configure pools in the Databricks console.

Cluster Workflows Process

Creation of a cluster workflow requires administrator and developer tasks.

The following image shows the process to create, configure, and run a cluster workflow:



The cluster workflow development process requires tasks from the following users:

Administrator

An administrator completes the following tasks:

1. Verify prerequisites for the domain and the cloud platform.
2. Create the cloud provisioning configuration in the Administrator tool.
3. Create a cluster configuration in the Administrator tool.
4. Create a dedicated cluster connection to associate with the cluster configuration.

For information about administrator tasks, see the *Data Engineering Administrator Guide*.

Developer

A developer completes the following tasks:

1. Create the workflow and the Create Cluster task.
2. Create Mapping tasks and other tasks as required.
3. Create a Delete Cluster task if you want to delete the cluster and resources when processing is complete.
4. Deploy and run the workflow.

Create Cluster Task Properties

When you configure a Create Cluster task, you configure options that are common for all cluster types, and you configure advanced options for specific clusters.

General Properties for the Create Cluster Task

The following table describes the General properties for the Create Cluster task:

Property	Description
Name	Task name.
Description	Optional description.
Connection Name	Name of the cloud provisioning configuration to use with the workflow.
Connection Type	Choose from the following options: - Amazon EMR. Create an Amazon EMR cluster in the Hadoop environment. - HDInsight. Create an Azure HDInsight cluster in the Hadoop environment. - Databricks. Create a Databricks cluster in the Databricks environment.

Input Properties for the Create Cluster Task

You can assign task input to workflow parameters and variables.

The following table describes the Input properties for the Create Cluster task:

Property	Description
Create Cluster Task Inputs	Parameters or variables to use for task input.

Output Properties for the Create Cluster Task

You can assign variables to the Create Cluster task output.

The following table describes the Output properties for the Create Cluster task:

Property	Description
Start Time	Date and time that the task started running.
End Time	Date and time that the task stopped running
Is Successful	Indicates whether the task ran successfully.
Cluster Identifier	The Cluster Identifier property of the Create Cluster task overrides the Cluster Identifier property of the Mapping task. Default is AutoDeployCluster.

Advanced Properties for the Create Cluster Task

Advanced properties for the Create Cluster task depend on the cluster type that you create.

For information about advanced properties for Microsoft Azure, see "Advanced Properties for Azure HDInsight" on page 177.

For information about advanced properties for Amazon Web Services, see "Advanced Properties for Amazon EMR" on page 174.

For information about advanced properties to run mappings on the Blaze engine, see "Advanced Properties for the Blaze Engine" on page 183.

For information about advanced properties for Databricks, see "Advanced Properties for Azure Databricks" on page 178.

Advanced Properties for Amazon EMR

Set the advanced properties for an Amazon EMR cluster.

General Options

The following table describes general options that you can set for an EMR cluster:

Property	Description
Cluster Name	Name of the cluster to create.
Release Version	EMR version to run on the cluster. Enter the AWS version tag string to designate the version. For example: `emr-5.8.0` Default is Latest version supported.
Connection Name	Name of the Hadoop connection that you configured for use with the cluster workflow.
S3 Log URI	Optional. S3 location of logs for cluster creation. Format: `s3://<bucket name>/<folder name>` If you do not supply a location, no cluster logs will be stored.

Master Instance Group Options

The following table describes master instance group options that you can set for an EMR cluster:

Property	Description
Master Instance Type	Master node EC2 instance type. You can specify any available EC2 instance type. Default is m4.4xlarge.
Master Instance Maximum Spot Price	Maximum spot price for the master node. Setting this property changes the purchasing option of the master instance group to Spot instead of On-demand.

Core Instance Group Options

The following table describes core instance group options that you can set for an EMR cluster:

Property	Description
Core Instance Type	Core node EC2 instance type. You can specify any available EC2 instance type. Default is m4.4xlarge.
Core Instance Count	Number of core EC2 instances to create in the cluster. Default is 2.
Core Instance Maximum Spot Price	Maximum spot price for core nodes. Setting this property changes the purchasing option of the core instance group to Spot instead of On-demand.
Core Auto-Scaling Policy	Optional. Auto-scaling policy for core instances. Type the policy JSON statement here, or provide a path to a file that contains a JSON statement. Format: `file:\\<path_to_policy_config_file>`

Task Instance Group Options

The following table describes task instance group options that you can set for an EMR cluster:

Property	Description
Task Instance Type	Task node EC2 instance type. You can specify any available EC2 instance type. Default is m4.4xlarge.
Task Instance Count	Number of task EC2 instances to create in the cluster. Default is 2.

Property	Description
Task Instance Maximum Spot Price	Maximum spot price for task nodes. Setting this property changes the purchasing option of the task instance group to Spot instead of On-demand. For more information on spot pricing, see the AWS documentation.
Task Auto-Scaling Policy	Optional. Auto-scaling policy for task instances. Type the policy JSON statement here, or provide a path to a file that contains a JSON statement. Format: `file:\\<path_to_policy_config_file>`

Additional Options

The following table describes additional options that you can set for an EMR cluster:

Property	Description
Root device EBS volume size (GB)	Number of GB of the EBS root device volume. Enter a value between 10 and 100. Default is 10.
Tags	Optional. Tags to propagate to cluster EC2 instances. Tags assist in identifying EC2 instances. Format: `TagName1=TagValue1,TagName2=TagValue2`
Bootstrap Actions	Optional. Actions to perform after EC2 instances are running, and before applications are installed. Type the JSON statement here, or provide a path to a file that contains a JSON statement. Format: `file:\\<path_to_policy_config_file>`
Custom AMI ID	Optional. ID of a custom Amazon Linux Amazon Machine Image (AMI). Copy the value from the AWS console.
Security Configuration	Optional. The name of a security configuration for authentication and encryption on the cluster. Amazon EMR supports server-side encryption (SSE) and client-side encryption (CSE) configurations. You can use the following at-rest security configurations: - SSE with Amazon S3-managed keys (SSE-S3) - SSE with AWS KMS-managed keys (SSE-KMS) - CSE with AWS KMS-managed keys (CSE-KMS) - Custom CSE configurations* You can use the following in-transit security configurations: - PEM - Custom in-transit configurations* You can also use custom AMIs for local disk security. ** To use custom security configurations, manually copy the .jar file to the Data Integration Service machine.*
Applications	Optional. Applications to add to the default applications that AWS installs. AWS installs certain applications when it creates an EMR cluster. In addition, you can specify additional applications. Select additional applications from the drop-down list. This field is equivalent to the Software Configuration list in the AWS EMR cluster creation wizard.

Property	Description
Software Settings	Optional. Custom configurations to apply to the applications installed on the cluster. This field is equivalent to the Edit Software Settings field in the AWS cluster creation wizard. You can use this as a method to modify the software configuration on the cluster. Type the configuration JSON statement here, or provide a path to a file that contains a JSON statement. Format: `file:\\<path_to_custom_config_file>`
Steps	Optional. Commands to run after cluster creation. For example, you can use this to run Linux commands or HDFS or Hive Hadoop commands. This field is equivalent to the Add Steps field in the AWS cluster creation wizard. Type the command statement here, or or provide a path to a file that contains a JSON statement. Format: `file:\\<path_to_command_file>`

Advanced Properties for Azure HDInsight

Set the advanced properties for an Azure HDInsight cluster.

The following table describes the Advanced properties for a Microsoft Azure HDInsight cluster:

Property	Description
Cluster Name	Name of the cluster to create.
Azure Cluster Type	Type of the cluster to be created. Choose one of the options in the drop-down list. Default is Hadoop.
HDInsight version	HDInsight version to run on the cluster. Enter the HDInsight version tag string to designate the version. Default is the latest version supported.
Azure Cluster Location	Use the drop-down list to choose the location in which to create the cluster.
Head Node VM Size	Size of the head node instance to create. Default is Standard_D12_v2.
Number of Worker Node Instances	Number of worker node instances to create in the cluster. Default is 2.
Worker Node VM Size	Size of the worker node instance to create. Default is Standard_D13_v2.
Default Storage Type	Primary storage type to be used for the cluster. Choose one of the following options: - Azure Data Lake Store - Azure BLOB storage account Default is BLOB storage

Property	Description
Default Storage Container or Root Mount Path	Default container for data. Type one of the following paths: - For ADLS storage, type the path to the storage. For example, you can type `storage-name` or `storage-name/folder-name`. - For blob storage, type the path to the container. Format: `/path/`
Log Location	Optional. Path to the directory to store workflow event logs. Default is `/app-logs`.
Attach External Hive Metastore	If you select this option, the workflow attaches an external Hive metastore to the cluster if you configured an external Hive metastore in the cloud provisioning configuration.

Bootstrap JSON String	JSON statement to run during cluster creation. You can use this statement to configure cluster details. For example, you could configure Hadoop properties on the cluster, add tags to cluster resources, or run script actions.
	Choose one of the following methods to populate the property:
	- Type the JSON statement. Use the following format:
	`{ "core-site" : { "<sample_property_key1>": "<sample_property_val1>", "<sample_property_key2>": "<sample_property_val2>" }, "tags": { "<tag_key>": "<tag_val>" }, "scriptActions": [ { "name": "setenvironmentvariable", "uri": "scriptActionUri", "parameters": "headnode" } ] }`
	- Provide a path to a file that contains a JSON statement. Format:
	`file://<path_to_bootstrap_file>`

Using ADLS Gen2 Resources with a Cluster Workflow

When a cluster workflow accesses ADLS Gen2 resources, verify that the Data Integration Service machine is configured to run in the same time zone that the cluster runs in.

Set the time zone environment variable on each Data Integration Service machine to the time zone of the cluster and then restart the domain.

To set the time zone, run the following command on the Data Integration Service machine:

```
setenv TZ <cluster time zone>
```

Advanced Properties for Azure Databricks

Set general and advanced options for advanced properties of the Azure Databricks Create Cluster task.

General Options

The following table describes the general options that you can set for a Databricks cluster:

Property	Description
Cluster Name	Name of the cluster to create.
Databricks Runtime Version	The Databricks version to run on the cluster. Default is the latest supported version. To manually enter a version, select *(Assign to Task Input)*. The Developer tool creates a Create Cluster Task Input called *Databricks Runtime Version*. Set the value of the input to the Databricks version tag string. For example, set the value to `5.5.x-scala2.11` to use Databricks version 5.5.
Python Version	The Python version to run if you include a Python transformation. Default is the latest supported version.
Warm Pool	Check this option to enable creation of the Databricks cluster using a warm pool. Default is unchecked (false).
Pool ID	Value of the DatabricksInstancePoolId property that identifies the warm pool resource. To retrieve the value of this property, go to the warm pool in the Databricks workspace and browse to **Configuration** > **Tags**. Required when the Warm Pool option is checked.
Driver Type	The type of node that you want to use for the driver node. Default is the worker type ID.
Worker Type	The type of node that you want to use for the worker node.
Workers	The number of worker nodes to create for the cluster. If you configure the cluster to scale automatically, this property is ignored. Default is 1.
Autoscale	Automatically scales the number of worker nodes based on workload.
Min Workers	The minimum number of worker nodes to use when the cluster is configured to scale automatically. Default is 0.
Max Workers	The maximum number of worker nodes to use when the cluster is configured to scale automatically. Default is 1.

Advanced Options

Configure advanced options such as environment variables and automatic termination.

The following table describes the advanced options that you can set for an Azure Databricks cluster:

Property	Description
Auto Termination	Enables automatic termination of the cluster.
Auto Termination Time	Terminates the cluster after it is inactive for the specified number of minutes. Enter a value between 10 and 10,000. If you do not configure this, or if you set to 0, the cluster will not automatically terminate.
Cluster Log Conf	The location to deliver logs for long-term storage. If configured, the Databricks Spark engine will deliver the logs every five minutes. Provide the path to DBFS.
Init Scripts	The location where you store init scripts. You can enter multiple destinations. The scripts are run sequentially in the order that you configure them. If you need to install additional Python libraries, specify the init script file location in this property. Use the following format: `dbfs:/<path to init script>,dbfs:/<path to init script>`
Cluster Tags	Labels that you can assign to resources for tracking purposes. Enter key-value pairs in the following format: <key1>=<value1>,<key2>=<value2>. You can also provide a path to a local file that contains the key-value pairs. Use the following format: `file:\\<file path>`
Spark Configurations	Performance configurations for the Databricks Spark engine. Enter key-value pairs in the following format: key1='value1' key2='value2'. You can also provide a path to a file that contains the key-value pairs.
Environment Variables	Environment variables that you can configure for the Databricks Spark engine. Enter key-value pairs in the following format: key1='value1' key2='value2'

Advanced Properties for AWS Databricks

Set the advanced properties for an AWS Databricks cluster.

General Options

The following table describes the general options that you can set for a Databricks cluster:

Property	Description
Cluster Name	Name of the cluster to create.
Databricks Runtime Version	The Databricks version to run on the cluster. Default is the latest supported version. To manually enter a version, select *(Assign to Task Input)*. The Developer tool creates a Create Cluster Task Input called *Databricks Runtime Version*. Set the value of the input to the Databricks version tag string. For example, set the value to `5.5.x-scala2.11` to use Databricks version 5.5.
Python Version	The Python version to run if you include a Python transformation. Default is the latest supported version.
Warm Pool	Check this option to enable creation of the Databricks cluster using a warm pool. Default is unchecked (false).
Pool ID	Value of the DatabricksInstancePoolId property that identifies the warm pool resource. To retrieve the value of this property, go to the warm pool in the Databricks workspace and browse to **Configuration** > **Tags**. Required when the Warm Pool option is checked.
Driver Type	The type of node that you want to use for the driver node. Default is the worker type ID.
Worker Type	The type of node that you want to use for the worker node.
Workers	The number of worker nodes to create for the cluster. If you configure the cluster to scale automatically, this property is ignored. Default is 1.
Autoscale	Automatically scales the number of worker nodes based on workload.
Min Workers	The minimum number of worker nodes to use when the cluster is configured to scale automatically. Default is 0.
Max Workers	The maximum number of worker nodes to use when the cluster is configured to scale automatically. Default is 1.

Advanced Options

Configure advanced options such as automatic termination and on-demand instances.

The following table describes advanced options that you can set for an AWS Databricks cluster:

Property	Description
Enable autoscaling local storage	Enables Databricks to monitor available disk space on worker nodes and automatically add additional EBS volumes.
EBS Volume Type	The type of volume that Databricks can add to cluster nodes. Set this property when you enable autoscaling.
Number of Volumes	The number of volumes to provision for each instance. Enter a value between 0 and 10. Set this property when you enable autoscaling and configure EBS volume type.
Size in GB	The size in gigabytes of each EBS volume. Set this property when you enable autoscaling and configure EBS volume type.
Auto Termination	Enables automatic termination of the cluster.
Auto Terminate Time	Terminates the cluster after it is inactive for the specified number of minutes. Enter a value between 10 and 10,000. If you do not configure this, or if you set to 0, the cluster will not automatically terminate.
On-Demand/Spot Composition	The number of on-demand nodes. Enter a value between 0 and the number of worker nodes set in General Options. Any remaining worker nodes will be spot instances. On-demand nodes are always available to use. Spot instances might terminate running jobs if they become unavailable. The driver node is always an on-demand node. Set this property when you enable **Spot fall back to On-Demand**. Default is 1.
Spot fall back to On-Demand	Enables on-demand instances to be used as a fallback. If you are using spot instances and the market price for spot instances surges above your spot bid price, AWS terminates the spot instances. When you enable this property, on-demand instances are used in place of the spot instances when they terminate.
Availability Zone	The AWS cluster availability zone. Default is us-east-1e.
Spot Bid Price	The maximum percent of the on-demand instance price that you bid on spot instances. Spot instances are priced as a percentage of the on-demand price and are not always available. If the market price for spot instances surges above the bid price set here and you do not enable **Spot fall back to On-Demand**, AWS terminates the spot instance. Default is 100%.
IAM Role ARN	The instance profile ARN (Amazon Resource Name) that corresponds to the AWS IAM (Identity and Access Management) role. Copy the value from the AWS console in the following format: `arn:aws:iam::<account-id>:instance-profile/<role-name>` IAM roles allow you to access data from Databricks clusters. Add new IAM roles in the Administrator tool.

Property	Description
Spark Configurations	Performance configurations for the Databricks Spark engine. Enter key-value pairs in the following format: key1='value1' key2='value2'. You can also provide a path to a file that contains the key-value pairs.
Environment Variables	Environment variables that you can configure for the Databricks Spark engine. Enter key-value pairs in the following format: key1='value1' key2='value2'
Cluster Tags	Labels that you can assign to resources for tracking purposes. Enter key-value pairs in the following format: <key1>=<value1>,<key2>=<value2>. You can also provide a path to a local file that contains the key-value pairs. Use the following format: `file:\\<file path>`
SSH Public Key	The SSH public key to log into the driver and worker instances if you enable SSH. Copy the value from the Databricks console.
Cluster Log Conf	The location to deliver logs for long-term storage. If configured, the Databricks Spark engine will deliver the logs every five minutes. Provide the path to DBFS.
Init Scripts	The location where you store init scripts. You can enter multiple destinations. The scripts are run sequentially in the order that you configure them. If you need to install additional Python libraries, specify the init script file location in this property. Use the following format: `dbfs:/<path to init script>,dbfs:/<path to init script>`

Advanced Properties for the Blaze Engine

If you want to use the Blaze engine to run mappings on the cloud platform cluster, you must set cluster configuration properties in the Software Setting property of the Create Cluster task.

Configure the Create Cluster task to set configuration properties in *-site.xml files on the cluster. Hadoop clusters run based on these settings.

Use a text file to specify *-site.xml file settings, and specify the path to the file in the Software Settings property.

The following text shows sample configuration of the Software Settings property file:

```
[
  {
     "Classification":"yarn-site",
     "Properties":{
     "yarn.scheduler.minimum-allocation-mb":"250",
     "yarn.scheduler.maximum-allocation-mb":"8192",
     "yarn.nodemanager.resource.memory-mb":"16000",
     "yarn.nodemanager.resource.cpu-vcores":"12"
   }
  },
  {
    "Classification":"core-site",
    "Properties":{
    "hadoop.proxyuser.<DIS/OSPUSER>.groups":"<group names>",
    "hadoop.proxyuser.<DIS/OSPUSER>.hosts":"*"
  }
 }
]
```

yarn-site

yarn.scheduler.minimum-allocation-mb

The minimum RAM available for each container. Required for Blaze engine resource allocation.

yarn.scheduler.maximum-allocation-mb

The maximum RAM available for each container. Required for Blaze engine resource allocation.

yarn.nodemanager.resource.memory-mb

The maximum RAM available for each container. Set the maximum memory on the cluster to increase resource memory available to the Blaze engine.

yarn.nodemanager.resource.cpu-vcores

The number of virtual cores for each container. Required for Blaze engine resource allocation.

core-site

hadoop.proxyuser.<proxy user>.groups

Defines the groups that the proxy user account can impersonate. On a secure cluster the <proxy user> is the Service Principal Name that corresponds to the cluster keytab file. On a non-secure cluster, the <proxy user> is the system user that runs the Informatica daemon.

Set to group names of impersonation users separated by commas. If less security is preferred, use the wildcard " * " to allow impersonation from any group.

hadoop.proxyuser.<user>.hosts

Defines the host machines from which impersonated connections are allowed. On a secure cluster the <proxy user> is the Service Principal Name that corresponds to the cluster keytab file. On a non-secure cluster, the <proxy user> is the system user that runs the Informatica daemon.

Set the property to " * " to allow impersonation from any host. This is required to run a Blaze mapping on a cloud platform cluster.

Advanced Properties for a Hive Metastore Database

If you want to use a relational database on the cloud platform as the Hive metastore database for the cluster, you must set cluster configuration properties in the Software Setting property of the Create Cluster task.

Configure the Create Cluster task to set configuration properties in the hive-site.xml configuration file on the cluster. Use a text file to specify hive-site settings, and specify the path to the file in the Software Settings property.

Create the following properties in the Software Settings property file:

javax.jdo.option.ConnectionURL

JDBC connection string for the data store.

javax.jdo.option.ConnectionDriverName

JDBC driver class name for the data store. Specify a JDBC driver that is compatible with the cloud platform.

javax.jdo.option.ConnectionUserName

User name to use to connect to the database.

javax.jdo.option.ConnectionPassword

Password for the database user account.

The following text shows sample configuration of the Software Settings property file:

```
{
      "Classification":"hive-site",
      "Properties":{
         "javax.jdo.option.ConnectionURL":"jdbc:mysql:\/\/<RDS_HOST>:<PORT>\/
<USER_SCHEMA>?createDatabaseIfNotExist=true",
         "javax.jdo.option.ConnectionDriverName":"<JDBC driver name>",
         "javax.jdo.option.ConnectionUserName":"<USER>",
         "javax.jdo.option.ConnectionPassword":"<USER>"
      }
   }
```

For example,

```
{
      "Classification":"hive-site",
      "Properties":{
         "javax.jdo.option.ConnectionURL":"jdbc:mysql:\/\/<host name>:<port number>\/
hive?createDatabaseIfNotExist=true",
         "javax.jdo.option.ConnectionDriverName":"org.mariadb.jdbc.Driver",
         "javax.jdo.option.ConnectionUserName":"hive",
         "javax.jdo.option.ConnectionPassword":"hive"
      }
   }
```

Mapping Task Properties

Set mapping and Mapping task properties to specify where the workflow runs Mapping tasks.

When you include a Mapping task in a cluster workflow, you must identify the cluster where you want the mapping to run.

The following table describes the Cluster Identifier property values that you can configure in the Advanced properties of a Mapping task:

Cluster Identifier Property	Description
Auto Deploy	Runs the mapping on the cluster that the workflow creates. The Data Integration Service generates a temporary cluster connection based on the connection associate with the cluster provisioning configuration. The connection property in the mapping must also be set to Auto Deploy.
Blank	Runs the mapping on the cluster configured in the cluster connection associated with the mapping. The connection property in the mapping must be set to a cluster connection associated with a cluster configuration. Validation fails if it is associated with the cluster provisioning configuration.
Assign to task input	Accept input from a source other than the Create Cluster task. Use this option if you configure input properties to use parameters.

Note: If the Mapping task and the Create Cluster task contain different values for the cluster identifier, the Create Cluster task value takes precedence.

Add a Delete Cluster Task

To create an ephemeral cluster, add a Delete Cluster task.

The Delete Cluster task terminates the cluster and deletes the cluster and other resources that the cluster workflow creates.

If you do not add a Delete Cluster task, the cluster that the workflow creates remains running when the workflow ends. You can delete the cluster at any time.

1. Drag a Delete Cluster task to the workflow editor.
2. In the General properties, optionally rename the Delete Cluster task.
3. Connect the final task in the workflow to the Delete Cluster task, and connect the Delete Cluster task to the workflow End_Event.

You can also use infacmd ccps deleteCluster to delete a cloud cluster.

Deploy and Run the Workflow

After you complete the cluster workflow, deploy and run the workflow.

You can monitor AWS and Azure cluster workflows on the web console. If you configured a log location, view the logs at the location that you configured in the Create Cluster task properties.

You can monitor Databricks cluster workflows on the Databricks workspace.

You can also monitor Data Integration Service jobs in the Administrator tool.

Note: After the workflow begins executing tasks, the task to provision the cluster may take several minutes.

Monitoring Azure HDInsight Cluster Workflow Jobs

You can access mapping log URLs through the **Monitoring** tab in the Administrator tool to monitor workflow jobs that run on an Azure HDInsight cluster. The log location depends on the run-time engine that each mapping uses.

To access the monitoring URL for mappings that run on the Blaze or Spark engine, expand the workflow and the mapping on the **Monitoring** tab. Select the Grid Task and view the value for the Monitoring URL property in the lower pane. Use this path to find the log.

CHAPTER 10

Profiles

This chapter includes the following topics:

- Profiles Overview, 187
- Native Environment, 187
- Hadoop Environment, 188
- Sampling Options, 189
- Creating a Single Data Object Profile in Informatica Developer, 189
- Creating an Enterprise Discovery Profile in Informatica Developer, 190
- Creating a Column Profile in Informatica Analyst, 191
- Creating an Enterprise Discovery Profile in Informatica Analyst, 193
- Creating a Scorecard in Informatica Analyst, 194
- Monitoring a Profile, 195
- Profiling Functionality Support, 195
- Troubleshooting, 196

Profiles Overview

You can create and run column profiles, enterprise discovery profiles, and scorecards in the native run-time environment or Hadoop run-time environment.

When you create or edit a profile or scorecard, you can choose Native or Hadoop validation environment. You can choose the Native, Blaze, or Spark run-time environment based on the validation environment. To process the profiles quickly, you can choose the Hadoop validation environment. You can run profiles with a Hive, Oracle, or HDFS data sources on the Spark engine in the Hadoop environment.

Native Environment

In Informatica Developer, you can run single object profiles, multiple object profiles, and enterprise discovery profiles in the native environment. In Informatica Analyst, you can run column profiles, enterprise discovery profiles, and scorecards in the native environment.

When you run a profile in the native run-time environment, the Analyst tool or Developer tool submits the profile jobs to the Profiling Service Module. The Profiling Service Module then breaks down the profile jobs

into a set of mappings. The Data Integration Service runs these mappings on the same machine where the Data Integration Service runs and writes the profile results to the profiling warehouse. By default, all profiles run in the native run-time environment.

You can use native sources to create and run profiles in the native environment. A native data source is a non-Hadoop source, such as a flat file, relational source, or mainframe source. You can also run a profile on a mapping specification or a logical data source with a Hive or HDFS data source in the native environment.

Hadoop Environment

You can run profiles and scorecards in the Hadoop environment on the Blaze or Spark engine.

Select a Hadoop connection to push the profile logic to the Blaze or Spark engine to run a profile. When you run a profile on the Blaze or Spark engine, the Analyst tool or Developer tool submits the profile job to the Profiling Service Module. The Profiling Service Module then breaks down the profile jobs into a set of mappings. The Data Integration Service pushes the mappings to the Hadoop environment through the Hadoop connection. The Blaze or Spark engine processes the mappings and the Data Integration Service writes the profile results to the profiling warehouse.

In the Developer tool, you can run single object profiles and multiple object profiles, and enterprise discovery profiles on the Blaze or Spark engine. In the Analyst tool, you can run column profiles, enterprise discovery profiles, and scorecards on the Blaze or Spark engine.

Column Profiles for Sqoop Data Sources

You can run a column profile on data objects that use Sqoop. After you choose Hadoop as a validation environment, you can select the Blaze engine or Spark engine on the Hadoop connection to run the column profiles.

When you run a column profile on a logical data object or customized data object, you can configure the num-mappers argument to achieve parallelism and optimize performance. You must also configure the split-by argument to specify the column based on which Sqoop must split the work units.

Use the following syntax:

```
--split-by <column_name>
```

If the primary key does not have an even distribution of values between the minimum and maximum range, you can configure the split-by argument to specify another column that has a balanced distribution of data to split the work units.

If you do not define the split-by column, Sqoop splits work units based on the following criteria:

- If the data object contains a single primary key, Sqoop uses the primary key as the split-by column.
- If the data object contains a composite primary key, Sqoop defaults to the behavior of handling composite primary keys without the split-by argument. See the Sqoop documentation for more information.
- If a data object contains two tables with an identical column, you must define the split-by column with a table-qualified name. For example, if the table name is CUSTOMER and the column name is FULL_NAME, define the split-by column as follows:
  `--split-by CUSTOMER.FULL_NAME`
- If the data object does not contain a primary key, the value of the m argument and num-mappers argument default to 1.

When you use Cloudera Connector Powered by Teradata or Hortonworks Connector for Teradata and the Teradata table does not contain a primary key, the split-by argument is required.

Sampling Options

Sampling options determine the number of rows on which the Analyst tool or Developer tool runs a profile. You can configure sampling options when you define a profile or run a profile.

The following table describes the sampling options for a profile:

Sampling Option	Description
All rows	Runs a profile on all the rows in the data object. Supported on Native, Blaze, and Spark run-time environment.
Sample first <number> rows	Runs a profile on the sample rows from the beginning of the rows in the data object. You can choose a maximum of 2,147,483,647 rows. Supported on Native and Blaze run-time environment.
Random sample <number> rows	Runs a profile on a randomly picked number of the rows in the data object. You can choose a maximum of 2,147,483,647 rows. Supported on Native and Blaze run-time environment.
Random sample (auto)	Runs a profile on the sample rows computed on the basis of the number of rows in the data object. Supported on Native and Blaze run-time environment.
Limit n <number> rows	Runs a profile based on the number of rows in the data object. When you choose to run a profile in the Hadoop validation environment, Spark engine collects samples from multiple partitions of the data object and pushes the samples to a single node to compute sample size. The Limit n sampling option supports Oracle, SQL Server, and DB2 databases. You cannot apply the Advanced filter with the Limit n sampling option. Supported on Spark run-time environment.
Random percentage	Runs a profile on a percentage of rows in the data object. Supported on Spark run-time environment.
Exclude approved data types and data domains from the data type and data domain inference in the subsequent profile run	Excludes the approved data type or data domain from data type and data domain inference from the next profile run.

Creating a Single Data Object Profile in Informatica Developer

You can create a single data object profile for one or more columns in a data object and store the profile object in the Model repository.

1. In the **Object Explorer** view, select the data object you want to profile.
2. Click **File** > **New** > **Profile** to open the profile wizard.
3. Select **Profile** and click **Next**.
4. Enter a name for the profile and verify the project location. If required, browse to a new location.

5. Optionally, enter a text description of the profile.
6. Verify that the name of the data object you selected appears in the **Data Objects** section.
7. Click **Next**.
8. Configure the profile operations that you want to perform. You can configure the following operations:
   - Column profiling
   - Primary key discovery
   - Functional dependency discovery
   - Data domain discovery

   Note: To enable a profile operation, select **Enabled as part of the "Run Profile" action** for that operation. Column profiling is enabled by default.
9. Review the options for your profile.

   You can edit the column selection for all profile types. Review the filter and sampling options for column profiles. You can review the inference options for primary key, functional dependency, and data domain discovery. You can also review data domain selection for data domain discovery.
10. Review the drill-down options, and edit them if necessary. By default, the **Enable Row Drilldown** option is selected. You can edit drill-down options for column profiles. The options also determine whether drill-down operations read from the data source or from staged data, and whether the profile stores result data from previous profile runs.
11. In the **Run Settings** section, choose a validation environment. Choose **Native**, **Hadoop** or **Databricks** as the validation environment. You can choose **Native**, **Blaze** or **Spark** as the run-time environment. After you choose **Blaze** or **Spark**, you can select a Hadoop connection. If you choose **Databricks**, you can select a Databricks connection.
12. Click **Finish**.

Creating an Enterprise Discovery Profile in Informatica Developer

You can create a profile on multiple data sources under multiple connections. The Developer tool creates individual profile tasks for each source.

1. In the **Object Explorer** view, select multiple data objects you want to run a profile on.
2. Click **File > New > Profile** to open the profile wizard.
3. Select **Enterprise Discovery Profile** and click **Next**.
4. Enter a name for the profile and verify the project location. If required, browse to a new location.
5. Verify that the name of the data objects you selected appears within the **Data Objects** section. Click **Choose** to select more data objects, if required.
6. Click **Next**.

   The **Add Resources to Profile Definition** pane appears. You can select multiple, external relational connections and data sources from this pane.
7. Click **Choose** to open the **Select Resources** dialog box.

   The **Resources** pane lists all the internal and external connections and data objects under the Informatica domain.

8. Click **OK** to close the dialog box.
9. Click **Next**.
10. Configure the profile types that you want to run. You can configure the following profile types:
    - Data domain discovery
    - Column profile
    - Primary key profile
    - Foreign key profile

    Note: Select **Enabled as part of "Run Enterprise Discovery Profile" action** for the profile types that you want to run as part of the enterprise discovery profile. Column profiling is enabled by default.
11. Review the options for the profile.

    You can edit the sampling options for column profiles. You can also edit the inference options for data domain, primary key, and foreign key profiles.
12. Select **Create profiles**.

    The Developer tool creates profiles for each individual data source.
13. Select **Run enterprise discovery profile on finish** to run the profile when you complete the profile configuration. If you enabled all the profiling operations, the Developer tool runs column, data domain, and primary key profiles on all selected data sources. Then, the Developer tool runs a foreign key profile across all the data sources.
14. Click **Finish**.

    After you run an enterprise discovery profile, you need to refresh the Model Repository Service before viewing the results. This step is required as the import of metadata for external connections happens in the Model repository. You need to refresh the Model Repository Service so that the Developer tool reflects the changes to the Model repository.

Creating a Column Profile in Informatica Analyst

You can create a custom profile or default profile. When you create a custom profile, you can configure the columns, sample rows, and drill-down options. When you create a default profile, the column profile and data domain discovery runs on the entire data set with all the data domains.

1. In the **Discovery** workspace, click **Profile**, or select **New** > **Profile** from the header area.

   Note: You can right-click on the data object in the **Library** workspace and create a profile. In this profile, the profile name, location name, and data object are extracted from the data object properties. You can create a default profile or customize the settings to create a custom profile.

   The **New Profile** wizard appears.
2. The **Single source** option is selected by default. Click **Next**.
3. In the **Specify General Properties** screen, enter a name and an optional description for the profile. In the Location field, select the project or folder where you want to create the profile. Click **Next**.
4. In the **Select Source** screen, click **Choose** to select a data object, or click **New** to import a data object. Click **Next**.
   - In the **Choose Data Object** dialog box, select a data object. Click **OK**.

The Properties pane displays the properties of the selected data object. The Data Preview pane displays the columns in the data object.

- In the **New Data Object** dialog box, you can choose a connection, schema, table, or view to create a profile on, select a location, and create a folder to import the data object. Click **OK**.

5. In the **Select Source** screen, select the columns that you want to run a profile on. Optionally, select **Name** to select all the columns. Click **Next**.

   All the columns are selected by default. The Analyst tool lists column properties, such as the name, data type, precision, scale, nullable, and participates in the primary key for each column.

6. In the **Specify Settings** screen, choose to run a column profile, data domain discovery, or a column profile and data domain discovery. By default, column profile option is selected.
   - Choose **Run column profile** to run a column profile.
   - Choose **Run data domain discovery** to perform data domain discovery. In the **Data domain** pane, select the data domains that you want to discover, select a conformance criteria, and select the columns for data domain discovery in the **Edit columns selection for data domin discovery** dialog box.
   - Choose **Run column profile** and **Run data domain discovery** to run the column profile and data domain discovery. Select the data domain options in the **Data domain** pane.

     Note: By default, the columns that you select is for column profile and data domain discovery. Click **Edit** to select or deselect columns for data domain discovery.
   - Choose Data, Columns, or Data and Columns to run data domain discovery on.
   - Choose a sampling option. You can choose **All rows (complete analysis)**, **Sample first**, **Random sample**, **Random sample (auto)**, **Limit n**, or **Random percentage** as a sampling option in the **Run profile on** pane. The sampling option applies to column profile and data domain discovery.
   - Choose a drilldown option. You can choose **Live** or **Staged** drilldown option, or you can choose **Off** to disable drilldown in the **Drilldown** pane. Optionally, click **Select Columns** to select columns to drill down on. You can choose to omit data type and data domain inference for columns with an approved data type or data domain.
   - Choose **Native**, **Blaze**, **Spark**, or **Databricks** as the run-time environment. If you choose **Blaze** or **Spark**, click **Choose** to select a Hadoop connection in the **Select a Hadoop Connection** dialog box. If you choose **Databricks**, click **Choose** to select a Databricks connection.
7. Click **Next**.

   The **Specify Rules and Filters** screen opens.

8. In the **Specify Rules and Filters** screen, you can perform the following tasks:
   - Create, edit, or delete a rule. You can apply existing rules to the profile.
   - Create, edit, or delete a filter.

     Note: When you create a scorecard on this profile, you can reuse the filters that you create for the profile.
9. Click **Save and Finish** to create the profile, or click **Save and Run** to create and run the profile.

Creating an Enterprise Discovery Profile in Informatica Analyst

You can run column profile and data domain discovery as part of enterprise discovery in Informatica Analyst.

1. In the **Discovery** workspace, select **New** > **Profile**.

   The **New Profile** wizard appears.

2. Select **Enterprise Discovery** . Click **Next**.

   The **Specify General Properties** tab appears.

3. In the **Specify General Properties** tab, enter a name for the enterprise discovery profile and an optional description. In the Location field, select the project or folder where you want to create the profile. Click **Next**.

   The **Select Data Objects** tab appears.

4. In the **Select Data Objects** tab, click **Choose**.

   The **Choose Data objects** dialog box appears.

5. In the **Choose Data objects** dialog box, choose one or more data objects to add to the profile. Click **Save**.

   The data objects appear in the **Data Objects** pane.

6. Click **Next**.

   The **Select Resources** tab appears.

7. In the **Select Resources** tab, click **Choose** to open the **Select Resources** tab.

   You can import data from multiple relational data sources.

8. In the **Select Resources** tab, select the connections, schemas, tables, and views that you want to include in the profile. Click **Save**.

   The left pane in the dialog box lists all the internal and external connections, schemas, tables, and views under the Informatica domain.

   The resources appear in the **Resource** pane.

9. Click **Next**.

   The **Specify Settings** tab appears.

10. In the **Specify Settings** tab, you can configure the column profile options and data domain discovery options. Click **Save and Finish** to save the enterprise discovery profile, or click **Save and Run** to run the profile.

    You can perform the following tasks in the **Specify Settings** tab.

    - Enable data domain discovery. Click **Choose** to select data domains that you want to discover from the **Choose Data Domains** dialog box. The selected data domains appear in the **Data Domains for Data Domain Discovery** pane.
    - Run data domain on data, column name, or on both data and column name.
    - Select all the rows in the data source, or choose a maximum number of rows to run domain discovery on.
    - Choose a minimum conformance percentage or specify the minimum number of conforming rows for data domain discovery.
    - Enable column profile settings and select all rows or first few rows in the data source for the column profile. You can exclude data type inference for columns with approved data types in the column profile.

- Choose **Native**, **Blaze**, **Spark**, or **Databricks** as the run-time environment. After you choose **Blaze**, or **Spark**, select a Hadoop connection to run profiles. If you choose **Databricks**, select a Databricks connection.

You can view the enterprise discovery results under the **Summary** and **Profiles** tabs.

Creating a Scorecard in Informatica Analyst

Create a scorecard and add columns from a profile to the scorecard. You must run a profile before you add columns to the scorecard.

1. In the **Library** workspace, select the project or folder that contains the profile.
2. Click the profile to open the profile.

   The profile results appear in the summary view in the **Discovery** workspace.
3. Click **Actions** > **Add to scorecard**.

   The **Add to Scorecard** wizard appears.
4. In the **Add to Scorecard** screen, you can choose to create a new scorecard, or edit an existing scorecard to add the columns to a predefined scorecard. The **New Scorecard** option is selected by default. Click **Next**.
5. In the **Step 2 of 8** screen, enter a name for the scorecard. Optionally, you can enter a description for the scorecard. Select the project and folder where you want to save the scorecard. Click **Next**.

   By default, the scorecard wizard selects the columns and rules defined in the profile. You cannot add columns that are not included in the profile.
6. In the **Step 3 of 8** screen, select the columns and rules that you want to add to the scorecard as metrics. Optionally, click the check box in the left column header to select all columns. Optionally, select **Column Name** to sort column names. Click **Next**.
7. In the **Step 4 of 8** screen, you can add a filter to the metric.

   You can apply the filter that you created for the profile to the metrics, or create a new filter. Select a metric in the **Metric Filters** pane, and click the **Manage Filters** icon to open the **Edit Filter: column name** dialog box. In the **Edit Filter: column name** dialog box, you can choose to perform one of the following tasks:

   - Choose a filter that you created for the profile. Click **Next**.
   - Select an existing filter. Click the edit icon to edit the filter in the **Edit Filter** dialog box. Click **Next**.
   - Click the plus (+) icon to create filters in the **New Filter** dialog box. Click **Next**.

   Optionally, you can choose to apply the selected filters to all the metrics in the scorecard.

   The filter appears in the **Metric Filters** pane.
8. In the **Step 4 of 8** screen, click **Next**.
9. In the **Step 5 of 8** screen, select each metric in the **Metrics** pane to perform the following tasks:

   - Configure valid values. In the **Score using: Values** pane, select one or more values in the **Available Values** pane, and click the right arrow button to move them to the **Valid Values** pane. The total number of valid values for a metric appears at the top of the **Available Values** pane.
   - Configure metric thresholds. In the **Metric Thresholds** pane, set the thresholds for **Good**, **Acceptable**, and **Unacceptable** scores.

- Configure the cost of invalid data. To assign a constant value to the cost for the metric, select **Fixed Cost**. To attach a numeric column as a variable cost to the metric, select **Variable Cost**, and click **Select Column** to select a numeric column. Optionally, click **Change Cost Unit** to change the unit of cost. If you do not want to configure the cost of invalid data for the metric, choose **None**.

10. Click **Next**.
11. In the **Step 6 of 8** screen, you can select a metric group to which you can add the metrics, or create a new metric group, To create a new metric group, click the group icon. Click **Next**.
12. In the **Step 7 of 8** screen, specify the weights for the metrics in the group and thresholds for the group.
13. In the **Step 8 of 8** screen, select **Native**, **Hadoop**, or **Databricks** as a run-time environment to run the scorecard. You can choose a **Blaze** or **Spark** engine in the Hadoop run-time environment. If you choose the **Databricks**, **Browse** to choose a Databricks connection to run the profile. click If you choose the **Blaze** or **Spark** engine, click **Browse** to choose a Hadoop connection to run the profile.
14. Click **Save** to save the scorecard, or click **Save & Run** to save and run the scorecard.

    The scorecard appears in the **Scorecard** workspace.

Monitoring a Profile

You can monitor a profile in the Administrator tool.

1. In the Administrator tool, click the **Monitor** tab.
2. In the Navigator workspace, select **Jobs**.
3. Select a profiling job.
4. In the **Summary Statistics** tab, you can view the general properties of the profile, summary statistics, and detailed statistics of the profile.
5. Click the **Execution Statistics** tab to view execution statistics for the profile.

Profiling Functionality Support

You can connect to the profiling warehouse through JDBC or native connectivity. You can perform specific functionality in profiling based on the profiling warehouse connection.

The following table lists the functionality that you can perform based on the type of profiling warehouse connection that you choose in Data Engineering Quality and Data Engineering Integration:

Functionality	JDBC Connection	Native Connection
Single data object column profile	Supported	Supported
Single data object data domain discovery	Supported	Supported
Single data object profile with primary key discovery	Not supported	Not supported

Functionality	JDBC Connection	Native Connection
Single data object profile with functional dependency discovery	Not supported	Not supported
Profile with *Sample first <number> rows* sampling	Supported	Supported
Profile with *Random sample <number> rows* sampling	Supported	Supported
Scorecard metrics and value cost	Not supported	Supported
Enterprise discovery profile with primary key and foreign key discovery	Not supported	Not supported
Enterprise discovery profile with join analysis and overlap discovery	Not supported	Not supported
Drill down on inferred values and value frequencies	Supported	Supported
Export profile results	Supported	Supported
Row count mapping*	Not supported	Supported
*Profiling runs Row Count Mapping when the stats helper fails to report the row count.		

Troubleshooting

Can I drill down on profile results if I run a profile in the Hadoop environment?

Yes, except for profiles in which you have set the option to drill down on staged data.

I get the following error message when I run a profile in the Hadoop environment: "[LDTM_1055] The Integration Service failed to generate a Hive workflow for mapping [Profile_CUSTOMER_INFO12_14258652520457390]." How do I resolve this?

This error can result from a data source, rule transformation, or run-time environment that is not supported in the Hadoop environment. For more information about objects that are not valid in the Hadoop environment, see the Mappings in a Hadoop Environment chapter.

You can change the data source, rule, or run-time environment and run the profile again. View the profile log file for more information on the error.

I see "N/A" in the profile results for all columns after I run a profile. How do I resolve this?

Verify that the profiling results are in the profiling warehouse. If you do not see the profile results, verify that the database path is accurate in the Cluster Environment Variables property of the Hadoop connection. You can also verify the database path from the Hadoop job tracker on the Monitoring tab of the Administrator tool.

After I run a profile on a Hive source, I do not see the results. When I verify the Hadoop job tracker, I see the following error when I open the profile job: "XML Parsing Error: no element found." What does this mean?

The Hive data source does not have any record and is empty. The data source must have a minimum of one row of data for successful profile run.

After I run a profile on a Hive source, I cannot view some of the column patterns. Why?

When you import a Hive source, the Developer tool sets the precision for string columns to 4000. The Developer tool cannot derive the pattern for a string column with a precision greater than 255. To resolve this issue, set the precision of these string columns in the data source to 255 and run the profile again.

When I run a profile on large Hadoop sources, the profile job fails and I get an "execution failed" error. What can be the possible cause?

One of the causes can be a connection issue. Perform the following steps to identify and resolve the connection issue:

1. Open the Hadoop job tracker.
2. Identify the profile job and open it to view the MapReduce jobs.
3. Click the hyperlink for the failed job to view the error message. If the error message contains the text "java.net.ConnectException: Connection refused", the problem occured because of an issue with the Hadoop cluster. Contact your network administrator to resolve the issue.

CHAPTER 11

Monitoring

This chapter includes the following topics:

- Overview of Monitoring, 198
- Hadoop Environment Logs, 198
- Blaze Engine Monitoring, 204
- Spark Engine Monitoring, 212

Overview of Monitoring

On the Monitor tab of the Administrator tool, you can view statistics and log events for mappings run in the Hadoop environment. The Monitor tab displays current and historical information about mappings run on the Blaze and Spark engines.

Use the Summary Statistics view to view graphical summaries of object state and distribution across the Data Integration Services. You can also view graphs of the memory and CPU that the Data Integration Services used to run the objects.

When you run a mapping in the Hadoop environment, the Data Integration Service generates log events. You can view log events relating to different types of errors such as Hadoop connection failures, Hive query failures, or other Hadoop job failures.

With the REST Operations Hub, you can also get monitoring statistics for deployed mappings that run on the Data Integration Service or the Blaze or Spark engines.

Note: You cannot monitor the Databricks engine on the Monitoring tab.

Hadoop Environment Logs

The Data Integration Service generates log events when you run a mapping in the Hadoop environment.

You can view logs for the Blaze and Spark engines. You can view log events relating to different types of errors such as Hadoop connection or job failures.

When you run a mapping on the Spark engine, you can view the Scala code in logs that the Logical Data Translation Generator generates from the Informatica mapping.

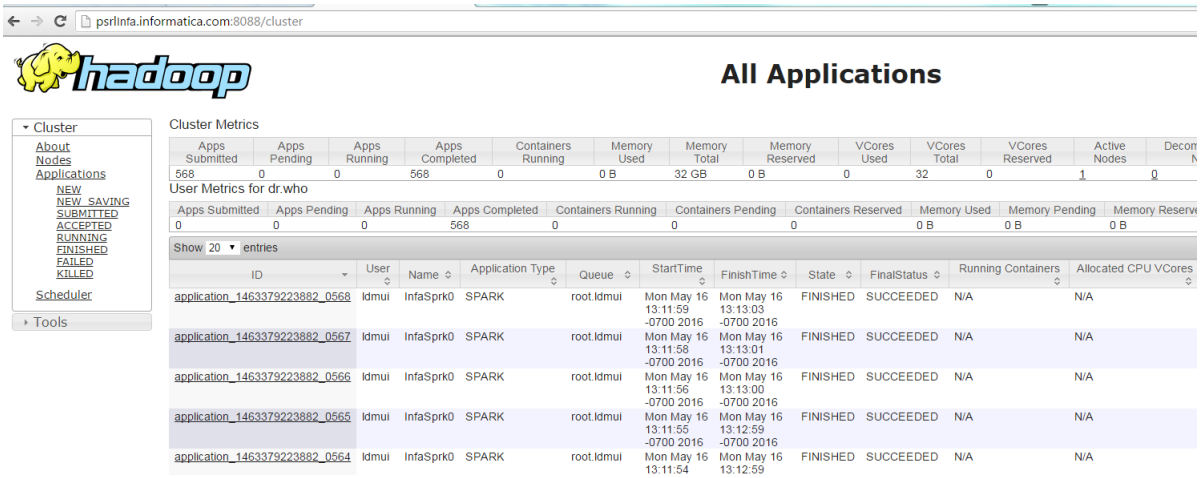
You can view reject files in the reject file directory specified for the Data Integration Service.

YARN Web User Interface

You can view the applications that ran on a cluster in the YARN web user interface. Click the Monitoring URL for Blaze or Spark jobs to access the YARN web user interface.

Blaze and Spark engines run on the Hadoop cluster that you configure in the Hadoop connection. The YARN web user interface shows each job that the engine runs as a YARN application.

The following image shows the Application Monitoring page of the YARN web user interface:



The **Application Type** indicates which engine submitted the YARN application.

The application ID is the unique identifier for the application. The application ID is a link to the application summary. The URL is the same as the Monitoring URL in the Administrator tool.

Click the **Logs** link in the application summary to view the application logs on the Hadoop cluster.

The amount of information in the application logs depends on the tracing level that you configure for a mapping in the Developer tool. The following table describes the amount of information that appears in the application logs for each tracing level:

Tracing Level	Messages
None	The log displays FATAL messages. FATAL messages include non-recoverable system failures that cause the service to shut down or become unavailable.
Terse	The log displays FATAL and ERROR code messages. ERROR messages include connection failures, failures to save or retrieve metadata, service errors.
Normal	The log displays FATAL, ERROR, and WARNING messages. WARNING errors include recoverable system failures or warnings.
Verbose initialization	The log displays FATAL, ERROR, WARNING, and INFO messages. INFO messages include system and service change messages.
Verbose data	The log displays FATAL, ERROR, WARNING, INFO, and DEBUG messages. DEBUG messages are user request logs.

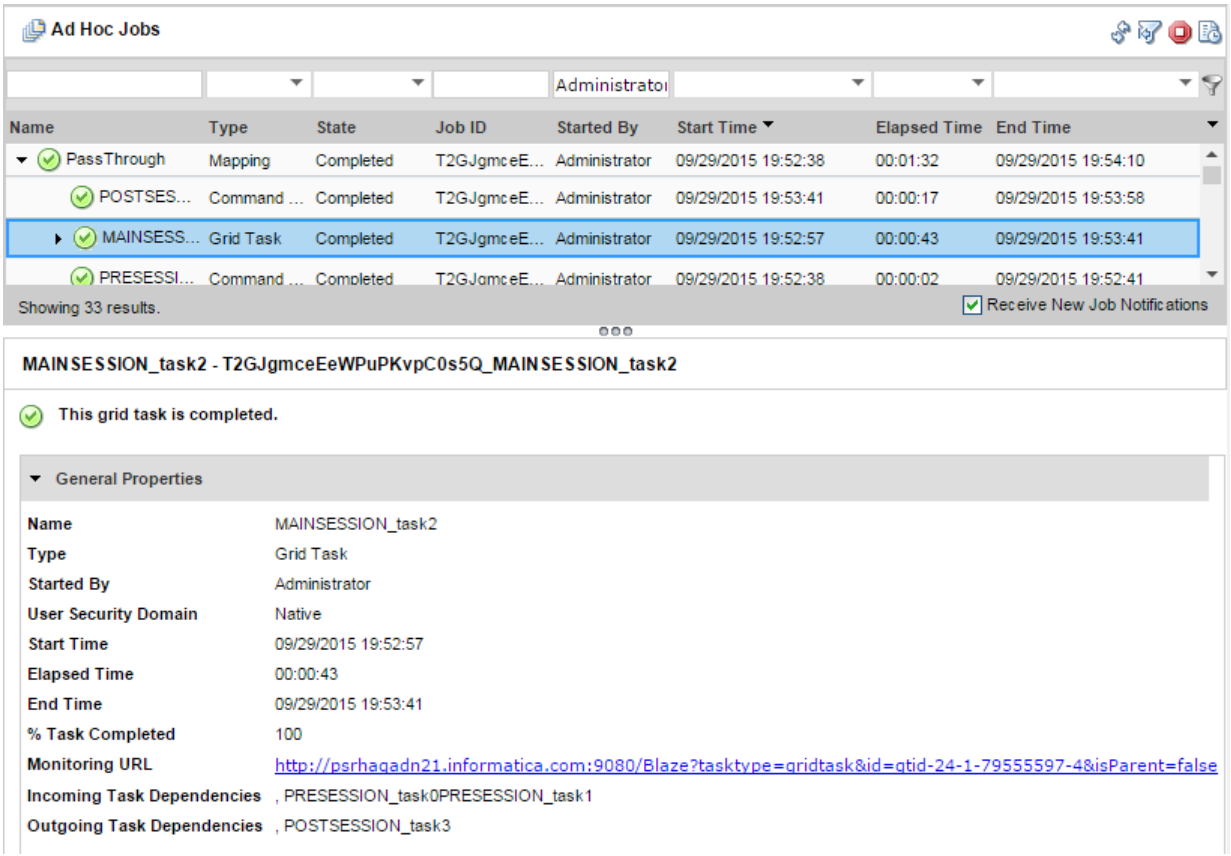
Accessing the Monitoring URL

The Monitoring URL opens the Blaze Job Monitor web application or the YARN web user interface. Access the Monitoring URL from the **Execution Statistics** view in the Administrator tool.

1. In the **Monitor** tab of the Administrator tool, click the **Execution Statistics** view.
2. Select **Ad Hoc Jobs** or select a deployed mapping job or workflow from an application in the Navigator.

   The list of jobs appears in the contents panel.
3. Select a mapping job and expand the mapping to select a grid task for the mapping.

   The Monitoring URL appears in the **Properties** view.



Viewing Hadoop Environment Logs in the Administrator Tool

You can view log events for a Blaze mapping from the Monitor tab of the Administrator tool.

1. In the Administrator tool, click the **Monitor** tab.
2. Select the **Execution Statistics** view.
3. In the Navigator, choose to open an ad hoc job, a deployed mapping job, or a workflow.
   - To choose an ad hoc job, expand a Data Integration Service and click **Ad Hoc Jobs**.
   - To choose a deployed mapping job, expand an application and click **Deployed Mapping Jobs**.
   - To choose a workflow, expand an application and click **Workflows**.

   The list of jobs appears in the contents panel.

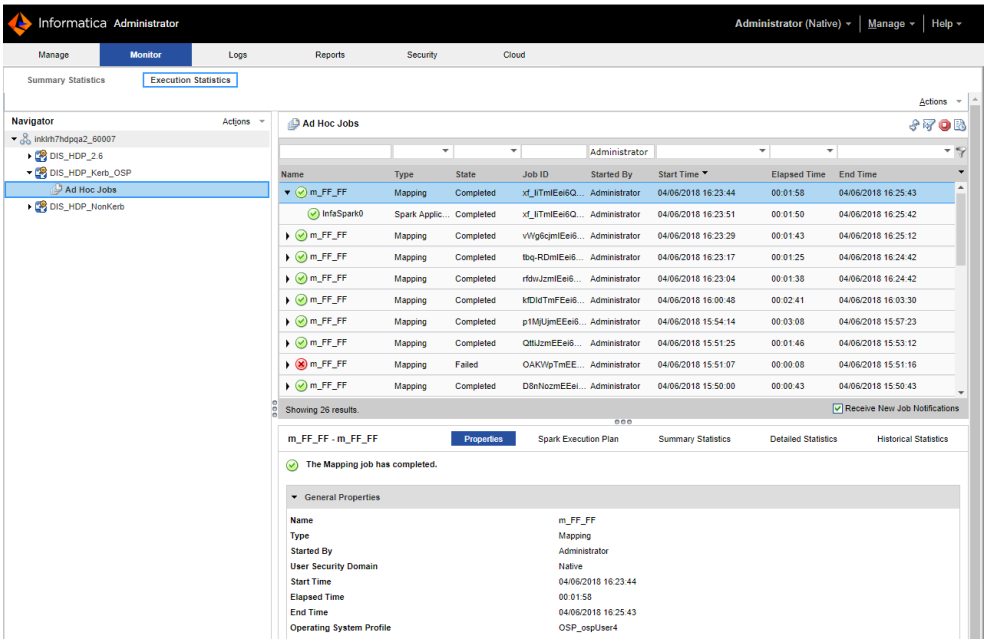
4. Click **Actions** > **View Logs for Selected Object** to view the run-time logs for the mapping.

   The log file shows the results of the Hive queries and Blaze engine queries run by the Data Integration Service. This includes the location of Hive session logs and Hive session history file.

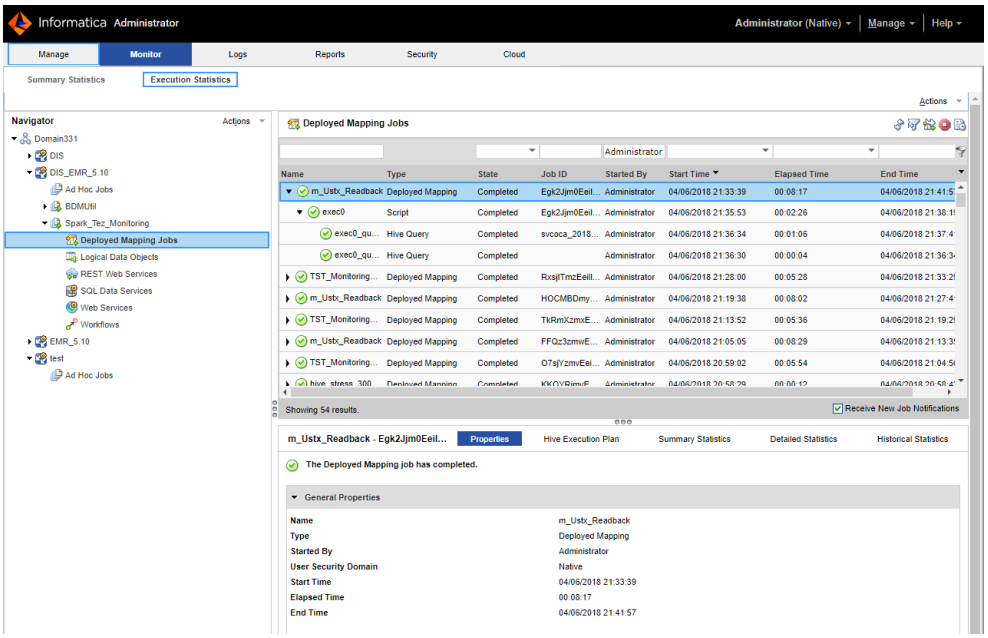
Monitoring a Mapping

You can monitor a mapping that runs in the Hadoop environment.

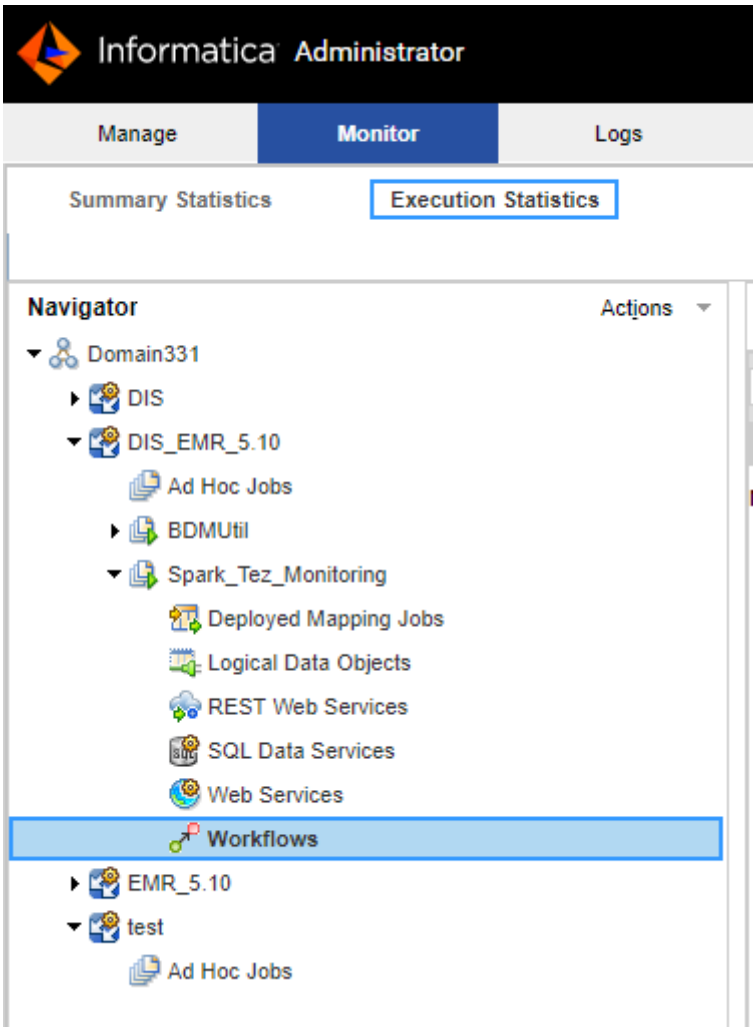
1. In the Administrator tool, click the **Monitor** tab.
2. Select the **Execution Statistics** view.
3. In the Navigator, choose to open an ad hoc job, a deployed mapping job, or a workflow.
   - To choose an ad hoc job, expand a Data Integration Service and click **Ad Hoc Jobs**.



- To choose a deployed mapping job, expand an application and click **Deployed Mapping Jobs**.



- To choose a workflow, expand an application and click **Workflows**.



The list of jobs appears in the contents panel.

4. Click a job to view its properties.

   The contents panel shows the default **Properties** view for the job. For a Blaze engine mapping, the Blaze engine monitoring URL appears in the general properties in the details panel. The monitoring URL is a link to the YARN web user interface for Spark jobs.

5. Choose a view in the contents panel to view more information about the job:
   - To view the execution plan for the mapping, select the **Execution Plan** view.
   - To view the summary statistics for a job, click the **Summary Statistics** view.
   - To view the detailed statistics for a job, click the **Detailed Statistics** view.

   Note: You can view the number of rows processed in the Summary Statistics for a Hive source or target. The remaining values do not appear for Hive sources and targets.

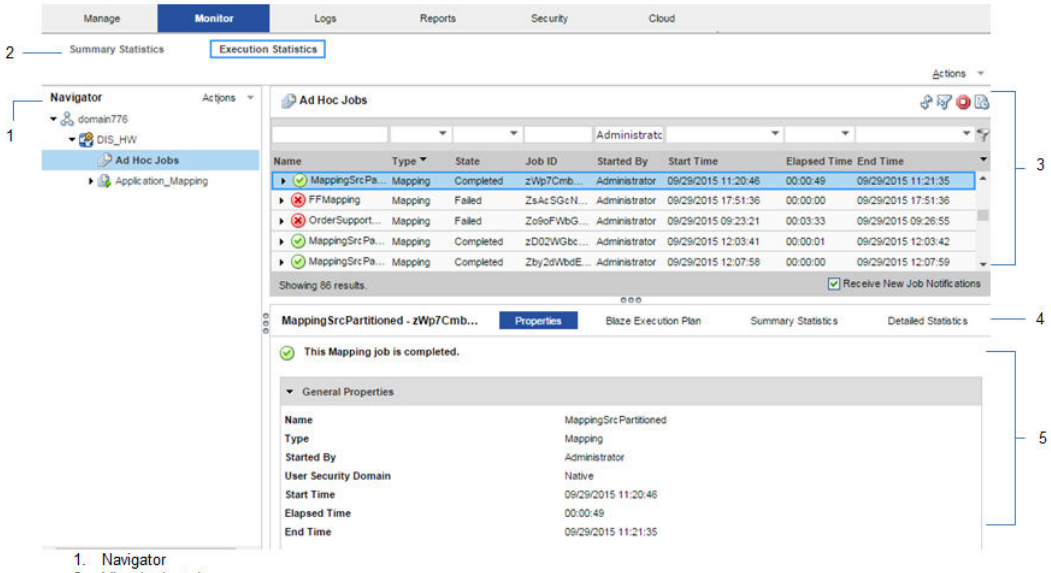
The following table describes the mapping job states in the Administrator tool contents panel:

Job Status	Rules and Guidelines
Queued	The job is in the queue.
Running	The Data Integration Service is running the job.
Completed	The job ran successfully.
Aborted	The job was flushed from the queue at restart or the node shut down unexpectedly while the job was running.
Failed	The job failed while running or the queue is full.
Canceled	The job was deleted from the queue or cancelled while running.
Unknown	The job status is unknown.

Blaze Engine Monitoring

You can monitor statistics and view log events for a Blaze engine mapping job in the Monitor tab of the Administrator tool. You can also monitor mapping jobs for the Blaze engine in the Blaze Job Monitor web application.

The following image shows the Monitor tab in the Administrator tool:



1. Navigator
2. View in the tab.
3. Contents panel
4. View in the contents panel.
5. Details panel

The Monitor tab has the following views:

Summary Statistics

Use the **Summary Statistics** view to view graphical summaries of object states and distribution across the Data Integration Services. You can also view graphs of the memory and CPU that the Data Integration Services used to run the objects.

Execution Statistics

Use the **Execution Statistics** view to monitor properties, run-time statistics, and run-time reports. In the Navigator, you can expand a Data Integration Service to monitor **Ad Hoc Jobs** or expand an application to monitor deployed mapping jobs or workflows

When you select **Ad Hoc Jobs**, deployed mapping jobs, or workflows from an application in the Navigator of the **Execution Statistics** view, a list of jobs appears in the contents panel. The contents panel displays jobs that are in the queued, running, completed, failed, aborted, and cancelled state. The Data Integration Service submits jobs in the queued state to the cluster when enough resources are available.

The contents panel groups related jobs based on the job type. You can expand a job type to view the related jobs under it.

Access the following views in the **Execution Statistics** view:

Properties

The **Properties** view shows the general properties about the selected job such as name, job type, user who started the job, and start time of the job. You can also monitor jobs on the Hadoop cluster from the Monitoring URL that appears for the mapping in the general properties. The Monitoring URL opens the Blaze Job Monitor in a web page. The Blaze Job Monitor displays detailed monitoring statistics for a mapping such as the number of grid tasks, grid segments, or tasklets, and recovery attempts for each tasklet.

Blaze Execution Plan

The Blaze execution plan displays the Blaze engine script that the Data Integration Service generates based on the mapping logic. The execution plan includes the tasks that the script depends on. Each script has a unique identifier.

Summary Statistics

The **Summary Statistics** view appears in the details panel when you select a mapping job in the contents panel. The **Summary Statistics** view displays throughput and resource usage statistics for the job.

You can view the following throughput statistics for the job:

- Source. The name of the mapping source file.
- Target name. The name of the target file.
- Rows. The number of rows read for source and target. If the target is Hive, this is the only summary statistic available.
- Average Rows/Sec. Average number of rows read per second for source and target.
- Bytes. Number of bytes read for source and target.
- Average Bytes/Sec. Average number of bytes read per second for source and target.
- First Row Accessed. The date and time when the Data Integration Service started reading the first row in the source file.
- Dropped rows. Number of source rows that the Data Integration Service did not read.

Detailed Statistics

The **Detailed Statistics** view appears in the details panel when you select a mapping job in the contents panel. The **Detailed Statistics** view displays graphs of the throughput and resource usage statistics for the job run.

Blaze Job Monitoring Application

Use the Blaze Job Monitor application to monitor Blaze engine jobs on the Hadoop cluster.

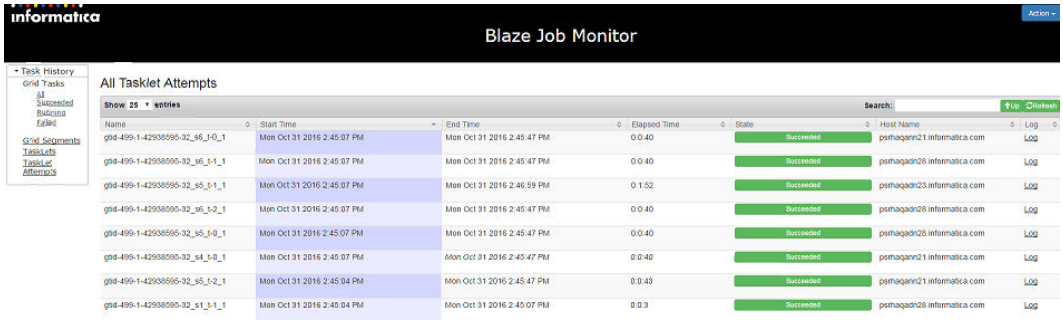
You configure the host that starts the Blaze Job Monitor in the Hadoop connection properties. You might want to configure the Blaze Job Monitor address to avoid conflicts with other users on the same cluster, or if you have access to a limited number of nodes. If you do not configure the Blaze Job Monitor address, the Grid Manager starts the host on the first alphabetical cluster node with a default port of 9080.

The Blaze engine monitoring URL appears in the Monitor tab of the Administrator tool when you view a Blaze engine mapping job. When you click the URL, the Blaze engine monitoring application opens in a web page.

Note: You can also access the Blaze Job Monitor through the LDTM log. After the session load summary, the log displays a list of segments within the grid task. Each segment contains a link to the Blaze Job Monitor. Click on a link to see the execution details of that segment.

You configure the host that starts the Blaze Job Monitor in the Hadoop connection properties. The default address is <hostname>:9080.

The following image shows the Blaze Job Monitor:



Use the **Task History** panel on the left to filter Blaze mapping jobs by the following criteria:

- Grid task. A parallel processing job request sent by the Blaze engine executor to the Grid Manager. You can further filter by all tasks, succeeded tasks, running tasks, or failed tasks.
- Grid segment. Part of a grid mapping that is contained in a grid task.
- Tasklet. A partition of a grid segment that runs on a separate DTM.
- Tasklet Attempts. The number of recovery attempts to restart a tasklet. Click **Log** to view the mapping grid task log.

The Blaze Job Monitor displays the task history for mapping jobs with the same namespace. You can monitor properties for a task such as start time, end time, elapsed time, or state of the task. You can also view log events. If you filter mapping jobs by grid segment, you can mouse over a grid segment to view the logical name of the segment.

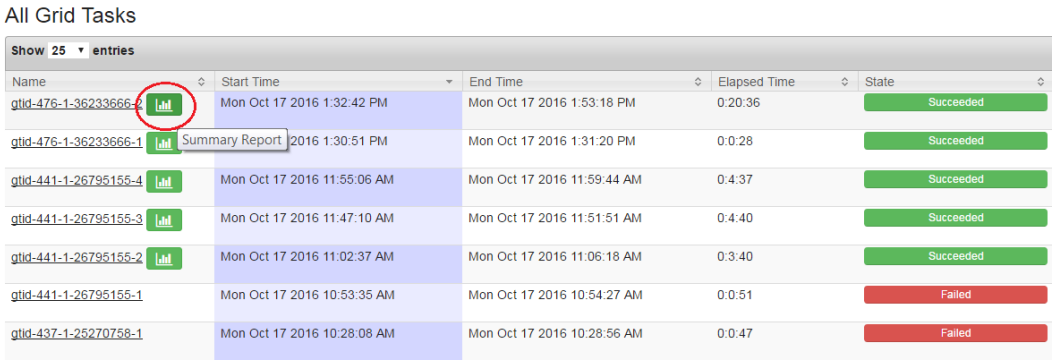
By default, the Blaze Job Monitor automatically refreshes the list of tasks every five seconds and reverts to the first page that displays tasks. Disable auto refresh if you want to browse through multiple pages. To turn off automatic refresh, click **Action** > **Disable Auto Refresh**.

The Blaze Job Monitor displays the first 100,000 grid tasks run in the past seven days. The Blaze Job Monitor displays the grid segments, tasklets, and tasklet attempts for grid tasks that are running and grid tasks that were accessed in the last 30 minutes.

Blaze Summary Report

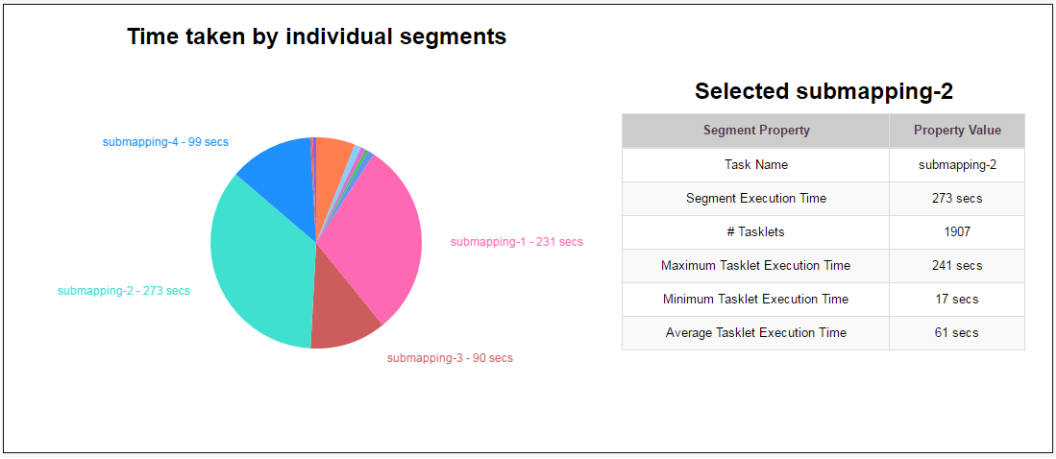
The Blaze Summary Report displays more detailed statistics about a mapping job. In the Blaze Job Monitor, a green summary report button appears beside the names of successful grid tasks. Click the button to open the Blaze Summary Report.

Note: The Blaze Summary Report is available for technical preview. Technical preview functionality is supported but is unwarranted and is not production-ready. Informatica recommends that you use in non-production environments only.



Time Taken by Individual Segments

A pie chart visually represents the time taken by individual segments contained within a grid task.



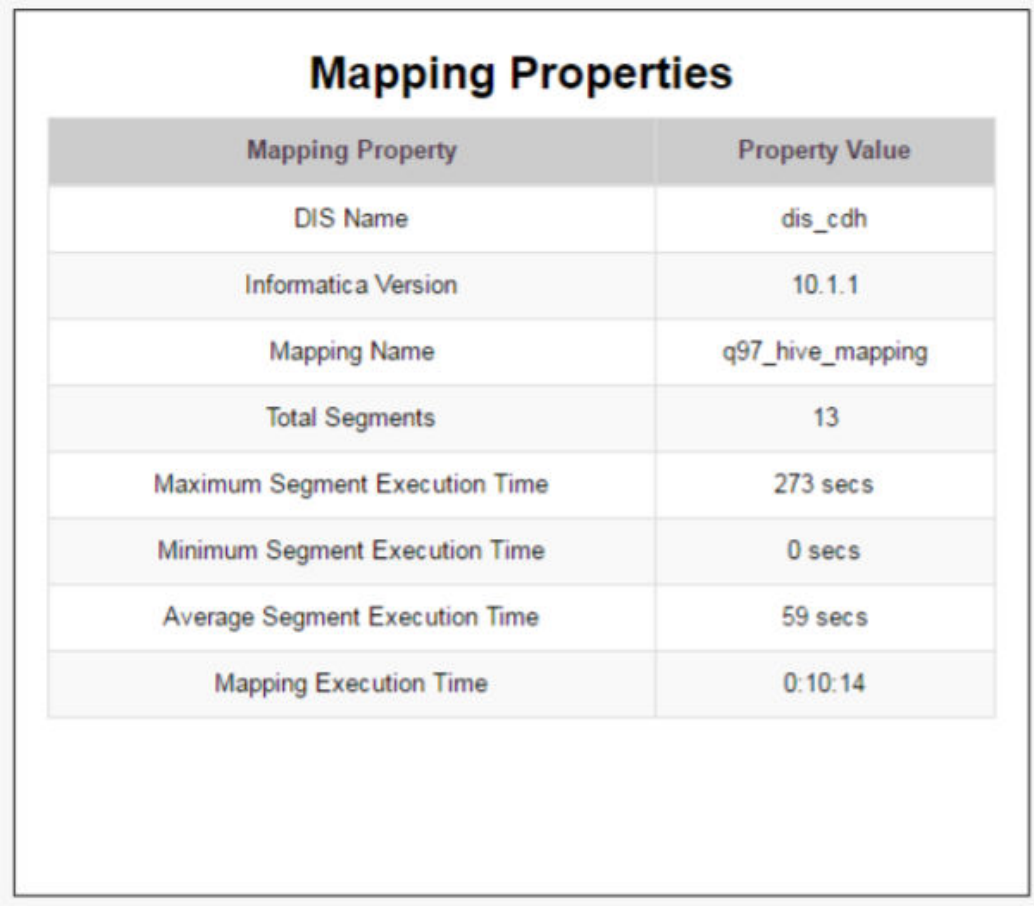
When you click on a particular segment in the pie chart, the **Selected Submapping** table displays detailed information about that segment. The table lists the following segment statistics:

- Task Name. The logical name of the selected segment.
- Segment Execution Time. The time taken by the selected segment.
- # Tasklets. The number of tasklets in the selected segment.
- Minimum Tasklet Execution Time. The execution time of the tasklet within the selected segment that took the shortest time to run.

- Maximum Tasklet Execution Time. The execution time of the tasklet within the selected segment that took the longest time to run.
- Average Tasklet Execution Time. The average execution time of all tasklets within the selected segment.

Mapping Properties

The Mapping Properties table lists basic information about the mapping job.



Mapping Properties

Mapping Property	Property Value
DIS Name	dis_cdh
Informatica Version	10.1.1
Mapping Name	q97_hive_mapping
Total Segments	13
Maximum Segment Execution Time	273 secs
Minimum Segment Execution Time	0 secs
Average Segment Execution Time	59 secs
Mapping Execution Time	0:10:14

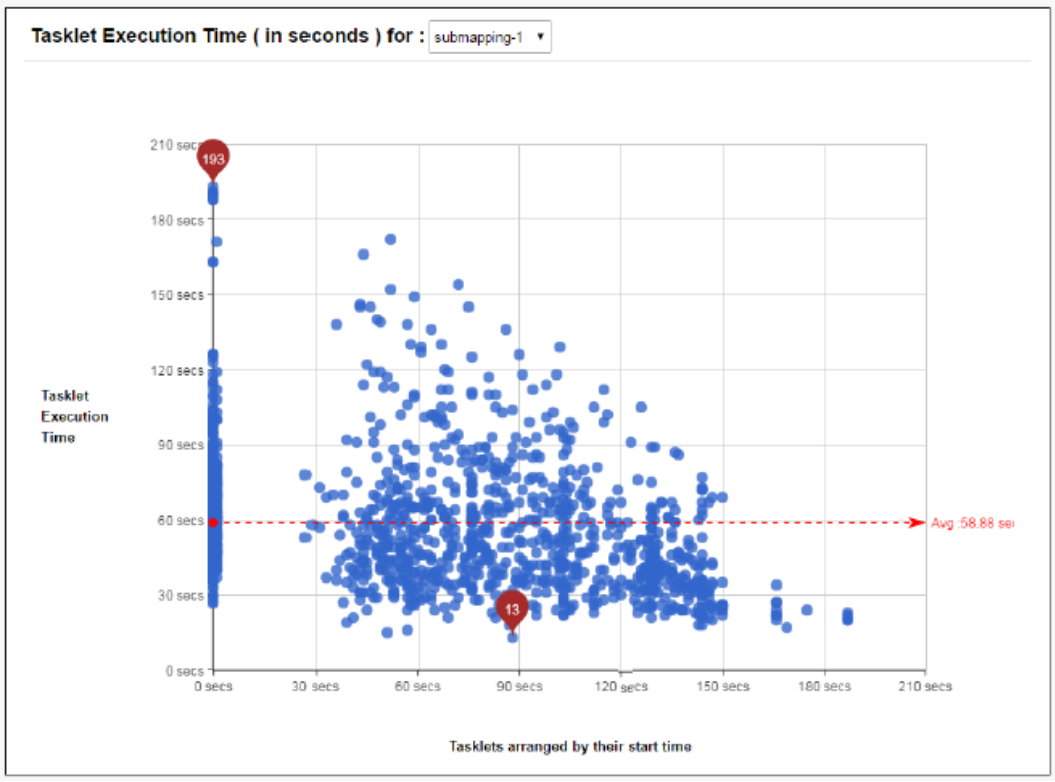
The Mapping Properties table displays the following information:

- The DIS name under which the mapping was run.
- The Informatica version.
- The name of the mapping.
- The total number of segments for the mapping.
- The execution time of the segment that took the longest time to run.
- The execution time of the segment that took the shortest time to run.
- The average execution time of all segments.
- The total mapping execution time.

Tasklet Execution Time

A time series graph displays the execution time of all tasklets within the selected segment.

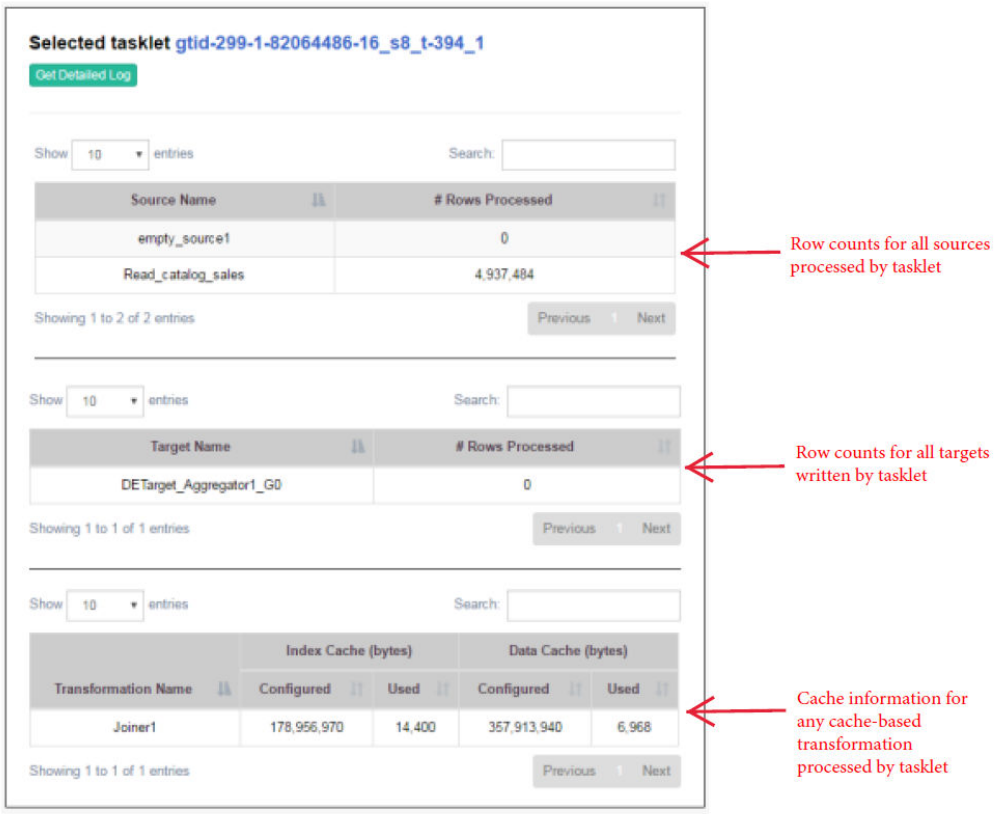
The x-axis represents the tasklet start time and the y-axis represents the actual tasklet execution time. The red dashed line represents the average execution time for all tasklets, and the two red markers show the minimum and maximum execution times within the segment.



Selected Tasklet Information

When you select a tasklet from the **Tasklet Execution Time** graph, you can see more data about that individual tasklet. This data includes source and target row counts as well as cache information for any

cache-based transformation processed by the tasklet. Click the **Get Detailed Log** button to see a full log of the selected tasklet.



Blaze Engine Logs

You can find information about the mapping run on the Blaze engine in the LDTM log file.

The LDTM logs the results of the mapping run on the Blaze engine. You can view the LDTM log from the Developer tool or the Monitoring tool for a mapping job.

You can configure the Data Integration Service to log details about the mapping execution to the session log. To enable logging of LDTM mapping execution details, set the log tracing level to verbose initialization or verbose data. For optimal logging performance, avoid setting the tracing level to verbose data. You can set the tracing level to verbose data if you need to debug a mapping run.

Logical mapping execution details include the following information:

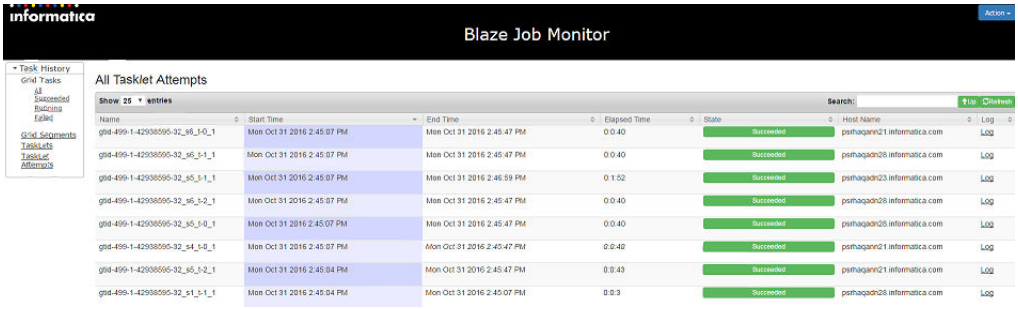
- Start time, end time, and state of each task
- Blaze Job Monitor URL
- Number of total, succeeded, and failed/cancelled tasklets
- Number of processed and rejected rows for sources and targets
- Data errors, if any, for transformations in each executed segment
- Number of healthy cluster nodes used by the Grid Manager

The mapping execution details include information about the Blaze engine run-time processing, such as the partition count for each executed segment.

Viewing Blaze Logs

You can view logs for a Blaze mapping from the Blaze Job Monitor.

1. In the Blaze Job Monitor, select a job from the list of jobs.
2. In the row for the selected job, click the **Logs** link.



The log events appear in another browser window.

Orchestrator Sunset Time

Orchestrator sunset time is the maximum lifetime for an Orchestrator service. Sunset time determines the maximum amount of time that the Blaze engine can run a mapping job. The default sunset time is 24 hours. After 24 hours, the Orchestrator shuts down, which causes the Blaze Grid Manager to shut down.

You can configure the Orchestrator sunset time to be greater than or less than 24 hours. Configure the following property in the Hadoop connection:

Property	Description
`infagrid.orchestrator.svc.sunset.time`	Maximum lifetime for an Orchestrator service, in hours. Default is 24 hours.

You can also disable sunset by setting the property to 0 or a negative value. If you disable sunset, the Orchestrator never shuts down during a mapping run.

Troubleshooting Blaze Monitoring

When I run a mapping on the Blaze engine and try to view the grid task log, the Blaze Job Monitor does not fetch the full log.

The grid task log might be too large. The Blaze Job Monitor can only fetch up to 2 MB of an aggregated log. The first line of the log reports this information and provides the location of the full log on HDFS. Follow the link to HDFS and search for "aggregated logs for grid mapping." The link to the full log contains the grid task number.

The Blaze Job Monitor will not start.

Check the Hadoop environment logs to locate the issue. If you do not find an issue, stop the Grid Manager with the infacmd stopBlazeService command and run the mapping again.

The Monitoring URL does not appear in the Properties view of the Administrator tool.

Locate the URL in the YARN log.

When Blaze processes stop unexpectedly, Blaze does not save logs in the expected location.

When Blaze stops unexpectedly, you can access Blaze service logs through the YARN monitor. Use one of these methods:

- The Grid Manager log contains all Blaze job container IDs and identifies the host on which Blaze ran. Alter the Grid Manager log URL with the container ID and host name of the Blaze host.
- Run the command yarn logs -applicationID <Blaze Grid Manager Application ID>.

A Blaze Job Monitor that has been running for several days loses its connection to the Application Timeline Server on the Hortonworks cluster.

The Blaze engine requires a running Application Timeline Server on the cluster. When the Blaze engine starts a mapping run, the Blaze Job Monitor checks the state of the Application Timeline Server. The Grid Manager will start it if it is not running. When the connection to the Application Timeline Server is lost, the Blaze engine attempts to reconnect to it. If the Application Timeline Server stops during a Blaze mapping run, you can restart it by restarting the Grid Manager.

Note: When the Application Timeline Server is configured to run on the cluster by default, the cluster administrator must manually restart it on the cluster.

When a mapping takes more than 24 hours to execute, the mapping fails.

When mappings run on the Blaze engine for more than 24 hours, some mappings might fail because the Orchestrator service has a default sunset time of 24 hours. After 24 hours, the Orchestrator shuts down, which causes the Blaze Grid Manager to shut down.

To increase the sunset time to be more than 24 hours, configure the following property in the Hadoop connection advanced properties:

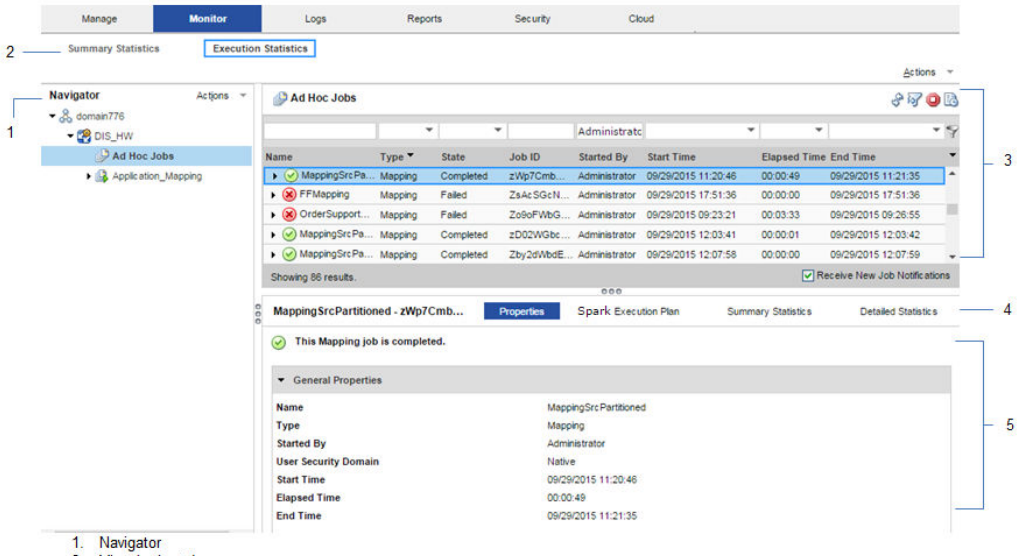
```
infagrid.orchestrator.svc.sunset.time=[HOURS]
```

You can also disable sunset by setting the property to 0 or a negative value. If you disable sunset, the Blaze Grid Manager never shuts down.

Spark Engine Monitoring

You can monitor statistics and view log events for a Spark engine mapping job in the Monitor tab of the Administrator tool. You can also monitor mapping jobs for the Spark engine in the YARN web user interface.

The following image shows the Monitor tab in the Administrator tool:



1. Navigator
2. View in the tab.
3. Contents panel
4. View in the contents panel.
5. Details panel

The Monitor tab has the following views:

Summary Statistics

Use the **Summary Statistics** view to view graphical summaries of object states and distribution across the Data Integration Services. You can also view graphs of the memory and CPU that the Data Integration Services used to run the objects.

Execution Statistics

Use the **Execution Statistics** view to monitor properties, run-time statistics, and run-time reports. In the Navigator, you can expand a Data Integration Service to monitor **Ad Hoc Jobs** or expand an application to monitor deployed mapping jobs or workflows

When you select **Ad Hoc Jobs**, deployed mapping jobs, or workflows from an application in the Navigator of the **Execution Statistics** view, a list of jobs appears in the contents panel. The contents panel displays jobs that are in the queued, running, completed, failed, aborted, and cancelled state. The Data Integration Service submits jobs in the queued state to the cluster when enough resources are available.

The contents panel groups related jobs based on the job type. You can expand a job type to view the related jobs under it.

Access the following views in the **Execution Statistics** view:

Properties

The **Properties** view shows the general properties about the selected job such as name, job type, user who started the job, and start time of the job.

Spark Execution Plan

When you view the Spark execution plan for a mapping, the Data Integration Service translates the mapping to a Scala program and an optional set of commands. The execution plan shows the commands and the Scala program code.

Summary Statistics

The **Summary Statistics** view appears in the details panel when you select a mapping job in the contents panel. The **Summary Statistics** view displays the following throughput statistics for the job:

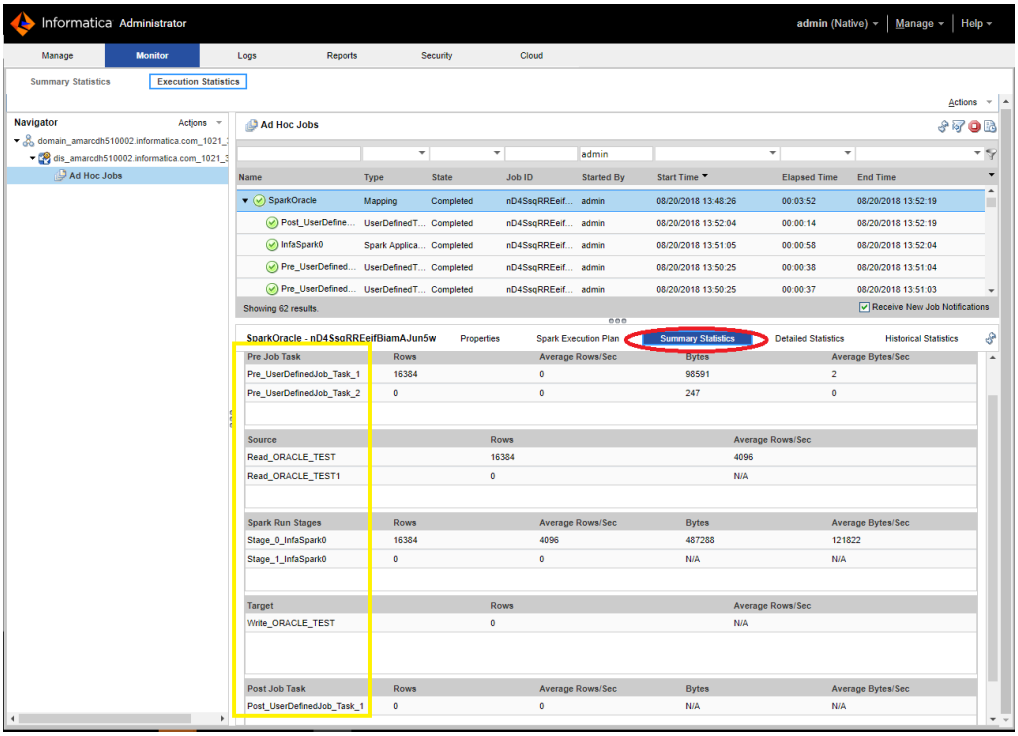
- Pre Job Task. The name of each job task that reads source data and stages the row data to a temporary table before the Spark job runs. You can also view the bytes and average bytes processed for each second.

  Note: If you enable recovery for a Sqoop mapping, the pre job task statistics does not display.
- Source. The name of the source.
- Target. The name of the target.
- Rows. For source, the number of rows read by the Spark application. For target, the sum of rows written to the target and rejected rows.
- Post Job Task. The name of each job task that writes target data from the staged tables. You can also view the bytes and average bytes processed for each second.

When a mapping contains a Union transformation with multiple upstream sources, the sources appear in a comma separated list in a single row under Sources.

Note: In a Hive mapping with an Update Strategy transformation containing the DD_UPDATE condition, the target contains only the temporary tables after the Spark job runs. The result of the mapping job statistics appears in the post job task and indicates twice the number of records updated.

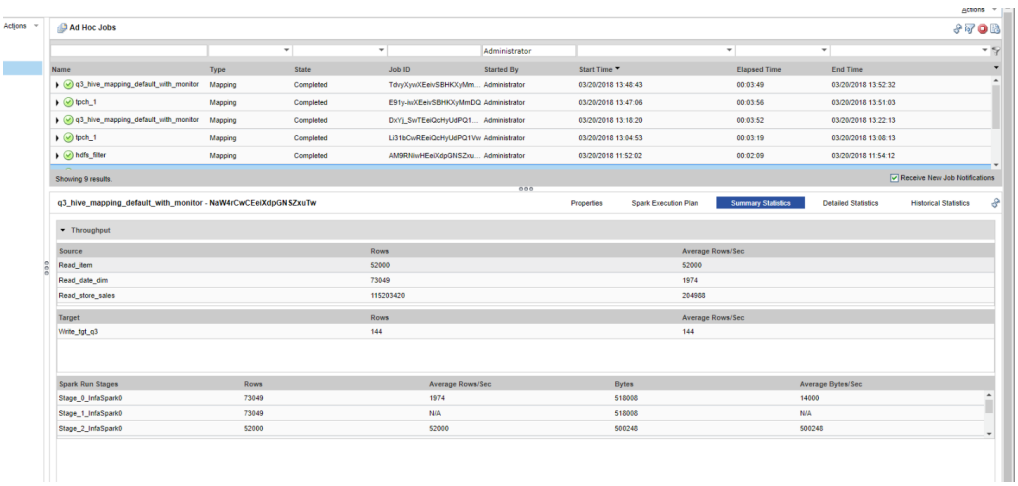
The following image shows the **Summary Statistics** view in the details panel for a mapping run on the Spark engine:



You can also view the Spark run stages information in the details pane of the Summary Statistics view on the Execution Statistics Monitor tab. It appears as a list after the sources and before the targets.

The **Spark Run Stages** displays the absolute counts and throughput of rows and bytes related to the Spark application stage statistics. Rows refer to the number of rows that the stage writes, and bytes refer to the bytes broadcasted in the stage.

The following image displays the Spark Run Stages:



For example, the Spark Run Stages column contains the Spark application staged information starting with `stage_<ID>`. In the example, `Stage_0` shows the statistics related to the Spark run stage with `ID=0` in the Spark application.

Consider when the Spark engine reads source data that includes a self-join with verbose data enabled. In this scenario, the optimized mapping from the Spark application does not contain any information on the second instance of the same source in the Spark engine logs.

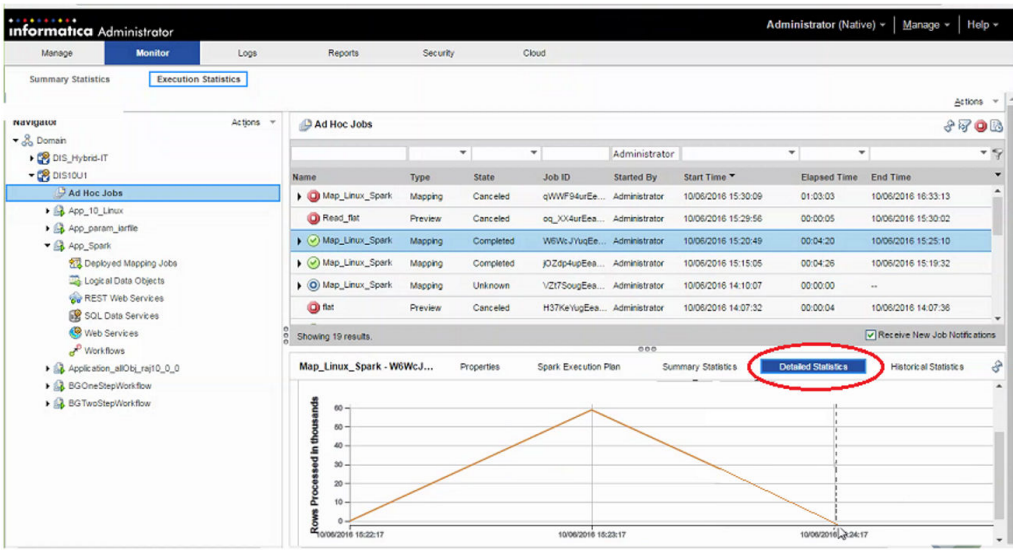
Consider when you read data from the temporary table and the Hive query of the customized data object leads to the shuffling of the data. In this scenario, the filtered source statistics appear instead of reading from the temporary source table in the Spark engine log.

When you run a mapping with Spark monitoring enabled, performance varies based on the mapping complexity. It can take up to three times longer than usual processing time with monitoring enabled. By default, monitoring is disabled.

Detailed Statistics

The **Detailed Statistics** view appears in the details panel when you select a mapping job in the contents panel. The **Detailed Statistics** view displays a graph of the row count for the job run.

The following image shows the **Detailed Statistics** view in the details panel for a mapping run on the Spark engine:



Viewing Hive Tasks

When you have a Hive source with a transactional table, you can view the Hive task associated with the Spark job.

When you run a mapping on Spark that launches Hive tasks, you can view the Hive query statistics in the session log and in the Administrator tool for monitoring along with the Spark application. For example, you can monitor information related to the Update Strategy transformation and SQL authorization associated to the mapping on Spark.

You can view the Summary Statistics for a Hive task in the Administrator tool. The Spark statistics continue to appear. When the Spark engine launches a Hive task, you can see Source Load Summary and Target Load summary including Spark data frame with Hive task statistics. Otherwise, when you have only a Spark task, the Source Load Summary and Target Load Summary do not appear in the session log.

Under Target Load Summary, all Hive instances will be prefixed with 'Hive_Target_'. You can see same instance name in the Administrator tool.

You can view the Tez job statistics in the Administrator tool when reading and writing to Hive tables that the Spark engine launches in any of the following scenarios:

- You have resources present in the Amazon buckets.
- You have transactional Hive tables.
- You have table columns secured with fine grained authorization.

Incorrect statistics appears for all the Hive sources and targets indicating zero rows for average rows for each second, bytes, average bytes for each second, and rejected rows. You can see that only processed rows contain correct values, and the remaining columns will contain either 0 or N/A.

Spark Engine Logs

The Spark engine logs appear in the LDTM log. The LDTM logs the results of the Spark engine execution plan run for the mapping. You can view the LDTM log from the Developer tool or the Monitoring tool for a mapping job.

The log for the Spark engine shows the following:

- Step to translate the mapping to an internal format
- Steps to optimize the mapping
- Steps to render the mapping to Spark code
- Steps to submit the code to the Spark executor
- Scala code that the Logical Data Translation Generator creates from the mapping logic
- Total number of cluster nodes used to execute the mapping

When you run Sqoop mappings on the Spark engine, the Data Integration Service prints the Sqoop log events in the mapping log.

Viewing Spark Logs

You can view logs for a Spark mapping from the YARN web user interface.

1. In the YARN web user interface, click an application ID to view.
2. Click the application **Details**.
3. Click the **Logs** URL in the application details to view the logs for the application instance.

   The log events appear in another browser window.

Troubleshooting Spark Engine Monitoring

Do I need to configure a port for Spark Engine Monitoring?

Spark engine monitoring requires the cluster nodes to communicate with the Data Integration Service over a socket. The Data Integration Service picks the socket port randomly from the port range configured for the domain. The network administrators must ensure that the port range is accessible from the cluster nodes to the Data Integration Service. If the administrators cannot provide a port range access, you can configure the Data Integration Service to use a fixed port with the SparkMonitoringPort custom property. The network administrator must ensure that the configured port is accessible from the cluster nodes to the Data Integration Service.

Recovered jobs show 0 elapsed time in monitoring statistics

When a job is recovered, the Monitoring tab shows the same start and end time for the job, and the elapsed time = 0. While this statistic is not the actual elapsed time, it enables you to identify jobs that were recovered. For a more accurate view of the elapsed time for the job, view the Spark job logs on the cluster or the session logs on the Data Integration Service.

I enabled data engineering recovery, but recovered jobs show missing or incorrect statistics in the monitoring tab

Consider the following behavior:

- When the Data Integration Service recovers a job, the Administrator tool might display incomplete job statistics in the Monitoring tab when the job is complete. For example, the job statistics might not correctly display the number of rows processed.

- The Monitoring tab does not display detailed statistics if the Data Integration Service process stops after it submits a job to the compute cluster.

CHAPTER 12

Hierarchical Data Processing

This chapter includes the following topics:

- Overview of Hierarchical Data Processing, 219
- How to Develop a Mapping to Process Hierarchical Data, 220
- Complex Data Types, 222
- Complex Ports, 227
- Complex Data Type Definitions, 229
- Type Configuration, 234
- Complex Operators, 238
- Complex Functions, 240
- Rules and Guidelines for Processing Hierarchical Data on the Spark Engine, 242
- Midstream Parsing of Hierarchical Data, 242

Overview of Hierarchical Data Processing

The Spark and Databricks Spark engines can process mappings that use complex data types, such as array, struct, and map. With complex data types, the run-time engine directly reads, processes, and writes hierarchical data in complex files.

The Spark and Databricks Spark engines can process hierarchical data in Avro, JSON, ORC, and Parquet complex files. They use complex data types to represent the native data types for hierarchical data in complex files. For example, a hierarchical data of type record in an Avro file is represented as a struct data type by the run-time engine.

Hierarchical Data Processing Scenarios

You can develop mappings for the following hierarchical data processing scenarios:

- To generate and modify hierarchical data.
- To transform relational data to hierarchical data.
- To transform hierarchical data to relational data.
- To convert data from one complex file format to another. For example, read hierarchical data from an Avro source and write to a JSON target.
- To generate struct target data after parsing hierarchical JSON or XML data midstream in a mapping.

Hierarchical Data Processing Configuration

You create a connection to access complex files and create a data object to represent data in the complex file. Then, configure the data object read and write operations to project columns as complex data types. To read and write hierarchical data in complex files, you create a mapping, add a Read transformation based on the read operation, and add a Write transformation based on the write operation. To process hierarchical data, configure the following objects and transformation properties in a mapping:

- Complex ports. To pass hierarchical data in a mapping, create complex ports. You create complex ports by assigning complex data types to ports.
- Complex data type definitions. To process hierarchical data of type struct, create or import complex data type definitions that represent the schema of struct data.
- Type configuration. To define the properties of a complex port, specify or change the type configuration.
- Complex operators and functions. To generate or modify hierarchical data, create expressions using complex operators and functions.

You can also use hierarchical conversion wizards to simplify some of the mapping development tasks.

How to Develop a Mapping to Process Hierarchical Data

Develop a mapping with complex ports, operators, and functions to process hierarchical data on the Spark or Databricks Spark engine.

Note: The tasks and the order in which you perform the tasks to develop the mapping depend on the mapping scenario.

The following is an outline of the high-level tasks to develop and run a mapping to read, write, and process hierarchical data in complex files.

Create a connection.

Create a connection to access data in complex files based on the file storage.

Create a data object.

1. Create a data object to represent the complex files as sources or targets.
2. Configure the data object properties.
3. In the read and write operations, enable the column file properties to project columns in the complex files as complex data types.

Create a mapping and add mapping objects.

1. Create a mapping, and add Read and Write transformations.
   To read from and write to a complex file, add Read and Write transformations based on the data object read and write operations.

   To write to an Avro or Parquet file, you can also create a complex file target from an existing transformation in the mapping.
2. Based on the mapping logic, add other transformations that are supported on the run-time engine.

Generate struct data.

Based on the mapping scenario, use one of the hierarchical conversion wizards to generate struct data. You can also perform the following steps manually:

Create or import complex data type definitions for struct ports.

1. Create or import complex data type definitions that represent the schema of the struct data. The complex data type definitions are stored in the type definition library, which is a Model repository object. The default name of the type definition library is m_Type_Definition_Library.
2. If a mapping uses one or more mapplets, rename the type definition libraries in the mapping and the mapplets to ensure that the names are unique.

Create and configure struct ports in transformations.

1. Create ports in transformations and assign struct data type.
2. Specify the type configuration for the struct ports.
   You must reference a complex data type definition for the struct port.
3. Create expressions with complex functions to generate struct data.

Modify struct data.

You can convert struct data to relational or hierarchical data. If the struct data contains elements of primitive data types, you can extract the elements as relational data. If the struct data contains elements of complex data types, you can extract the elements as hierarchical data. Based on the mapping scenario, use one of the hierarchical conversion wizards to modify struct data. You can also perform the following steps manually:

1. Create output ports with port properties that match the element of the struct data that you want to extract.
2. Create expressions with complex operators or complex functions to modify the struct data.

Generate array data.

1. Create ports in transformations and assign array data type.
2. Specify the type configuration for the array ports.
3. Create expressions with complex functions to generate array data.

Modify array data.

You can convert array data to relational or hierarchical data. If the array data contains elements of primitive data types, you can extract the elements as relational data. If the array data contains elements of complex data types, you can extract the elements as hierarchical data. Based on the mapping scenario, use one of the hierarchical conversion wizards to modify array data. You can also perform the following steps manually:

1. Create output ports with port properties that match the element of the array data that you want to extract.
2. Create expressions with complex operators or complex functions to modify the array data.

Generate map data.

1. Create ports in transformations and assign map data type.
2. Specify the type configuration for the map ports.
3. Create expressions with complex functions to generate map data.

Modify map data.

You can convert map data to relational or hierarchical data. If the map data contains elements of primitive data types, you can extract the elements as relational data. If the map data contains elements of complex data types, you can extract the elements as hierarchical data. Based on the mapping

scenario, use one of the hierarchical conversion wizards to modify map data. You can also perform the following steps manually:

1. Create output ports with port properties that match the element of the map data that you want to extract.
2. Create expressions with complex operators or complex functions to modify the map data.

Configure the transformations.

Link the ports and configure the transformation properties based on the mapping logic.

Configure the mapping properties.

Configure the following mapping run-time properties:

- To run on the Spark engine, choose the Spark validation environment and Hadoop as the execution environment.
- To run on the Databricks Spark engine, choose the Databricks Spark validation environment and Hadoop as the execution environment.

Validate and run the mapping.

1. Validate the mapping to identify and correct any errors.
2. Optionally, view the engine execution plan to debug the logic.
3. Run the mapping.

Complex Data Types

A complex data type is a transformation data type that represents multiple data values in a single column position. The data values are called elements. Elements in a complex data type can be of primitive or complex data types. Use complex data types to process hierarchical data in mappings that run on the Spark or Databricks Spark engine.

Transformation data types include the following data types:

Primitive data type

A transformation data type that represents a single data value in a single column position. Data types such as decimal, integer, and string are primitive data types. You can assign primitive data types to ports in any transformation.

Complex data type

A transformation data type that represents multiple data values in a single column position. Data types such as array, map, and struct are complex data types. You can assign complex data types to ports in some transformations.

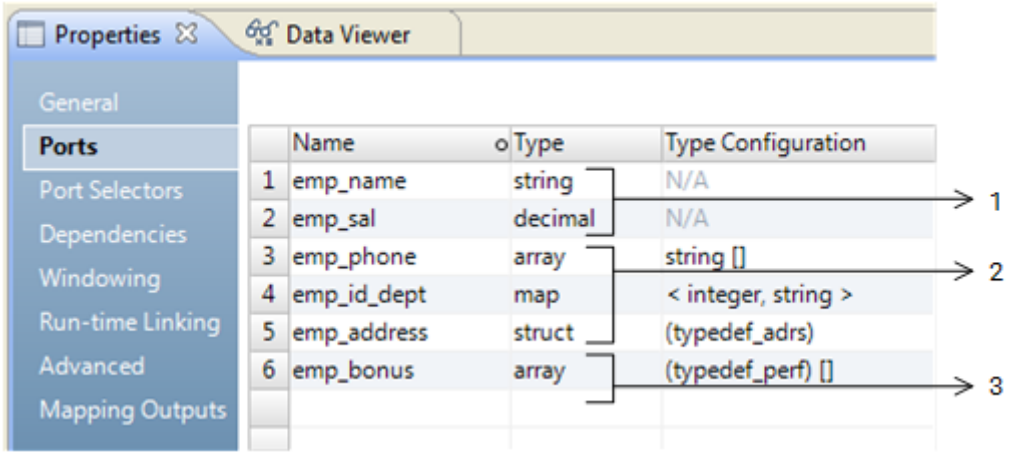
Nested data type

A complex data type that contains elements of complex data types. Complex data types such as an array of structs or a struct with an array of other structs are nested data types.

The following table lists the complex data types:

Complex Data Type	Description
array	An array is an ordered collection of elements. The elements can be of a primitive or complex data type. All elements in the array must be of the same data type.
map	A map is an unordered collection of key-value pair elements. The key must be of a primitive data type. The value can be of a primitive or complex data type.
struct	A struct is a collection of elements of different data types. The elements can be of primitive or complex data types. Struct has a schema that defines the structure of the data.

The following image shows primitive, complex, and nested data types assigned to ports in a transformation:



1. Primitive data types
2. Complex data types
3. Nested data type

The ports emp_name and emp_sal are of primitive data types. The ports emp_phone, emp_id_dept, and emp_address are of complex data types. The port emp_bonus is of a nested data type. The array contains elements of type struct.

Array Data Type

An array data type represents an ordered collection of elements. To pass, generate, or process array data, assign array data type to ports.

An array is a zero-based indexed list. An array index indicates the position of the array element. For example, the array index 0 indicates the first element in an array. The transformation language includes operators to access array elements and functions to generate and process array data.

An array can be one-dimensional or multidimensional. A one-dimensional array is a linear array. A multidimensional array is an array of arrays. Array transformation data types can have up to five dimensions.

Format

```
array <data_type> []
```

The following table describes the arguments for this data type:

Argument	Description
`array`	Name of the array column or port.
`data_type`	Data type of the elements in an array. The elements can be primitive data types or complex data types. All elements in the array must be of the same data type.
`[]`	Dimension of the array represented as subscript. A single subscript `[]` represents a one-dimensional array. Two subscripts `[][]` represent a two-dimensional array. Elements in each dimension are of the same data type.

The elements of an array do not have names. The number of elements in an array can be different for each row.

Array Examples

One-dimensional array

The following array column represents a one-dimensional array of string elements that contains customer phone numbers:

```
custphone string[]
```

The following example shows data values for the custphone column:

custphone
[205-128-6478,722-515-2889]
[107-081-0961,718-051-8116]
[107-031-0961,NULL]

Two-dimensional array

The following array column represents a two-dimensional array of string elements that contains customer work and personal email addresses.

```
email_work_pers string[][]
```

The following example shows data values for the email_work_pers column:

email_work_pers
[john_baer@xyz.com,jbaer@xyz.com][john.baer@fgh.com,jbaer@ijk.com]
[bobbi_apperley@xyz.com,bapperl@xyz.com][apperlbob@fgh.com,bobbi@ijk.com]
[linda_bender@xyz.com,lbender@xyz.com][l.bender@fgh.com,NULL]

Map Data Type

A map data type represents an unordered collection of key-value pair elements. A map element is a key and value pair that maps one thing to another. To pass, generate, or process map data, assign map data type to ports.

The key must be of a primitive data type. The value can be of a primitive or complex data type. A map data type with values of a complex data type is a nested map. A nested map can contain up to three levels of nesting of map data type.

Note: The Spark engine allows duplicate keys in the map data.

The transformation language includes subscript operator to access map elements. It also includes functions to generate and process map data.

Format

```
map <primitive_type -> data_type>
```

The following table describes the arguments for this data type:

Argument	Description
map	Name of the map column or port.
primitive_type	Data type of the key in a map element. The key must be of a primitive data type.
data_type	Data type of the value in a map element. The value can be of a primitive or complex data type.

Map Example

The following map column represents map data with an integer key and a string value to map customer ids with customer names:

```
custid_name <integer -> string>
```

The following example shows data values for the custid_name column:

```
custid_name

<26745 -> 'John Baer'>

<56743 -> 'Bobbi Apperley'>

<32879 -> 'Linda Bender'>
```

Struct Data Type

A struct data type represents a collection of elements of different data types. A struct data type has an associated schema that defines the structure of the data. To pass, generate, or process struct data, assign the struct data type to ports.

The schema for the struct data type determines the element names and the number of elements in the struct data. The schema also determines the order of elements in the struct data. Informatica uses complex data type definitions to represent the schema of struct data.

The transformation language includes operators to access struct elements. It also includes functions to generate, process, and parse struct data and to modify the schema of the data.

Format

```
struct {element_name1:value1 [, element_name2:value2, ...]}
```

Schema for the struct is of the following format:

```
schema {element_name1:data_type1 [, element_name2:data_type2, ...]}
```

The following table describes the arguments for this data type:

Argument	Description
struct	Name of the struct column or port.
schema	A definition of the structure of data. Schema is a name-type pair that determines the name and data type of the struct elements.
element_name	Name of the struct element.
value	Value of the struct element.
data_type	Data type of the element value. The element values can be of a primitive or complex data type. Each element in the struct can have a different data type.

Struct Example

The following schema is for struct data to store customer addresses:

```
address
{st_number:integer,st_name:string,city:string,state:string,zip:string}
```

The following example shows struct data values for the cust_address column:

```
cust_address

{st_number:154,st_name:Addison Ave,city:Redwood City,state:CA,zip:94065}

{st_number:204,st_name:Ellis St,city:Mountain View,state:CA,zip:94043}

{st_number:357,st_name:First St,city:Sunnyvale,state:CA,zip:94085}
```

Rules and Guidelines for Complex Data Types

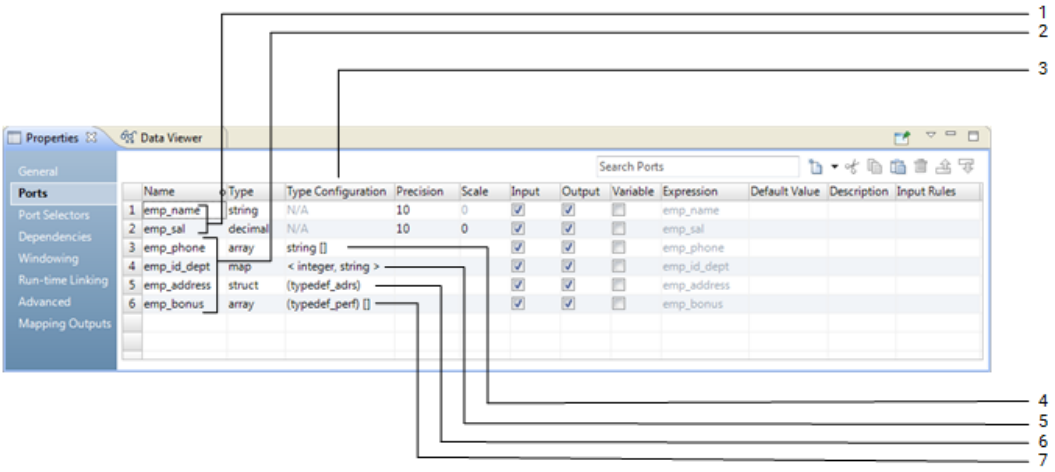
Consider the following rules and guidelines when you work with complex data types:

- A nested data type can contain up to 10 levels of nesting.
- A nested map can contain up to three levels of nesting of map data types.
- An array data type cannot directly contain an element of type array. Use multidimensional arrays to create a nested array. For example, an array with two dimensions is an array of arrays.
- A multidimensional array can contain up to five levels of nesting. The array dimension determines the levels of nesting.
- Each array in a multidimensional array must have elements of the same data type.

Complex Ports

A complex port is a port that is assigned a complex data type. Based on the complex data type, you must specify the complex port properties. Use complex ports in transformations to pass or process hierarchical data in a mapping.

The following image shows the complex ports and complex port properties on the Ports tab for a transformation:



1. Port
2. Complex port
3. Type configuration
4. Type configuration for an array port
5. Type configuration for a map port
6. Type configuration for a struct port
7. Type configuration for a port of nested data type

Based on the data type, a transformation can include the following ports and port properties:

Port

A port of a primitive data type that you can create in any transformation.

Complex port

A port of a complex or nested data type that you can create in some transformations. Array, map, and struct ports are complex ports. Based on the complex data type, you specify the complex port properties in the type configuration column.

Type configuration

A set of properties that you specify for the complex port. The type configuration determines the data type of the complex data type elements or the schema of the data. You specify the data type of the elements for array and map ports. You specify a complex data type definition for the struct port.

Type configuration for an array port

Properties that determine the data type of the array elements. In the image, the array port emp_phone is a one-dimensional array with an ordered collection of string elements. An array with string elements is also called an array of strings.

Type configuration for a map port

Properties that determine the data type of the key-value pair of the map elements. In the image, the map port emp_id_dept is an unordered collection of key-value pairs of type integer and string.

Type configuration for a struct port

Properties that determine the schema of the data. To represent the schema, you create or import a complex data type definition. In the image, the struct port emp_address references a complex data type definition typedef_adrs.

Type configuration for a port of nested data type

Properties that determine the nested data type. In the image, the array port emp_bonus is a one-dimensional array with an ordered collection of struct elements. The struct elements reference a complex data type definition typedef_bonus. An array with struct elements is also called an array of structs.

Complex Ports in Transformations

You can create complex ports in some transformations that are supported on the Spark and Databricks Spark engines. Read and Write transformations can represent ports that pass hierarchical data as complex data types.

You can create complex ports in the following transformations:

- Aggregator
- Expression
- Filter
- Java
- Joiner
- Lookup
- Normalizer
- Router
- Sorter
- Union

Note: The Databricks Spark engine does not support the Java transformation.

The Read and Write transformations can read and write hierarchical data in complex files. To read and write hierarchical data, the Read and Write transformations must meet the following requirements:

- The transformation must be based on a complex file data object.
- The data object read and write operations must project columns as complex data types.

Rules and Guidelines for Complex Ports

Consider the following rules and guidelines when you work with complex ports:

- Aggregator transformation. You cannot define a group by value as a complex port.
- Filter transformation. You cannot use the operators >, < , >=, and <= in a filter condition to compare data in complex ports.
- Joiner transformation. You cannot use the operators >, < , >=, and <= in a join condition to compare data in complex ports.

- Lookup transformation. You cannot use a complex port in a lookup condition.
- Rank transformation. You cannot define a group by or rank value as a complex port.
- Router transformation. You cannot use the operators >, < , >=, and <= in a group filter condition to compare data in complex ports.
- Sorter transformation. You cannot define a sort key value as a complex port.
- You can use complex operators to specify an element of a complex port that is of a primitive data type. For example, an array port "emp_names" contains string elements. You can define a group by value as emp_names[0], which is of type string.

Creating a Complex Port

Create complex ports in transformations to pass or process hierarchical data in mappings that run on the Spark or Databricks Spark engine.

1. Select the transformation in the mapping.
2. Create a port.
3. In the **Type** column for the port, select a complex data type.

   The complex data type for the port appears in the Type column.

After you create a complex port, specify the type configuration for the complex port.

Complex Data Type Definitions

A complex data type definition represents the schema of the data. Use complex data type definitions to define the schema of the data for a struct port. If the complex port is of type struct or contains elements of type struct, you must specify the type configuration to reference a complex data type definition.

You can create or import complex data type definitions. You import complex data type definitions from complex files. Complex data type definitions are stored in the type definition library, which is a Model repository object.

When you create a mapping or a mapplet, the Developer tool creates a type definition library with the name Type_Definition_Library. You can view and rename the type definition library and the complex data type definitions in the Outline view of a mapping. The complex data type definitions appear on the type definition library tab of the mapping editor. The tab name is based on the name of the type definition library. You create or import complex data type definitions on the type definition library tab. When a mapping uses one or more mapplets, rename the type definition libraries in the mapping and the mapplets to ensure that the names are unique.

Type definition library and complex data type definitions are specific to a mapping or mapplet. A mapping or a mapplet can have one or more complex data type definitions. You can reference the same complex data type definition in one or more complex ports in the mapping or mapplet.

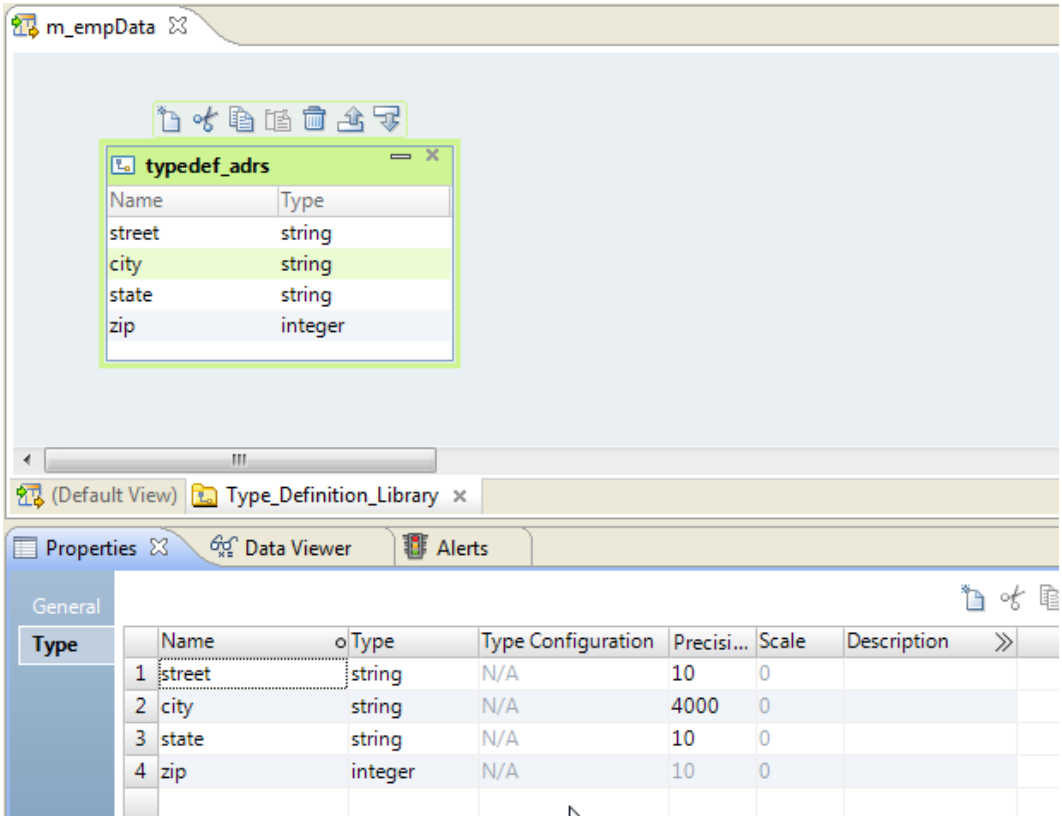
Complex Data Type Definition Example

The complex ports work_address and home_address of type struct have the following schema:

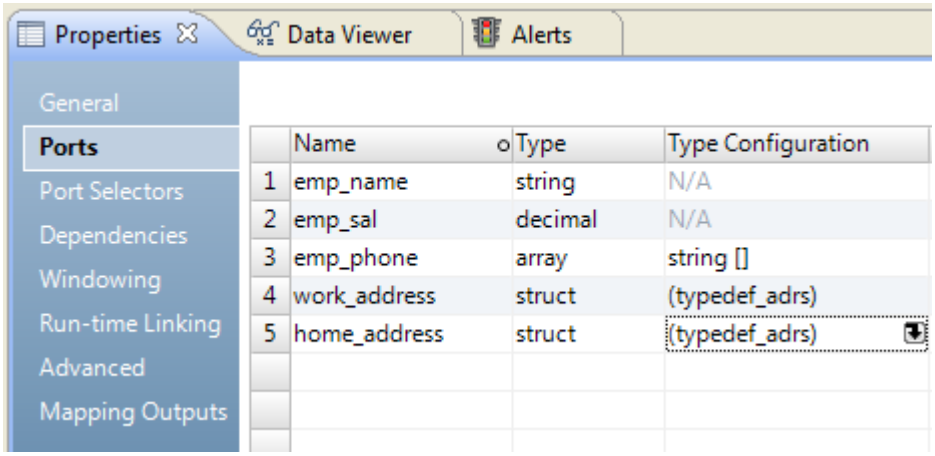
{street:string, city:string, state:string, zip:integer}

In the mapping, you create a complex data type definition typedef_adrs that defines the schema of the data. Then, you specify the type configuration of the struct ports work_address and home_address to use the typedef_adrs complex data type definition.

The following image shows a complex data type definition typedef_adrs:



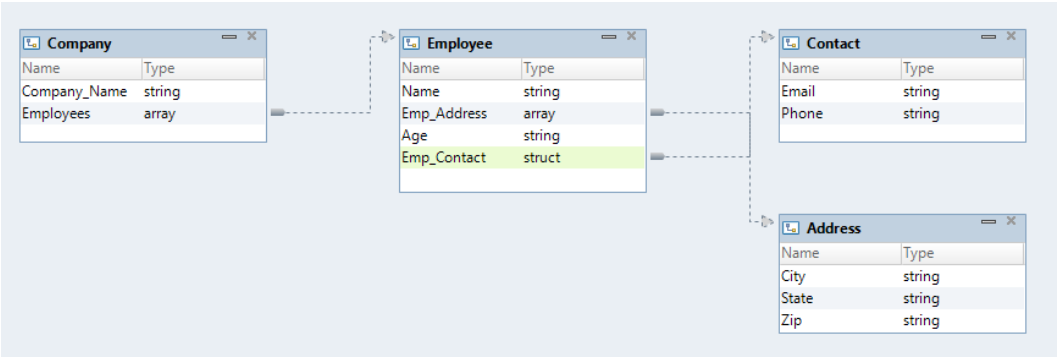
The following image shows two struct ports work_address and home_address that reference the complex data type definition typedef_adrs:



Nested Data Type Definitions

Elements of a complex data type definition can reference one or more complex data type definitions in the type definition library. Such complex data type definitions are called nested data type definitions.

The following image shows a nested data type definition Company on the type definition library tab:



The nested data type definition Company references the following complex data type definitions:

- In the complex data type definition Company, the array element Employees references the complex data type definition Employee.
- In the complex data type definition Employee, the Emp_Address element references the complex data type definition Address.
- In the complex data type definition Employee, the Emp_Contact element references the complex data type definition Contact.

Note: In a recursive data type definition, one of the complex data type definitions at any level is the same as any of its parents. You cannot reference a recursive data type definition to a struct port or a struct element in a complex port.

Rules and Guidelines for Complex Data Type Definitions

Consider the following rules and guidelines when you work with complex data type definitions:

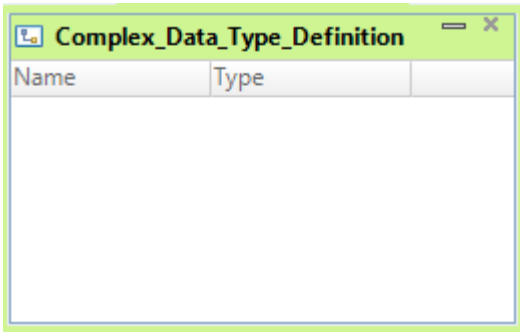
- A struct port or a struct element in a complex port must reference a complex data type definition.
- You cannot reference a complex data type definition in one mapping to a complex port in another mapping.
- You cannot reference a recursive data type definition to a struct port or a struct element in a complex port.
- Changes to the elements in a complex data type definition are automatically propagated to complex ports that reference the complex data type definition.
- If you change the name of the type definition library or the complex data type definition, you must update the name in expressions that use complex functions such as STRUCT_AS, RESPEC, and CAST.
- In a midstream parsing mapping, the source string that contains JSON or XML elements is defined with a complex data type definition. For more information, see "Midstream Parsing of Hierarchical Data" on page 242.

Creating a Complex Data Type Definition

A complex data type definition represents the schema of the data of type struct. You can create a complex data type definition for a mapping or a mapplet on the Type Definition Library tab of the mapping editor.

1. In the Developer tool, open the mapping or mapplet where you want to create a complex data type definition.
2. In the **Outline** view, select the **Type Definition Library** to view the **Type_Definition_Library** tab in the mapping editor.
3. Right-click an empty area of the editor and click **New Complex Data Type Definition**.

   An empty complex data type definition appears on the Type Definition Library tab of the mapping editor.



4. Optionally, change the name of the complex data type definition.
5. Click the **New** button to add elements to the data type definition with a name and a type.

   The data type of the element can be of primitive or complex data type.

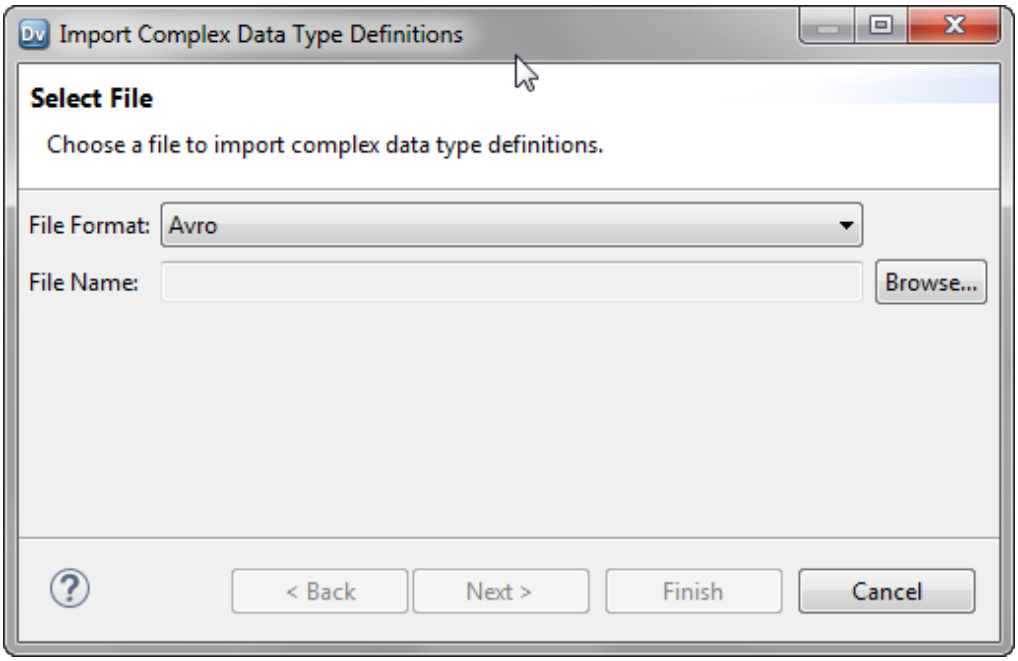
Importing a Complex Data Type Definition

A complex data type definition represents the schema of the data of type struct. You can import the schema of the hierarchical data in the complex file to the type definition library as a complex data type definition.

Import complex data type definitions from an Avro, JSON, or Parquet schema file. For example, you can import an Avro schema from an .avsc file as a complex data type definition.

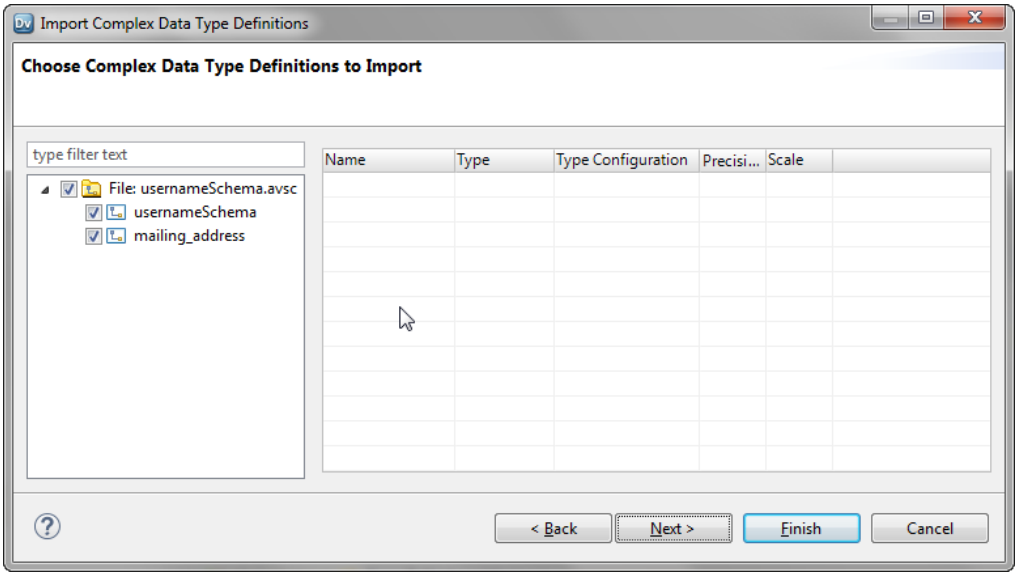
1. In the Developer tool, open the mapping or mapplet where you want to import a complex data type definition.
2. In the **Outline** view, select the **Type Definition Library** to view the **Type_Definition_Library** tab in the mapping editor.
3. Right-click an empty area of the editor and click **Import Complex Data Type Definitions**.

The **Import Complex Data Type Definitions** dialog box appears.



4. Select a complex file format from the **File Format** list.
5. Click **Browse** to select the complex file schema from which you want to import the complex data type definition.
6. Navigate to the complex file schema and click **Open**.
7. Click **Next**.

   The **Choose Complex Data Type Definitions to Import** page appears.



8. Select one or more schemas from the list to import.
9. Click **Next**.

   The **Complex Data Type Definitions to Import** page appears.

10. Review the complex data type definitions that you want to import and click **Finish**.

    The complex data type definition appears in the **Type Definition Library** tab of the mapping or the mapplet.

Type Configuration

A type configuration is a set of complex port properties that specify the data type of the complex data type elements or the schema of the data. After you create a complex port, you specify or change the type configuration for the complex port on the Ports tab.

The type configuration for a complex port depends on the complex data type assigned to the port. You must specify a complex data type definition for a struct port or a struct element in a complex port. Array and map ports have a default type configuration that you can change.

The following table lists the type configuration properties for complex ports:

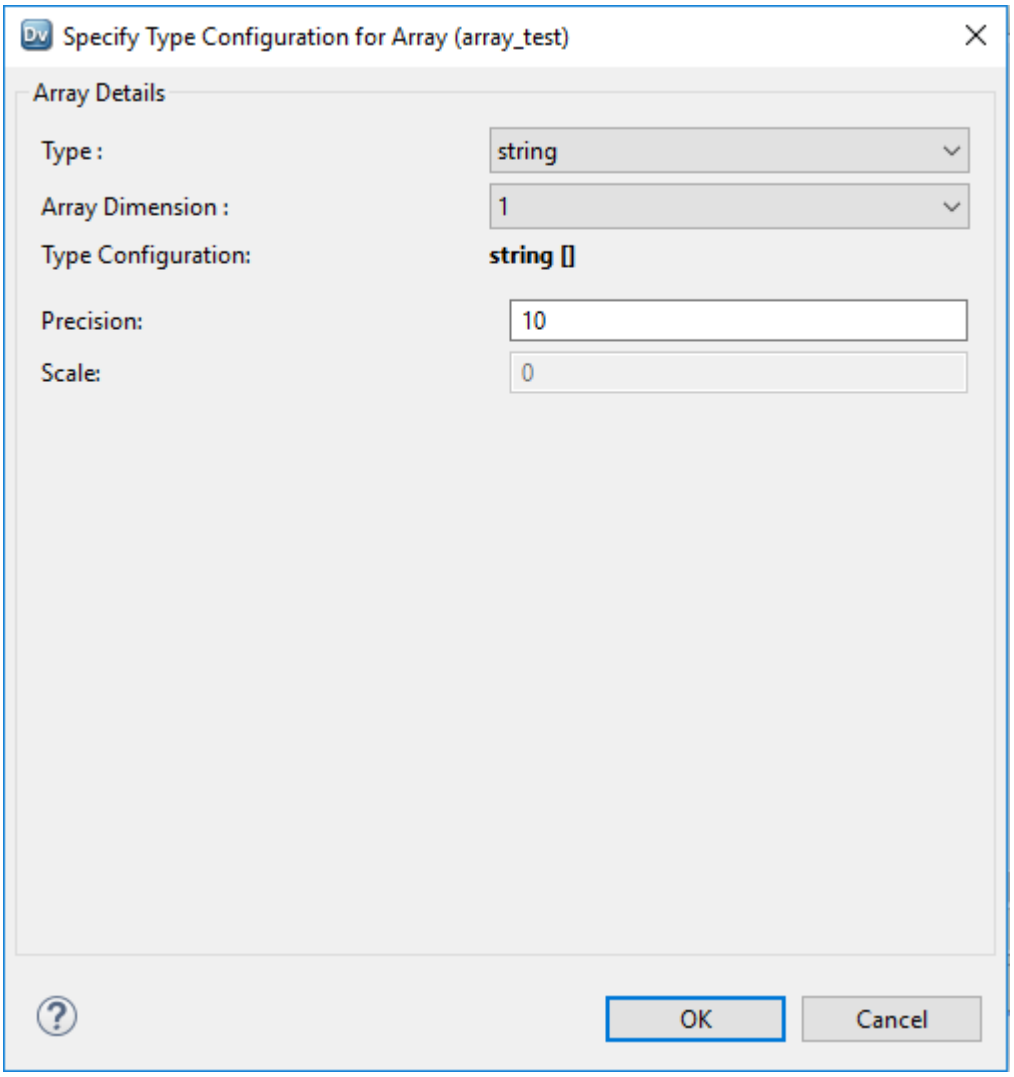
Complex Port	Type Configuration Properties
array	- Type. Data type of the elements of an array. Default is string. - Array Dimension. Number of levels for an array. Default is 1. - Precision. Maximum number of significant digits for numeric data types, or maximum number of characters for string data types. - Scale. Maximum number of digits after the decimal point of numeric values.
map	- Type. Data type of the key-value pair. Default is string for key and value. - Precision. Maximum number of significant digits for numeric data types, or maximum number of characters for string data types. - Scale. Maximum number of digits after the decimal point of numeric values.
struct	- Complex data type definition. The schema of the data that you created or imported as complex data type definition in the type definition library.

Changing the Type Configuration for an Array Port

The default type configuration for an array is `string[]`, which is a one-dimensional array of strings. You can change the type configuration for the array port. Specify the data type of the array elements and the array dimension.

1. In the transformation, select the array port for which you want to specify the type configuration.
2. In the **Type Configuration** column, click the **Open** button.

The **Specify Type Configuration for Array** dialog box appears.



3. In the **Type** drop-down menu, change the data type of the elements in the array.

   For example, if the elements in the array are of type integer, select **integer**.

   The type configuration value appears as **integer []**.

4. Optionally, in the **Array Dimension** drop-down menu, select a number to create a nested array.

   For example, array dimension 3 creates a nested array of 3 levels. The type configuration value appears as **integer [] [] []**.

5. Optionally, configure the following properties:

   - **Precision**. Maximum number of significant digits for numeric data types, or maximum number of characters for string data types.
   - **Scale**. Maximum number of digits after the decimal point of numeric values.

6. Click **OK**.

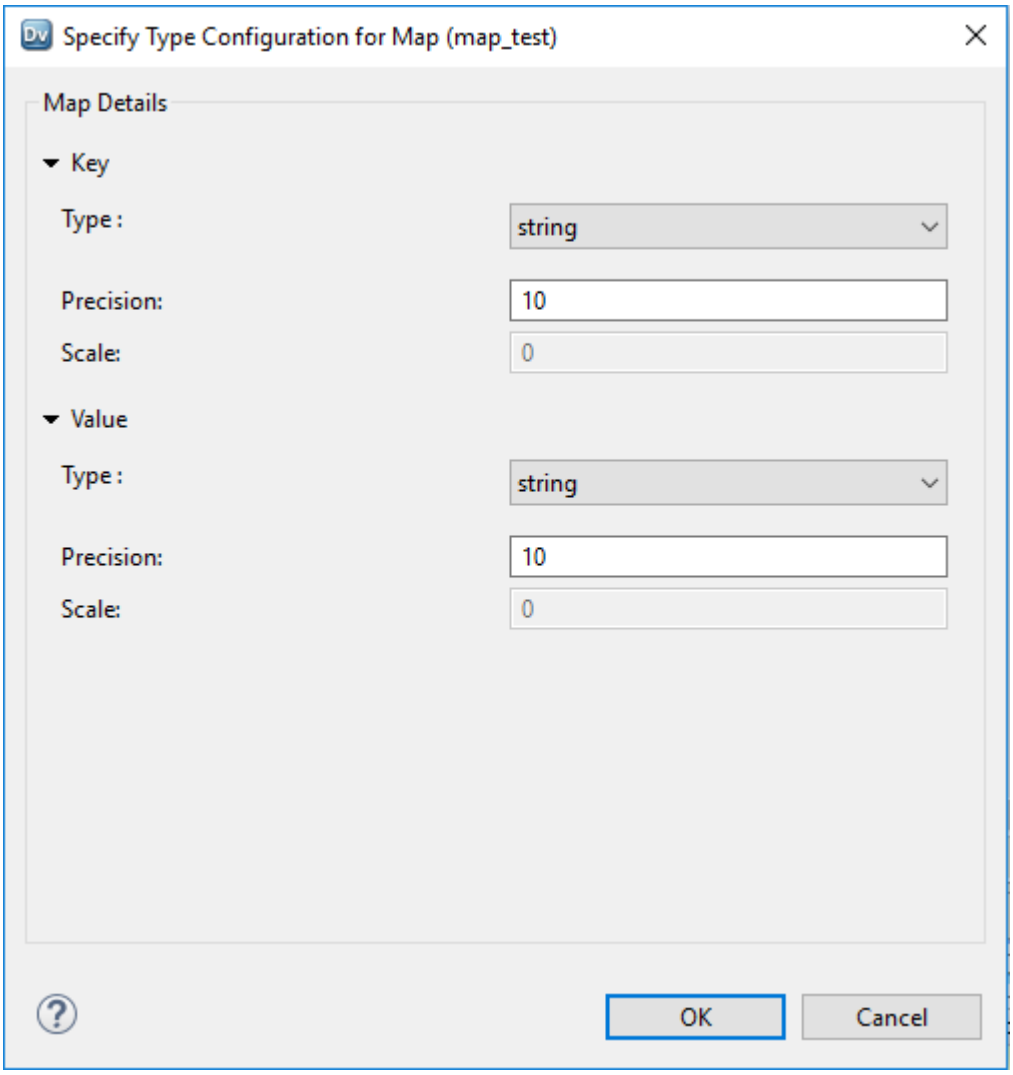
   On the **Ports** tab, the specified type configuration appears in the **Type Configuration** column of the array port.

Changing the Type Configuration for a Map Port

The default type configuration for a map is `<string, string>`. You can change the type configuration for a map port. Specify the data type of the key-value pair.

1. In the transformation, select the map port for which you want to set the type configuration.
2. In the **Type Configuration** column, click the **Open** button.

   The **Type Configuration** dialog box appears:



3. Specify the properties for the key and value of the map data type.
   a. In the **Type** drop-down menu, select the data type.
   b. Optionally, in the **Precision** box, specify the maximum number of significant digits for numeric data types, or maximum number of characters for string data types.
   c. Optionally, in the **Scale** box, specify the maximum number of digits after the decimal point of numeric values.
4. Click **OK**.

   On the **Ports** tab, the specified type configuration appears in the **Type Configuration** column of the map port.

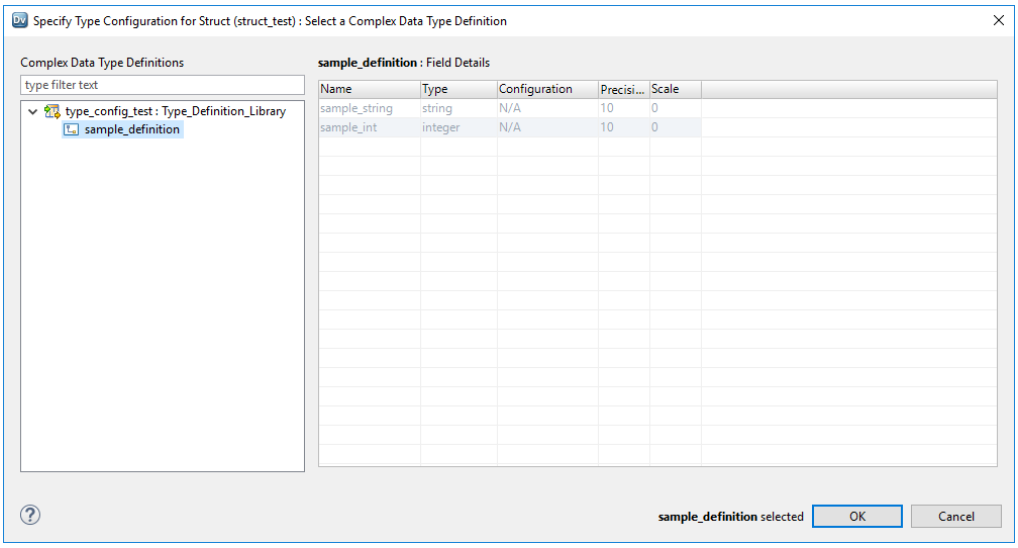
Specifying the Type Configuration for a Struct Port

The type configuration for a struct port requires a complex data type definition in the type definition library. Specify the complex data type definition that represents the schema of the struct data.

Before you specify the type configuration for the struct port, create or import a complex data type definition. You can also reference an existing complex data type definition in the type definition library.

1. In the transformation, select the struct port for which you want to specify the type configuration.
2. In the **Type Configuration** column, click the **Open** button.

   The **Type Configuration** dialog box appears:



3. Select a complex data type definition from the list of definitions in the type definition library for the mapping or the mapplet.
4. Click **OK**.

   On the **Ports** tab, the specified type configuration appears in the **Type Configuration** column of the struct port.

Complex Operators

A complex operator is a type of operator to access elements in a complex data type. The transformation language includes complex operators for complex data types. Use complex operators to access or extract elements in hierarchical data.

The following table lists the complex operators:

Complex Operators	Description
[]	Subscript operator. Use to access array and map elements. Syntax: *array*[*index*] - *array* is the array from which you want to access an element. - *index* is the position of an element within an array. For example, use [0] to access the first element in an one-dimensional array. Syntax: *map*[*key*] - *map* is the map from which you want to access the value corresponding to a key. - *key* is the map key for which you want to retrieve the map value.
.	Dot operator. Use to access struct elements. Syntax: *struct.element_name* - *struct* is the struct from which you want to access an element. - *element_name* is the name of the struct element.

Use complex operators in expressions to convert hierarchical data to relational data.

For more information about the operator syntax, return values, and examples, see the *Informatica Developer Transformation Language Reference*.

Complex Operator Examples

- Convert array data in a JSON file to relational data.
  A JSON file has a quarterly_sales column that is of the array type with integer elements. You develop a mapping to transform the hierarchical data to relational data. The array port quarterly_sales contains four elements. Use subscript operators in expressions to extract the four elements into the four integer ports, q1_sales, q2_sales, q3_sales, and q4_sales.

  The expressions for the integer output port are as follows:

  ```
  quarterly_sales[0]
  quarterly_sales[1]
  quarterly_sales[2]
  quarterly_sales[3]
  ```

- Convert struct data in a Parquet file to relational data.
  A Parquet file has an address column that contains customer address as a struct type. You develop a mapping to transform hierarchical data to relational data. The struct port address contains three elements city, state, and zip, of type string. Use dot operators in expressions to extract the three struct elements into the three string ports, city, state, and zip.

  The expressions for the string output port are as follows:

  ```
  address.city
  address.state
  address.zip
  ```

Extracting an Array Element Using a Subscript Operator

Use the subscript operator in expressions to extract one or more elements of an array.

1. In the transformation, create an output port of the same data type as the return value of the array element that you want to extract.

   To extract more than one element in an array, create an array output port. The data type of the elements in the type configuration for the array port must match the data type of the return value of the array elements.
2. In the **Expression** column for the output port, click the **Open** button.

   The **Expression Editor** opens.
3. Delete the existing expression in the editor.
4. Click the **Ports** tab and select an array port.
5. Enter one or more index values within the subscript operator.

   To extract array elements in a multidimensional array, enter index values within each subscript operator.
6. Click **Validate** to validate the expression.
7. Click **OK** to exit the **Validate Expression** dialog box.
8. Click **OK** to exit the **Expression Editor**.

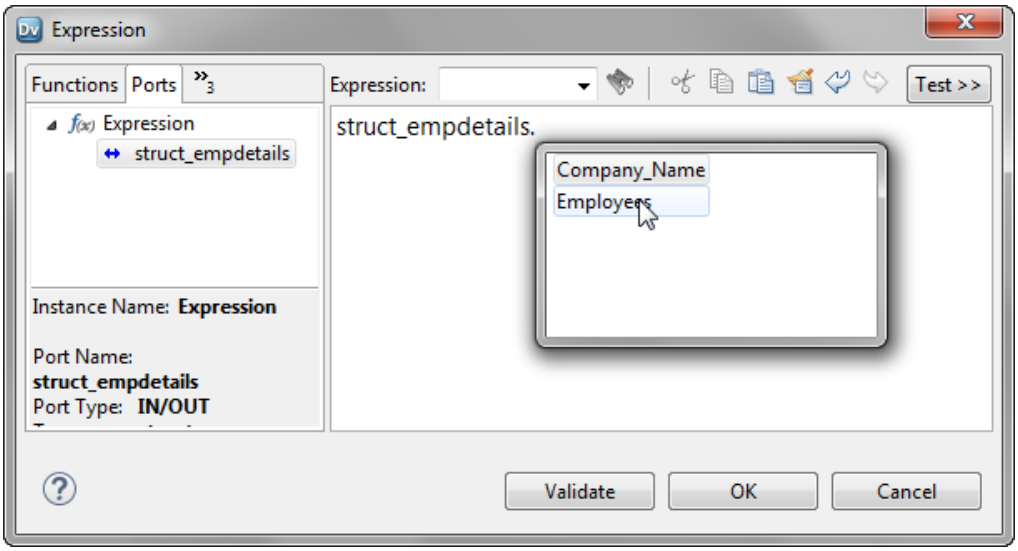
Extracting a Struct Element Using the Dot Operator

Use the dot operator in expressions to extract elements of a struct.

1. In the transformation, create an output port of the same data type as the return value of the struct element that you want to extract.
2. In the **Expression** column for the output port, click the **Open** button.

   The Expression Editor opens.
3. Delete the existing expression in the editor.
4. Click the **Ports** tab and select the struct port.
5. Enter a dot after the port name.

The Developer tool displays a list of elements in the struct.



6. Select the element that you want to extract.
7. Click **Validate** to validate the expression.
8. Click **OK** to exit the Validate Expression dialog box.
9. Click **OK** to exit the Expression Editor.

Complex Functions

A complex function is a type of pre-defined function in which the value of the input or the return type is of a complex data type. The transformation language includes complex functions for complex data types. Use complex functions to generate, process, and parse hierarchical data.

The following table describes the complex functions for array data type:

Complex Function	Description
ARRAY (*element1* [,*element2*] ...)	Generates an array with the specified elements. The data type of the argument determines the data type of the array.
COLLECT_LIST (*value*)	Returns an array with elements in the specified port. The data type of the argument determines the data type of the array. COLLECT_LIST is an aggregate function.
CONCAT_ARRAY (' ', *array_of_strings*)	Concatenates string elements in an array based on a separator that you specify and returns a string.
SIZE (*array*)	Returns the size of the array.

The following table describes the complex functions for map data type:

Complex Function	Description
MAP (*key1*, *value1* [,*key2*, *value2*] ...)	Generates a map with elements based on the specified key-value pair. The data type of arguments determines the data type of the map elements.
MAP_FROM_ARRAYS (*key_array* ,*value_array*)	Generates a map from the specified key and value arrays. The data type of arguments determines the data type of the map elements.
MAP_KEYS (*map*)	Returns an array of keys for the specified map.
MAP_VALUES (*map*)	Returns an array of values for the specified map.
COLLECT_MAP (*value*)	Returns a map with elements based on the specified arguments. The data type of the argument determines the data type of the map. COLLECT_MAP is an aggregate function.
SIZE (*map*)	Returns the size of the map.

The following table describes the complex functions for struct data type:

Complex Function	Description
STRUCT_AS (*type_definition*, *struct*)	Generates a struct with a schema based on the specified complex data type definition and the values you pass as argument.
STRUCT (*element_name1*, *value1*)	Generates a struct with the specified names and values for the elements. The data type of the value is based on the specified value argument.
RESPEC (*type_definition*, *struct*)	Renames each element of the given struct value based on the names of the elements in the specified complex data type definition.
CAST (*type_definition*, *struct*)	Changes the data type and the name of each element for the given struct value based on the corresponding data type and name of the element in the specified complex data type definition.

The following table describes the complex functions that parse hierarchical data midstream in a mapping:

Complex Function	Description
PARSE_JSON (*upstream_string*, *:Type.type_definition_library.type_definition*)	Parses hierarchical JSON data in a string data type and generates a struct with a schema based on the specified complex data type definition.
PARSE_XML (*upstream_string*, *:Type.type_definition_library.type_definition*)	Parses hierarchical XML data in a string data type and generates a struct with a schema based on the specified complex data type definition.

For more information about the function syntax, return value, and examples, see the *Informatica Developer Transformation Language Reference*.

Rules and Guidelines for Processing Hierarchical Data on the Spark Engine

There are processing differences when you work with complex data types in a mapping that runs on the Spark engine.

Consider the following rules and guidelines when you use complex data types in a mapping that runs on the Spark engine:

- You cannot read hierarchical data from or write hierarchical data to a Hive source in a dynamic mapping.
- When you read hierarchical data from a Hive source, you cannot enable Hive LLAP for Hive queries.
- When you read hierarchical data from a Hive source, the Spark engine converts float type data to double. Use the double data type when you read from and write to a Hive source to prevent precision errors.
- When you write date/time data within a complex data type to a Hive target using HDP 3.1, configure the timezone as UTC. In the Hadoop connection Spark advanced properties, append "-Duser.timezone=UTC" to the end of the value for the following properties:
  - spark.driver.extraJavaOptions
  - spark.executor.extraJavaOptions

Midstream Parsing of Hierarchical Data

Midstream Parsing Overview

You can use complex functions to parse up to 5 MB of data midstream in a mapping on the Spark engine.

The Spark engine can parse raw string source data using the following complex functions:

- PARSE_JSON
- PARSE_XML

The complex functions parse JSON or XML data in the source string and generate struct target data.

For more information about the complex functions, see the *Informatica Developer Transformation Language Reference*.

Midstream Parsing Use Case

You work in the IT department for an online retailer. Your management team wants to understand and categorize decreased conversion rates.

The online sales application team reports that there has been an increase in the number of shoppers and in the number of shopping carts with large value items. However, the sales bottom line does not reflect that. In fact, the marketing team reports a decrease in conversion rates. These changes are happening fast, and the amount of data is daunting. The management team wants answers.

You have an upstream Java transformation web service that provides the shopping cart data in JSON, within a string field. However, you are unsure of the specific data structure. The sales team is able to provide a sample file with the JSON data.

The sales and marketing teams have some ideas about what might be causing these issues, but they want actual shopping data to confirm their theories. The teams do not have the resources or expertise to read and analyze raw JSON data.

With the sample JSON data, you are able to use CLAIRE® Intelligent Structure Discovery to parse the shopping cart data, creating a complex data type definition. Now that you know the data structure, you create a midstream mapping to parse the data, and categorize the shopping cart information. You can use the categorized data in downstream transformations, allowing you to further refine the information for the sales and marketing teams.

When the teams see the details you provide regarding the abandoned shopping carts, they realize what they need to do to reduce the number of lookers, and turn them into buyers.

How to Use a Midstream Mapping to Parse Hierarchical Data

Develop a midstream mapping to parse hierarchical data in a source string on the Spark engine.

The following high-level tasks describe how to develop and run a mapping that parses hierarchical data in a source string:

Create a connection.

Create a connection to access data in complex files based on the file storage.

Create data objects.

1. Create a data object to represent the source file with the source string that contains hierarchical data in JSON or XML format.
2. Create a data object to represent the target that will include the hierarchical data in a struct.
3. Configure the data objects' properties.

Create or import a complex data type definition.

Intelligent Structure Discovery parses sample data and discovers the schema for hierarchical data in a source string. The following alternatives are available:

- Create a complex data type definition using a representative sample file in Informatica Intelligent Cloud Services Intelligent Structure Discovery.
- Import an existing Intelligent Structure Discovery `.amodel` complex data type definition.
- Create a complex data type definition using a representative sample file in the Developer tool.

The complex data type definitions are stored in the type definition library that is a Model repository object. The default name of the type definition library is m_Type_Definition_Library.

Note: You cannot manually create a complex data type definition for midstream mapping.

Create a mapping and add mapping objects.

1. Create a mapping, and add Read and Write transformations.
2. Create a Read transformation to read the hierarchical data from the source string.
3. Create a Write transformation to write the hierarchical data to a target struct.
4. Create an Expression transformation for the PARSE_JSON or PARSE_XML function.
5. Based on the mapping logic, add other transformations that are supported on the run-time engine.

Create and configure ports in transformations.

1. Create the Read ports including the string Type that contains the hierarchical data.
2. Create the Write ports including the struct Type that contains the parsed hierarchical data.

3. Create the Expression ports:
   Configure the input string as input and output.

   Configure the output struct as output. The Type Definition must reference the complex data type definition you created or imported. Configure the PARSE_JSON or PARSE_XML function for the expression.

Configure the transformations.

Link the ports and configure the transformation properties based on the mapping logic.

Configure the mapping properties.

Configure the mapping run-time properties: choose the Spark validation environment and Hadoop as the execution environment.

Validate and run the mapping.

1. Validate the mapping to identify and correct any errors.
2. Optionally, view the engine execution plan to debug the logic.
3. Run the mapping.

Rules and Guidelines for Midstream Parsing

Consider the following rules and guidelines when you work with midstream parsing:

- For optimal performance, the size of hierarchical data to be parsed should be 5 MB or less.
- If the defined schema does not match the data, the parser ignores the unrecognized field and does not fail the mapping job.
- Dynamic complex ports are not supported with midstream mapping.
- Intelligent Structure Discovery converts Boolean data to a string in the intelligent structure model. Use a string field in the midstream mapping for any Boolean data.
- The midstream mapping complex functions cannot parse Avro binary boolean data types.

Troubleshooting Midstream Parsing

Consider the following troubleshooting tips when you use midstream mappings to parse hierarchical data:

The midstream mapping did not parse the complex data completely.

If the parser encounters a data field that it cannot parse, the parser ignores the unrecognized field and does not fail the mapping job.

The PARSE_JSON and PARSE_XML complex functions ignore data in the following situations:

- The JSON or XML structure is invalid.
- The JSON or XML source file contains invalid or unrecognized data.
- The data does not match the schema provided in the type definition library.
- The complex data schema of the source string changes. For example, a software update on the server might change automated JSON or XML input.
- The wrong element is selected as the root, which can cause the parser to flatten parent data in an array.

To troubleshoot unparsed data, capture the data from the source string and save it in an `UnassignedData` array as an `unidentifiedDataItem`. Then, use this information to analyze the data.

Contact Informatica Global Customer Support and refer to the internal KB 627032.

Note: This debugging tool can have an impact on system performance.

The midstream mapping loses data when parsing arrays.

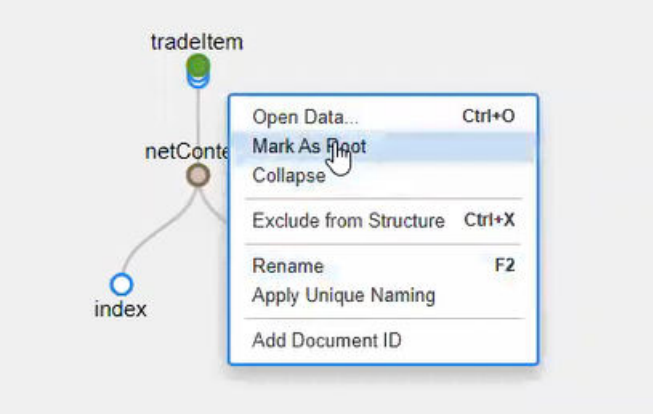
Data loss occurs when the PARSE_JSON or PARSE_XML functions parse a schema that contains an array or a multidimensional array with struct elements. The resulting data loss depends on the array type. For example:

- Array with struct elements. The parser function returns only the first element of an array. The remaining array elements are ignored.
- Multidimensional array with struct elements. The parser function returns a struct instead of an array, for an intermediate array element.

When you create the intelligent structure model using sample data, Intelligent Structure Discovery selects an intermediate element as the root node. For example, when an array contains struct elements, Intelligent Structure Discovery selects the first struct element as the root node. This behavior causes data loss for midstream parsing. To avoid data loss, select the top-level node as root.

Use Informatica Intelligent Cloud Services Intelligent Structure Discovery to modify the intelligent structure model.

Select the top-level node, and then right-click and select **Mark As Root**. The following image shows how to modify the root node:



Note: The brown-colored node shows the current root node. The green-colored node shows the selected root node.

For more information, see the "Intelligent structure models" section of the Informatica Intelligent Cloud Services Data Integration help.

The midstream mapping does not work when the data or field names contain special characters.

The input data is encoded as a UTF-8 string. Check the input flat file data object configuration in the Developer tool.

In the Object Explorer view, navigate to the flat file data object in the Physical Data Objects **Advanced tab** > **Format** section and set the **Code page** attribute to `UTF-8`.

CHAPTER 13

Hierarchical Data Processing Configuration

This chapter includes the following topics:

- Hierarchical Data Conversion, 246
- Convert Relational or Hierarchical Data to Struct Data, 247
- Convert Relational or Hierarchical Data to Nested Struct Data, 249
- Extract Elements from Hierarchical Data, 257
- Flatten Hierarchical Data, 259

Hierarchical Data Conversion

In the Developer tool, use **Hierarchical Conversion** wizards that simplify the tasks for developing a mapping to process hierarchical data on the Spark engine. These wizards add one or more transformations to the mapping and create output ports. The expressions in the output ports use complex functions and operators to convert relational data to hierarchical data or hierarchical data to relational data.

The following table lists the hierarchical conversion wizards and when to use these wizards:

Hierarchical Conversion Wizard	When to Use
Create Struct Port	To convert relational and hierarchical data to struct data. For example, you want to convert data in one or more columns of any data type to hierarchical data column of type struct.
Create Nested Complex Port	To convert relational and hierarchical data in two tables to a nested struct data. You can select one or more columns of any data type. For example, you want to convert data in columns from two tables to a hierarchical data column of type struct with a nested schema. The wizard creates a struct with an array of structs.

Hierarchical Conversion Wizard	When to Use
Extract from Complex Port	To convert hierarchical data to relational data or modify the hierarchical data. For example, you want to perform the following conversions: - Convert one or more elements of primitive data types in a hierarchical data to relational data columns. - Convert one or more elements of complex data types in a hierarchical data to hierarchical data columns.
Flatten Complex Port	To convert hierarchical data to relational data. For example, you want to convert one or more elements of a hierarchical data to relational data columns.

Convert Relational or Hierarchical Data to Struct Data

You can convert relational or hierarchical data in one or more columns to hierarchical data of type struct. Use the **Create Struct Port** wizard in the Developer tool to perform the conversion.

For example, a relational table contains three columns city, state, and zip. You create a mapping to convert the data in the three columns to one hierarchical column. Select the three ports in the transformation and use the Create Struct Port wizard to generate struct data with the selected ports as its elements.

The wizard performs the following tasks:

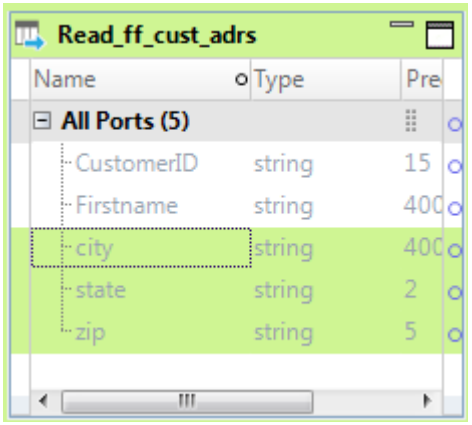
- Creates a complex data type definition based on the ports that you select.
- Adds an Expression transformation to the mapping.
- Creates a struct output port to represent the struct data.
- Creates an expression that uses the STRUCT_AS function to generate struct data.

Creating a Struct Port

Use the **Create Struct Port** wizard to convert data that passes through one or more ports to struct data.

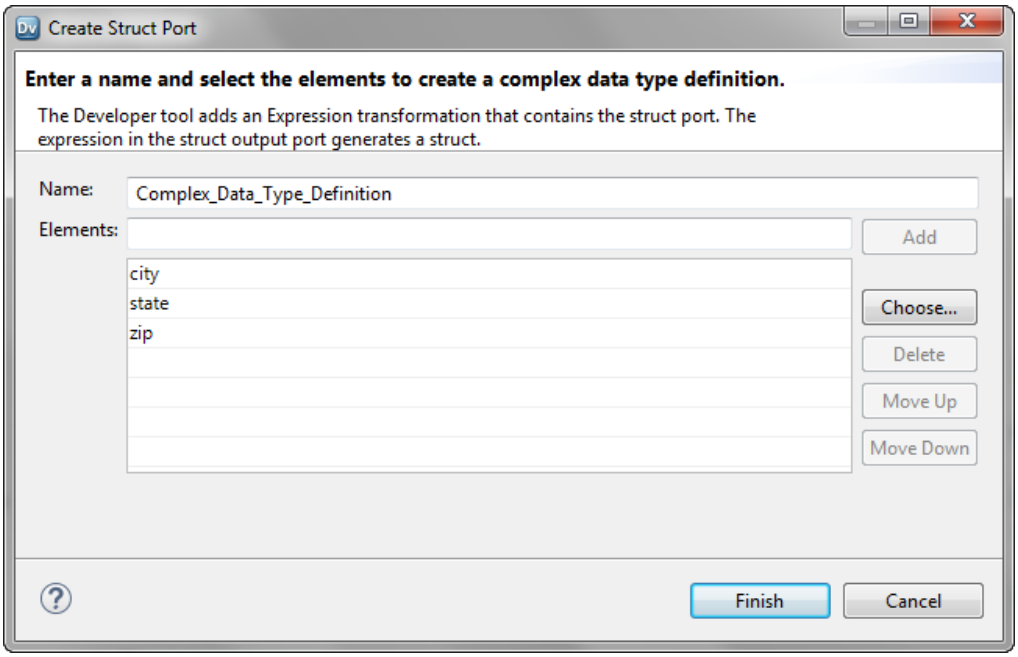
1. In the transformation, select one or more ports that you want to convert as elements of the struct data.

   The ports you select also determine the elements of the complex data type definition.



2. Right-click the selected ports, and select **Hierarchical Conversions** > **Create Struct Port**.

The **Create Struct Port** wizard appears with the list of ports that you selected.



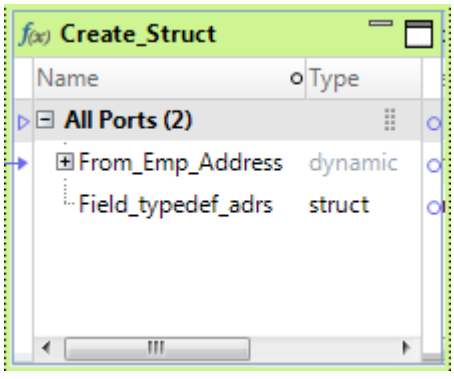
3. Optionally, in the **Name** box, change the name of the complex data type definition.

   For example, typedef_address.

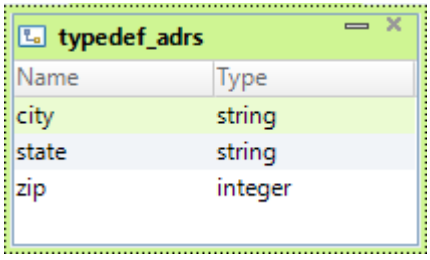
4. Optionally, click **Choose** to select other ports in the transformation.
5. Click **Finish**.

You can see the following changes in the mapping:

- The mapping contains a new Expression transformation `Create_Struct` with a struct output port and a dynamic port with ports from the upstream transformation.



- The type definition library contains the new complex data type definition.



- The struct output port references the complex data type definition.
- The struct output port contains an expression with the STRUCT_AS function. For example,

```
STRUCT_AS(:Type.Type_Definition_Library.typedef_address,city,state,zip)
```

Convert Relational or Hierarchical Data to Nested Struct Data

You can convert relational or hierarchical data in one or more columns in two transformations to nested struct data. Use the **Create Nested Complex Port** wizard in the Developer tool to perform the conversion.

The wizard converts relational data from two tables to struct data with a nested data type definition.

For example, a relational table contains employee bank account details. Another table contains the bank details. You create a mapping to convert data in the two tables into a hierarchical format that the payment system can process. Select the ports in the two transformations and use the Create Nested Complex Port wizard. The wizard generates struct data that contains an array element for each bank. The array contains a struct element for the employee bank account details.

Relational input

The Emp_Details table contains the following columns: employee ID, name, address, bank id, and bank account number.

The Bank_Details table contains the following columns: bank id, bank name, SWIFT code.

Struct output

```
banks{
bank_swift:integer
[bank_name{account_number:integer,employee_id:integer,name:string,address:string]
}
```

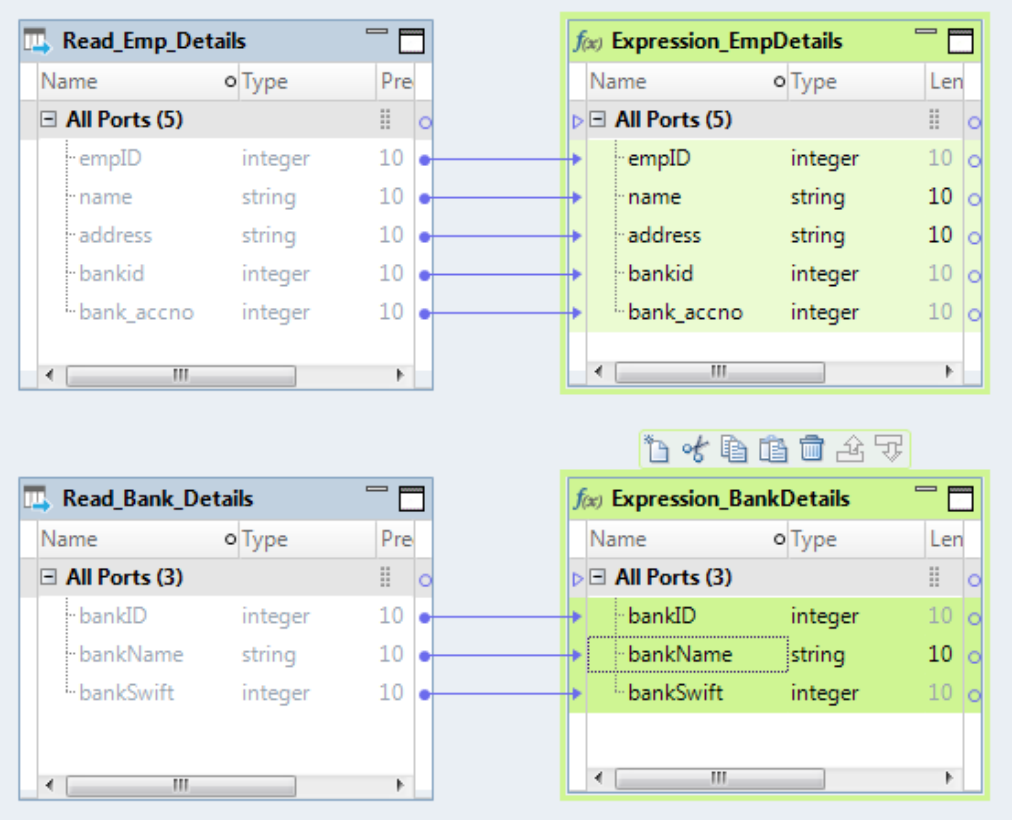
The wizard performs the following tasks:

- Adds a Joiner transformation to join source data from the parent and child tables based on the join keys. The transformation that contains ports for the parent complex data type definition is the parent transformation, and the other transformation is the child transformation.
- Creates a child complex data type definition and adds an Expression transformation that contains the child struct port.
  The expression in the struct port generates a struct based on the child complex data type definition.
- Adds an Aggregator transformation that contains an array port.
  The expression in the array port generates an array with the child struct as its element, and groups the struct values based on the group by port.
- Creates a parent complex data type definition and adds an Expression transformation that contains the nested complex port.
  The expression in the nested complex port generates a struct with an array of structs as one of its elements.

Creating A Nested Complex Port

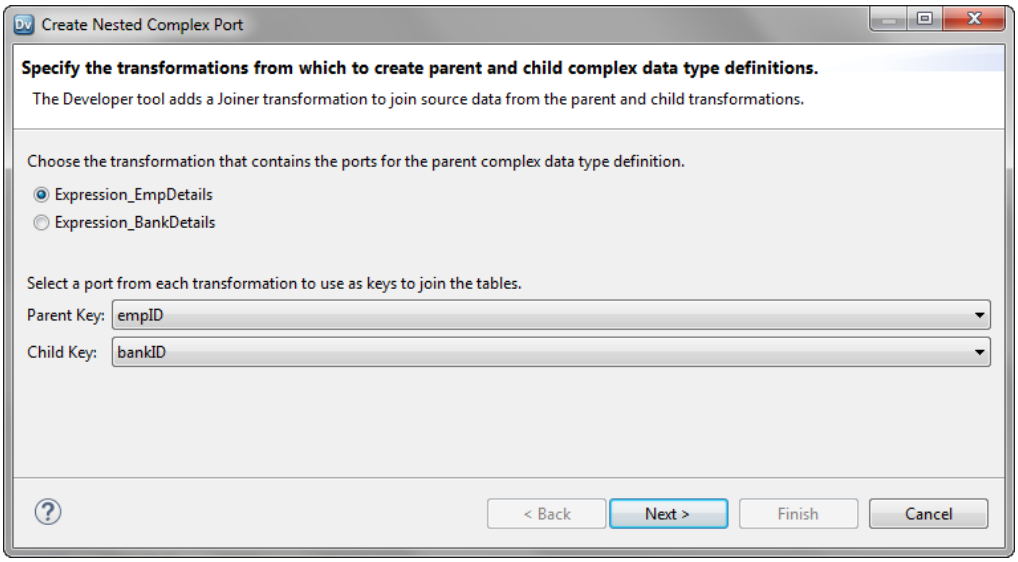
Use the **Create Nested Complex Port** wizard to convert data that passes through one or more ports in two transformations to a struct data that references a nested data type definition. You specify the transformations from which the wizard creates the parent and child complex data type definitions.

1. Open the mapping or mapplet in which you want to create a nested complex port.
2. Press Ctrl and select the ports in the two transformations that you want to convert as elements of the nested complex port.

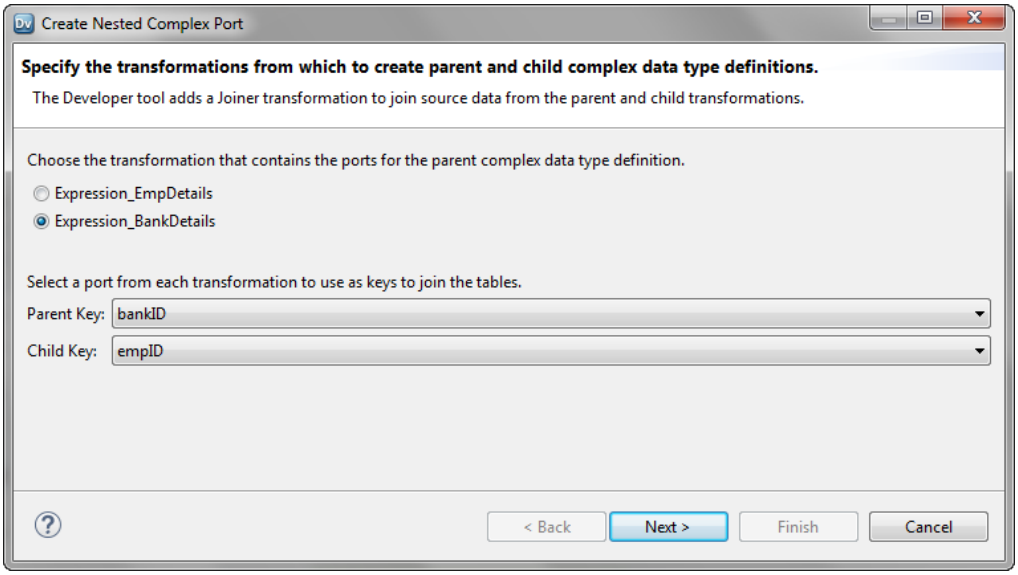


3. Right-click the selected ports, and select **Hierarchical Conversions** > **Create Nested Complex Port**.

The **Create Nested Complex Port** wizard appears.

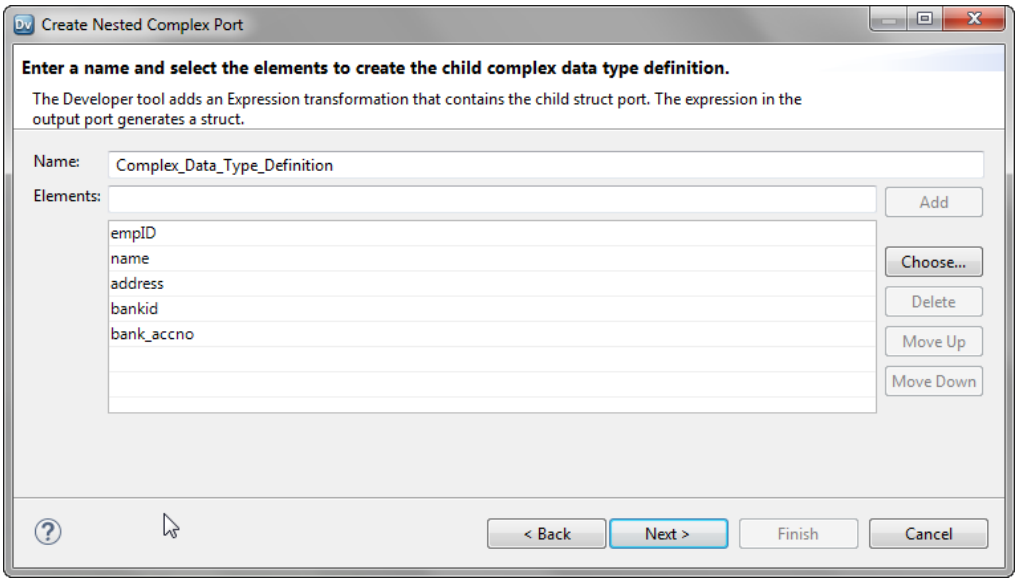


4. Choose the transformation that contains ports for the parent complex data type definition and select the join keys to join the tables.

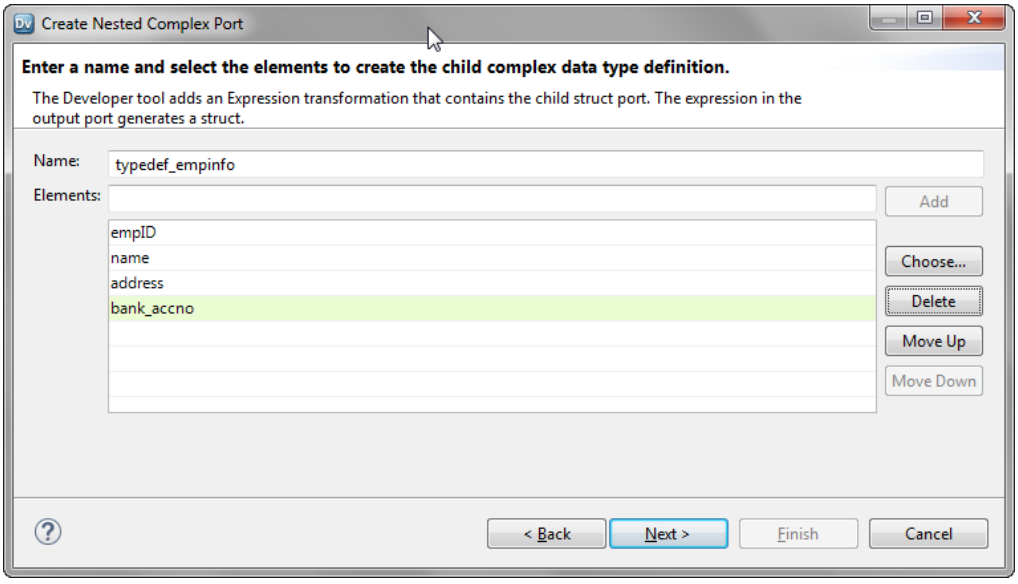


5. Click **Next**.

The wizard page to create child complex data type definition appears.

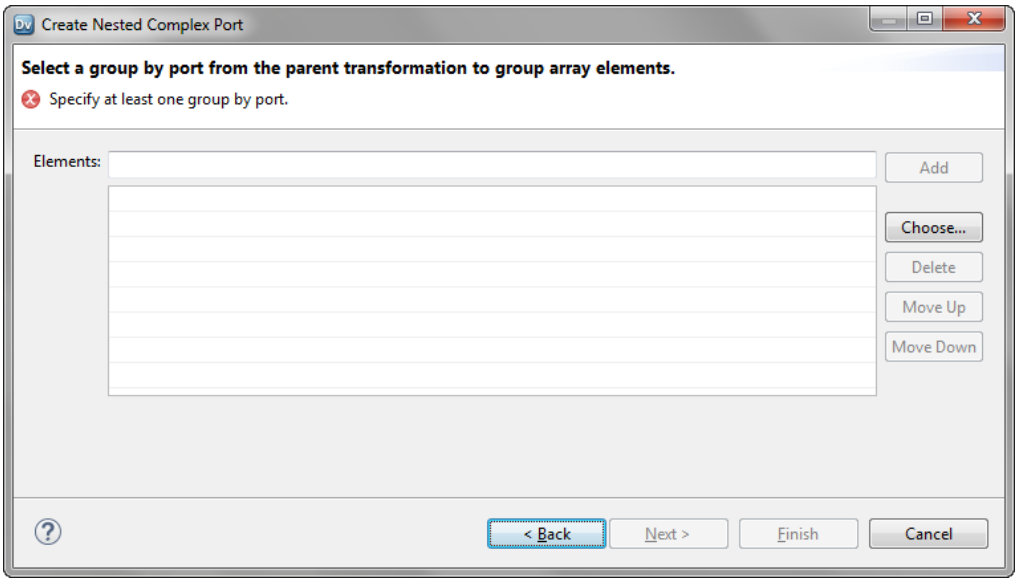


6. Optionally, change the name of the child complex data type definition and make any changes to the elements.



7. Click **Next**.

The wizard page to select group by port from the parent transformation appears.

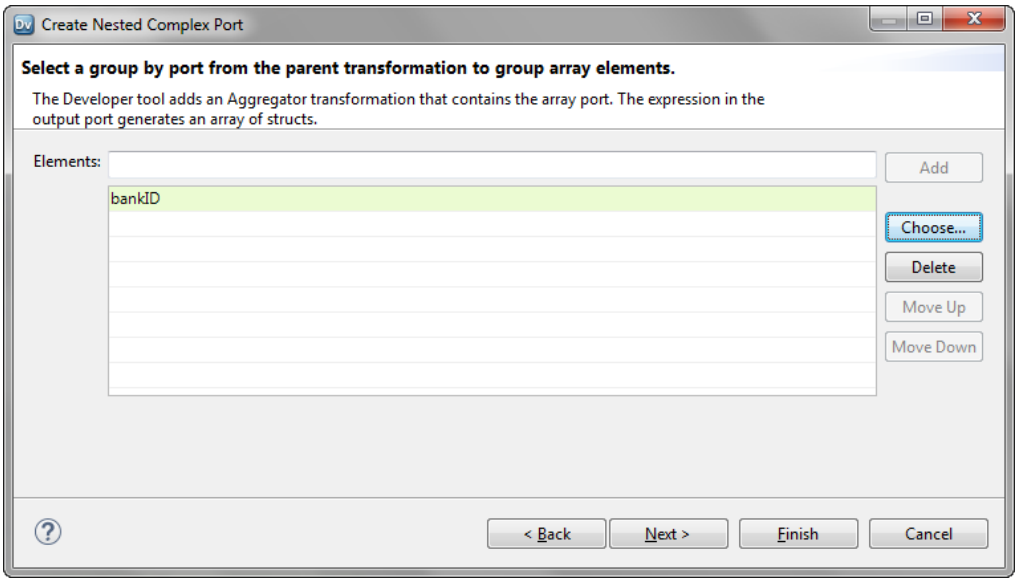


8. Click **Choose**.

   The **Ports** dialog box appears.

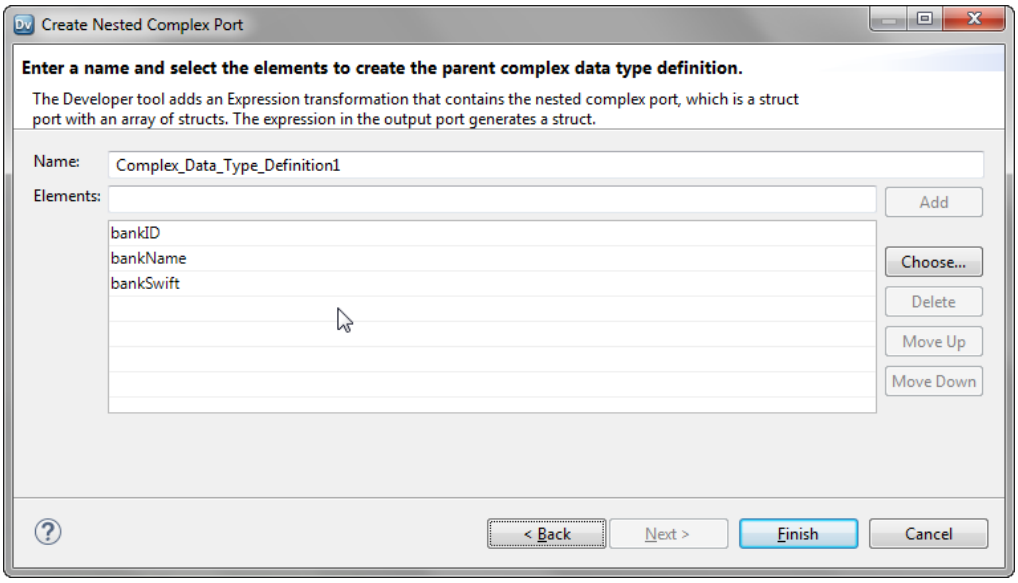
9. Select a group by port from the parent transformation and click **OK**.

   The selected port appears on the wizard page.



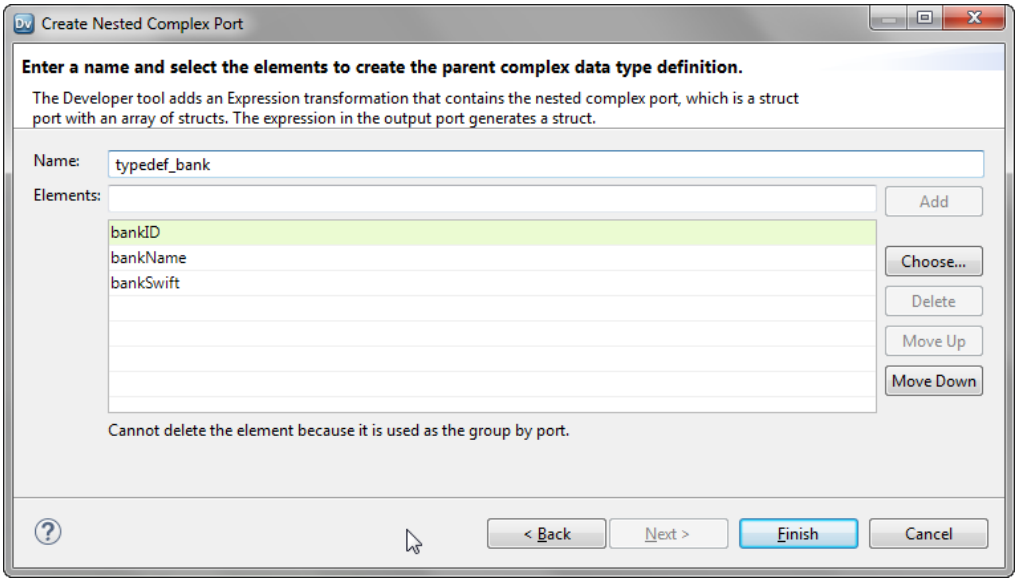
10. Click **Next**.

The final wizard page to create parent complex data type definition and nested complex port appears.



11. Optionally, change the name of the parent complex data type definition and make any changes to the elements.

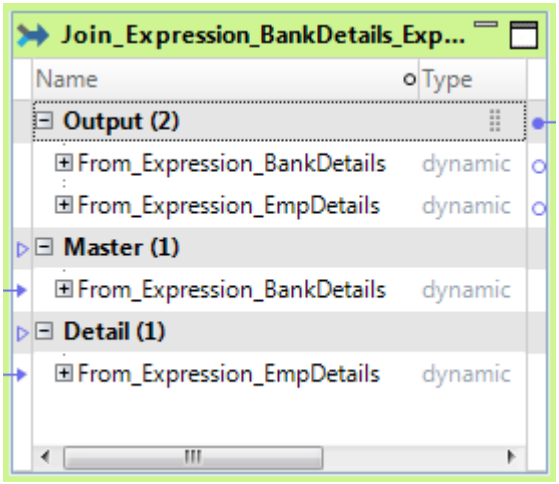
    You cannot delete the port that you selected as the group by port.



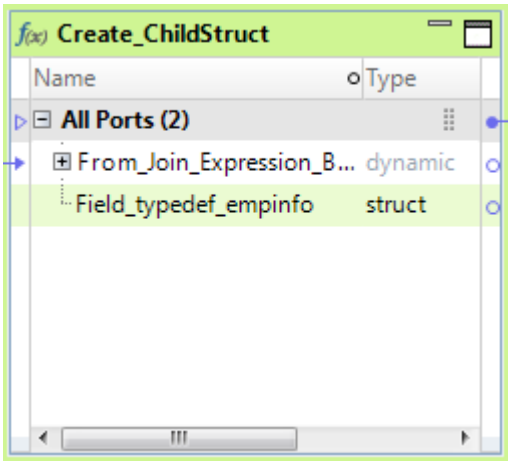
12. Click **Finish**.

You can see the following changes in the mapping:

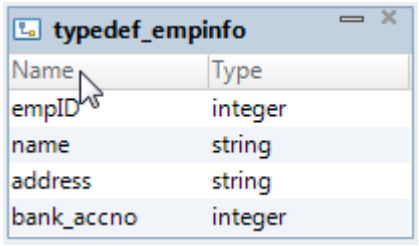
- The mapping contains a new Joiner transformation `Join_<transformation1>_<transformation1>` that joins the two tables.



- The mapping contains a new Expression transformation `Create_ChildStruct` with a struct output port and a dynamic port with ports from the upstream transformation.



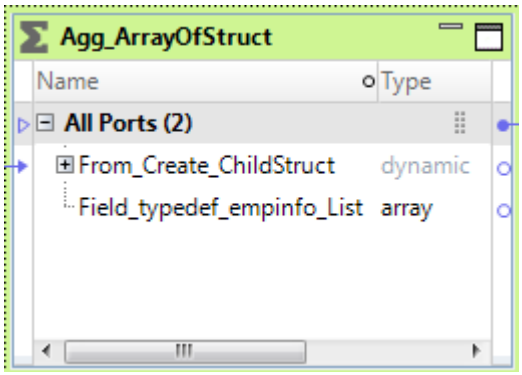
- The type definition library contains the new child complex data type definition. The struct output port references the child complex data type definition.



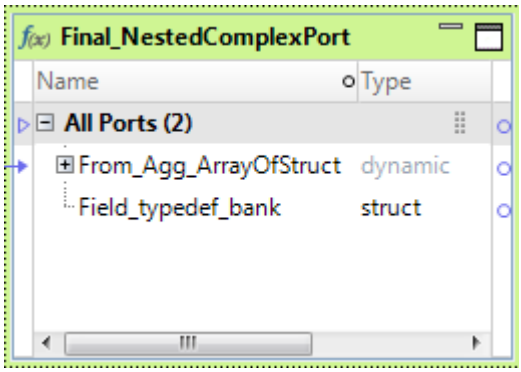
The output port contains an expression with the STRUCT_AS function:

```
STRUCT_AS(:Type.Type_Definition_Library.<child_typedef>,<comma_delimited_child_elements>)
```

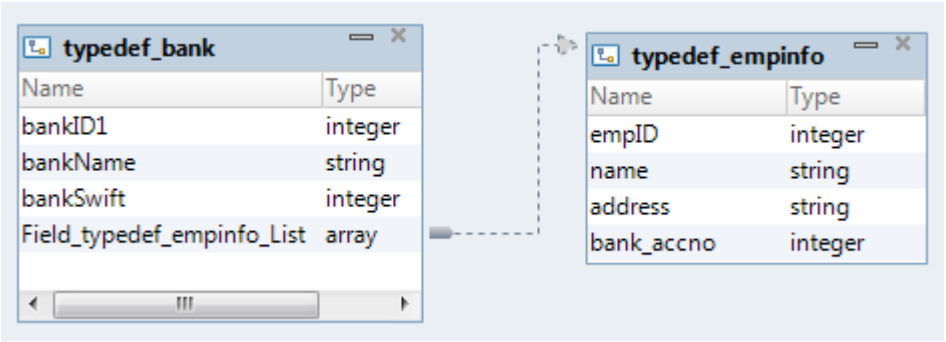
- The mapping contains a new Aggregator transformation `Agg_ArrayOfStruct` with the group by port to group the child structs into an array.



- The mapping contains a new Expression transformation `Final_NestedComplexPort` with a struct output port and a dynamic port with ports from the upstream transformation.



- The type definition library contains the new parent complex data type definition, which is a nested data type definition. The struct output port references the parent complex data type definition.



The output port contains an expression with the STRUCT_AS function:

```
STRUCT_AS(:Type.Type_Definition_Library.<parent_typedef>,<comma_delimited_struct_elements>)
```

Extract Elements from Hierarchical Data

You can extract elements of a primitive or complex data type from hierarchical data. Use the **Extract from Complex Port** wizard in the Developer tool to perform the conversion.

Based on the data type of the elements in the complex port that you select, the wizard performs the following conversions:

- If the elements are of primitive data types, the wizard converts the element to a relational data.
- If the elements are of complex data types, the wizard converts the element to a hierarchical data.

The wizard adds an Expression transformation that contains one or more extracted ports. The number of elements that you select to extract determine the number of output ports in the transformation. The data type of the element determines the data type of the output port. The expression in the output ports use complex operators to extract elements from the complex port.

The following table describes the wizard conversions based on the data type of the complex port that you select:

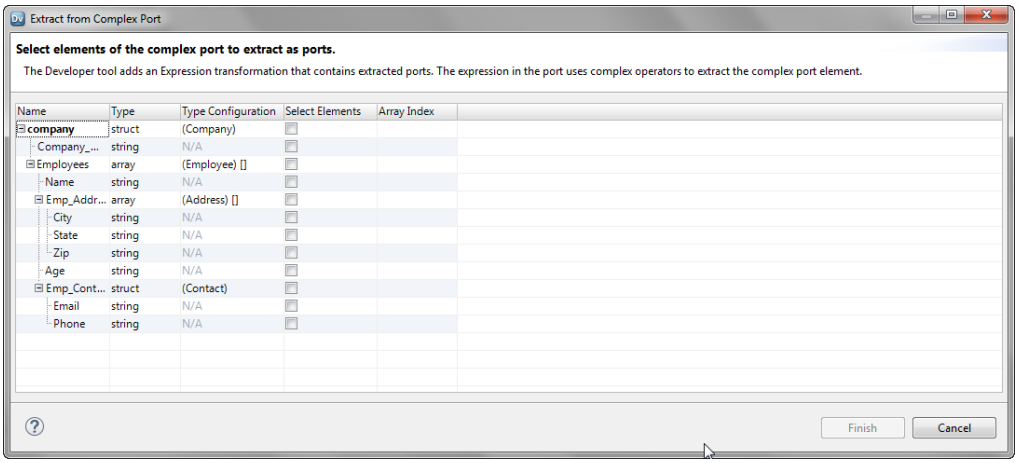
Complex Port Data Type	Wizard Conversion
array	Relational data. If you specify an array index, the wizard extracts an element in the array. Default is 0. The wizard extracts the first element of the array. Hierarchical data. If you do not specify an array index, the wizard extracts the entire array.
struct	Relational data. The wizard extracts the element in the struct that you selected.
array of structs	Relational data. If you select an element in the struct, the wizard extracts the element. The wizard requires an array index value. Default is 0. Hierarchical data. If you select the array and specify an array index, the wizard extracts the struct element in the array. If you do not specify an array index, the wizard extracts the entire array.
array of maps	Relational data. If you select the key or value element in the array and specify an array index, the wizard extracts the element. Hierarchical data. If you select the array and specify an array index, the wizard extracts the map element in the array. If you do not specify an array index, the wizard extracts the entire array.
struct of structs	Relational data. If you select an element in the parent or the child struct, the wizard extracts the element. Hierarchical data. If you select the parent struct or child struct, the wizard extracts that struct.
struct of maps	Hierarchical data. If you select the map element, the wizard extracts the map data. If you select the struct, the wizard extracts the struct data in another port.

Extracting Elements from a Complex Port

Use the **Extract from Complex Port** wizard to extract elements from hierarchical data.

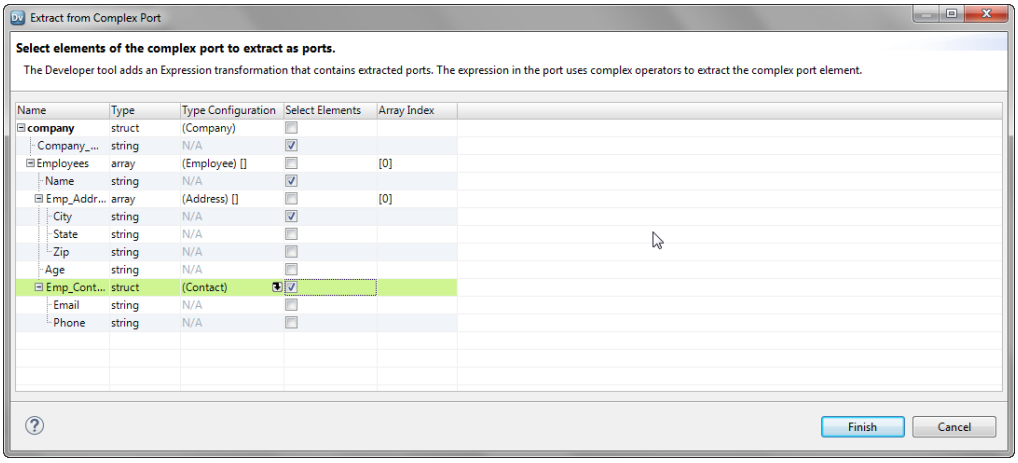
1. In the transformation, select a complex port from which you want to extract elements.
2. Right-click the selected ports, and select **Hierarchical Conversions** > **Extract from Complex Port**.

The **Extract from Complex Port** wizard appears with the list of elements in the complex port.



3. In the **Select Elements** column, select the check box for each element that you want to extract.
4. To extract an array element, specify an array index in the **Array Index** column.

   If you delete the array index, the wizard extracts the entire array. To extract an element in the struct from a complex port for an array of structs, you must specify an array index value.

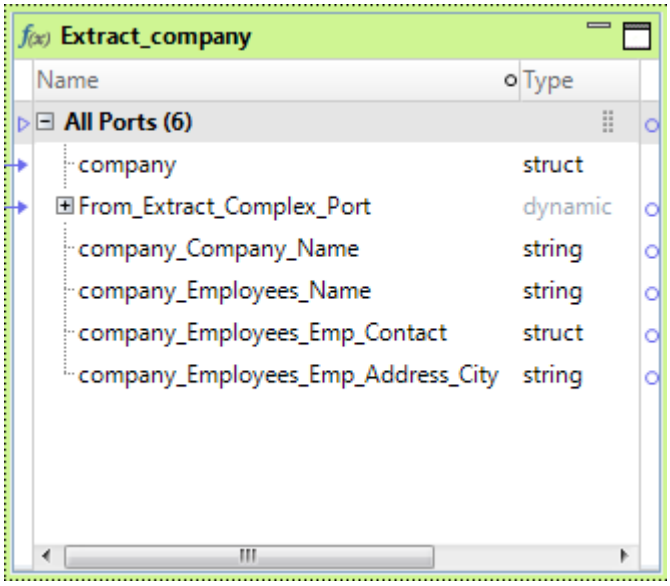


5. Click **Finish**.

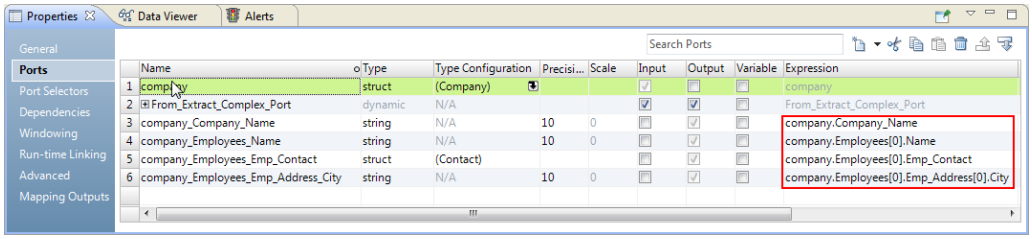
You can see the following changes in the mapping:

- The mapping contains a new Expression transformation `Extract_<complex_port_name>` with the following ports:
  - The complex port from which you want to extract elements as the input port.
  - One or more output ports for the extracted elements. The number of elements that you selected to extract determines the number of output ports in the transformation.

- A dynamic port with ports from the upstream transformation.



- The output ports contain expressions that use complex operators to extract elements.
  The following image shows the expressions for the output ports in the Expression column on the Ports tab:



Flatten Hierarchical Data

You can flatten elements of hierarchical data into relational data. Use the **Flatten Complex Port** wizard in the Developer tool to perform the conversion.

The wizard converts hierarchical data to relational data. When you have hierarchical data with nested data type, you can select specific elements or all elements of complex data type to flatten.

Based on the data type of the complex port, the wizard performs the following tasks:

struct

- Adds an Expression transformation with flattened output ports. The expression for the output ports uses the dot operator to extract elements in the struct.
- Adds a final Expression transformation that contains a dynamic port with all ports from the upstream transformation including the flattened struct ports.

array

Adds a Normalizer transformation with flattened output ports. The wizard flattens the array field in the Normalizer view.

Adds a final Expression transformation that contains a dynamic port with all ports from the upstream transformation including the flattened array ports.

nested data type

Adds one or more Expression and Normalizer transformations with flattened output ports. If you select a child element of a nested complex port, the wizard flattens both the parent and child elements.

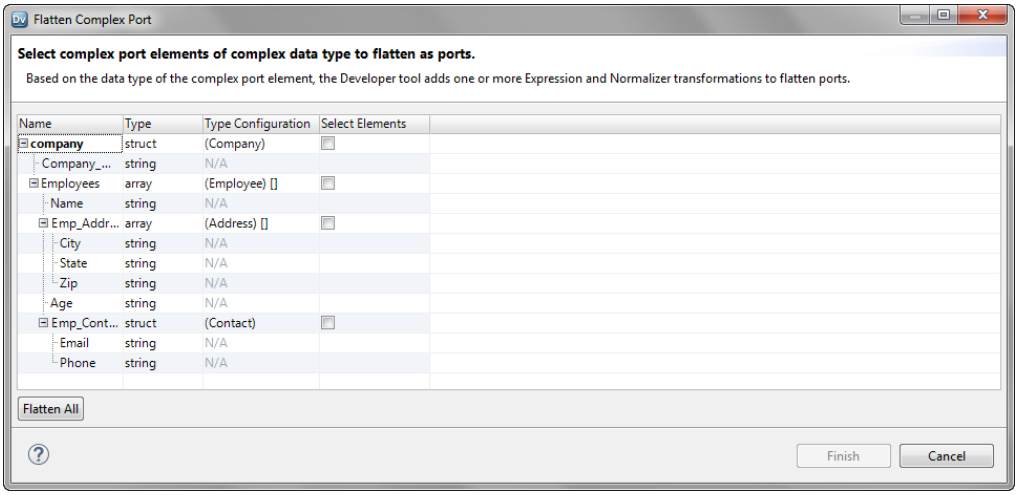
Adds a final Expression transformation that contains a dynamic port with all ports from the upstream transformation including the flattened ports.

Flattening a Complex Port

Use the **Flatten Complex Port** wizard to convert hierarchical data to relational data.

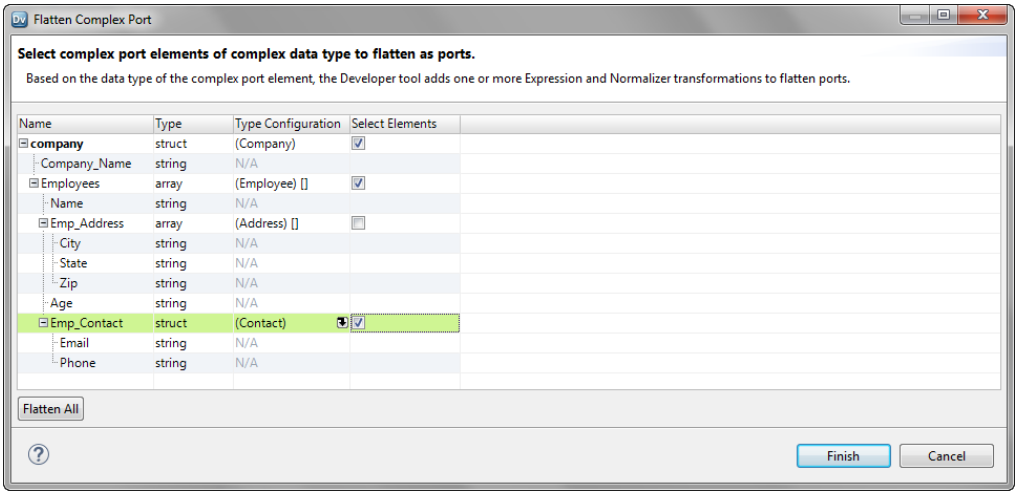
1. In the transformation, select a complex port that you want to flatten.
2. Right-click the selected ports, and select **Hierarchical Conversions** > **Flatten Complex Port**.

   The **Flatten Complex Port** wizard appears with the list of elements in the complex port.



3. In the **Select Elements** column, select the check box for each element of a struct or an array data type that you want to extract.

   If you select a child element of a nested complex port, the wizard automatically selects the parent element to flatten.



4. Optionally, click **Flatten All** to flatten all elements of struct or data type in the complex port.

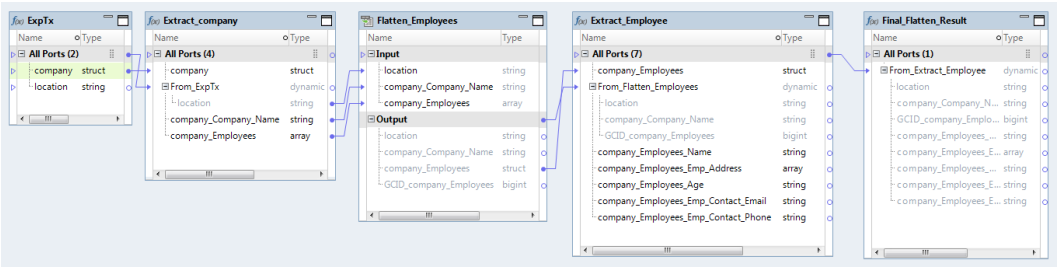
   The wizard selects all elements of struct or data type in the complex port to flatten.

5. Click **Finish**.

You can see the following changes in the mapping:

- For each struct element that you selected to flatten, the mapping contains a new Expression transformation `Extract_<element_of_struct_type>` with the following ports:
  - The complex port from which you want to extract elements.
  - One or more output ports for the extracted elements.
    The output ports contain expressions that use complex operators to extract elements.
  - A dynamic port with ports from the upstream transformation
- For each array element that you selected to flatten, the mapping contains a new Normalizer transformation `Flatten_<element_of_array_type>` with an input group and an output group. The output group contains the flattened normalized fields for the array element.
- A final Expression transformation `Final_Flatten_Result` with a dynamic port that contains the flattened ports for the complex port and other ports in the upstream transformation.

The following image shows an example of mapping changes:



CHAPTER 14

Hierarchical Data Processing with Schema Changes

This chapter includes the following topics:

- Overview of Hierarchical Data Processing with Schema Changes, 262
- How to Develop a Dynamic Mapping to Process Schema Changes in Hierarchical Data, 263
- Flatten Hierarchical Data with Schema Changes, 264
- Dynamic Complex Ports, 265
- Input Rules for a Dynamic Complex Port, 267
- Port Selectors for Dynamic Complex Ports, 270
- Dynamic Expressions, 270
- Complex Operators, 272
- Complex Functions, 272
- Rules and Guidelines for Dynamic Complex Ports, 272
- Optimized Mappings, 272

Overview of Hierarchical Data Processing with Schema Changes

You can use dynamic complex ports to manage schema changes to hierarchical data. You use dynamic ports to receive new and changed hierarchical data columns from a dynamic complex file source. Then, to manage schema or metadata changes of a complex port, you use dynamic complex ports.

To process hierarchical data on the Spark engine, you use complex ports in transformations. To handle metadata changes at run time, you create dynamic mappings with dynamic ports. A dynamic port can propagate hierarchical data through a generated port of a complex data type. However, if the schema of the hierarchical data changes at run time, you must use dynamic complex ports to pass hierarchical data in the mapping.

Complex ports require you to specify the type configuration based on the data type of the elements in an array or a map or the schema of a struct. With dynamic complex ports, you do not have to specify the type configuration for a hierarchical column. For example, you do not have to specify a complex data type definition for a dynamic struct.

With dynamic complex ports, you can perform the following tasks in a dynamic mapping:

- Receive new or changed elements of a complex port if the schema of the complex port changes at run time.
- Modify hierarchical data or filter elements of hierarchical data with input rules for a dynamic complex port.
- Extract elements of hierarchical data that might have schema changes at run time using complex operators.
- Generate arrays and structs with dynamic expressions that use complex functions.

For more information about dynamic mappings, see the *Informatica Developer Mapping Guide*.

Note: You cannot perform midstream parsing in a dynamic mapping.

How to Develop a Dynamic Mapping to Process Schema Changes in Hierarchical Data

To process hierarchical data with schema changes on the Spark engine, develop a dynamic mapping with dynamic complex sources and targets, dynamic ports, complex ports, dynamic complex ports, complex operators and functions, port selectors, and dynamic expressions.

Note: The tasks and the order in which you perform the tasks to develop a dynamic mapping depend on the mapping scenario.

The following list outlines the high-level tasks to develop and run a mapping to read, write, and process hierarchical data that can change at run time.

Create dynamic complex sources and targets.

If a complex file source changes, you can configure the Read transformation to accommodate changes. For example, you can configure the Read transformation to use a different data source or to update the data object based on the data source. If a complex file target changes, you can configure the Write transformation to accommodate target changes. For example, you can configure the Write transformation to generate columns based on the mapping flow.

You can configure a Read or a Write transformation to get columns from a complex file data object at run time.

Create dynamic ports and define input rules.

When you configure transformations in a mapping, you create dynamic ports to receive new or changed columns based on the data flow. You configure input rules to determine the columns that a dynamic port receives and to rename or reorder the generated ports.

Create dynamic complex ports and define input rules.

Create a dynamic complex port to handle changes to the schema of a complex port. Configure input rules to determine the schema or element types of a dynamic complex port. Use a user rename on the upstream dynamic field.

Create port selectors and dynamic expressions for dynamic complex output ports.

Create a dynamic expression by using dynamic ports or port selectors in the expressions. When you include a dynamic port, the expression generates a complex data type with elements based on each port that the dynamic port generates. When you include a port selector, the expression generates a complex data type with elements based on each port determined by the selection rule.

Create parameters.

You can use parameters to change values at run time. Use parameters to change values such as sources, targets, connections, and rules within the mapping.

Create run-time links.

Transformations might change in such a way that you cannot create direct links when you design the mapping. If you cannot create links at design time, you can configure run-time links.

Flatten Hierarchical Data with Schema Changes

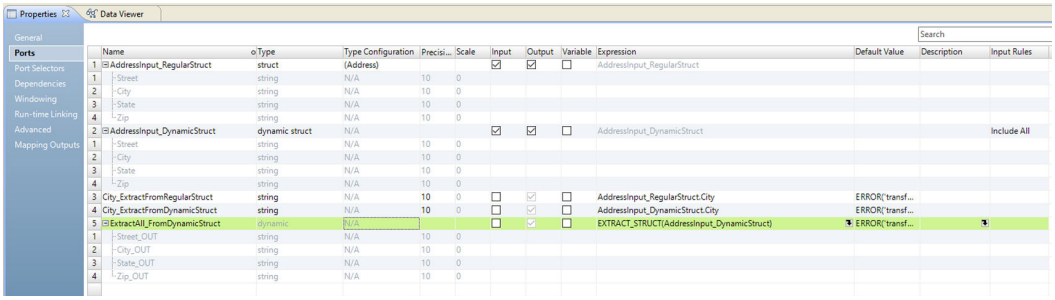
You can flatten dynamic arrays, maps, and structs with schema changes into relational data. Use the flatten action in the Normalizer transformation to flatten dynamic arrays and maps. Use the EXTRACT_STRUCT function in dynamic expressions to extract all elements from a dynamic struct port in an Expression transformation.

Flatten a Dynamic Struct

Use the EXTRACT_STRUCT function to flatten dynamic structs in an Expression transformation.

Use the dot operator in expressions to extract elements of a struct.

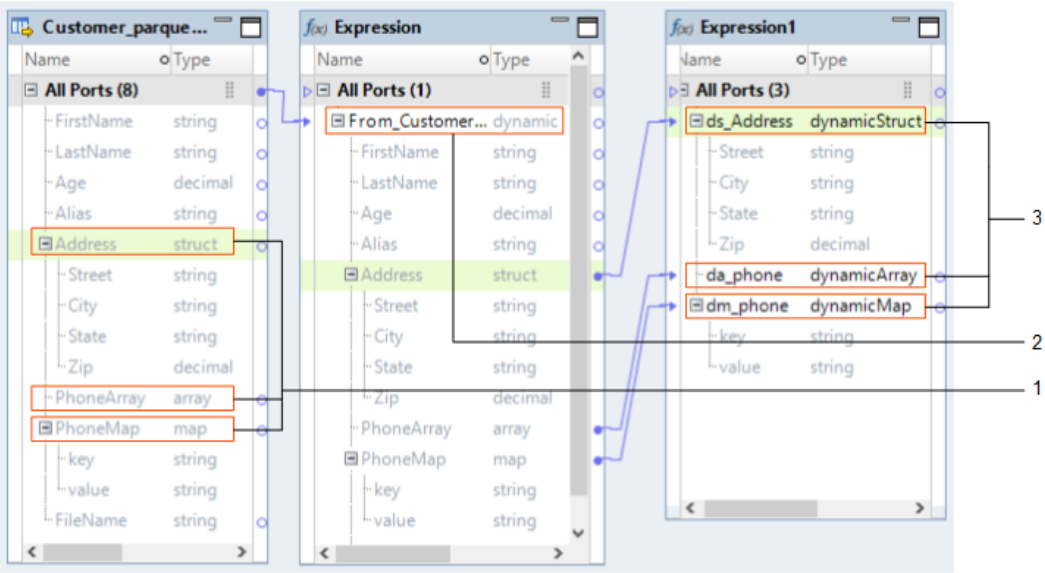
For example, you can use the EXTRACT_STRUCT function to extract all data from an address without specifying each address field individually.



Dynamic Complex Ports

A dynamic complex port receives one or more elements of a complex port or a hierarchical data column based on input rules. You can use dynamic complex ports such as dynamic array, dynamic map, and dynamic struct in some transformations on the Spark engine.

The following image shows the ports that you can use in a mapping to process hierarchical data with schema changes:



1. Complex port
2. Dynamic port
3. Dynamic complex port

Complex port

A port of a complex data type in a transformation that can pass or process hierarchical data. Array, map, and struct ports are complex ports. Based on the complex data type, you specify the complex port properties as the type configuration.

Dynamic port

A port in a transformation that can receive one or more columns from an upstream transformation. Dynamic ports can receive new or changed columns as generated ports based on the metadata that passes through the mapping. The input rules for a dynamic port determine the generated ports.

Dynamic complex port

A port in a transformation that can pass or process hierarchical data with new or changed elements based on the schema changes of the hierarchical data. The input rules for a dynamic complex port determine the data type of the hierarchical data elements or the schema of the hierarchical data.

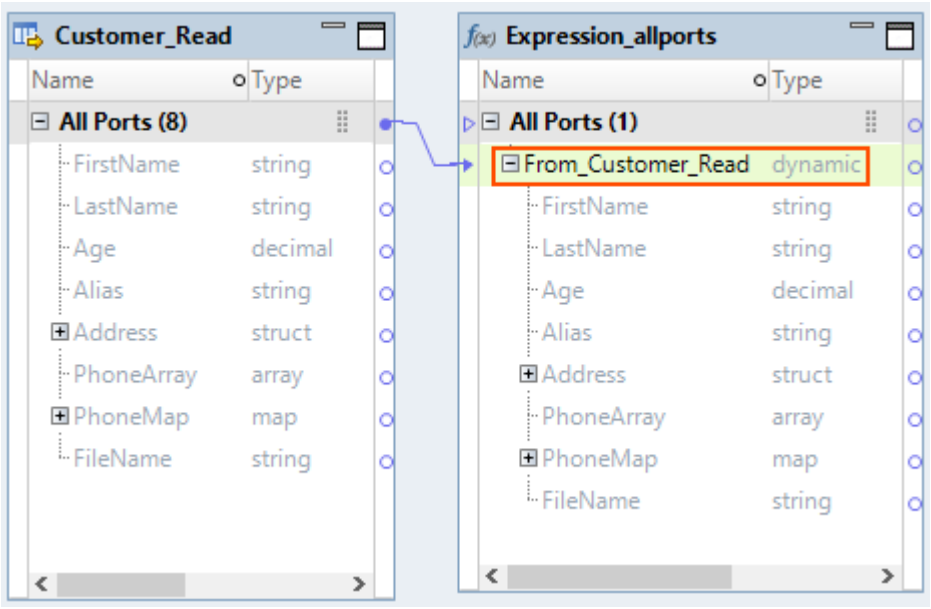
Dynamic Ports and Dynamic Complex Ports

Dynamic complex ports are different from dynamic ports. A dynamic complex port represents one hierarchical column of a complex data type, whereas a dynamic port represents one or more columns of any data type.

You create dynamic ports to accommodate changes to columns at run time based on the metadata that passes through the mapping. You create dynamic complex ports to accommodate changes to the schema of the hierarchical column at run time.

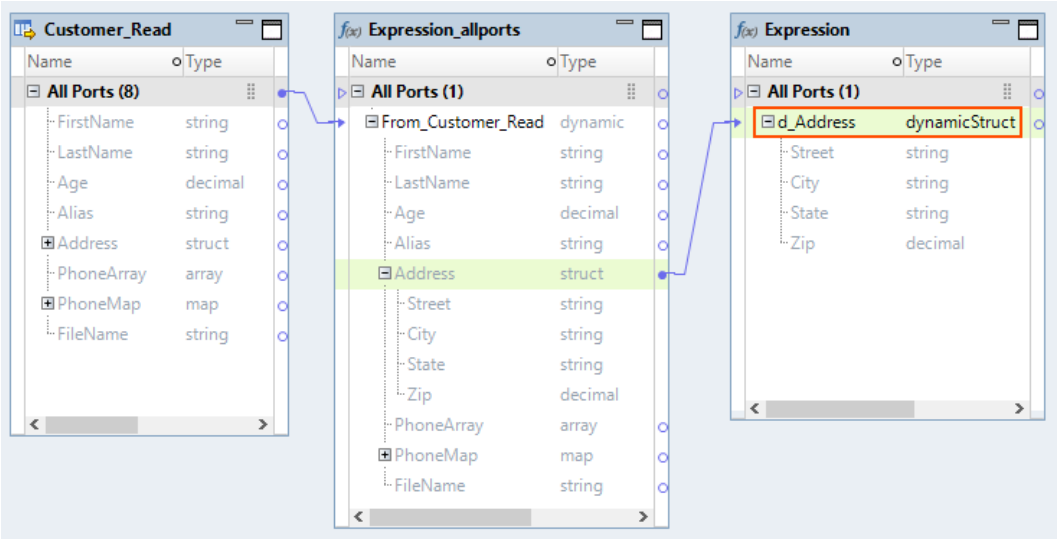
You cannot use dynamic complex ports in a mapplet, whereas you can use dynamic ports in a mapplet. For example, you can add a dynamic port to transformations in a mapplet. But you cannot use dynamic ports in the input and output instances of a mapplet.

The following image shows a dynamic port From_Customer_Read:



The dynamic port From_Customer_Read in the Expression_allports transformation includes generated ports of type string, decimal, struct, array, and map. The generated ports can change at run time based on the columns in the upstream transformation.

The following image shows a dynamic complex port d_Address:



The dynamic struct port d_Address in the Expression transformation includes elements of type string and decimal. The elements of a dynamic complex port can change at run time based on the schema changes to the hierarchical column in the upstream transformation.

Dynamic Complex Ports in Transformations

You can create dynamic complex ports in some transformations on the Spark engine.

The following transformations support dynamic complex ports:

- Aggregator
- Expression
- Filter
- Joiner
- Lookup
- Normalizer
- Rank
- Router
- Sorter

Input Rules for a Dynamic Complex Port

Input rules for a dynamic complex port are one or more rules that determine the data type of the hierarchical data elements or the schema of the hierarchical data. Define input rules to modify hierarchical data that is represented as a complex port in an upstream mapping.

Based on the dynamic complex port, you define the input rules in the corresponding **Input Rules** dialog box as follows:

dynamic array

Select one or more data types for the array elements. The default input rule includes elements of all data types.

dynamic map

Select one or more data types for the key and value of map elements. The default input rules for key and map include elements of all data types.

dynamic struct

Specify which elements to include or exclude in a struct based on the element names or data type. Include all elements is the default input rule.

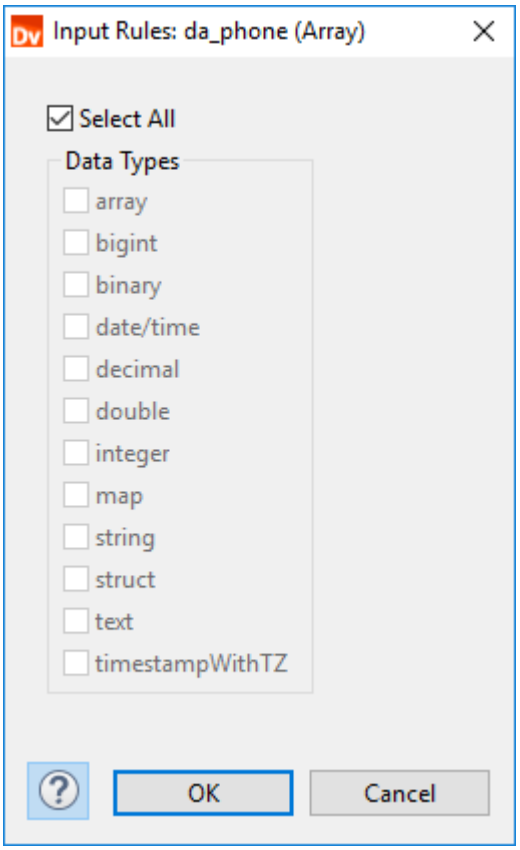
Input Rule for a Dynamic Array

An input rule for a dynamic array determines the data type of the array elements. You can select one or more data types. The default input rule includes elements of all data types.

For example, you develop a dynamic mapping to process two different complex file sources. You expect that an array in one of the complex files is of string elements and an array in the other complex file is of integer elements. Then, you can select string and integer data types as input rules for the dynamic array port.

If you are not certain about the data type of the array elements, you can select all data types.

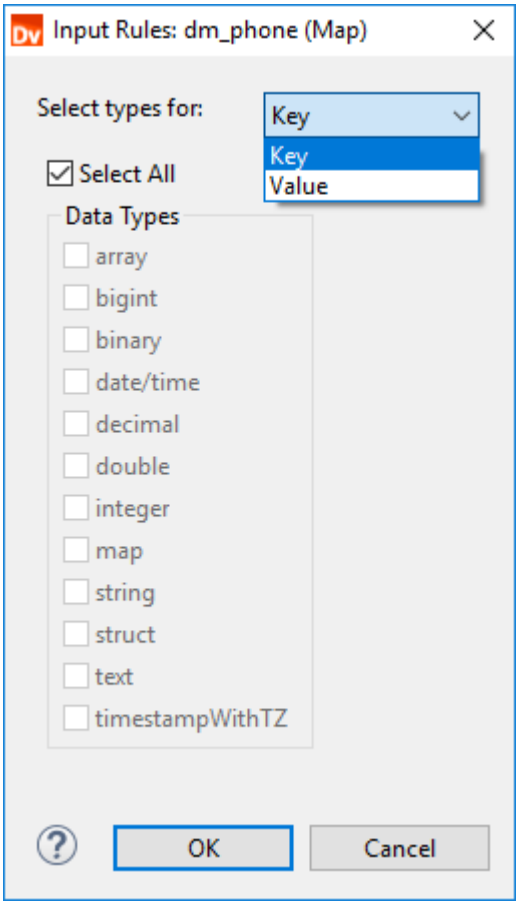
The following image shows the Dynamic Array Input Rule dialog box for a dynamic array port:



Input Rules for a Dynamic Map

An input rule for a dynamic map determines the data type of the key and value pair of a map. You can select one or more data types for the key and value. The default input rules for key and map include elements of all data types.

The following image shows the Input Rules dialog box for a dynamic map port:



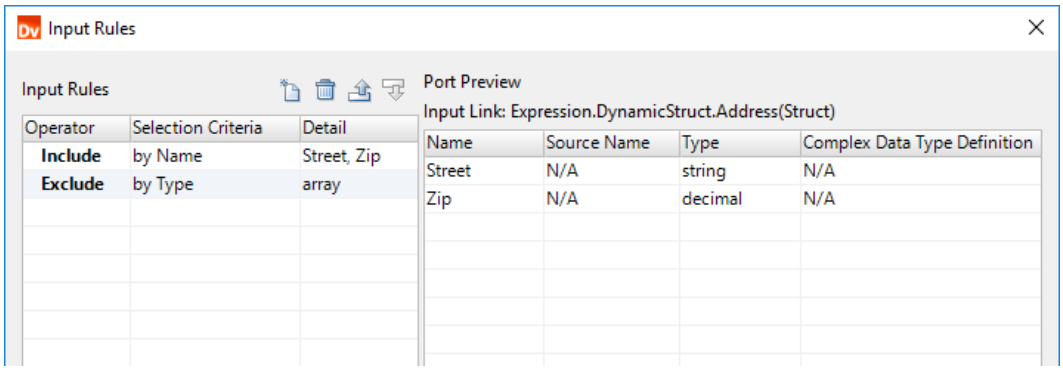
Input Rules for a Dynamic Struct

An input rule for a dynamic struct determines the schema of the struct. You can define the input rules based on the name or type of the struct element in an upstream transformation.

Specify which elements to include or exclude in a struct based on the element names or data type. You can define multiple rules. The Data Integration Service applies rules in the order in which they appear in the input rules list. Include all ports is the default input rule. Create at least one include input rule for a dynamic struct port.

Note: You cannot use input rules to rename elements of a struct port.

The following image shows the Input Rules dialog box for a dynamic struct port:



Port Selectors for Dynamic Complex Ports

You can create port selectors for dynamic complex ports to use in a dynamic expression or a join condition.

You can use port selectors in the following transformations for complex dynamic ports:

- Expression transformation. You can reference a port selector in STRUCT and ARRAY functions as a dynamic expression to construct a dynamic array or a dynamic struct.
- Joiner transformation. You can reference port selectors in STRUCT and ARRAY functions in a join condition.

For more information about port selectors, see the *Informatica Developer Mapping Guide*.

Dynamic Expressions

You can configure a dynamic expression to construct a dynamic complex port. Use STRUCT or ARRAY functions in dynamic expressions to construct a dynamic struct or a dynamic array. You can reference a port selector or a dynamic port in a dynamic expression for a dynamic complex output port.

A dynamic expression for an output port of a primitive data type runs the expression against each port to generate multiple output ports. A dynamic expression for a dynamic complex output port runs the expression to construct a dynamic complex port and returns a single output port of a complex data type.

You must specify a base port for the dynamic expression. Base port setting determines how the elements of the output complex port must look like. For example, elements of a port selector determine the elements of a dynamic output array. Base port for a struct determines the types of the elements of the output struct port.

Note: When you configure a dynamic expression, the Developer tool does not validate whether you have specified a base port.

Example - Dynamic Expression to Construct a Dynamic Struct

An Expression transformation has a dynamic port from which you want to construct a struct.

An Expression transformation has the following generated input ports:

```
Name        String
Street    String
City        String
State        String
Zip        Integer
```

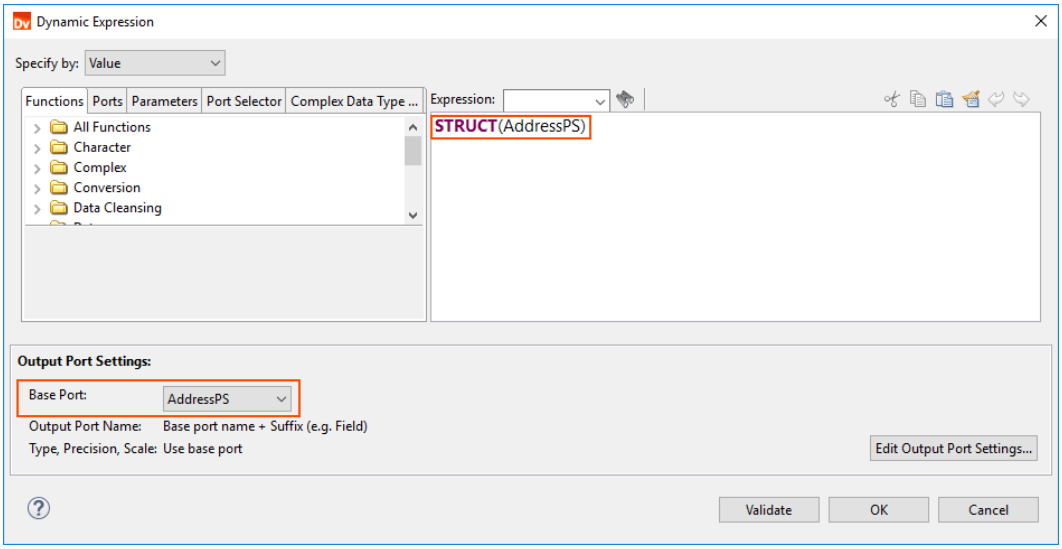
The transformation contains a dynamic complex output port called d_structAddress. You want to construct a struct and return the result to d_structAddress.

You create a port selector AddressPS and define the selection rules to include the following ports by name:

```
Street    String
City        String
State        String
Zip        Integer
```

Then, you create a dynamic expression with the STRUCT function.

The following image shows a dynamic expression that references a port selector AddressPS:



In the **Output Port Settings** area, specify the port selector AddressPS as the base port.

The dynamic expression creates the following struct with the elements from the port selector:

```
d_structAddress{
  Street    String
  City        String
  State        String
  Zip        Integer
}
```

Complex Operators

You can use complex operators in expressions to access or extract elements of a dynamic complex port.

For example, you can use the dot and subscript operators as follows to access elements of a dynamic complex port:

```
dynamicStruct.element
dynamicArray[0]
dynamicMap[key]
```

For more information about complex operators, see "Complex Operators" on page 238.

Complex Functions

You can use ARRAY and STRUCT functions in dynamic expressions to construct dynamic arrays and dynamic structs. Additionally, you can use the EXTRACT_STRUCT function in dynamic expressions to extract all elements from a dynamic struct port in an Expression transformation. You use a dynamic port or a port selector as the complex function argument.

When you construct a dynamic struct, the number of elements of the struct changes based on the number of ports in the dynamic port or the port selector.

When you construct a dynamic array, the size of the array changes based on the number of ports in the dynamic port or the port selector.

For more information about complex functions, see "Complex Functions" on page 240.

Rules and Guidelines for Dynamic Complex Ports

Consider the following rules and guidelines when you work with dynamic complex ports:

- You cannot use dynamic complex ports in a mapplet.
- You cannot use a filter condition to compare dynamic complex ports. However, you can use complex operators to extract elements of a dynamic complex port and use them in a filter condition. For example, you can use the following filter condition:

  ```
  dynamicStruct.field == 100
  ```
- You cannot use the PARSE_JSON or PARSE_XML complex functions in dynamic expressions.

Optimized Mappings

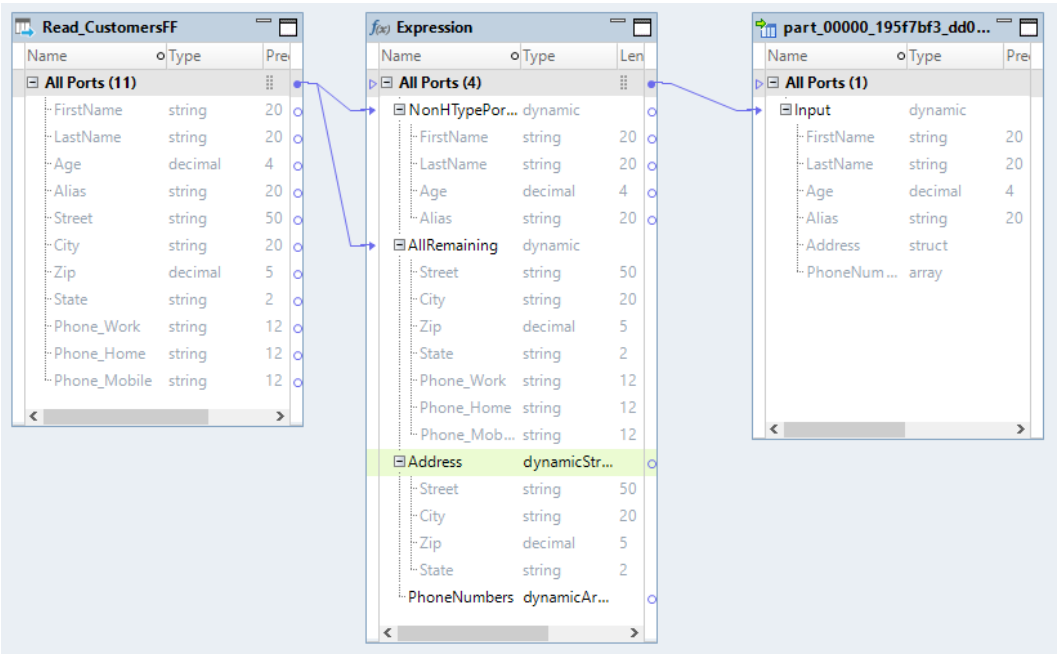
When you develop a dynamic mapping, you might want to review the dynamic mapping and fix any issues before you run the mapping. To review the mapping, you can view an optimized mapping. An optimized mapping is a compiled version of the dynamic mapping.

To view the optimized mapping, right-click an empty area in the mapping editor and click **Show Optimized Mapping**.

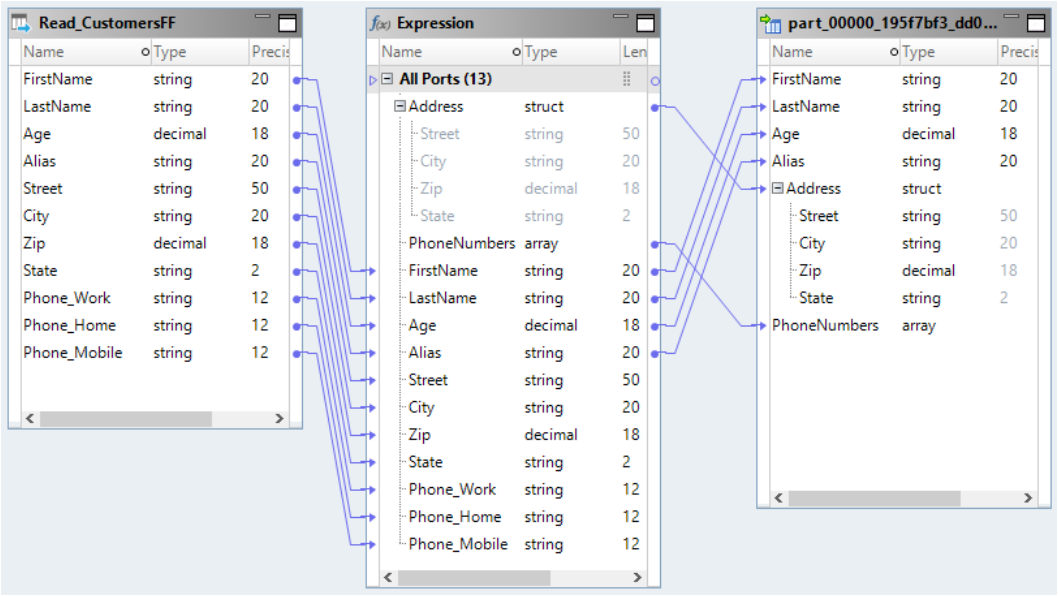
The Data Integration Service generates the optimized mapping. The Data Integration Service compiles the mapping to complete the following tasks:

- Expand dynamic ports and convert generated ports to static ports.
- Convert dynamic complex ports to complex ports.
- Resolve dynamic expressions.
- Resolve parameter values to default values.
- Link static ports.
- Resolve run-time links to connect the ports.

The following image shows a dynamic mapping with dynamic ports, dynamic complex ports, and a mapping flow link:



The following image shows the optimized mapping in which the generated ports are converted to static ports, dynamic complex ports are converted to complex ports, and the port links are resolved:



CHAPTER 15

Intelligent Structure Models

This chapter includes the following topics:

- Overview of Intelligent Structure Models, 275
- Intelligent Structure Discovery Process, 276
- Use Case, 277
- Using an Intelligent Structure Model in a Mapping, 277
- Rules and Guidelines for Intelligent Structure Models, 278
- How to Create a Mapping with an Intelligent Structure Model , 278
- Create an Intelligent Structure Model in Cloud Data Integration, 281

Overview of Intelligent Structure Models

You can use CLAIRE® Intelligent Structure Discovery to parse semi-structured or unstructured data in mappings that run on the Spark engine or the Databricks Spark engine.

Long, complex files with little or no structure can be difficult to understand, much less parse. CLAIRE Intelligent Structure Discovery can automatically discover the structure in unstructured data.

CLAIRE uses machine learning algorithms to decipher data in semi-structured or unstructured data files and create a model of the underlying structure of the data. You can generate an intelligent structure model, a model of the pattern, repetitions, relationships, and types of fields of data discovered in a file, in Informatica Intelligent Cloud Services.

To use the model, you associate it with a data object that represents a complex file source in a data engineering mapping. You can run the mapping on the Spark engine or the Databricks Spark engine to process the data. The mapping uses the intelligent structure model to extract and parse data from input files based on the structure expressed in the model.

To associate a model with a data object, you can perform one of the following tasks:

- Create and export an intelligent structure model in Cloud Data Integration and then select the model when you create the data object.
- Select an XML, JSON, ORC, Avro, or Parquet sample file when you create the data object. Intelligent Structure Discovery creates an intelligent structure model based on the sample file that you select.

  Note: You cannot edit or refine a model that Intelligent Structure Discovery creates automatically. You can only edit a model that you create in Cloud Data Integration.

Intelligent Structure Discovery Process

You can create a CLAIRE® intelligent structure model in Intelligent Structure Discovery. Intelligent Structure Discovery is a service in Cloud Data Integration.

When you provide a sample file, Intelligent Structure Discovery determines the underlying structure of the information and creates a model of the structure. After you create an intelligent structure model you can view, edit, and refine it. For example, you can choose to exclude or combine structure elements. You can normalize repeating groups.

You can create models from the following input types:

- Delimited files, for example, CSV files
- Machine generated files such as weblogs and clickstreams
- JSON files
- XML files
- ORC files
- Avro files
- Parquet files
- Microsoft Excel files
- Data within PDF form fields
- Data within Microsoft Word tables
- XSD files

When you finish refining the model, you can export it and then associate it with a data object that represents a complex file source in a data engineering mapping.

Alternatively, when you create the data object, you can select an XML, JSON, ORC, Avro, or Parquet sample file. Intelligent Structure Discovery creates an intelligent structure model based on the sample file that you select.

Note: You cannot edit or refine a model that Intelligent Structure Discovery creates automatically. You can only edit a model that you create in Cloud Data Integration.

The following image shows the process by which Intelligent Structure Discovery deciphers the underlying patterns of data and creates a model of the data patterns:

Use Case

You work in an operations group for an insurance company. Your team wants to process web logs to identify operations and security issues.

Your back-end system collects data on system access and security activities in various branches of the company. When the data is collected, the data is stored in the corporate data center and hosted in Amazon S3 storage.

Your team wants to understand the types of operations issues that have caused the most errors and system downtime in the past few weeks. You want to store data afterwards for auditing purposes.

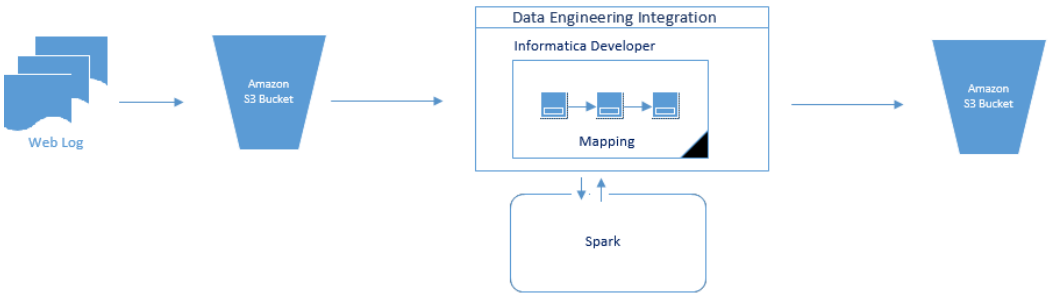
Before your data analysts can begin working with the data, you need to parse the data in Amazon S3 input buckets and produce actionable data. But you cannot spend the time and resources required to sift through the data to create models of analysis. You might have to develop numerous mappings and parameter sets to parse the data to make sure that actionable data is created from the web logs.

Instead of manually creating individual transformations, your team can use automatically generated intelligent structure models to determine the relevant data sets. You create an intelligent structure model in Intelligent Structure Discovery, an application in Data Integration that uses machine learning algorithms to decipher data in structured or unstructured data files and discover the underlying structure of the data.

Intelligent Structure Discovery creates an intelligent structure model that represents the input file data structure. You create a mapping with a data object that uses the intelligent structure model to output actionable data sets.

After the mapping fetches data from Amazon S3 input buckets, the mapping processes the data with an intelligent structure model to prepare the data, and can then write the data to Amazon S3 output buckets.

The following image shows the process to fetch the data from Amazon S3 input buckets, parse and prepare the data, and then write the data to Amazon S3 output buckets. Analysts can then use the data to handle security issues and improve operations management.



Using an Intelligent Structure Model in a Mapping

To use an intelligent structure model in a mapping, you associate it with a data object that represents a complex file source when you create the object. The data object can be used to process files that are similar to the one used to create the model.

When you create the data object, you either select a model that you created and exported in Cloud Data Integration or select an XML, JSON, ORC, Avro, or Parquet sample file. Intelligent Structure Discovery creates the model based on the sample file that you select.

You can add a Read transformation based on the data object to a mapping. If you want to process the data any further, such as checking data quality, or structuring the data into relational format, you add relevant downstream transformations to the mapping.

When the mapping runs, the Read transformation reads one or more input files and parses the data into fields, arrays, and structs.

Depending on the model and input, the data object output might contain primitive data types, complex data types, or nested data types. For more information about working with these data types in transformations, see "Complex Data Types" on page 222.

If the input contains unidentified data, Intelligent Structure Discovery arranges the output of unidentified data in the sample file in structured JSON format.

Rules and Guidelines for Intelligent Structure Models

Consider the following rules and guidelines when you work with an intelligent structure model:

- When you create an intelligent structure model in Cloud Data Integration, select the Data Integration version that corresponds to your current Data Engineering Integration version.
- The model or the sample file that you select for a data object should match the structure of the expected input files for the data object as much as possible. If an input file does not match the model that you select or that Intelligent Structure Discovery creates based on the sample file, or partially matches the model, there might be a large amount of unidentified data. Therefore, it is important to choose a file that represents the expected input files.
- If you select a sample XML file that tags special characters that javax.xml.parsers can't parse, the data object fails to load the file. For example, javax.xml.parsers can't parse the U+035D unicode character.
- You cannot use an intelligent structure model in a Write transformation.
- A data object with an intelligent structure model parses PDF forms, Microsoft Word tables, and Microsoft Excel files whose size is less than the supported Hadoop split size of 256 MB.
- An intelligent structure model can parse the data within PDF form fields but not data outside the fields.
- An intelligent structure model can parse data within Microsoft Word tables. Other data is unidentified.

How to Create a Mapping with an Intelligent Structure Model

You can create a mapping with a data object that incorporates an intelligent structure model to parse data. You run the mapping on the Spark engine or the Databricks Spark engine to process the data.

The following high-level tasks describe how to develop and run a mapping to read and process data in files of any type that an intelligent structure model can process, and then write the data to a target.
Note: The tasks and the order in which you perform the tasks to develop the mapping depend on the mapping scenario.

Optionally, create an intelligent structure model.

You can create an intelligent structure model in Cloud Data Integration. Create the model using a sample file in Data Integration and export the model to the relevant file storage system, following the instructions in "Create an Intelligent Structure Model in Cloud Data Integration" on page 281.

Create a connection.

Create a connection to access data in files that are stored in that system. You can create the following types of connections that work with the data objects that can incorporate an intelligent structure model:

- Amazon S3
- Hadoop Distributed File System
- Microsoft Azure Blob
- Microsoft Azure Data Lake Store

Create a read data object.

1. Create a data object with an intelligent structure model to represent the files stored as sources. You can create the following types of data objects with an intelligent structure model:
   - Complex file
   - Amazon S3
   - Microsoft Azure Blob
   - Microsoft Azure Data Lake Store
2. Configure the data object properties. Note the following guidelines:
   - In **Resource Format**, select **Intelligent Structure Model or Sample File**.
   - If you created an intelligent structure model in Data Integration, select that file. Alternatively, select a sample file to base the model on. When you select an XML, JSON, ORC, Avro, or Parquet sample file, Intelligent Structure Discovery creates an intelligent structure model based on the sample file that you select.

     Note: To import an intelligent structure model as a data object, you must have the relevant Informatica Intelligent Cloud Services license. For more information about prerequisites for creating an intelligent structure model, see "Before You Begin" on page 281.
3. In the data object read operation, configure columns to project hierarchical data as a complex data type. Enable the Project Column as Complex Data Type property in the Column Projection properties.

Create a write data object.

1. Create a data object to write the data to target storage.
2. Configure the data object properties.

Note: Do not associate an intelligent structure model with a data object write operation. If you use a write operation that is associated with an intelligent structure model in a mapping, the mapping is not valid.

Create a mapping.

1. Create a mapping.
2. Add a Read transformation based on the data object with the intelligent structure model.
3. Based on the mapping logic, add other transformations that are supported on the run-time engine. Link the ports and configure the transformation properties based on the mapping logic.
4. Add a Write transformation based on the data object that passes the data to the target storage or output. Link the ports and configure the transformation properties based on the mapping logic.

Configure the mapping.

1. Choose a validation environment and a run-time engine.
2. Choose an execution environment and select a connection.

Run the mapping.

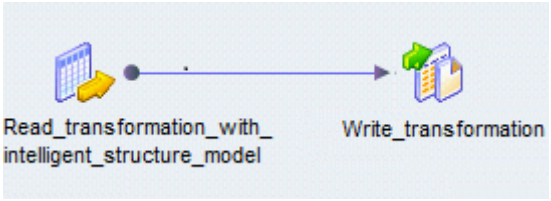
1. Validate the mapping and fix any errors.
2. Optionally, view the execution plan to debug the logic.
3. Run the mapping.

Mapping Example

Your organization stores purchase order details, such as customer ID, item codes, and item quantity, in Microsoft Excel spreadsheets in HDFS. You need to remove all private customer data and change the data into text files for storage, so you can use the data for auditing.

Create a mapping that reads all the purchase records from the file in HDFS using a data object with an intelligent structure. Then use the mapping to parse the data and write it to a storage target.

The following figure shows the example mapping:



You can use the following objects in the HDFS mapping:

HDFS input

The input object, Read_transformation_with_intelligent_structure_model, is a Read transformation that processed a Microsoft Excel file stored in HDFS and creates field output.

Amazon S3 output

The output object, Write_transformation, is a Write transformation that represents an Amazon S3 bucket.

When you run the mapping, the Data Integration Service reads the file in a binary stream and passes it to the Read transformation. The Read transformation extracts the relevant data in accordance to the intelligent structure and passes data to the Write transformation. The Write transformation writes the data to the storage target.

You can configure the mapping to run in a Hadoop run-time environment.

To configure the mapping, perform the following tasks:

1. Create an HDFS connection to read files from the Hadoop cluster.
2. Create a complex file data object read operation. Specify the following parameters:
   - The intelligent structure as the resource in the data object. The intelligent structure was configured so that it does not pass sensitive data.
   - The HDFS file location.
   - The input file folder location in the read data object operation.
3. Drag and drop the complex file data object read operation into a mapping.
4. Create an Amazon S3 connection.
5. Create a Write transformation for the Amazon S3 data object and add it to the mapping.

Create an Intelligent Structure Model in Cloud Data Integration

Create an intelligent structure model to use in a Data Engineering Integration mapping.

You create and export the model in Data Integration, a service in Informatica Intelligent Cloud Services. For more information, see the Data Integration *Components* help.

Before You Begin

Before you create intelligent structure models in Data Integration, perform the following tasks:

- Ensure that you have an active Informatica Intelligent Cloud Services account.
- In Informatica Intelligent Cloud Services, configure a separate organization to create and export models to use in a mapping in the Developer tool.
- Verify that your Informatica Intelligent Cloud Services administrator has set the correct user-defined roles to create and export intelligent structure models.
- Verify that the version of the license of the Informatica Intelligent Cloud Services that you use to create intelligent structure models corresponds to your current Data Engineering Integration version.

For more information about registration and roles in Informatica Intelligent Cloud Services Administrator, see the Administrator help.

Creating an Informatica Intelligent Cloud Services Account

Create an Informatica Intelligent Cloud Services account if you do not already have one.

1. To create an account, access the Informatica Intelligent Cloud Services login page at the following link: https://dm-us.informaticacloud.com/identity-service/home.
2. On the Informatica Intelligent Cloud Services login page, click **Don't have an account?**.
3. In the setup page, enter your contact details and email address. You can select to use your email address as a username.

   Note: You will receive a registration email with access details and a link to confirm your account. You must confirm the account within 48 hours or register again.
4. In the registration email, click the account confirmation access link.

   The Informatica Intelligent Cloud Services password setup window appears.
5. On the password setup window, define your Informatica Intelligent Cloud Services password, and then click **Log In**.

   The Informatica Intelligent Cloud Services service picker appears. You can now access Data Integration.

Creating an Intelligent Structure Model

Create an intelligent structure model in Data Integration. It is recommended to use a simplified sample file to generate the model. The sample file should have a representative data structure that is similar to the files that you want to parse.

1. In the Informatica Intelligent Cloud Services **My Services** page, select **Data Integration**.
2. Click **New** > **Components** > **Intelligent Structure Model** > **Create**.

3. In the **Intelligent Structure Model** page, enter a name for the model. The name can contain alphanumeric characters and underscores. Optionally, enter a description of the model.
4. Select a location. Navigate to the project and folder where you want to save the model, or use the default folder.
5. Browse for the sample file on which to base the model and click **Discover Structure**.

   Intelligent Structure Discovery creates and displays a model of the data.
6. To refine the intelligent structure model and customize the structure of the output data, you can select a data node and select to combine, exclude, flatten, or collapse the node. For more details about refining and customizing a model, see the "Intelligent structure models" section of the Data Integration help.
7. Click **Save**.

Exporting an Intelligent Structure Model

After you create or edit a model, you can export the model to your local drive.

1. In Data Integration, select the **Explore** page.
2. On the **Explore** page, navigate to the project and folder with the intelligent structure model.

   The **Explore** page displays all the assets in the folder.
3. Click to select the row that contains the relevant intelligent structure model. In the **Actions** menu, select **Edit**. In the **Intelligent Structure Details** panel, click **Edit**.

   The intelligent structure model appears in a separate page.
4. In the **Intelligent Structure** page, find the icon menu in the upper right corner of the Visual Model tab, and click the **Export model** icon.

   Intelligent Structure Discovery downloads the `.amodel` file to your local drive.

CHAPTER 16

Blockchain

This chapter includes the following topics:

- Blockchain Overview, 283
- Blockchain Data Objects, 284
- Blockchain Data Object Operations, 288
- Use Case: Using a Blockchain Source to Improve Services in a Vehicle Lifecycle, 292

Blockchain Overview

You can read or look up data from a blockchain and transform the data according to your business needs. You can also write data to the blockchain where the data can be stored as an immutable record.

To connect to a blockchain in a mapping, create a blockchain connection, a blockchain data object, and the associated blockchain data object operations. A mapping that connects to a blockchain must run on the Spark engine.

A blockchain data object represents the data on the blockchain, and a data object operation specifies the transactions that you want to perform on the blockchain. When you run a mapping that uses a blockchain data object operation, the Spark engine communicates with the REST web service that exposes the blockchain to trigger the transactions according to the business rules on the blockchain.

To design a mapping that uses a blockchain data source, assess how you participate in the blockchain. As a member, you can read from, look up, or write to blockchain data sources. As a regulator or a third party, you can read the data into a mapping and transform the data for use in external applications.

For example, you can read from, look up, and write to blockchain data sources in the following ways:

Read

You can read the data on the blockchain and integrate it with data that you store in your corporate data center.

Look Up

If you are a member in a supply chain that purchases materials from producers, you can develop a mapping that looks up data about the materials from the blockchain.

Write

You store contextual data in an enterprise resource planning system and you want to store hashes of the data on the blockchain to improve data security and trust when you share the data with other members. You can use the contextual data as a source in the mapping and write the data to a blockchain target.

Technical Preview Notice: Effective in version 10.4.0, connecting to a blockchain is available for technical preview. Technical preview functionality is supported for evaluation purposes but is unwarranted and is not supported in production environments or any environment that you plan to push to production. Informatica intends to include the preview functionality in an upcoming release for production use, but might choose not to in accordance with changing market or technical circumstances. For more information, contact Informatica Global Customer Support.

Blockchain Process

Using blockchain as a source, lookup, or target in a mapping requires administrator and developer tasks.

The blockchain process requires tasks from the following users:

Administrator

An administrator creates a blockchain connection. You can create a blockchain connection using the Administrator tool or a command line program. The blockchain connection contains the information to connect to the blockchain and to communicate with the blockchain through a REST API.

For information about creating a blockchain connection using the Administrator tool, see "Blockchain Connection Properties" on page 326. For information about creating a blockchain connection using a command line program, see the *Informatica Command Reference*.

Developer

A developer completes the following tasks:

1. Creates a blockchain data object to represent data on the blockchain.
2. Creates one or more blockchain data object operations.
3. Adds the blockchain data object to the mapping as a Read, Lookup, or Write transformation.

Blockchain Data Objects

A blockchain data object is a physical data object that represents data on the blockchain.

When you create a blockchain data object, you select resource URIs to specify the assets that you want to access. After you select a resource URI, you can select the transactions that you want to use. Each transaction is associated with an HTTP GET, POST, PUT, or DELETE request.

Response Ports

If you use a transaction that uses an HTTP GET request, the Developer tool creates response ports in addition to the ports that contain the data that the transaction returns.

The Developer tool creates the following additional ports:

Response_200

The `Response_200` port is a struct that represents the asset. To view the nested elements in the struct, create a read or write operation.

responseCode

The `responseCode` port stores the HTTP status code that the request returns.

For example, the response code `200 (OK)` indicates that data was returned. You can pass the code to a Filter transformation that uses a filter condition such as `responseCode=200` to filter out data that is not available.

Similarly, the response code `401 (Unauthorized)` indicates that the client is not authorized to perform the transaction. You might use the response code to detect issues in a security configuration on the blockchain or in the blockchain connection properties.

Blockchain Data Object Overview Properties

Overview properties are general properties for the blockchain data object.

The properties are populated according to the REST API. If you change a property value in the Developer tool, make the same changes on the blockchain if necessary.

General Properties

The following table describes the general properties for the blockchain data object:

Property	Description
Name	Name of the blockchain data object.
Description	Description of the blockchain data object.
Connection	Name of the blockchain connection.

Objects Properties

The following table describes the properties for each transaction:

Property	Description
Name	Name of the transaction in the Informatica environment.
Description	Description of the transaction.
Native Name	Native name of the transaction on the blockchain.
Path Information	Path of the transaction on the REST web service.

Columns Properties

The following table describes the properties for the columns in an asset that a transaction operates on:

Property	Description
Name	Name of the column.
Native Name	Native name of the column on the blockchain. If you change the native name in the Developer tool, you must also change the name of the column on the blockchain.
Type	Data type of the column.

Property	Description
Precision	Maximum number of significant digits for numeric data types, or the maximum number of characters for string data types. You can edit the precision based on your requirements. If the data exceeds the precision, the data is truncated.
Scale	Scale of the data type.
Advanced Properties	Advanced properties that are set according to the REST API. The advanced property `isRequired=true` is set for columns that are required.
Access Type	Access type for the column. The Read access type indicates that you can use the column only as a source. The Read and Write access type indicates that you can use the column as both a source and a target. For example, you must have read and write access to the column that you reference in a lookup condition. The Write access type indicates that you can use the column only as a target.
Description	Description of the column. If the REST API restricts column values, the description lists the values that are valid.

Advanced Properties

The advanced properties have the precedence level, **Operation level** > **Object level** > **Connection level**. The properties configured at the operation level will override the properties configured at the object or connection level. The following table describes advanced properties for each transaction:

Property	Description
Request Type	HTTP method, such as GET, POST, PUT, or DELETE.
Producer Content Type	Content type of the request that is sent to the web service. The content type must be application/json.
Consumer Content Type	Content type of the response that the web service returns. The content type must be application/json.
Object Location	A location to access an asset on the blockchain.
Base URL	The base URL to access assets on the blockchain. The base URL must match the beginning of the object location.

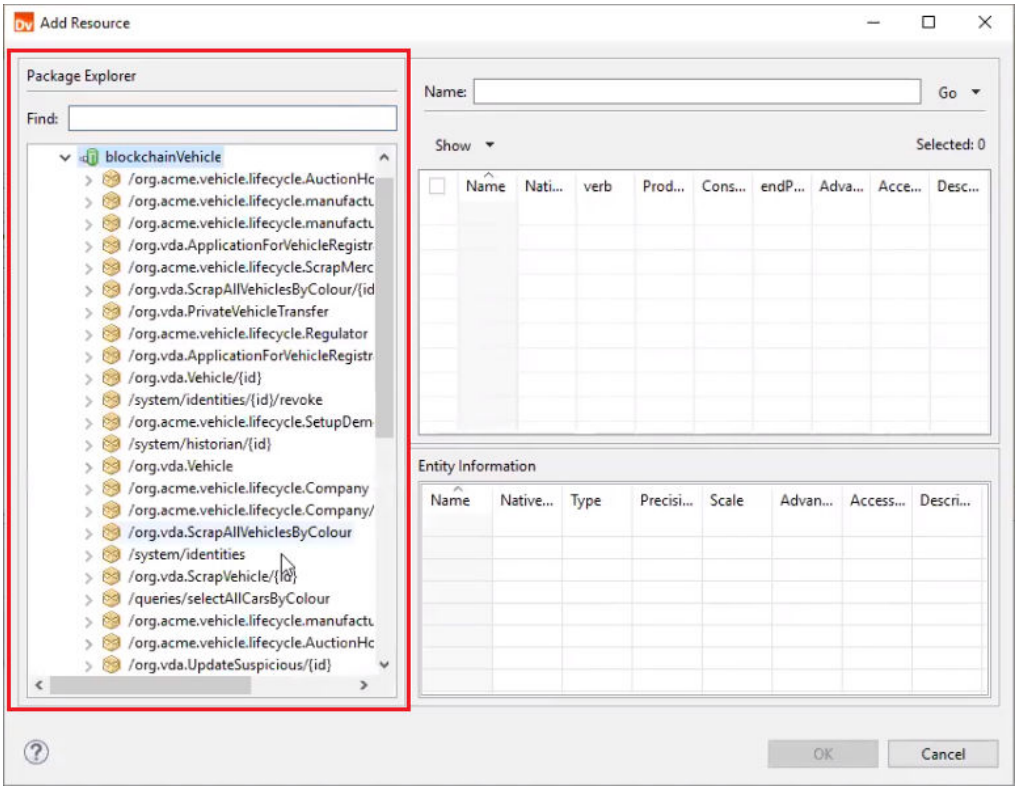
Creating a Blockchain Data Object

Create a blockchain data object to represent the data that you access on the blockchain.

1. Click **File** > **New** > **Data Object**.
2. Select **Physical Data Objects** > **Blockchain Data Object**.
3. Enter a name for the data object.
4. Click **Browse** to select the location for the data object in the Model repository.

5. Click **Browse** to select the blockchain connection.
6. Optionally, select an operating system profile.
7. Click **Add** to add resources to the data object.
8. In the Package Explorer section, expand the blockchain connection to view the resource URIs.

   The following image shows the Package Explorer section:



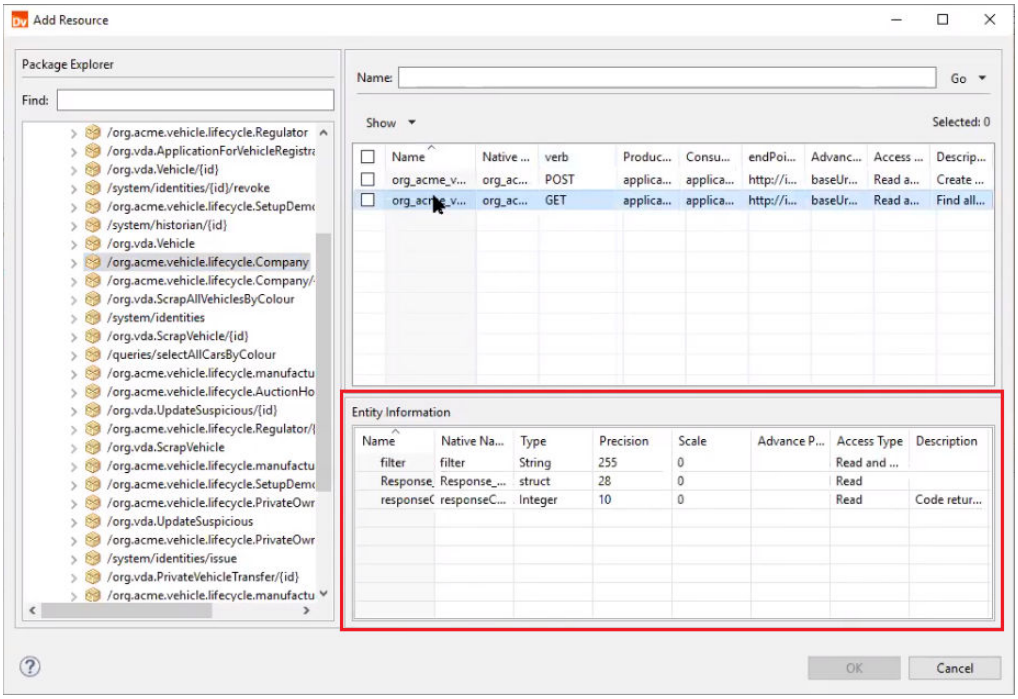
9. Select a resource URI.

   The available transactions appear on the right side of the dialog box. Each transaction uses an HTTP GET, PUT, POST, or DELETE method.

10. Select the transactions that you want to add to the data object.

    For each transaction, you can preview the columns that the transaction operates on in the Entity Information section.

    The following image shows the Entity Information section:



Blockchain Data Object Operations

Create a blockchain data object operation to use a blockchain data object in a mapping.

You can create a read operation to use as a source in a mapping or to look up data on the blockchain. You can create a write operation to use as a target in a mapping. When you run the mapping, the Spark engine uses the transactions in the data object operation to send the appropriate HTTP requests to the REST web service that exposes the blockchain.

Requests and responses use hierarchical data in JSON format.

Blockchain Data Object Read Operation Properties

Create a blockchain data object read operation to read or to look up data on a blockchain. Use the read operation as a source or a lookup in a mapping.

For more information about read operations, see the *Informatica Developer Mapping Guide*.

General Properties

The following table describes general properties for a read operation:

Property	Description
Name	Name of the data object operation.
Description	Description of the data object operation.

Ports Properties

The following table describes the port properties in the read operation:

Property	Description
Name	Name of the port.
Type	Data type of the port.
Detail	Type configuration for the port.
Precision	Maximum number of significant digits for numeric data types, or the maximum number of characters for string data types. You can edit the precision based on your requirements. If the data exceeds the precision, the data is truncated.
Scale	Scale of the data type.
Description	Description of the port.

Run-time Properties

The following table describes the properties that the read operation uses at run time:

Property	Description
Connection	Name of the blockchain connection.

Advanced Properties

The following table describes the advanced properties for the read operation:

Property	Description
Operation Type	Type of blockchain data object operation. For a read operation, the operation type is `blockchainRead`.
Advanced Properties	Additional advanced properties for the operation. Advanced properties are name-value pairs that are separated by a semicolon.

Property	Description
Pagination Properties	A name-value pair URL property for pagination. You can add more than one name-value pair separated by a semicolon. For example, you can add `page=1` to view one page at a time and the URL appears as `https://jobs.github.com/positions.json?description=ruby&page=1`.
Query Parameters	A name-value pair URL property for querying results. You can add more than one name-value pair separated by a semicolon. For example, you can query `description=ruby` to fetch results and the URL appears as `https://jobs.github.com/positions.json?description=ruby&page=1`.

Sources Properties

The following table describes the sources properties for the read operation:

Property	Description
Sources	Transactions that the blockchain data object operation uses to access an asset on the blockchain.

Blockchain Data Object Write Operation Properties

Create a blockchain data object write operation to write data to a blockchain. Use the write operation as a target in a mapping.

For more information about write operations, see the *Informatica Developer Mapping Guide*.

When you write data to a blockchain, make sure that the target does not contain extra ports that the REST API does not support. If the target contains an extra port, the REST API call fails at run time and reports that there is an unexpected property. To resolve the issue, perform one of the following tasks:

- Remove the corresponding column in the swagger file.
- Do not pass data to the extra port in the target.

Target Properties

The following table describes the target properties for the write operation:

Property	Description
Target	Transactions that the blockchain data object operation uses to write data to the blockchain.

General Properties

The following table describes general properties for the write operation:

Property	Description
Name	Name of the data object operation.
Description	Description of the data object operation.

Ports Properties

The following table describes the port properties in the write operation:

Property	Description
Name	Name of the port.
Type	Data type of the port.
Detail	Type configuration for the port.
Precision	Maximum number of significant digits for numeric data types, or the maximum number of characters for string data types. You can edit the precision based on your requirements. If the data exceeds the precision, the data is truncated.
Scale	Scale of the data type.
Description	Description of the port.

Run-time Properties

The following table describes the properties that the write operation uses at run time:

Property	Description
Connection	Name of the blockchain connection.

Advanced Properties

The following table describes the advanced properties for the write operation:

Property	Description
Operation Type	Type of blockchain data object operation. For a write operation, the operation type is `blockchainWrite`.
Advanced Properties	Additional advanced properties for the operation. Advanced properties are name-value pairs that are separated by a semicolon.

Creating a Blockchain Data Object Operation

Create a blockchain data object operation that you can use as a source, lookup, or target in a mapping.

1. In the **Object Explorer** view, open the blockchain data object.

   The blockchain data object appears in the **Editor**.
2. Select the **Data Object Operations** tab.
3. Click the **New (Insert)** icon to create a new blockchain data object operation.

   The **Data Object Operation** dialog box appears.
4. Edit the name of the data object operation.
5. For capabilities, select either `blockchainRead` or `blockchainWrite`.

The `blockchainRead` capability indicates that the data object operation is a read operation. The `blockchainWrite` capability indicates that the data object operation is a write operation.

6. Click **Add** to select resources.
7. Select the resources that you want the data object operation to use.
8. Click **Finish**

   The data object operation appears in the **Editor**. Edit the links between the source object and the output object according to your requirements.

Use Case: Using a Blockchain Source to Improve Services in a Vehicle Lifecycle

You participate as an insurance agent in a blockchain that manages the vehicle lifecycle. Your goal is to improve how manufacturers and dealers handle vehicles as they are built and sold to consumers.

Before a manufacturer or dealer prepares insurance for a vehicle, the blockchain verifies that the manufacturer and the dealer have the proper certificates and comply with government regulations. If verification fails, the blockchain logs a suspicious message for the vehicle.

As an insurance agent, you access the suspicious messages to learn about the state of a vehicle and approve the vehicle to be insured. To make sure that manufacturers and dealers handle the suspicious messages promptly, you design a mapping that connects to the blockchain to get the suspicious message for each vehicle and generates an alert for the manufacturer or dealer that is responsible for the vehicle.

Mapping Overview

Create a mapping that reads data from a blockchain and uses the data to generate an alert message for vehicles that have a suspicious message.

For example, a vehicle has the following suspicious message:

```
Vehicle registration expired
```

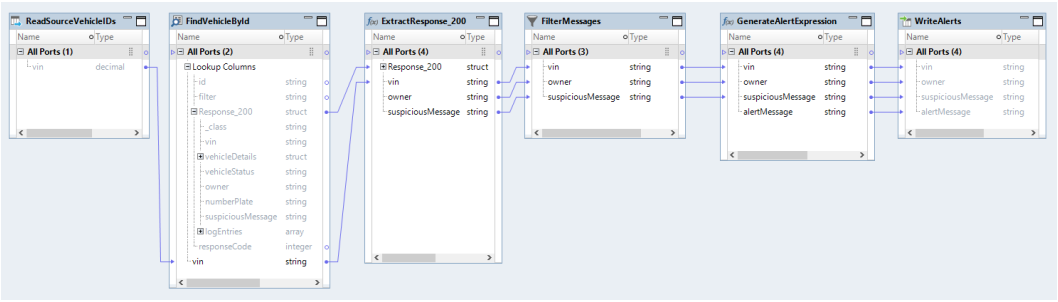
You want to generate the following alert message:

```
Dear Arianne Fitzgerald, Vehicle # 1B3HB48B67D562726 in your possession has the issue:
Vehicle registration expired. Please resolve the issue in 10 days.
```

The mapping performs the following steps:

1. Takes a list of vehicle identification numbers as input.
2. Uses the list of vehicle identification numbers to look up vehicles on the blockchain.
3. Extracts the suspicious message and owner name for the vehicle.
4. Filters out vehicles that do not have a suspicious message.
5. Generates an alert message.
6. Writes the vehicle identification number, owner name, suspicious message, and alert message to a target.

The following image shows the mapping:



Step 1. Read Vehicle Numbers

Add a Read transformation to get vehicle identification numbers and pass the numbers to downstream transformations in the mapping.

Step 2. Look Up Vehicles

To look up vehicles on the blockchain, complete the following tasks:

1. Add a blockchain data object to the mapping as a Lookup transformation and select a read operation that uses an HTTP GET request to get vehicles by ID. For example, the read operation might send an HTTP GET request to the following resource URI:

   ```
   /org.company/vehicles/{id}
   ```

2. Configure the following lookup condition:

   ```
   id = vin
   ```

Step 3. Extract Vehicle Information

To extract the suspicious message and the name of the vehicle owner, complete the following tasks:

1. Add an Expression transformation to the mapping.
2. Create the following output ports:
   - owner
   - suspiciousMessage
3. In the output port `owner`, configure the following port expression:

   ```
   Response_200.owner
   ```

4. In the output port `suspiciousMessage`, configure the following port expression:

   ```
   Response_200.suspiciousMessage
   ```

Step 4. Filter Out Vehicles

To filter out vehicles that do not have a suspicious message, complete the following tasks:

1. Add a Filter transformation to the mapping.
2. Configure the following filter condition:

   ```
   suspiciousMessage != NULL
   ```

Step 5. Generate an Alert Message

To generate an alert message, complete the following tasks:

1. Add an Expression transformation to the mapping.
2. Create the output port `alertMessage`.

3. In the output port `alertMessage`, configure the following port expression:

```
'Dear ' || owner || ', Vehicle # ' || vin || ' in your possession has the issue: '
|| suspiciousMessage || '. Please resolve the issue in 10 days.'
```

Step 6. Write the Data

To write the data to a target, add a Write transformation to the mapping. When the mapping runs, the Write transformation populates the target with the vehicle identification number, owner name, suspicious message, and alert message.

A third-party tool can use the data in the target to perform the following tasks:

1. Use the owner name to look up the owner's e-mail address.
2. Use the e-mail address and alert message to populate an e-mail.
3. Send the e-mail immediately or on a schedule.
4. Archive the data in the target for future reference.

CHAPTER 17

Stateful Computing

This chapter includes the following topics:

- Overview of Stateful Computing, 295
- Windowing Configuration, 296
- Window Functions, 299
- Windowing Examples, 304

Overview of Stateful Computing

You can use window functions in Expression transformations to perform stateful computations on the Spark engine.

A stateful computation is a function that takes some state and returns a value along with a new state.

You can use a window function to perform stateful computations. A window function takes a small subset of a larger data set for processing and analysis.

Window functions operate on a group of rows and calculate a return value for every input row. This characteristic of window functions makes it easy to express data processing tasks that are difficult to express concisely without window functions.

Use window functions to perform the following tasks:

- Retrieve data from upstream or downstream rows.
- Calculate a cumulative sum based on a group of rows.
- Calculate a cumulative average based on a group of rows.

Before you define a window function in an Expression transformation, you must describe the window by configuring the windowing properties. Windowing properties include a frame specification, partition keys, and order keys. The frame specification states which rows are included in the overall calculation for the current row. The partition keys determine which rows are in the same partition. The order keys determine how rows in a partition are ordered.

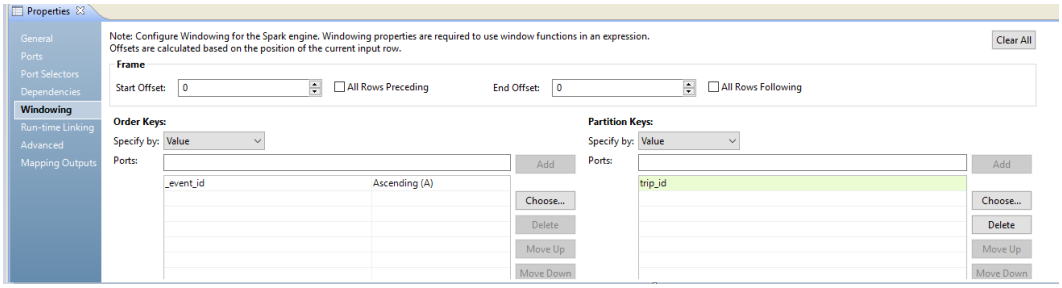
After you configure windowing properties, you define a window function in the Expression transformation. Spark supports the window functions LEAD and LAG. You can also use aggregate functions as window functions in an Expression transformation.

Windowing Configuration

When you include a window function in an Expression transformation, you configure the windowing properties associated with the function. Windowing properties define the partitioning, ordering, and frame boundaries associated with a particular input row.

Configure a transformation for windowing on the Windowing tab.

The following image shows the Windowing tab:



You configure the following groups of properties on the Windowing tab:

Frame

Defines the rows that are included in the frame for the current input row, based on physical offsets from the position of the current input row.

You configure a frame if you use an aggregate function as a window function. The window functions LEAD and LAG reference individual rows and ignore the frame specification.

Partition Keys

Separate the input rows into different partitions. If you do not define partition keys, all rows belong to a single partition.

Order Keys

Define how rows in a partition are ordered. The ports you choose determine the position of a row within a partition. The order key can be ascending or descending. If you do not define order keys, the rows have no particular order.

Frame

The frame determines which rows are included in the calculation for the current input row, based on their relative position to the current row.

If you use an aggregate function instead of LEAD or LAG, you must specify a window frame. LEAD and LAG reference individual row sand ignore the frame specification.

The start offset and end offset describe the number of rows that appear before and after the current input row. An offset of "0" represents the current input row. For example, a start offset of -3 and an end offset of 0 describes a frame including the current input row and the three rows before the current row.

The following image shows a frame with a start offset of -1 and an end offset of 1:

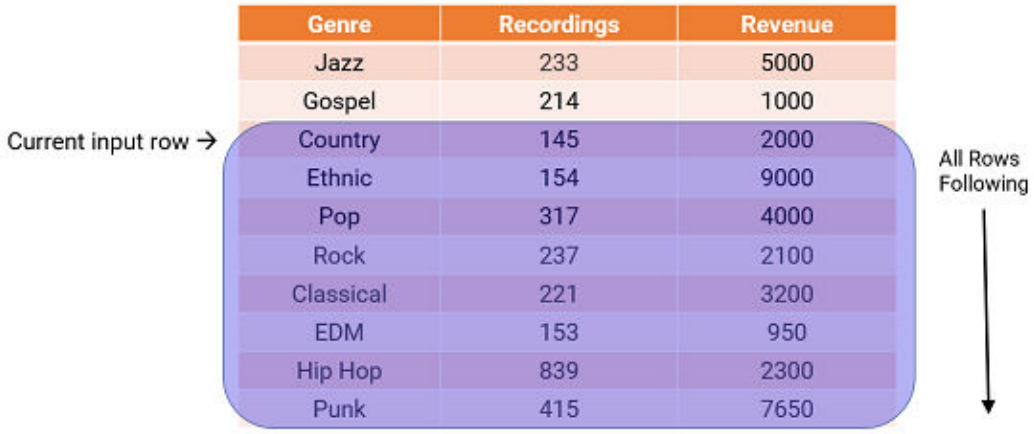
For every input row, the function performs an aggregate operation on the rows inside the frame. If you configure an aggregate expression like SUM with the preceding frame, the expression calculates the sum of the values within the frame and returns a value of 6000 for the input row.

You can also specify a frame that does not include the current input row. For example, a start offset of 10 and an end offset of 15 describes a frame that includes six total rows, from the tenth to the fifteenth row after the current row.

Note: The start offset must be less than or equal to the end offset.

Offsets of **All Rows Preceding** and **All Rows Following** represent the first row of the partition and the last row of the partition. For example, if the start offset is All Rows Preceding and the end offset is -1, the frame includes one row before the current row and all rows before that.

The following figure illustrates a frame with a start offset of 0 and an end offset of All Rows Following:



Partition and Order Keys

Configure partition and order keys to form groups of rows and define the order or sequence of rows within each partition.

Use the following keys to specify how to group and order the rows in a window:

Partition keys

Configure partition keys to define partition boundaries, rather than performing the calculation across all inputs. The window function operates across the rows that fall into the same partition as the current row.

You can specify the partition keys by value or parameter. Select **Value** to use port names. Choose **Parameter** to use a sort key list parameter. A sort key list parameter contains a list of ports to sort by. If you do not specify partition keys, all the data is included in the same partition.

Order keys

Use order keys to determine how rows in a partition are ordered. Order keys define the position of a particular row in a partition.

You can specify the order keys by value or parameter. Select **Value** to use port names. Choose **Parameter** to use a sort key list parameter. A sort key list parameter contains a list of ports to sort by. You must also choose to arrange the data in ascending or descending order. If you do not specify order keys, the rows in a partition are not arranged in any particular order.

Example

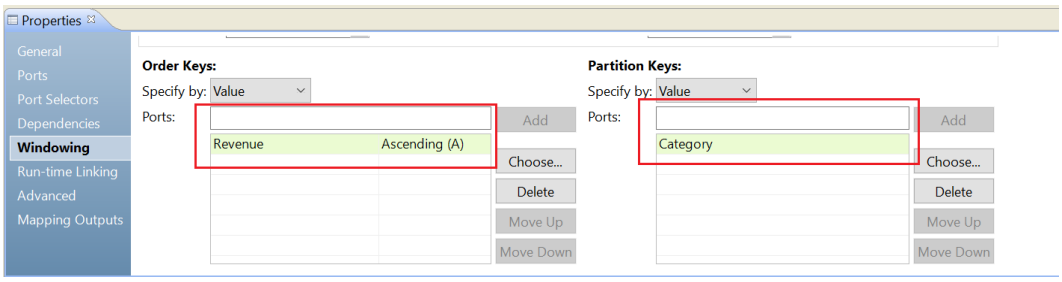
You are the owner of a coffee and tea shop. You want to calculate the best-selling and second best-selling coffee and tea products.

The following table lists the products, the corresponding product categories, and the revenue from each product:

Product	Category	Revenue
Espresso	Coffee	600
Black	Tea	550
Cappuccino	Coffee	500
Americano	Coffee	600
Oolong	Tea	250
Macchiato	Coffee	300
Green	Tea	450
White	Tea	650

You partition the data by category and order the data by descending revenue.

The following image shows the properties you configure on the Windowing tab:



The following table shows the data grouped into two partitions according to category. Within each partition, the revenue is organized in descending order:

Product	Category	Revenue
Espresso	Coffee	600
Americano	Coffee	600
Cappuccino	Coffee	500
Macchiato	Coffee	300
White	Tea	650
Black	Tea	550
Green	Tea	450
Oolong	Tea	250

Based on the partitioning and ordering specifications, you determine that the two best-selling coffees are espresso and Americano, and the two best-selling teas are white and black.

Rules and Guidelines for Windowing Configuration

Certain guidelines apply when you configure a transformation for windowing.

Consider the following rules and guidelines when you define windowing properties for a window function:

- When you configure a frame, the start offset must be less than or equal to the end offset. Otherwise, the frame is not valid.
- Configure a frame specification if you use an aggregate function as a window function. LEAD and LAG operate based on the offset value and ignore the frame specification.
- You cannot use complex ports as partition or order keys.
- Assign unique port names to partition and order keys to avoid run-time errors.
- The partition and order keys cannot use both a dynamic port and one or more generated ports of the same dynamic port. You must select either the dynamic port or the generated ports.

Window Functions

Window functions calculate a return value for every input row of a table, based on a group of rows.

A window function performs a calculation across a set of table rows that are related to the current row. You can also perform this type of calculation with an aggregate function. But unlike regular aggregate functions, a window function does not group rows into a single output row. The rows retain unique identities.

You can define the LEAD and LAG analytic window functions in an Expression transformation. LEAD and LAG give access to multiple rows within a table, without the need for a self-join.

LEAD

The LEAD function returns data from future rows.

LEAD uses the following syntax:

```
LEAD ( Column name, Offset, Default )
```

LEAD returns the value at an offset number of rows after the current row. Use the LEAD function to compare values in the current row with values in a following row. Use the following arguments with the LEAD function:

- Column name. The column name whose value from the subsequent row is to be returned.
- Offset. The number of rows following the current row from which the data is to be retrieved. For example, an offset of "1" accesses the next immediate row, and an offset of "3" accesses the third row after the current row.
- Default. The default value to be returned if the offset is outside the scope of the partition. If you do not specify a default, the default is NULL.

For more information about the LEAD function, see the *Informatica Transformation Language Reference*.

LAG

The LAG function returns data from preceding rows.

LAG uses the following syntax:

```
LAG ( Column name, Offset, Default )
```

LAG returns the value at an offset number of rows before the current row. Use the LAG function to compare values in the current row with values in a previous row. Use the following arguments with the LAG function:

- Column name. The column name whose value from the prior row is to be returned.
- Offset. The number of rows preceding the current row from which the data is to be retrieved. For example, an offset of "1" accesses the previous row, and an offset of "3" accesses the row that is three rows before the current row.
- Default. The default value to be returned if the offset is outside the scope of the partition. If you do not specify a default, the default is NULL.

For more information about the LAG function, see the *Informatica Transformation Language Reference*.

Aggregate Functions as Window Functions

In addition to LEAD and LAG, you can also use aggregate functions as window functions. When you use aggregate functions like SUM and AVG as window functions, you can perform running calculations that are similar to the stateful functions MOVINGSUM, MOVINGAVG, and CUME. Window functions are more flexible than stateful functions because you can set a specific end offset.

To use an aggregate function as a window function, you must define a frame in the windowing properties. You define a frame to limit the scope of the calculation. The aggregate function performs a calculation across the frame and produces a single value for each row.

Example

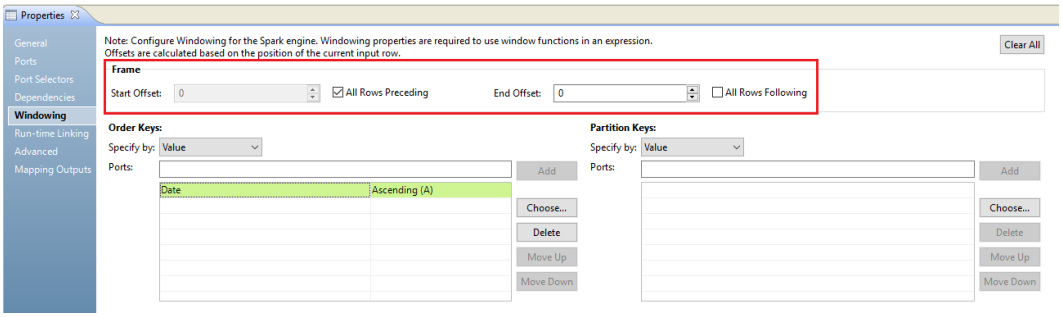
You are a lumber salesperson who sold different quantities of wood over the past two years. You want to calculate a running total of sales quantities.

The following table lists each sale ID, the date, and the quantity sold:

Sale_ID	Date	Quantity
30001	2016-08-02	10
10001	2016-12-24	10
10005	2016-12-24	30
40001	2017-01-09	40
10006	2017-01-18	10
20001	2017-02-12	20

A SUM function adds all the values and returns one output value. To get a running total for each row, you can define a frame for the function boundaries.

The following image shows the frame you specify on the Windowing tab:



You configure the following windowing properties:

- Start offset: All Rows Preceding
- End offset: 0
- Order Key: Date Ascending
- Partition Key: Not specified

You define the following aggregate function:

```
SUM (Quantity)
```

SUM adds the quantity in the current row to the quantities in all the rows preceding the current row. The function returns a running total for each row.

The following table lists a running sum for each date:

Sale_ID	Date	Quantity	Total
30001	2016-08-02	10	10
10001	2016-12-24	10	20
10005	2016-12-24	30	50
40001	2017-01-09	40	90

Sale_ID	Date	Quantity	Total
10006	2017-01-18	10	100
20001	2017-02-12	20	120

Aggregate Offsets

An aggregate function performs a calculation on a set of values inside a partition. If the frame offsets are outside the partition, the aggregate function ignores the frame.

If the offsets of a frame are not within the partition or table, the aggregate function calculates within the partition. The function does not return NULL or a default value.

For example, you partition a table by seller ID and you order by quantity. You set the start offset to -3 and the end offset to 4.

The following image shows the partition and frame for the current input row:

The frame includes eight total rows, but the calculation remains within the partition. If you define an AVG function with this frame, the function calculates the average of the quantities inside the partition and returns 18.75.

Nested Aggregate Functions

A nested aggregate function in a window function performs a calculation separately for each partition. A nested aggregate function behaves differently in a window function and an Aggregator transformation.

A nested aggregate function in an Aggregator transformation performs a calculation globally across all rows. A nested aggregate function in an Expression transformation performs a separate calculation for each partition.

For example, you configure the following nested aggregate function in an Aggregator transformation and an Expression transformation:

```
MEDIAN ( COUNT ( P1 ) )
```

You define P2 as the group by port in the Aggregator transformation.

The function takes the median of the count of the following set of data:

P1	P2
10	1
7	1
12	1
11	2
13	2
8	2
10	2

RETURN VALUE: 3.5

When you include nested aggregate functions in an Expression transformation and configure the transformation for windowing, the function performs the calculation separately for each partition.

You partition the data by P2 and specify a frame of all rows preceding and all rows following. The window function performs the following calculations:

1. COUNT (P1) produces one value for every row. COUNT returns the number of rows in the partition that have non-null values.
2. MEDIAN of that value produces the median of a window of values generated by COUNT.

The window function produces the following outputs:

P1	P2	Output
10	1	3
7	1	3
12	1	3
11	2	4
13	2	4
8	2	4
10	2	4

You can nest aggregate functions with multiple window functions. For example:

```
LAG ( LEAD( MAX( FIRST ( p1 )))
```

Note: You can nest any number of the window functions LEAD and LAG, but you cannot nest more than one aggregate function within another aggregate function.

Rules and Guidelines for Window Functions

Certain guidelines apply when you use window functions on the Spark engine.

Consider the following rules and guidelines when you define window functions in a transformation:

- Specify a constant integer as the offset argument in a window function.
- Specify a default argument that is the same data type as the input value.
- You cannot specify a default argument that contains complex data type or a SYSTIMESTAMP argument.
- To use the LEAD and LAG window functions, you must configure partition and order keys in the windowing properties.
- To use an aggregate function as a window function in an Expression transformation, you must configure a frame specification in the windowing properties.

Windowing Examples

The examples in this section demonstrate how to use LEAD, LAG, and other aggregate functions as window functions in an Expression transformation.

Financial Plans Example

You are a banker with information about the financial plans of two of your customers. Each plan has an associated start date.

For each customer, you want to know the expiration date for the current plan based on the activation date of the next plan. The previous plan ends when a new plan starts, so the end date for the previous plan is the start date of the next plan minus one day.

The following table lists the customer codes, the associated plan codes, and the start date of each plan:

CustomerCode	PlanCode	StartDate
C1	00001	2014-10-01
C2	00002	2014-10-01
C2	00002	2014-11-01
C1	00004	2014-10-25
C1	00001	2014-09-01
C1	00003	2014-10-10

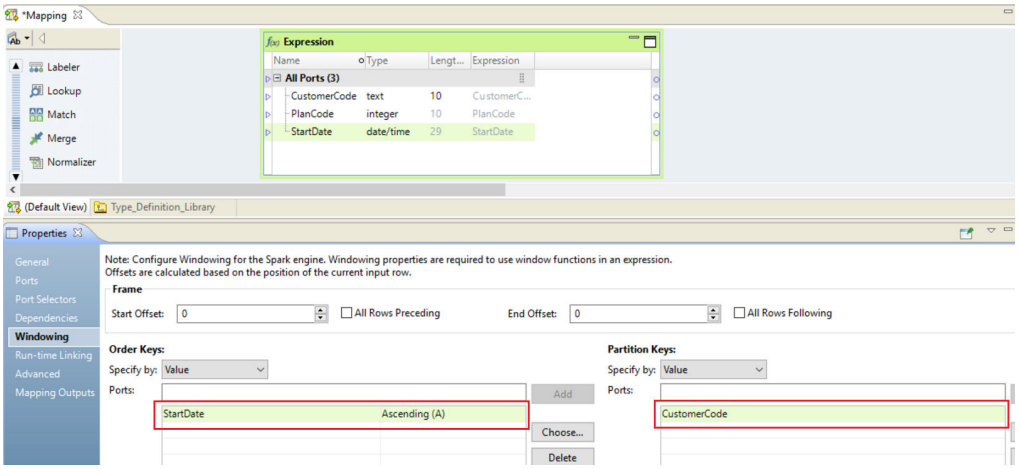
Define partition and order keys

You partition the data by customer code and order the data by ascending start date.

You configure the following windowing properties:

Property	Description
Order key	`StartDate Ascending`. Arranges the data chronologically by ascending start date.
Partition key	`CustomerCode`. Groups the rows according to customer code so that calculations are based on individual customers.
Frame	Not specified. Window functions access rows based on the offset argument and ignore the frame specification.

The following image shows the windowing properties you configure on the Windowing tab:



The following table lists the data grouped by customer code and ordered by start date:

CustomerCode	PlanCode	StartDate
C1	00001	2014-09-01
C1	00002	2014-10-01
C1	00003	2014-10-10
C1	00004	2014-10-25
C2	00001	2014-10-01
C2	00002	2014-11-01

The start dates for each customer are arranged in ascending order so that the dates are chronological.

Define a window function

You define a LEAD function to access the subsequent row for every input.

You define the following function on the Ports tab of the Expression transformation:

```
LEAD ( StartDate, 1, '01-Jan-2100' )
```

Where:

- `StartDate` indicates the target column that the function operates on.
- `1` is the offset. This value accesses the next immediate row.
- `01-Jan-2100` is the default value. The expression returns "01-Jan-2100" if the returned value is outside the bounds of the partition.

Define an ADD_TO_DATE function

You use an ADD_TO_DATE function to subtract one day from the date you accessed.

You define the following expression on the Ports tab of the Expression transformation:

```
ADD_TO_DATE ( LEAD ( StartDate, 1, '01-Jan-2100' ), 'DD', -1, )
```

By subtracting one day from the start date of the next plan, you find the end date of the current plan.

The following table lists the end dates of each plan:

CustomerCode	PlanCode	StartDate	EndDate
C1	00001	2014-09-01	2014-09-30
C1	00002	2014-10-01	2014-10-09
C1	00003	2014-10-10	2014-10-24
C1	00004	2014-10-25	2099-12-31*
C2	00001	2014-10-01	2014-10-31
C2	00002	2014-11-01	2099-12-31*

*The LEAD function returned the default value because these plans have not yet ended. The rows were outside the partition, so the ADD_TO_DATE function subtracted one day from 01-Jan-2100, returning 2099-12-31.

GPS Pings Example

Your organization receives GPS pings from vehicles that include trip and event IDs and a time stamp. You want to calculate the time difference between each ping and flag the row as skipped if the time difference with the previous row is less than 60 seconds.

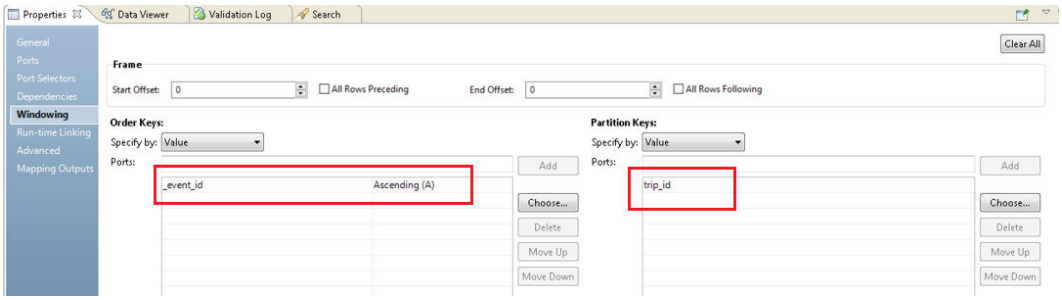
You order the events chronologically and partition the events by trip. You define a window function that accesses the event time from the previous row, and you use an ADD_TO_DATE function to calculate the time difference between the two events.

Windowing Properties

You define the following windowing properties on the Windowing tab:

Property	Description
Order key	`_event_id Ascending`. Arranges the data chronologically by ascending event ID.
Partition key	`trip_id`. Groups the rows according to trip ID so calculations are based on events from the same trip.
Frame	Not specified. Window functions access rows based on the offset argument and ignore the frame specification.

The following image shows the windowing properties you configure in the Expression transformation:



Window Function

You define the following LAG function to get the event time from the previous row:

```
LAG ( _event_time, 1, NULL )
```

Where:

- *_event_time* is the column name whose value from the previous row is to be returned.
- *1* is the offset. This value represents the row immediately before the current row.
- *NULL* is the default value. The function returns NULL if the return value is outside the bounds of the partition.

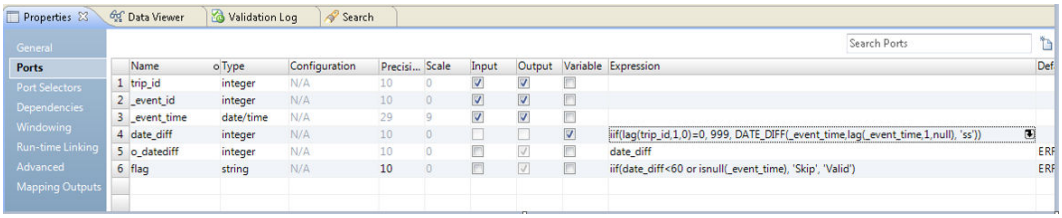
You define the following DATE_DIFF function to calculate the length of time between the two dates:

```
DATE_DIFF ( _event_time, LAG ( _event_time, 1, NULL ), 'ss' )
```

You flag the row as skipped if the DATE_DIFF is less than 60 seconds, or if the _event_time is NULL:

```
IIF ( DATE_DIFF < 60 or ISNULL ( _event_time ), 'Skip', 'Valid' )
```

The following image shows the expressions you define in the transformation properties:



Output

The transformation produces the following outputs:

Trip ID	Event ID	Event Time	Time Difference	Flag
101	1	2017-05-03 12:00:00	NULL*	Skip
101	2	2017-05-03 12:00:34	34	Skip
101	3	2017-05-03 12:02:00	86	Valid
101	4	2017-05-03 12:02:23	23	Skip
102	1	2017-05-03 12:00:00	NULL*	Skip
102	2	2017-05-03 12:01:56	116	Valid
102	3	2017-05-03 12:02:00	4	Skip
102	4	2017-05-03 13:00:00	3480	Valid
103	1	2017-05-03 12:00:00	NULL*	Skip
103	2	2017-05-03 12:00:12	12	Skip
103	3	2017-05-03 12:01:12	60	Valid

*The rows preceding these rows are outside the bounds of the partition, so the transformation produced NULL values.

Aggregate Function as Window Function Example

You work for a human resources group and you want to compare each of your employees' salaries with the average salary in his or her department:

The following table lists the department names, the employee identification number, and the employee's salary:

Department	Employee	Salary
Development	11	5200
Development	7	4200
Development	9	4500
Development	8	6000
Development	10	5200
Personnel	5	3500
Personnel	2	3900
Sales	3	4800

Department	Employee	Salary
Sales	1	5000
Sales	4	4800

You set an unbounded frame to include all employees in the calculation, and you define an aggregate function to calculate the difference between each employee's salary and the average salary in the department.

Windowing Properties

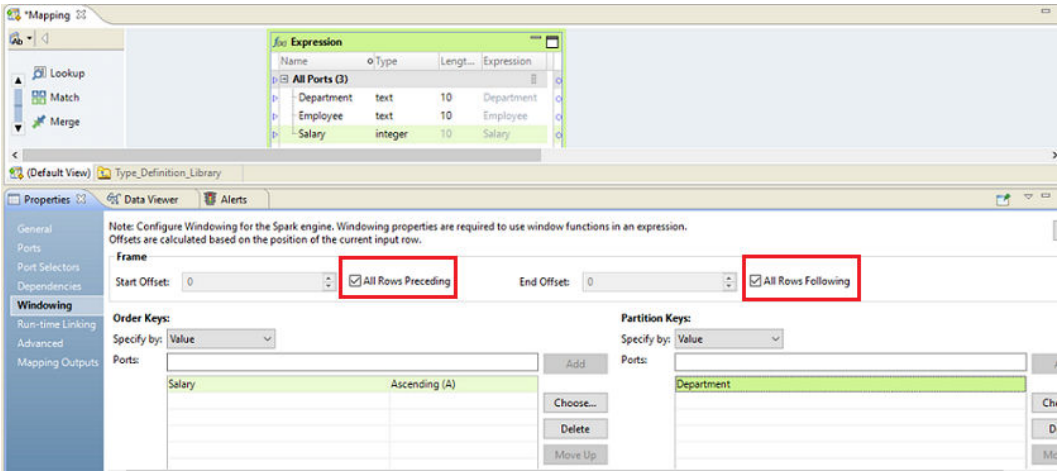
You define the following windowing properties on the Windowing tab:

Property	Description
Order key	`Salary Ascending`. Arranges the data by increasing salary.
Partition key	`Department`. Groups the rows according to department.
Start offset	`All Rows Preceding`
End offset	`All Rows Following`

With an unbounded frame, the aggregate function includes all partition rows in the calculation.

For example, suppose the current row is the third row. The third row is in the "Development" partition, so the frame includes the third row in addition to all rows before and after the third row in the "Development" partition.

The following image shows the windowing properties you configure in the Expression transformation:



Window Function

An aggregate function acts as a window function when you configure the transformation for windowing.

You define the following aggregate function to calculate the difference between each employee's salary and the average salary in his or her department:

```
Salary - AVG ( Salary ) = Salary_Diff
```

Output

The transformation produces the following salary differences:

Department	Employee	Salary	Salary_Diff
Development	11	5200	-820
Development	7	4200	-520
Development	9	4500	180
Development	8	6000	180
Development	10	5200	980
Personnel	5	3500	200
Personnel	2	3900	200
Sales	3	4800	-66
Sales	1	5000	-66
Sales	4	4800	134

You can identify which employees are making less or more than the average salary for his or her department. Based on this information, you can add other transformations to learn more about your data. For example, you might add a Rank transformation to produce a numerical rank for each employee within his or her department.

APPENDIX A

Connections Reference

This appendix includes the following topics:

- Connections Overview, 312
- Cloud Provisioning Configuration, 312
- Amazon Redshift Connection Properties, 322
- Amazon S3 Connection Properties, 324
- Blockchain Connection Properties, 326
- Cassandra Connection Properties, 328
- Confluent Kafka Connection, 329
- Databricks Connection Properties, 331
- Google Analytics Connection Properties, 333
- Google BigQuery Connection Properties, 334
- Google Cloud Spanner Connection Properties, 336
- Google Cloud Storage Connection Properties, 337
- Google PubSub Connection Properties, 338
- Hadoop Connection Properties, 338
- HDFS Connection Properties, 344
- HBase Connection Properties, 346
- HBase Connection Properties for MapR-DB, 346
- Hive Connection Properties, 347
- JDBC Connection Properties, 350
- JDBC V2 Connection Properties, 357
- Kafka Connection Properties, 359
- Kudu Connection Properties , 362
- Microsoft Azure Blob Storage Connection Properties, 363
- Microsoft Azure Cosmos DB SQL API Connection Properties, 364
- Microsoft Azure Data Lake Storage Gen1 Connection Properties, 365
- Microsoft Azure Data Lake Storage Gen2 Connection Properties, 366
- Microsoft Azure SQL Data Warehouse Connection Properties, 367
- Snowflake Connection Properties, 369
- Creating a Connection to Access Sources or Targets, 370

- Creating a Hadoop Connection, 370
- Configuring Hadoop Connection Properties, 371

Connections Overview

Create a connection to access non-native environments, Hadoop and Databricks. If you access HBase, HDFS, or Hive sources or targets in the Hadoop environment, you must also create those connections. You can create the connections using the Developer tool, Administrator tool, and infacmd.

You can create the following types of connections:

Hadoop connection

Create a Hadoop connection to run mappings in the Hadoop environment.

HBase connection

Create an HBase connection to access HBase. The HBase connection is a NoSQL connection.

HDFS connection

Create an HDFS connection to read data from or write data to the HDFS file system on a Hadoop cluster.

Hive connection

Create a Hive connection to access Hive as a source or target. You can access Hive as a source if the mapping is enabled for the native or Hadoop environment. You can access Hive as a target if the mapping runs on the Blaze engine.

JDBC connection

Create a JDBC connection and configure Sqoop properties in the connection to import and export relational data through Sqoop.

Databricks connection

Create a Databricks connection to run mappings in the Databricks environment.

Note: For information about creating connections to other sources or targets such as social media web sites or Teradata, see the respective PowerExchange adapter user guide for information.

Cloud Provisioning Configuration

The cloud provisioning configuration establishes a relationship between the Create Cluster task and the cluster connection that the workflows use to run mapping tasks. The Create Cluster task must include a reference to the cloud provisioning configuration. In turn, the cloud provisioning configuration points to the cluster connection that you create for use by the cluster workflow.

The properties to populate depend on the Hadoop distribution you choose to build a cluster on. Choose one of the following connection types:

- AWS Cloud Provisioning. Connects to an Amazon EMR cluster on Amazon Web Services.
- Azure Cloud Provisioning. Connects to an HDInsight cluster on the Azure platform.
- Databricks Cloud Provisioning. Connects to a Databricks cluster on the Azure Databricks platform.

AWS Cloud Provisioning Configuration Properties

The properties in the AWS cloud provisioning configuration enable the Data Integration Service to contact and create resources on the AWS cloud platform.

General Properties

The following table describes cloud provisioning configuration general properties:

Property	Description
Name	Name of the cloud provisioning configuration.
ID	ID of the cloud provisioning configuration. Default: Same as the cloud provisioning configuration name.
Description.	Optional. Description of the cloud provisioning configuration.
AWS Access Key ID	Optional. ID of the AWS access key, which AWS uses to control REST or HTTP query protocol requests to AWS service APIs. If you do not specify a value, Informatica attempts to follow the Default Credential Provider Chain.
AWS Secret Access Key	Secret component of the AWS access key. Required if you specify the AWS Access Key ID.
Region	Region in which to create the cluster. This must be the region in which the VPC is running. Use AWS region values. For a list of acceptable values, see AWS documentation. **Note:** The region where you want to create the cluster can be different from the region in which the Informatica domain is installed.

Permissions

The following table describes cloud provisioning configuration permissions properties:

Property	Description
EMR Role	Name of the service role for the EMR cluster that you create. The role must have sufficient permissions to create a cluster, access S3 resources, and run jobs on the cluster. When the AWS administrator creates this role, they select the “EMR” role. This contains the default AmazonElasticMapReduceRole policy. You can edit the services in this policy.
EC2 Instance Profile	Name of the EC2 instance profile role that controls permissions on processes that run on the cluster. When the AWS administrator creates this role, they select the “EMR Role for EC2” role. This includes S3 access by default.
Auto Scaling Role	Required if you configure auto-scaling for the EMR cluster. This role is created when the AWS administrator configures auto-scaling on any cluster in the VPC. Default: When you leave this field blank, it is equivalent to setting the Auto Scaling role to “Proceed without role” when the AWS administrator creates a cluster in the AWS console.

EC2 Configuration

The following table describes cloud provisioning configuration EC2 configuration properties:

Property	Description
EC2 Key Pair	EC2 key pair to enable communication with the EMR cluster master node. Optional. This credential enables you to log into the cluster. Configure this property if you intend the cluster to be non-ephemeral.
EC2 Subnet	ID of the subnet on the VPC in which to create the cluster. Use the subnet ID of the EC2 instance where the cluster runs.
Master Security Group	Optional. ID of the security group for the cluster master node. Acts as a virtual firewall to control inbound and outbound traffic to cluster nodes. Security groups are created when the AWS administrator creates and configures a cluster in a VPC. In the AWS console, the property is equivalent to ElasticMapReduce-master. You can use existing security groups, or the AWS administrator might create dedicated security groups for the ephemeral cluster. If you do not specify a value, the cluster applies the default security group for the VPC.
Additional Master Security Groups	Optional. IDs of additional security groups to attach to the cluster master node. Use a comma-separated list of security group IDs.
Core and Task Security Group	Optional. ID of the security group for the cluster core and task nodes. When the AWS administrator creates and configures a cluster In the AWS console, the property is equivalent to the ElasticMapReduce-slave security group If you do not specify a value, the cluster applies the default security group for the VPC.
Additional Core and Task Security Groups	Optional. IDs of additional security groups to attach to cluster core and task nodes. Use a comma-separated list of security group IDs.
Service Access Security Group	EMR managed security group for service access. Required when you provision an EMR cluster in a private subnet.

General Properties

The following table describes cloud provisioning configuration general properties:

Property	Description
Name	Name of the cloud provisioning configuration.
ID	ID of the cloud provisioning configuration. Default: Same as the cloud provisioning configuration name.
Description.	Optional. Description of the cloud provisioning configuration.
AWS Access Key ID	Optional. ID of the AWS access key, which AWS uses to control REST or HTTP query protocol requests to AWS service APIs. If you do not specify a value, Informatica attempts to follow the Default Credential Provider Chain.

Property	Description
AWS Secret Access Key	Secret component of the AWS access key. Required if you specify the AWS Access Key ID.
Region	Region in which to create the cluster. This must be the region in which the VPC is running. Use AWS region values. For a list of acceptable values, see AWS documentation. **Note:** The region where you want to create the cluster can be different from the region in which the Informatica domain is installed.

Permissions

The following table describes cloud provisioning configuration permissions properties:

Property	Description
EMR Role	Name of the service role for the EMR cluster that you create. The role must have sufficient permissions to create a cluster, access S3 resources, and run jobs on the cluster. When the AWS administrator creates this role, they select the “EMR” role. This contains the default AmazonElasticMapReduceRole policy. You can edit the services in this policy.
EC2 Instance Profile	Name of the EC2 instance profile role that controls permissions on processes that run on the cluster. When the AWS administrator creates this role, they select the “EMR Role for EC2” role. This includes S3 access by default.
Auto Scaling Role	Required if you configure auto-scaling for the EMR cluster. This role is created when the AWS administrator configures auto-scaling on any cluster in the VPC. Default: When you leave this field blank, it is equivalent to setting the Auto Scaling role to “Proceed without role” when the AWS administrator creates a cluster in the AWS console.

EC2 Configuration

The following table describes cloud provisioning configuration EC2 configuration properties:

Property	Description
EC2 Key Pair	EC2 key pair to enable communication with the EMR cluster master node. Optional. This credential enables you to log into the cluster. Configure this property if you intend the cluster to be non-ephemeral.
EC2 Subnet	ID of the subnet on the VPC in which to create the cluster. Use the subnet ID of the EC2 instance where the cluster runs.
Master Security Group	Optional. ID of the security group for the cluster master node. Acts as a virtual firewall to control inbound and outbound traffic to cluster nodes. Security groups are created when the AWS administrator creates and configures a cluster in a VPC. In the AWS console, the property is equivalent to ElasticMapReduce-master. You can use existing security groups, or the AWS administrator might create dedicated security groups for the ephemeral cluster. If you do not specify a value, the cluster applies the default security group for the VPC.

Property	Description
Additional Master Security Groups	Optional. IDs of additional security groups to attach to the cluster master node. Use a comma-separated list of security group IDs.
Core and Task Security Group	Optional. ID of the security group for the cluster core and task nodes. When the AWS administrator creates and configures a cluster In the AWS console, the property is equivalent to the ElasticMapReduce-slave security group If you do not specify a value, the cluster applies the default security group for the VPC.
Additional Core and Task Security Groups	Optional. IDs of additional security groups to attach to cluster core and task nodes. Use a comma-separated list of security group IDs.
Service Access Security Group	EMR managed security group for service access. Required when you provision an EMR cluster in a private subnet.

Azure Cloud Provisioning Configuration Properties

The properties in the Azure cloud provisioning configuration enable the Data Integration Service to contact and create resources on the Azure cloud platform.

Authentication Details

The following table describes authentication properties to configure:

Property	Description
Name	Name of the cloud provisioning configuration.
ID	ID of the cloud provisioning configuration. Default: Same as the cloud provisioning configuration name.
Description	Optional. Description of the cloud provisioning configuration.
Subscription ID	ID of the Azure account to use in the cluster creation process.
Tenant ID	A GUID string associated with the Azure Active Directory.
Client ID	A GUID string that is the same as the Application ID associated with the Service Principal. The Service Principal must be assigned to a role that has permission to create resources in the subscription that you identified in the Subscription ID property.
Client Secret	An octet string that provides a key associated with the client ID.

Storage Account Details

Choose to configure access to one of the following storage types:

- Azure Data Lake Storage (ADLS). See Azure documentation.
- An Azure Storage Account, known as general or blob storage. See Azure documentation.

The following table describes the information you need to configure Azure Data Lake Storage (ADLS) with the HDInsight cluster:

Property	Description
Azure Data Lake Store Name	Name of the ADLS storage to access. The ADLS storage and the cluster to create must reside in the same region.
Data Lake Service Principal Client ID	A credential that enables programmatic access to ADLS storage. Enables the Informatica domain to communicate with ADLS and run commands and mappings on the HDInsight cluster. The service principal is an Azure user that meets the following requirements: - Permissions to access required directories in ADLS storage. - Certificate-based authentication for ADLS storage. - Key-based authentication for ADLS storage.
Data Lake Service Principal Certificate Contents	The Base64 encoded text of the public certificate used with the service principal. Leave this property blank when you create the cloud provisioning configuration. After you save the cloud provisioning configuration, log in to the VM where the Informatica domain is installed and run infacmd ccps updateADLSCertificate to populate this property.
Data Lake Service Principal Certificate Password	Private key for the service principal. This private key must be associated with the service principal certificate.
Data Lake Service Principal Client Secret	An octet string that provides a key associated with the service principal.
Data Lake Service Principal OAUTH Token Endpoint	Endpoint for OAUTH token based authentication.

The following table describes the information you need to configure Azure General Storage, also known as blob storage, with the HDInsight cluster:

Property	Description
Azure Storage Account Name	Name of the storage account to access. Get the value from the Storage Accounts node in the Azure web console. The storage and the cluster to create must reside in the same region.
Azure Storage Account Key	A key to authenticate access to the storage account. To get the value from the Azure web console, select the storage account, then Access Keys. The console displays the account keys.

Cluster Deployment Details

The following table describes the cluster deployment properties that you configure:

Property	Description
Resource Group	Resource group in which to create the cluster. A resource group is a logical set of Azure resources.
Virtual Network Resource Group	Optional. Resource group to which the virtual network belongs. If you do not specify a resource group, the Data Integration Service assumes that the virtual network is a member of the same resource group as the cluster.
Virtual Network	Name of the virtual network or vnet where you want to create the cluster. Specify a vnet that resides in the resource group that you specified in the Virtual Network Resource Group property. The vnet must be in the same region as the region in which to create the cluster.
Subnet Name	Subnet in which to create the cluster. The subnet must be a part of the vnet that you designated in the previous property. Each vnet can have one or more subnets. The Azure administrator can choose an existing subnet or create one for the cluster.

External Hive Metastore Details

You can specify the properties to enable the cluster to connect to a Hive metastore database that is external to the cluster.

If you do not specify an existing external database in this dialog box, the cluster creates its own database on the cluster. This database is terminated when the cluster is terminated.

You can use an external relational database like MySQL or Amazon RDS as the Hive metastore database. The external database must be on the same cloud platform as the cluster to create.

The following table describes the Hive metastore database properties that you configure:

Property	Description
Database Name	Name of the Hive metastore database.
Database Server Name	Server on which the database resides. **Note:** The database server name on the Azure web console commonly includes the suffix `database.windows.net`. For example: `server123xyz.database.windows.net`. You can specify the database server name without the suffix and Informatica will automatically append the suffix. For example, you can specify `server123xyz`.
Database User Name	User name of the account for the domain to use to access the database.
Database Password	Password for the user account.

Authentication Details

The following table describes authentication properties to configure:

Property	Description
Name	Name of the cloud provisioning configuration.
ID	ID of the cloud provisioning configuration. Default: Same as the cloud provisioning configuration name.
Description	Optional. Description of the cloud provisioning configuration.
Subscription ID	ID of the Azure account to use in the cluster creation process.
Tenant ID	A GUID string associated with the Azure Active Directory.
Client ID	A GUID string that is the same as the Application ID associated with the Service Principal. The Service Principal must be assigned to a role that has permission to create resources in the subscription that you identified in the Subscription ID property.
Client Secret	An octet string that provides a key associated with the client ID.

Storage Account Details

Choose to configure access to one of the following storage types:

- Azure Data Lake Storage (ADLS). See Azure documentation.
- An Azure Storage Account, known as general or blob storage. See Azure documentation.

The following table describes the information you need to configure Azure Data Lake Storage (ADLS) with the HDInsight cluster:

Property	Description
Azure Data Lake Store Name	Name of the ADLS storage to access. The ADLS storage and the cluster to create must reside in the same region.
Data Lake Service Principal Client ID	A credential that enables programmatic access to ADLS storage. Enables the Informatica domain to communicate with ADLS and run commands and mappings on the HDInsight cluster. The service principal is an Azure user that meets the following requirements: - Permissions to access required directories in ADLS storage. - Certificate-based authentication for ADLS storage. - Key-based authentication for ADLS storage.
Data Lake Service Principal Certificate Contents	The Base64 encoded text of the public certificate used with the service principal. Leave this property blank when you create the cloud provisioning configuration. After you save the cloud provisioning configuration, log in to the VM where the Informatica domain is installed and run infacmd ccps updateADLSCertificate to populate this property.
Data Lake Service Principal Certificate Password	Private key for the service principal. This private key must be associated with the service principal certificate.

Property	Description
Data Lake Service Principal Client Secret	An octet string that provides a key associated with the service principal.
Data Lake Service Principal OAUTH Token Endpoint	Endpoint for OAUTH token based authentication.

The following table describes the information you need to configure Azure General Storage, also known as blob storage, with the HDInsight cluster:

Property	Description
Azure Storage Account Name	Name of the storage account to access. Get the value from the Storage Accounts node in the Azure web console. The storage and the cluster to create must reside in the same region.
Azure Storage Account Key	A key to authenticate access to the storage account. To get the value from the Azure web console, select the storage account, then Access Keys. The console displays the account keys.

Cluster Deployment Details

The following table describes the cluster deployment properties that you configure:

Property	Description
Resource Group	Resource group in which to create the cluster. A resource group is a logical set of Azure resources.
Virtual Network Resource Group	Optional. Resource group to which the virtual network belongs. If you do not specify a resource group, the Data Integration Service assumes that the virtual network is a member of the same resource group as the cluster.
Virtual Network	Name of the virtual network or vnet where you want to create the cluster. Specify a vnet that resides in the resource group that you specified in the Virtual Network Resource Group property. The vnet must be in the same region as the region in which to create the cluster.
Subnet Name	Subnet in which to create the cluster. The subnet must be a part of the vnet that you designated in the previous property. Each vnet can have one or more subnets. The Azure administrator can choose an existing subnet or create one for the cluster.

External Hive Metastore Details

You can specify the properties to enable the cluster to connect to a Hive metastore database that is external to the cluster.

You can use an external relational database like MySQL or Amazon RDS as the Hive metastore database. The external database must be on the same cloud platform as the cluster to create.

If you do not specify an existing external database in this dialog box, the cluster creates its own database on the cluster. This database is terminated when the cluster is terminated.

The following table describes the Hive metastore database properties that you configure:

Property	Description
Database Name	Name of the Hive metastore database.
Database Server Name	Server on which the database resides. **Note:** The database server name on the Azure web console commonly includes the suffix `database.windows.net`. For example: `server123xyz.database.windows.net`. You can specify the database server name without the suffix and Informatica will automatically append the suffix. For example, you can specify `server123xyz`.
Database User Name	User name of the account for the domain to use to access the database.
Database Password	Password for the user account.

Databricks Cloud Provisioning Configuration Properties

The properties in the Databricks cloud provisioning configuration enable the Data Integration Service to contact and create resources on the Databricks cloud platform.

The following table describes the Databricks cloud provisioning configuration properties:

Property	Description
Name	Name of the cloud provisioning configuration. **Tip:** Because the Administrator tool lists cloud provisioning configuration objects with other connections, use a naming convention such as "CPC" as part of the name of the object to help identify it.
ID	The cluster ID of the Databricks cluster.
Description	Optional description of the cloud provisioning configuration.
Databricks domain	Domain name of the Databricks deployment.
Databricks token ID	The token ID created within Databricks required for authentication. **Note:** If the token has an expiration date, verify that you get a new token from the Databricks administrator before it expires.
Advanced Properties	Advanced properties that are unique to the Databricks cloud provisioning configuration.

Advanced Properties

Configure the following properties in the **Advanced Properties** of the Databricks configuration section:

infaspark.pythontx.exec

Required to run a Python transformation on the Databricks Spark engine. Set to the location of the Python executable binary on the worker nodes in the Databricks cluster.

When you provision the cluster at run time, set this property in the Databricks cloud provisioning configuration. Otherwise, set on the Databricks connection.

For example, set to:

```
infaspark.pythontx.exec=/databricks/python3/bin/python3
```

infaspark.pythontx.executorEnv.PYTHONHOME

Required to run a Python transformation on the Databricks Spark engine. Set to the location of the Python installation directory on the worker nodes in the Databricks cluster.

When you provision the cluster at run time, set this property in the Databricks cloud provisioning configuration. Otherwise, set on the Databricks connection.

For example, set to:

```
infaspark.pythontx.executorEnv.PYTHONHOME=/databricks/python3
```

Amazon Redshift Connection Properties

When you set up an Amazon Redshift connection, you must configure the connection properties.

The following table describes the Amazon Redshift connection properties:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters:~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	The description of the connection. The description cannot exceed 4,000 characters.
Location	The domain where you want to create the connection.
Type	The connection type. Select Amazon Redshift in the Database.

The **Details** tab contains the connection attributes of the Amazon Redshift connection. The following table describes the connection attributes:

Property	Description
Username	User name of the Amazon Redshift account.
Password	Password for the Amazon Redshift account.
Access Key ID	Amazon S3 bucket access key ID. **Note:** Required if you do not use AWS Identity and Access Management (IAM) authentication.
Secret Access Key	Amazon S3 bucket secret access key ID. **Note:** Required if you do not use AWS Identity and Access Management (IAM) authentication.

Property	Description
Master Symmetric Key	Optional. Provide a 256-bit AES encryption key in the Base64 format when you enable client-side encryption. You can generate a key using a third-party tool. If you specify a value, ensure that you specify the encryption type as client side encryption in the advanced target properties.
JDBC URL	Amazon Redshift connection URL.
Cluster Region	Optional. The AWS cluster region in which the bucket you want to access resides. Select a cluster region if you choose to provide a custom JDBC URL that does not contain a cluster region name in the **JDBC URL** connection property. If you specify a cluster region in both **Cluster Region** and **JDBC URL** connection properties, the Data Integration Service ignores the cluster region that you specify in the **JDBC URL** connection property. To use the cluster region name that you specify in the **JDBC URL** connection property, select **None** as the cluster region in this property. Select one of the following cluster regions: Select one of the following regions: - Asia Pacific (Mumbai) - Asia Pacific (Seoul) - Asia Pacific (Singapore) - Asia Pacific (Sydney) - Asia Pacific (Tokyo) - AWS GovCloud (US) - Canada (Central) - China (Beijing) - China (Ningxia) - EU (Ireland) - EU (Frankfurt) - EU (London) - EU (Paris) - South America (Sao Paulo) - US East (Ohio) - US East (N. Virginia) - US West (N. California) - US West (Oregon) Default is None. You can only read data from or write data to the cluster regions supported by AWS SDK used by PowerExchange for Amazon Redshift.
Customer Master Key ID	Optional. Specify the customer master key ID generated by AWS Key Management Service (AWS KMS) or the Amazon Resource Name (ARN) of your custom key for cross-account access. You must generate the customer master key corresponding to the region where Amazon S3 bucket resides. You can specify any of the following values: **Customer generated customer master key** Enables client-side or server-side encryption. **Default customer master key** Enables client-side or server-side encryption. Only the administrator user of the account can use the default customer master key ID to enable client-side encryption.

Amazon S3 Connection Properties

When you set up an Amazon S3 connection, you must configure the connection properties.

The following table describes the Amazon S3 connection properties:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters:~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	Optional. The description of the connection. The description cannot exceed 4,000 characters.
Location	The domain where you want to create the connection.
Type	The Amazon S3 connection type.
Access Key	Access key to access the Amazon S3 bucket. Provide the access key value based on the following authentication methods: - Basic authentication: provide the actual access key value. - IAM authentication: do not provide the access key value. - Temporary security credentials via assume role: provide access key of an IAM user with no permissions to access Amazon S3 bucket.
Secret Key	Secret access key to access the Amazon S3 bucket. The secret key is associated with the access key and uniquely identifies the account. Provide the access key value based on the following authentication methods: - Basic authentication: provide the actual access secret value. - IAM authentication: do not provide the access secret value. - Temporary security credentials via assume role: provide access secret of an IAM user with no permissions to access Amazon S3 bucket.
IAM Role ARN	The ARN of the IAM role assumed by the user to use the dynamically generated temporary security credentials. Enter the value of this property if you want to use the temporary security credentials to access the AWS resources. If you want to use the temporary security credentials with IAM authentication, do not provide the Access Key and Secret Key connection properties. If you want to use the temporary security credentials without IAM authentication, you must enter the value of the Access Key and Secret Key connection properties. For more information about how to obtain the ARN of the IAM role, see the AWS documentation.
Folder Path	The complete path to Amazon S3 objects. The path must include the bucket name and any folder name. Do not use a slash at the end of the folder path. For example, `<bucket name>/<my folder name>`.
Master Symmetric Key	Optional. Provide a 256-bit AES encryption key in the Base64 format when you enable client-side encryption. You can generate a master symmetric key using a third-party tool.

Property	Description
S3 Account Type	The type of the Amazon S3 account. Select **Amazon S3 Storage** or **S3 Compatible Storage**. Select the Amazon S3 storage option to use the Amazon S3 services. Select the S3 compatible storage option to specify the endpoint for a third-party storage provider such as Scality RING. By default, Amazon S3 storage is selected.
REST Endpoint	The S3 storage endpoint. Specify the S3 storage endpoint in HTTP/HTTPs format when you select the S3 compatible storage option. For example, `http://s3.isv.scality.com`.
Region Name	Select the AWS region in which the bucket you want to access resides. Select one of the following regions: - Asia Pacific (Mumbai) - Asia Pacific (Seoul) - Asia Pacific (Singapore) - Asia Pacific (Sydney) - Asia Pacific (Tokyo) - AWS GovCloud (US) - Canada (Central) - China (Beijing) - China (Hong Kong) - China (Ningxia) - EU (Ireland) - EU (Frankfurt) - EU (London) - EU (Paris) - South America (Sao Paulo) - US East (Ohio) - US East (N. Virginia) - US West (N. California) - US West (Oregon) Default is US East (N. Virginia). Not applicable for S3 compatible storage.
Customer Master Key ID	Optional. Specify the customer master key ID or alias name generated by AWS Key Management Service (AWS KMS) or the Amazon Resource Name (ARN) of your custom key for cross-account access. You must generate the customer master key for the same region where Amazon S3 bucket reside. You can specify any of the following values: **Customer generated customer master key** Enables client-side or server-side encryption. **Default customer master key** Enables client-side or server-side encryption. Only the administrator user of the account can use the default customer master key ID to enable client-side encryption.
Federated SSO IdP	SAML 2.0-enabled identity provider for the federated user single sign-on to use with the AWS account. PowerExchange for Amazon S3 supports only the `ADFS 3.0` identity provider. Select `None` if you do not want to use federated user single sign-on.

Federated user single sign-on connection properties

Configure the following properties when you select `ADFS 3.0` in **Federated SSO IdP**:

Property	Description
Federated User Name	User name of the federated user to access the AWS account through the identity provider.
Federated User Password	Password for the federated user to access the AWS account through the identity provider.
IdP SSO URL	Single sign-on URL of the identity provider for AWS.
SAML Identity Provider ARN	ARN of the SAML identity provider that the AWS administrator created to register the identity provider as a trusted provider.
Role ARN	ARN of the IAM role assumed by the federated user.

Blockchain Connection Properties

When you set up a blockchain connection, you must configure the connection properties.

The following table describes the general connection properties for a blockchain connection:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: `~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /`
ID	The string that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	The description of the connection. Enter a string that you can use to identify the connection. The description cannot exceed 4,000 characters.
Swagger File Path	The absolute path of the swagger file path that contains the REST API to communicate with the blockchain. The swagger file must be a JSON file that is stored on the Data Integration Service machine. If the swagger file is in a different file format, such as YAML, convert the file to JSON format.
Base URL	Required. The base URL that is used to access assets on the blockchain.
Auth Type*	Authentication method that the run-time engine uses to connect to the REST server. You can use none, basic, digest, or OAuth.
Auth User ID*	User name to authenticate to the REST server.
Auth Password*	Password for the user name to authenticate to the REST server.

Property	Description
OAuth Consumer Key*	Required for the OAuth authentication type. Client key that is associated with the REST server.
OAuth Consumer Secret*	Required for the OAuth authentication type. Client password to connect to the REST server.
OAuth Token*	Required for the OAuth authentication type. Access token to connect to the REST server.
OAuth Token Secret*	Required for the OAuth authentication type. Password associated with the OAuth token.
Proxy Type*	Type of proxy. You can use no proxy, platform proxy, or custom.
Proxy Details*	Proxy configuration using the format `<host>:<port>`.
TrustStore File Path*	The absolute path of the truststore file that contains the SSL certificate.
TrustStore Password*	Password for the truststore file.
KeyStore File Path*	The absolute path of the keystore file that contains the keys and certificates required to establish a two-way secure connection with the REST server.
KeyStore Password*	Password for the keystore file.
Advanced Properties	List of advanced properties to access an asset on the blockchain. Specify the advanced properties using name-value pairs that are separated by a semicolon. You can use the following advanced properties: - X-API-KEY. Required if you authenticate to the REST server using an API key. The advanced properties that you configure in the connection override the values for the corresponding advanced properties in the blockchain data object. For example, if the connection and the data object both specify a base URL, the value in the connection overrides the value in the data object. **Note:** The advanced properties have the precedence level,**Operation level** > **Object level** > **Connection level**. The properties configured at the operation level will override the properties configured at the object or connection level.
Cookies	Required based on how the REST API is implemented. List of cookie properties to specify the cookie information that is passed to the REST server. Specify the properties using name-value pairs that are separated by a semicolon. The cookie properties that you configure in the connection override the values for the corresponding cookie properties in the blockchain data object.
* The property is ignored. To use the functionality, configure the property as an advanced property and provide a name-value pair based on the property name in the swagger file. For example, configure the following name-value pair to use basic authorization: `Authorization=Basic <credentials>` **Note:** You cannot use **Test Connection** to validate a Blockchain connection.	

Cassandra Connection Properties

When you set up a Cassandra connection, you must configure the connection properties.

Note: The order of the connection properties might vary depending on the tool where you view them.

The following table describes the Cassandra connection properties:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: ~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. The ID must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	Optional. The description of the connection. The description cannot exceed 4,000 characters.
Location	The domain where you want to create the connection. Not applicable for Data Engineering Streaming.
Type	The connection type. Select **Cassandra**.
Host Name	Host name or IP address of the Cassandra server.
Port	Cassandra server port number. Default is 9042.
User Name	User name to access the Cassandra server.
Password	Password corresponding to the user name to access the Cassandra server.
Default Keyspace	Name of the Cassandra keyspace to use by default.
SQL Identifier Character	Type of character that the database uses to enclose delimited identifiers in SQL or CQL queries. The available characters depend on the database type. Select **None** if the database uses regular identifiers. When the Data Integration Service generates SQL or CQL queries, the service does not place delimited characters around any identifiers. Select a character if the database uses delimited identifiers. When the Data Integration Service generates SQL or CQL queries, the service encloses delimited identifiers within this character.
SSL Mode	Select **disabled**. Not applicable for PowerExchange for Cassandra JDBC. SSL mode indicates the encryption type to use for the connection. You can choose a mode from the following SSL modes: - Disabled - One way - Two way

Property	Description
SSL Truststore Path	Not applicable for PowerExchange for Cassandra JDBC or when **disabled** SSL mode is selected. Absolute path and file name of the SSL truststore file that contains certificates of the trusted SSL server.
SSL Truststore Password	Not applicable for PowerExchange for Cassandra JDBC or when **disabled** SSL mode is selected. Password for the SSL truststore.
SSL Keystore Path	Not applicable for PowerExchange for Cassandra JDBC or when **disabled** SSL mode is selected. Absolute path and file name of the SSL keystore file that contains private keys and certificates for the SSL server.
SSL Keystore Password	Not applicable for PowerExchange for Cassandra JDBC or when **disabled** SSL mode is selected. Password for the SSL keystore.
Additional Connection Properties	Enter one or more JDBC connection parameters in the following format: `<param1>=<value>;<param2>=<value>;<param3>=<value>` PowerExchange for Cassandra JDBC supports the following JDBC connection parameters: - BinaryColumnLength - DecimalColumnScale - EnableCaseSensitive - EnableNullInsert - EnablePaging - RowsPerPage - StringColumnLength - VTTableNameSeparator

Confluent Kafka Connection

The Confluent Kafka connection is a Messaging connection. Use the Confluent Kafka connection to access a Kafka broker or a Confluent Kafka broker as a source or a target. You can create and manage a Confluent Kafka connection in the Developer tool or through infacmd.

When you configure a Confluent Kafka connection, you configure the following properties:

- List of Kafka brokers or Confluent Kafka brokers that the connection reads from or writes to.
- Number of seconds the Integration Service attempts to reconnect to the database if the connection fails.
- Version of the Confluent Kafka messaging broker.

General Properties

The following table describes the general connection properties for the Confluent Kafka connection:

Property	Description
Name	Name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: ~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	The string that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	Description of the connection. Enter a string that you can use to identify the connection. The description cannot exceed 4,000 characters.
Location	Domain where you want to create the connection. Select the domain name.
Type	Connection type. Select `Messaging/ConfluentKafka`.

Confluent Kafka Broker Properties

The following table describes the Kafka broker properties for the Confluent Kafka connection:

Property	Description
Kafka Broker List	Comma-separated list of Confluent Kafka brokers that maintain the configuration of the Confluent Kafka messaging broker. To specify a Confluent Kafka broker, use the following format: `<IP address>:<port>`
Retry Timeout	Number of seconds after which the Data Integration Service attempts to reconnect to the Confluent Kafka broker to read or write data. If the source or target is not available for the time you specify, the mapping execution stops to avoid any data loss.
Kafka Broker Version	Version of the Confluent Kafka messaging broker.
Additional Connection Properties	Optional. Comma-separated list of connection properties to connect to the Kafka broker.
Schema Registry URL	Location and port of the schema registry provider on which to connect.

Additional Connection Properties

You can use the following syntax for specifying the additional connection properties:

```
request.timeout.ms=<value>,session.timeout.ms=<value>,
fetch.max.wait.ms=<value>,heartbeat.interval.ms=<value>,
security.protocol=SASL_PLAINTEXT,sasl.kerberos.
service.name=<kerberos name>,sasl.mechanism=GSSAPI,
sasl.jaas.config=com.sun.security.auth.module.
Krb5Login Modulerequired useKeyTab=true
doNotPrompt=true storeKey=true client=true
keyTab="<Keytab Location>" principal="<principal>";
```

SSL Properties

The following table describes the SSL properties for the Confluent Kafka connection:

Property	Description
SSL Mode	Optional. SSL mode indicates the encryption type to use for the connection. You can choose one of the following SSL modes: - Disabled - One way - Two way The default value is `Disabled`.
SSL TrustStore File Path	Required when `One way` SSL mode is selected. Absolute path and file name of the SSL truststore file that contains certificates of the trusted SSL server.
SSL TrustStore Password	Required when `One way` SSL mode is selected. Password for the SSL truststore.
SSL KeyStore File Path	Required when `Two way` SSL mode is selected. Absolute path and file name of the SSL keystore file that contains private keys and certificates for the SSL server.
SSL KeyStore Password	Required when `Two way` SSL mode is selected. Password for the SSL keystore.
Additional Security Properties	Optional. Comma-separated list of connection properties to connect to the Confluent Kafka broker in a secured way.

Creating a Confluent Kafka Connection Using infacmd

You can use the infacmd command line program to create a Confluent Kafka connection.

To create a Confluent Kafka connection on UNIX, run the following command:

```
sh infacmd.sh createConnection -dn <domain name> -un <domain user> -pd <domain password>
-cn <connection name> -cid <connection id> -ct ConfluentKafka -o
"kfkBrkList='<host1:port1>,<host2:port2>,<host3:port3>' kafkabrokerversion='<version>'
schemaregistryurl='<schema registry URL>'"
```

For more information about the CreateConnection command, see the *Informatica Command Reference*.

Databricks Connection Properties

Use the Databricks connection to run mappings on a Databricks cluster.

A Databricks connection is a cluster type connection. You can create and manage a Databricks connection in the Administrator tool or the Developer tool. You can use infacmd to create a Databricks connection. Configure properties in the Databricks connection to enable communication between the Data Integration Service and the Databricks cluster.

The following table describes the general connection properties for the Databricks connection:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters:~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	Optional. The description of the connection. The description cannot exceed 4,000 characters.
Connection Type	Choose Databricks.
Cluster Configuration	Name of the cluster configuration associated with the Databricks environment. Required if you do not configure the cloud provisioning configuration.
Cloud Provisioning Configuration	Name of the cloud provisioning configuration associated with a Databricks cloud platform. Required if you do not configure the cluster configuration.
Staging Directory	The directory where the Databricks Spark engine stages run-time files. If you specify a directory that does not exist, the Data Integration Service creates it at run time. If you do not provide a directory path, the run-time staging files are written to */<cluster staging directory>/DATABRICKS*.
Advanced Properties	List of advanced properties that are unique to the Databricks environment. You can configure run-time properties for the Databricks environment in the Data Integration Service and in the Databricks connection. You can override a property configured at a high level by setting the value at a lower level. For example, if you configure a property in the Data Integration Service custom properties, you can override it in the Databricks connection. The Data Integration Service processes property overrides based on the following priorities: 1. Databricks connection advanced properties 2. Data Integration Service custom properties **Note:** Informatica does not recommend changing these property values before you consult with third-party documentation, Informatica documentation, or Informatica Global Customer Support. If you change a value without knowledge of the property, you might experience performance degradation or other unexpected results.

Advanced Properties

Configure the following properties in the **Advanced Properties** of the Databricks configuration section:

infaspark.json.parser.mode

Specifies the parser how to handle corrupt JSON records. You can set the value to one of the following modes:

- DROPMALFORMED. The parser ignores all corrupted records. Default mode.
- PERMISSIVE. The parser accepts non-standard fields as nulls in corrupted records.
- FAILFAST. The parser generates an exception when it encounters a corrupted record and the Spark application goes down.

infaspark.json.parser.multiLine

Specifies whether the parser can read a multiline record in a JSON file. You can set the value to true or false. Default is false. Applies only to non-native distributions that use Spark version 2.2.x and above.

infaspark.flatfile.writer.nullValue

When the Databricks Spark engine writes to a target, it converts null values to empty strings (" "). For example, 12, AB,"",23p09udj.

The Databricks Spark engine can write the empty strings to string columns, but when it tries to write an empty string to a non-string column, the mapping fails with a type mismatch.

To allow the Databricks Spark engine to convert the empty strings back to null values and write to the target, configure the property in the Databricks Spark connection.

Set to: TRUE

infaspark.pythontx.exec

Required to run a Python transformation on the Databricks Spark engine. Set to the location of the Python executable binary on the worker nodes in the Databricks cluster.

When you provision the cluster at run time, set this property in the Databricks cloud provisioning configuration. Otherwise, set on the Databricks connection.

For example, set to:

```
infaspark.pythontx.exec=/databricks/python3/bin/python3
```

infaspark.pythontx.executorEnv.PYTHONHOME

Required to run a Python transformation on the Databricks Spark engine. Set to the location of the Python installation directory on the worker nodes in the Databricks cluster.

When you provision the cluster at run time, set this property in the Databricks cloud provisioning configuration. Otherwise, set on the Databricks connection.

For example, set to:

```
infaspark.pythontx.executorEnv.PYTHONHOME=/databricks/python3
```

Google Analytics Connection Properties

When you set up a Google Analytics connection, you must configure the connection properties.

Note: The order of the connection properties might vary depending on the tool where you view them.

The following table describes the Google Analytics connection properties:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters:~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. The ID must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	Optional. The description of the connection. The description cannot exceed 4,000 characters.
Location	The domain where you want to create the connection.
Type	The connection type. Select **Google Analytics**.
Service Account ID	Specifies the client_email value present in the JSON file that you download after you create a service account.
Service Account Key	Specifies the private_key value present in the JSON file that you download after you create a service account.
APIVersion	API that PowerExchange for Google Analytics uses to read from Google Analytics reports. Select **Core Reporting API v3**. **Note:** PowerExchange for Google Analytics does not support Analytics Reporting API v4.

Google BigQuery Connection Properties

When you set up a Google BigQuery connection, you must configure the connection properties.

Note: The order of the connection properties might vary depending on the tool where you view them.

The following table describes the Google BigQuery connection properties:

Property	Description
Service Account ID	Specifies the client_email value present in the JSON file that you download after you create a service account in Google BigQuery.
Service Account Key	Specifies the private_key value present in the JSON file that you download after you create a service account in Google BigQuery.

Property	Description
Connection mode	The mode that you want to use to read data from or write data to Google BigQuery. Select one of the following connection modes: - Simple. Flattens each field within the Record data type field as a separate field in the mapping. - Hybrid. Displays all the top-level fields in the Google BigQuery table including Record data type fields. PowerExchange for Google BigQuery displays the top-level Record data type field as a single field of the String data type in the mapping. - Complex. Displays all the columns in the Google BigQuery table as a single field of the String data type in the mapping. Default is Simple.
Schema Definition File Path	Specifies a directory on the client machine where the Data Integration Service must create a JSON file with the sample schema of the Google BigQuery table. The JSON file name is the same as the Google BigQuery table name. Alternatively, you can specify a storage path in Google Cloud Storage where the Data Integration Service must create a JSON file with the sample schema of the Google BigQuery table. You can download the JSON file from the specified storage path in Google Cloud Storage to a local machine.
Project ID	Specifies the project_id value present in the JSON file that you download after you create a service account in Google BigQuery. If you have created multiple projects with the same service account, enter the ID of the project that contains the dataset that you want to connect to.
Storage Path	This property applies when you read or write large volumes of data. Path in Google Cloud Storage where the Data Integration Service creates a local stage file to store the data temporarily. You can either enter the bucket name or the bucket name and folder name. For example, enter `gs://<bucket_name>` or `gs://<bucket_name>/<folder_name>`
Dataset ID	Not applicable for PowerExchange for Google BigQuery.
Use Legacy SQL For Custom Query	Not applicable for PowerExchange for Google BigQuery.
Dataset Name for Custom Query	Not applicable for PowerExchange for Google BigQuery.
Region ID	The region name where the Google BigQuery dataset resides. For example, if you want to connect to a Google BigQuery dataset that resides in Las Vegas region, specify **us-west4** as the **Region ID**. **Note:** In the **Storage Path** connection property, ensure that you specify a bucket name or the bucket name and folder name that resides in the same region as the dataset in Google BigQuery. For more information about the regions supported by Google BigQuery, see the following Google BigQuery documentation:https://cloud.google.com/bigquery/docs/locations

Property	Description
Optional Properties	Specifies whether you can configure certain source and target functionalities through custom properties. You can select one of the following options: - None. Select if you do not want to configure any custom properties. - Required. If you want to specify custom properties to configure the source and target functionalities. Default is None.
Provide Optional Properties	Comma-separated key-value pairs of custom properties to enable certain source and target functionalities. Appears only when you select **Required** in the Optional Properties. For more information about the list of custom properties that you can specify, see the Informatica Knowledge Base article: https://kb.informatica.com/faq/7/Pages/26/632722.aspx

Google Cloud Spanner Connection Properties

When you set up a Google Cloud Spanner connection, you must configure the connection properties.

Note: The order of the connection properties might vary depending on the tool where you view them.

The following table describes the Google Cloud Spanner connection properties:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters:~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. The ID must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	Optional. The description of the connection. The description cannot exceed 4,000 characters.
Location	The domain where you want to create the connection.
Type	The connection type. Select Google Cloud Spanner.
Project ID	Specifies the project_id value present in the JSON file that you download after you create a service account. If you have created multiple projects with the same service account, enter the ID of the project that contains the bucket that you want to connect to.
Service Account ID	Specifies the client_email value present in the JSON file that you download after you create a service account.

Property	Description
Service Account Key	Specifies the private_key value present in the JSON file that you download after you create a service account.
Instance ID	Name of the instance that you created in Google Cloud Spanner.

Google Cloud Storage Connection Properties

When you set up a Google Cloud Storage connection, you must configure the connection properties.

Note: The order of the connection properties might vary depending on the tool where you view them.

The following table describes the Google Cloud Storage connection properties:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters:~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. The ID must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	Optional. The description of the connection. The description cannot exceed 4,000 characters.
Location	The domain where you want to create the connection.
Type	The connection type. Select **Google Cloud Storage**.
Project ID	Specifies the project_id value present in the JSON file that you download after you create a service account. If you have created multiple projects with the same service account, enter the ID of the project that contains the bucket that you want to connect to.
Service Account ID	Specifies the client_email value present in the JSON file that you download after you create a service account.
Service Account Key	Specifies the private_key value present in the JSON file that you download after you create a service account.

Google PubSub Connection Properties

When you create a Google PubSub connection, you must configure the connection properties.

The following table describes the Google PubSub connection properties:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: ~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	The string that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	The description of the connection. Enter a string that you can use to identify the connection. The description cannot exceed 4,000 characters.
Connection Type	The connection type. Select **Pub Sub** connection type.
Client Email	The `client_email` value available in the JSON file that you download after you create a service account.
Client Id	The `client_id` value available in the JSON file that you download after you create a service account.
Private Key Id	The `private_key_id` value available in the JSON file that you download after you create a service account.
Private Key	The `private_key` value available in the JSON file that you download after you create a service account.
Project Id	The `project_id` value available in the JSON file that you download after you create a service account.

Hadoop Connection Properties

Use the Hadoop connection to configure mappings to run on a Hadoop cluster. A Hadoop connection is a cluster type connection. You can create and manage a Hadoop connection in the Administrator tool or the Developer tool. You can use infacmd to create a Hadoop connection. Hadoop connection properties are case sensitive unless otherwise noted.

Hadoop Cluster Properties

Configure properties in the Hadoop connection to enable communication between the Data Integration Service and the Hadoop cluster.

The following table describes the general connection properties for the Hadoop connection:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: `~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /`
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	The description of the connection. Enter a string that you can use to identify the connection. The description cannot exceed 4,000 characters.
Cluster Configuration	The name of the cluster configuration associated with the Hadoop environment. Required if you do not configure the Cloud Provisioning Configuration.
Cloud Provisioning Configuration	Name of the cloud provisioning configuration associated with a cloud platform such as Amazon AWS or Microsoft Azure. Required if you do not configure the Cluster Configuration.
Cluster Environment Variables*	Environment variables that the Hadoop cluster uses. If you use a Cloudera CDH 6.x cluster or a Cloudera CDP cluster, configure the locale setting as cluster environment variables. In Cloudera Manager, you must also add the environment variables to the following YARN property: `yarn.nodemanager.env-whitelist` For example, the variable ORACLE_HOME represents the directory where the Oracle database client software is installed. You can configure run-time properties for the Hadoop environment in the Data Integration Service, the Hadoop connection, and in the mapping. You can override a property configured at a high level by setting the value at a lower level. For example, if you configure a property in the Data Integration Service custom properties, you can override it in the Hadoop connection or in the mapping. The Data Integration Service processes property overrides based on the following priorities: 1. Mapping custom properties set using infacmd ms runMapping with the `-cp` option 2. Mapping run-time properties for the Hadoop environment 3. Hadoop connection advanced properties for run-time engines 4. Hadoop connection advanced general properties, environment variables, and classpaths 5. Data Integration Service custom properties
Cluster Library Path*	The path for shared libraries on the cluster. The $DEFAULT_CLUSTER_LIBRARY_PATH variable contains a list of default directories.

Property	Description
Cluster Classpath*	The classpath to access the Hadoop jar files and the required libraries. The $DEFAULT_CLUSTER_CLASSPATH variable contains a list of paths to the default jar files and libraries. You can configure run-time properties for the Hadoop environment in the Data Integration Service, the Hadoop connection, and in the mapping. You can override a property configured at a high level by setting the value at a lower level. For example, if you configure a property in the Data Integration Service custom properties, you can override it in the Hadoop connection or in the mapping. The Data Integration Service processes property overrides based on the following priorities: 1. Mapping custom properties set using infacmd ms runMapping with the `-cp` option 2. Mapping run-time properties for the Hadoop environment 3. Hadoop connection advanced properties for run-time engines 4. Hadoop connection advanced general properties, environment variables, and classpaths 5. Data Integration Service custom properties
Cluster Executable Path*	The path for executable files on the cluster. The $DEFAULT_CLUSTER_EXEC_PATH variable contains a list of paths to the default executable files.
* Informatica does not recommend changing these property values before you consult with third-party documentation, Informatica documentation, or Informatica Global Customer Support. If you change a value without knowledge of the property, you might experience performance degradation or other unexpected results.	

Common Properties

The following table describes the common connection properties that you configure for the Hadoop connection:

Property	Description
Impersonation User Name	Required if the Hadoop cluster uses Kerberos authentication. Hadoop impersonation user. The user name that the Data Integration Service impersonates to run mappings in the Hadoop environment. Data Engineering Integration supports operating system profiles on all Hadoop distributions. In the Hadoop run-time environment, the Data Integration Service pushes the processing to the Hadoop cluster and the run-time engines run mappings with the operating system profile specified Hadoop impersonation properties.
Temporary Table Compression Codec	Hadoop compression library for a compression codec class name. **Note:** The Spark engine does not support compression settings for temporary tables. When you run mappings on the Spark engine, the Spark engine stores temporary tables in an uncompressed file format.
Codec Class Name	Codec class name that enables data compression and improves performance on temporary staging tables.
Hive Staging Database Name	Namespace for Hive staging tables. Use the name `default` for tables that do not have a specified database name. If you do not configure a namespace, the Data Integration Service uses the Hive database name in the Hive target connection to create staging tables. When you run a mapping in the native environment to write data to Hive, you must configure the Hive staging database name in the Hive connection. The Data Integration Service ignores the value you configure in the Hadoop connection.

Property	Description
Environment SQL	SQL commands to set the Hadoop environment. The Data Integration Service executes the environment SQL at the beginning of each Hive script generated by a HiveServer2 job. The following rules and guidelines apply to the usage of environment SQL: - You can use environment SQL to define Hadoop or Hive parameters that you want to use in the PreSQL commands or in custom queries. - If you use multiple values for the Environment SQL property, ensure that there is no space between the values.
Engine Type	The Data Integration Service uses HiveServer2 to process portions of some jobs by running HiveServer2 tasks on the Spark engine. When you import the cluster configuration through the Administrator tool, you can choose to create connections. The engine type property is populated by default based on the distribution. When you manually create a connection, you must configure the engine type. You can specify the engine type based on the following Hadoop distributions: - Amazon EMR. Tez - Azure HDI. Tez - Cloudera CDH. MRv2 - Cloudera CDP. Tez - Dataproc. MRv2 - Hortonworks HDP. Tez - MapR. MRv2
Advanced Properties	List of advanced properties that are unique to the Hadoop environment. The properties are common to the Blaze and Spark engines. The advanced properties include a list of default properties. You can configure run-time properties for the Hadoop environment in the Data Integration Service, the Hadoop connection, and in the mapping. You can override a property configured at a high level by setting the value at a lower level. For example, if you configure a property in the Data Integration Service custom properties, you can override it in the Hadoop connection or in the mapping. The Data Integration Service processes property overrides based on the following priorities: 1. Mapping custom properties set using infacmd ms runMapping with the -cp option 2. Mapping run-time properties for the Hadoop environment 3. Hadoop connection advanced properties for run-time engines 4. Hadoop connection advanced general properties, environment variables, and classpaths 5. Data Integration Service custom properties **Note:** Informatica does not recommend changing these property values before you consult with third-party documentation, Informatica documentation, or Informatica Global Customer Support. If you change a value without knowledge of the property, you might experience performance degradation or other unexpected results.

Reject Directory Properties

The following table describes the connection properties that you configure to the Hadoop Reject Directory.

Property	Description
Write Reject Files to Hadoop	If you use the Blaze engine to run mappings, select the check box to specify a location to move reject files. If checked, the Data Integration Service moves the reject files to the HDFS location listed in the property, Reject File Directory. By default, the Data Integration Service stores the reject files based on the RejectDir system parameter.
Reject File Directory	The directory for Hadoop mapping files on HDFS when you run mappings.

Blaze Configuration

The following table describes the connection properties that you configure for the Blaze engine:

Property	Description
Blaze Staging Directory	The HDFS file path of the directory that the Blaze engine uses to store temporary files. Verify that the directory exists. The YARN user, Blaze engine user, and mapping impersonation user must have write permission on this directory. Default is `/blaze/workdir`. If you clear this property, the staging files are written to the Hadoop staging directory `/tmp/blaze_<user name>`.
Blaze User Name	The owner of the Blaze service and Blaze service logs. When the Hadoop cluster uses Kerberos authentication, the default user is the Data Integration Service SPN user. When the Hadoop cluster does not use Kerberos authentication and the Blaze user is not configured, the default user is the Data Integration Service user.
Minimum Port	The minimum value for the port number range for the Blaze engine. Default is 12300.
Maximum Port	The maximum value for the port number range for the Blaze engine. Default is 12600.
YARN Queue Name	The YARN scheduler queue name used by the Blaze engine that specifies available resources on a cluster. **Note:** If YARN preemption is enabled on the cluster, verify with the Hadoop administrator that preemption is disabled on the queue associated with the Blaze engine.
Blaze Job Monitor Address	The host name and port number for the Blaze Job Monitor. Use the following format: `<hostname>:<port>` Where - <hostname> is the host name or IP address of the Blaze Job Monitor server. - <port> is the port on which the Blaze Job Monitor listens for remote procedure calls (RPC). For example, enter: `myhostname:9080`

Property	Description
Blaze YARN Node Label	Node label that determines the node on the Hadoop cluster where the Blaze engine runs. If you do not specify a node label, the Blaze engine runs on the nodes in the default partition. If the Hadoop cluster supports logical operators for node labels, you can specify a list of node labels. To list the node labels, use the operators `&&` (AND), `||` (OR), and `!` (NOT). **Note:** You cannot use node labels on a Cloudera CDH cluster.
Advanced Properties	List of advanced properties that are unique to the Blaze engine. The advanced properties include a list of default properties. You can configure run-time properties for the Hadoop environment in the Data Integration Service, the Hadoop connection, and in the mapping. You can override a property configured at a high level by setting the value at a lower level. For example, if you configure a property in the Data Integration Service custom properties, you can override it in the Hadoop connection or in the mapping. The Data Integration Service processes property overrides based on the following priorities: 1. Mapping custom properties set using infacmd ms runMapping with the `-cp` option 2. Mapping run-time properties for the Hadoop environment 3. Hadoop connection advanced properties for run-time engines 4. Hadoop connection advanced general properties, environment variables, and classpaths 5. Data Integration Service custom properties **Note:** Informatica does not recommend changing these property values before you consult with third-party documentation, Informatica documentation, or Informatica Global Customer Support. If you change a value without knowledge of the property, you might experience performance degradation or other unexpected results.

Spark Configuration

The following table describes the connection properties that you configure for the Spark engine:

Property	Description
Spark Staging Directory	The HDFS file path of the directory that the Spark engine uses to store temporary files for running jobs. The YARN user, Data Integration Service user, and mapping impersonation user must have write permission on this directory. If you do not specify a file path, by default, the temporary files are written to the Hadoop staging directory `/tmp/SPARK_<user name>`. When you run Sqoop jobs on the Spark engine, the Data Integration Service creates a Sqoop staging directory within the Spark staging directory to store temporary files: `<Spark staging directory>/sqoop_staging`
Spark Event Log Directory	Optional. The HDFS file path of the directory that the Spark engine uses to log events.

Property	Description
YARN Queue Name	The YARN scheduler queue name used by the Spark engine that specifies available resources on a cluster. The name is case sensitive.
Advanced Properties	List of advanced properties that are unique to the Spark engine. The advanced properties include a list of default properties. You can configure run-time properties for the Hadoop environment in the Data Integration Service, the Hadoop connection, and in the mapping. You can override a property configured at a high level by setting the value at a lower level. For example, if you configure a property in the Data Integration Service custom properties, you can override it in the Hadoop connection or in the mapping. The Data Integration Service processes property overrides based on the following priorities: 1. Mapping custom properties set using infacmd ms runMapping with the `-cp` option 2. Mapping run-time properties for the Hadoop environment 3. Hadoop connection advanced properties for run-time engines 4. Hadoop connection advanced general properties, environment variables, and classpaths 5. Data Integration Service custom properties **Note:** Informatica does not recommend changing these property values before you consult with third-party documentation, Informatica documentation, or Informatica Global Customer Support. If you change a value without knowledge of the property, you might experience performance degradation or other unexpected results.

HDFS Connection Properties

Use a Hadoop File System (HDFS) connection to access data in the Hadoop cluster. The HDFS connection is a file system type connection. You can create and manage an HDFS connection in the Administrator tool, Analyst tool, or the Developer tool. HDFS connection properties are case sensitive unless otherwise noted.

Note: The order of the connection properties might vary depending on the tool where you view them.

The following table describes HDFS connection properties:

Property	Description
Name	Name of the connection. The name is not case sensitive and must be unique within the domain. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: `~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /`
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	The description of the connection. The description cannot exceed 765 characters.
Location	The domain where you want to create the connection. Not valid for the Analyst tool.
Type	The connection type. Default is Hadoop File System.

Property	Description
User Name	User name to access HDFS.
NameNode URI	The URI to access the storage system. You can find the value for `fs.defaultFS` in the `core-site.xml` configuration set of the cluster configuration. If you create connections when you import the cluster configuration, the NameNode URI property is populated by default, and it is updated each time you refresh the cluster configuration. If you use a Cloudera CDP Public Cloud compute cluster and the HDFS is on a Cloudera Data Lake cluster, set the property *spark.yarn.access.hadoopFileSystems* in the Spark properties of the Hadoop Connection to the same value as set here.

Accessing Multiple Storage Types

Use the NameNode URI property in the connection parameters to connect to various storage types. The following table lists the storage type and the NameNode URI format for the storage type:

Storage	NameNode URI Format
HDFS	`hdfs://<namenode>:<port>` where: - `<namenode>` is the host name or IP address of the NameNode. - `<port>` is the port that the NameNode listens for remote procedure calls (RPC). `hdfs://<nameservice>` in case of NameNode high availability.
MapR-FS	`maprfs:///`

WASB in HDInsight	`wasb://<container_name>@<account_name>.blob.core.windows.net/<path>`
	where:
	- `<container_name>` identifies a specific Azure Storage Blob container.
	Note: `<container_name>` is optional.
	- `<account_name>` identifies the Azure Storage Blob object.
	Example:
	`wasb://infabdmoffering1storage.blob.core.windows.net/infabdmoffering1cluster/mr-history`

ADLS in HDInsight	`adl://home`

When you create a cluster configuration from an Azure HDInsight cluster, the cluster configuration uses either ADLS or WASB as the primary storage. You cannot create a cluster configuration with ADLS or WASB as the secondary storage. You can edit the NameNode URI property in the HDFS connection to connect to a local HDFS location.

HBase Connection Properties

Use an HBase connection to access HBase. The HBase connection is a NoSQL connection. You can create and manage an HBase connection in the Administrator tool or the Developer tool. HBase connection properties are case sensitive unless otherwise noted.

The following table describes HBase connection properties:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: ~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	The description of the connection. The description cannot exceed 4,000 characters.
Location	The domain where you want to create the connection.
Type	The connection type. Select HBase.
Database Type	Type of database that you want to connect to. Select **HBase** to create a connection for an HBase table.

HBase Connection Properties for MapR-DB

Use an HBase connection to connect to a MapR-DB table. The HBase connection is a NoSQL connection. You can create and manage an HBase connection in the Administrator tool or the Developer tool. HBase connection properties are case sensitive unless otherwise noted.

The following table describes the HBase connection properties for MapR-DB:

Property	Description
Name	Name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: ~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.

Property	Description
Description	Description of the connection. The description cannot exceed 4,000 characters.
Location	Domain where you want to create the connection.
Type	Connection type. Select **HBase**.
Database Type	Type of database that you want to connect to. Select **MapR-DB** to create a connection for a MapR-DB table.
Cluster Configuration	The name of the cluster configuration associated with the Hadoop environment.
MapR-DB Database Path	Database path that contains the MapR-DB table that you want to connect to. Enter a valid MapR cluster path. When you create an HBase data object for MapR-DB, you can browse only tables that exist in the MapR-DB path that you specify in the **Database Path** field. You cannot access tables that are available in sub-directories in the specified path. For example, if you specify the path as `/user/customers/`, you can access the tables in the `customers` directory. However, if the `customers` directory contains a sub-directory named `regions`, you cannot access the tables in the following directory: `/user/customers/regions`

Hive Connection Properties

Use the Hive connection to access Hive data. A Hive connection is a database type connection. You can create and manage a Hive connection in the Administrator tool, Analyst tool, or the Developer tool. Hive connection properties are case sensitive unless otherwise noted.

Note: The order of the connection properties might vary depending on the tool where you view them.

The following table describes Hive connection properties:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: `~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /`
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	The description of the connection. The description cannot exceed 4000 characters.

Property	Description
Location	The domain where you want to create the connection. Not valid for the Analyst tool.
Type	The connection type. Select Hive.
LDAP username	LDAP user name of the user that the Data Integration Service impersonates to run mappings on a Hadoop cluster. The user name depends on the JDBC connection string that you specify in the Metadata Connection String or Data Access Connection String for the native environment. If the Hadoop cluster uses Kerberos authentication, the principal name for the JDBC connection string and the user name must be the same. Otherwise, the user name depends on the behavior of the JDBC driver. With Hive JDBC driver, you can specify a user name in many ways and the user name can become a part of the JDBC URL. If the Hadoop cluster does not use Kerberos authentication, the user name depends on the behavior of the JDBC driver. If you do not specify a user name, the Hadoop cluster authenticates jobs based on the following criteria: - The Hadoop cluster does not use Kerberos authentication. It authenticates jobs based on the operating system profile user name of the machine that runs the Data Integration Service. - The Hadoop cluster uses Kerberos authentication. It authenticates jobs based on the SPN of the Data Integration Service. LDAP username will be ignored.
Password	Password for the LDAP username.
Environment SQL	SQL commands to set the Hadoop environment. In native environment type, the Data Integration Service executes the environment SQL each time it creates a connection to a Hive metastore. If you use the Hive connection to run profiles on a Hadoop cluster, the Data Integration Service executes the environment SQL at the beginning of each Hive session. The following rules and guidelines apply to the usage of environment SQL in both connection modes: - Use the environment SQL to specify Hive queries. - Use the environment SQL to set the classpath for Hive user-defined functions and then use environment SQL or PreSQL to specify the Hive user-defined functions. You cannot use PreSQL in the data object properties to specify the classpath. If you use Hive user-defined functions, you must copy the .jar files to the following directory: `<Informatica installation directory>/services/shared/hadoop/<Hadoop distribution name>/extras/hive-auxjars` - You can use environment SQL to define Hadoop or Hive parameters that you want to use in the PreSQL commands or in custom queries. - If you use multiple values for the Environment SQL property, ensure that there is no space between the values.
SQL Identifier Character	The type of character used to identify special characters and reserved SQL keywords, such as WHERE. The Data Integration Service places the selected character around special characters and reserved SQL keywords. The Data Integration Service also uses this character for the **Support mixed-case identifiers** property.

Properties to Access Hive as Source or Target

The following table describes the connection properties that you configure to access Hive as a source or target:

Property	Description
JDBC Driver Class Name	Name of the Hive JDBC driver class. If you leave this option blank, the Developer tool uses the default Apache Hive JDBC driver shipped with the distribution. If the default Apache Hive JDBC driver does not fit your requirements, you can override the Apache Hive JDBC driver with a third-party Hive JDBC driver by specifying the driver class name.
Metadata Connection String	The JDBC connection URI used to access the metadata from the Hadoop server. You can use PowerExchange for Hive to communicate with a HiveServer service or HiveServer2 service. To connect to HiveServer, specify the connection string in the following format: `jdbc:hive2://<hostname>:<port>/<db>` Where - <hostname> is name or IP address of the machine on which HiveServer2 runs. - <port> is the port number on which HiveServer2 listens. - <db> is the database name to which you want to connect. If you do not provide the database name, the Data Integration Service uses the default database details. To connect to HiveServer2, use the connection string format that Apache Hive implements for that specific Hadoop Distribution. For more information about Apache Hive connection string formats, see the Apache Hive documentation. For user impersonation, you must add `hive.server2.proxy.user=<xyz>` to the JDBC connection URI. If you do not configure user impersonation, the current user's credentials are used connect to the HiveServer2. If the Hadoop cluster uses SSL or TLS authentication, you must add `ssl=true` to the JDBC connection URI. For example: `jdbc:hive2://<hostname>:<port>/<db>;ssl=true` If you use self-signed certificate for SSL or TLS authentication, ensure that the certificate file is available on the client machine and the Data Integration Service machine. For more information, see the*Data Engineering Integration Guide*.
Bypass Hive JDBC Server	JDBC driver mode. Select the check box to use the embedded JDBC driver mode. To use the JDBC embedded mode, perform the following tasks: - Verify that Hive client and Informatica services are installed on the same machine. - Configure the Hive connection properties to run mappings on a Hadoop cluster. If you choose the non-embedded mode, you must configure the Data Access Connection String. Informatica recommends that you use the JDBC embedded mode.
Fine Grained Authorization	When you select the option to observe fine grained authorization in a Hive source, the mapping observes the following: - Row and column level restrictions. Applies to Hadoop clusters where Sentry or Ranger security modes are enabled. - Data masking rules. Applies to masking rules set on columns containing sensitive data by Dynamic Data Masking. If you do not select the option, the Blaze and Spark engines ignore the restrictions and masking rules, and results include restricted or sensitive data.

Property	Description
Data Access Connection String	The connection string to access data from the Hadoop data store. To connect to HiveServer, specify the non-embedded JDBC mode connection string in the following format: `jdbc:hive2://<hostname>:<port>/<db>` Where - <hostname> is name or IP address of the machine on which HiveServer2 runs. - <port> is the port number on which HiveServer2 listens. - <db> is the database to which you want to connect. If you do not provide the database name, the Data Integration Service uses the default database details. To connect to HiveServer2, use the connection string format that Apache Hive implements for the specific Hadoop Distribution. For more information about Apache Hive connection string formats, see the Apache Hive documentation. For user impersonation, you must add `hive.server2.proxy.user=<xyz>` to the JDBC connection URI. If you do not configure user impersonation, the current user's credentials are used connect to the HiveServer2. If the Hadoop cluster uses SSL or TLS authentication, you must add `ssl=true` to the JDBC connection URI. For example: `jdbc:hive2://<hostname>:<port>/<db>;ssl=true` If you use self-signed certificate for SSL or TLS authentication, ensure that the certificate file is available on the client machine and the Data Integration Service machine. For more information, see the *Data Engineering Integration Guide*.
Hive Staging Directory on HDFS	HDFS directory for Hive staging tables. You must grant execute permission to the Hadoop impersonation user and the mapping impersonation users. This option is applicable and required when you write data to a Hive target in the native environment.
Hive Staging Database Name	Namespace for Hive staging tables. The Hive Staging Database Name is automatically updated from the Data Access Connection String. If you want to override the default name, you need to configure the Hive Staging Database Name in the Hive connection. This option is applicable when you run a mapping in the native environment to write data to a Hive target. If you run the mapping on the Blaze or Spark engine, you do not need to configure the Hive staging database name in the Hive connection. The Data Integration Service uses the value that you configure in the Hadoop connection.

JDBC Connection Properties

You can use a JDBC connection to access tables in a database. You can create and manage a JDBC connection in the Administrator tool, the Developer tool, or the Analyst tool.

Note: The order of the connection properties might vary depending on the tool where you view them.

The following table describes JDBC connection properties:

Property	Description
Database Type	The database type.
Name	Name of the connection. The name is not case sensitive and must be unique within the domain. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: `~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /`
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	The description of the connection. The description cannot exceed 765 characters.
User Name	The database user name.
Password	The password for the database user name.

JDBC Driver Class Name	Name of the JDBC driver class.
	The following list provides the driver class name that you can enter for the applicable database type:
	- DataDirect JDBC driver class name for Oracle:
	`com.informatica.jdbc.oracle.OracleDriver`
	- DataDirect JDBC driver class name for IBM DB2:
	`com.informatica.jdbc.db2.DB2Driver`
	- DataDirect JDBC driver class name for Microsoft SQL Server:
	`com.informatica.jdbc.sqlserver.SQLServerDriver`
	- DataDirect JDBC driver class name for Sybase ASE:
	`com.informatica.jdbc.sybase.SybaseDriver`
	- DataDirect JDBC driver class name for Informix:
	`com.informatica.jdbc.informix.InformixDriver`
	- DataDirect JDBC driver class name for MySQL:
	`com.informatica.jdbc.mysql.MySQLDriver`
	- JDBC driver for Databricks Delta Lake: the name of the driver that you downloaded from Databricks. For information about the driver, see the topic on configuring storage access in the "Before You Begin Databricks Integration" chapter of the *Data Engineering Integration Guide*.
	For more information about which driver class to use with specific databases, see the vendor documentation.

Connection String	Connection string to connect to the database. Use the following connection string: `jdbc:<subprotocol>:<subname>` For more information about the connection string to use with specific drivers, see the vendor documentation.
Environment SQL	Optional. Enter SQL commands to set the database environment when you connect to the database. The Data Integration Service executes the connection environment SQL each time it connects to the database. **Note:** If you enable Sqoop, Sqoop ignores this property.
Transaction SQL	Optional. Enter SQL commands to set the database environment when you connect to the database. The Data Integration Service executes the transaction environment SQL at the beginning of each transaction. **Note:** If you enable Sqoop, Sqoop ignores this property.

Property	Description
SQL Identifier Character	Type of character that the database uses to enclose delimited identifiers in SQL queries. The available characters depend on the database type. Select (None) if the database uses regular identifiers. When the Data Integration Service generates SQL queries, the service does not place delimited characters around any identifiers. Select a character if the database uses delimited identifiers. When the Data Integration Service generates SQL queries, the service encloses delimited identifiers within this character. **Note:** If you enable Sqoop, Sqoop ignores this property.
Support Mixed-case Identifiers	Enable if the database uses case-sensitive identifiers. When enabled, the Data Integration Service encloses all identifiers within the character selected for the **SQL Identifier Character** property. When the **SQL Identifier Character** property is set to none, the **Support Mixed-case Identifiers** property is disabled. **Note:** If you enable Sqoop, Sqoop honors this property when you generate and execute a DDL script to create or replace a target at run time. In all other scenarios, Sqoop ignores this property.
Use Sqoop Connector	Enables Sqoop connectivity for the data object that uses the JDBC connection. The Data Integration Service runs the mapping in the Hadoop run-time environment through Sqoop. You can configure Sqoop connectivity for relational data objects, customized data objects, and logical data objects that are based on a JDBC-compliant database. Select **Sqoop v1.x** to enable Sqoop connectivity. Default is **None**.

Sqoop Arguments	Enter the arguments that Sqoop must use to connect to the database. Separate multiple arguments with a space.
	To run the mapping on the Blaze engine with the Teradata Connector for Hadoop (TDCH) specialized connectors for Sqoop, you must define the TDCH connection factory class in the Sqoop arguments. The connection factory class varies based on the TDCH Sqoop Connector that you want to use.
	- To use Cloudera Connector Powered by Teradata, configure the following Sqoop argument:
	`-Dsqoop.connection.factories=com.cloudera.connector.teradata.TeradataManagerFactory`
	- To use Hortonworks Connector for Teradata (powered by the Teradata Connector for Hadoop), configure the following Sqoop argument:
	`-Dsqoop.connection.factories=org.apache.sqoop.teradata.TeradataManagerFactory`
	To run the mapping on the Spark engine, you do not need to define the TDCH connection factory class in the Sqoop arguments. The Data Integration Service invokes the Cloudera Connector Powered by Teradata and Hortonworks Connector for Teradata (powered by the Teradata Connector for Hadoop) by default.
	Note: To run the mapping with a generic JDBC connector instead of the specialized Cloudera or Hortonworks connector, you must define the --driver and --connection-manager Sqoop arguments in the JDBC connection. If you define the --driver and --connection-manager arguments in the Read or Write transformation of the mapping, Sqoop ignores the arguments.
	If you do not enter Sqoop arguments, the Data Integration Service constructs the Sqoop command based on the JDBC connection properties.

JDBC Connection String

Specify the connection string in the JDBC connection to connect to the database.

Specify the connection string in the following format:

```
jdbc:<subprotocol>:<subname>
```

Use the following sample connection strings that you can enter for the applicable database type:

- DataDirect Oracle JDBC driver:

```
jdbc:informatica:oracle://<host>:<port>;SID=<value>
```

- Oracle JDBC driver:

```
jdbc:oracle:thin:@//<host>:<port>:<SID>
```

- DataDirect IBM DB2 JDBC driver:

```
jdbc:informatica:db2://<host>:<port>;DatabaseName=<value>
```

- IBM DB2 JDBC driver:

```
jdbc:db2://<host>:<port>/<database_name>
```

- DataDirect Microsoft SQL Server JDBC driver:

```
jdbc:informatica:sqlserver://<host>;DatabaseName=<value>
```

- Microsoft SQL Server JDBC driver:

```
jdbc:sqlserver://<host>;DatabaseName=<value>
```

- Netezza JDBC driver:

```
jdbc:netezza://<host>:<port>/<database_name>
```

- Pivotal Greenplum driver:

```
jdbc:pivotal:greenplum://<host>:<port>;/database_name=<value>
```

- Postgres Greenplum driver:

```
jdbc:postgressql://<host>:<port>/<database_name>
```

- Teradata JDBC driver:

```
jdbc:teradata://<host>/database_name=<value>,tmode=<value>,charset=<value>
```

- JDBC driver for Delta Lake:

```
jdbc:spark://<host name>:443/default;transportMode=http;ssl=1;httpPath=sql/
protocolv1/o/<cluster ID>;AuthMech=3;UID=token;PWD=<token string>
```

Use the following sample connection strings that you can enter for an SSL-enabled applicable database type:

- DataDirect Oracle JDBC driver:

```
jdbc:informatica:oracle://
<host_name>:<port>;CatalogOptions=6;ServiceName=<service_name>;trustStorePassword=<tru
ststore_password>;
keyStorePassword=<keystore_password>;CryptoProtocolVersion=TLSv1.2;keyStore=<keystore_
location_of_ewallet.p12_file>;
trustStore=<truststore_location_of_truststore.p12_file>;HostNameInCertificate=<databas
e_host_name>;encryptionMethod=SSL;
ValidateServerCertificate=True;
```

- Oracle JDBC driver:

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCPS)(HOST=<host>)
(PORT=<port_number>))
(CONNECT_DATA=(SERVICE_NAME=<service_name>)))"
```

- DataDirect Microsoft SQL Server JDBC driver:

```
jdbc:informatica:sqlserver://
<host_name>:<port>;databaseName=<database_name>;EncryptionMethod=SSL;
CryptoProtocolVersion=<TLSv1.2_or_TLSv1.1_or_TLSv1>;ValidateServerCertificate=false;Tr
ustStore=<truststore_location>;
TrustStorePassword=<truststore_password>
```

- Microsoft SQL Server JDBC driver:

```
jdbc:sqlserver://
<host_name>:<port>;databaseName=<database_name>;integratedSecurity=false;encrypt=true;
trustServerCertificate=true;
TrustStore=/
<truststore_location>;TrustStorePassword=<truststore_password>;user=<user_name>;passwo
rd=<password>
```

For more information about the connection string to use with specific drivers, see the vendor documentation.

Sqoop Connection-Level Arguments

In the JDBC connection, you can define the arguments that Sqoop must use to connect to the database. The Data Integration Service merges the arguments that you specify with the default command that it constructs based on the JDBC connection properties. The arguments that you specify take precedence over the JDBC connection properties.

If you want to use the same driver to import metadata and run the mapping, and do not want to specify any additional Sqoop arguments, select **Sqoop v1.x** from the **Use Sqoop Version** list and leave the **Sqoop Arguments** field empty in the JDBC connection. The Data Integration Service constructs the Sqoop command based on the JDBC connection properties that you specify.

However, if you want to use a different driver for run-time tasks or specify additional run-time Sqoop arguments, select **Sqoop v1.x** from the **Use Sqoop Version** list and specify the arguments in the **Sqoop Arguments** field.

A mapping that contains an Update Strategy transformation cannot use a Sqoop-enabled JDBC connection to write to a target. To run the mapping, disable the Sqoop connector in the Write transformation.

You can configure the following Sqoop arguments in the JDBC connection:

driver

Defines the JDBC driver class that Sqoop must use to connect to the database.

Use the following syntax:

```
--driver <JDBC driver class>
```

For example, use the following syntax depending on the database type that you want to connect to:

- Aurora: `--driver com.mysql.jdbc.Driver`
- Greenplum: `--driver org.postgresql.Driver`
- IBM DB2: `--driver com.ibm.db2.jcc.DB2Driver`
- IBM DB2 z/OS: `--driver com.ibm.db2.jcc.DB2Driver`
- Microsoft SQL Server: `--driver com.microsoft.sqlserver.jdbc.SQLServerDriver`
- Netezza: `--driver org.netezza.Driver`
- Oracle: `--driver oracle.jdbc.driver.OracleDriver`
- Teradata: `--driver com.teradata.jdbc.TeraDriver`

connect

Defines the JDBC connection string that Sqoop must use to connect to the database. The JDBC connection string must be based on the driver that you define in the driver argument.

Use the following syntax:

```
--connect <JDBC connection string>
```

For example, use the following syntax depending on the database type that you want to connect to:

- Aurora: `--connect "jdbc:mysql://<host_name>:<port>/<schema_name>"`
- Greenplum: `--connect jdbc:postgresql://<host_name>:<port>/<database_name>`
- IBM DB2: `--connect jdbc:db2://<host_name>:<port>/<database_name>`
- IBM DB2 z/OS: `--connect jdbc:db2://<host_name>:<port>/<database_name>`

- Microsoft SQL Server: `--connect jdbc:sqlserver://<host_name>:<port or named_instance>;databaseName=<database_name>`
- Netezza: `--connect "jdbc:netezza://<database_server_name>:<port>/<database_name>;schema=<schema_name>"`
- Oracle: `--connect jdbc:oracle:thin:@<database_host_name>:<database_port>:<database_SID>`
- Teradata: `--connect jdbc:teradata://<host_name>/database=<database_name>`

Use the following syntax to connect to an SSL-enabled database:

```
--connect <JDBC connection string>
```

For example, use the following syntax depending on the database type that you want to connect to:

- Microsoft SQL Server: `--connect jdbc:sqlserver://<host_name>:<port>;databaseName=<database_name>;integratedSecurity=false;encrypt=true;trustServerCertificate=true;TrustStore=/<truststore_location>;TrustStorePassword=<truststore_password>;user=<user_name>;password=<password>`
- Oracle: `--connect jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCPS)(HOST=<host>)(PORT=<port_number>))(CONNECT_DATA=(SERVICE_NAME=<service_name>)))"`

connection-param-file

Defines the extra JDBC parameters through a property file that Sqoop must use to connect to the database. The contents of this file are parsed as standard Java properties and passed into the driver when you create a connection.

Use the following syntax:

```
--connection-param-file <parameter file name>
```

For example, use the following syntax to use the parameter file when you connect to the Oracle database.

```
--connection-param-file param_file
```

connection-manager

Defines the connection manager class name that Sqoop must use to connect to the database.

Use the following syntax:

```
--connection-manager <connection manager class name>
```

For example, use the following syntax to use the generic JDBC manager class name:

```
--connection-manager org.apache.sqoop.manager.GenericJdbcManager
```

direct

When you read data from or write data to Oracle, you can configure the direct argument to enable Sqoop to use OraOop. OraOop is a specialized Sqoop plug-in for Oracle that uses native protocols to connect to the Oracle database. When you configure OraOop, the performance improves.

You can configure OraOop when you run Sqoop mappings on the Spark engine.

Use the following syntax:

```
--direct
```

When you use OraOop, you must use the following syntax to specify multiple arguments:

```
-D<argument=value> -D<argument=value>
```

Note: If you specify multiple arguments and include a space character between -D and the argument name-value pair, Sqoop considers only the first argument and ignores the remaining arguments.

If you do not direct the job to a specific queue, the Spark engine uses the default queue.

-Dsqoop.connection.factories

To run the mapping on the Blaze engine with the Teradata Connector for Hadoop (TDCH) specialized connectors for Sqoop, you must configure the -Dsqoop.connection.factories argument. Use the argument to define the TDCH connection factory class that Sqoop must use. The connection factory class varies based on the TDCH Sqoop Connector that you want to use.

- To use Cloudera Connector Powered by Teradata, configure the -Dsqoop.connection.factories argument as follows:
  `-Dsqoop.connection.factories=com.cloudera.connector.teradata.TeradataManagerFactory`
- To use Hortonworks Connector for Teradata (powered by the Teradata Connector for Hadoop), configure the -Dsqoop.connection.factories argument as follows:
  `-Dsqoop.connection.factories=org.apache.sqoop.teradata.TeradataManagerFactory`

Note: To run the mapping on the Spark engine, you do not need to configure the -Dsqoop.connection.factories argument. The Data Integration Service invokes Cloudera Connector Powered by Teradata and Hortonworks Connector for Teradata (powered by the Teradata Connector for Hadoop) by default.

--infaoptimize

Use this argument to disable the performance optimization of Sqoop pass-through mappings on the Spark engine.

When you run a Sqoop pass-through mapping on the Spark engine, the Data Integration Service optimizes mapping performance in the following scenarios:

- You read data from a Sqoop source and write data to a Hive target that uses the Text format.
- You read data from a Sqoop source and write data to an HDFS target that uses the Flat, Avro, or Parquet format.

If you want to disable the performance optimization, set the --infaoptimize argument to false. For example, if you see data type issues after you run an optimized Sqoop mapping, you can disable the performance optimization.

Use the following syntax:

```
--infaoptimize false
```

For a complete list of the Sqoop arguments that you can configure, see the Sqoop documentation.

Delta Lake JDBC Connection Properties

To enable the domain to access Delta Lake resources in the cloud platform environment, you must manually create and configure a JDBC connection.

Configure the following connection properties:

Property	Description
Name	Type a name for the connection. Example: DatabricksDeltaCxn
ID	Default: Automatically populated with the connection name. Changing this default value is optional.
Username	Type the following value to use the Databricks user token: `token`
Password	Value of the token that you configured for the Databricks user.
JDBC Driver Class Name	Type the following value: `com.simba.spark.jdbc4.Driver`
Connection String	Connection to the Delta Lake resource. This connection string contains all the information that the domain needs to connect to the resource. The connection string contains the following elements: - jdbc:spark://<server host name>. - Port number. - Transport mode. - ssl. Use `1` to enable SSL. - httpPath. - UID. User ID that will be used to run jobs on the cluster. Use `token`. - PWD. Value of the token that you configured for the Databricks user. Example: `jdbc:spark://westus.azuredatabricks.net:443/default;transportMode=http;ssl=1;httpPath=sql/protocolv1/o/1654523072521724/0123-456789-films259;AuthMech=3;UID=token;PWD=<token string>` To get the value of these parameters from the Advanced Options area of the cluster configuration settings: 1. In the Databricks environment, select Clusters. 2. Select the cluster to connect to. 3. Expand the Advanced Options and click the **JDBC/ODBC** tab. For more information about the JDBC connection string for Databricks, see the Databricks documentation.

JDBC V2 Connection Properties

When you set up a JDBC V2 connection, you must configure the connection properties.

Note: The order of the connection properties might vary depending on the tool where you view them.

The following table describes the JDBC V2 connection properties:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters:~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	The description of the connection. The description cannot exceed 4,000 characters.
Location	The domain where you want to create the connection.
Type	The connection type. Select JDBC V2.

The **Details** tab contains the connection attributes of the JDBC V2 connection. The following table describes the connection attributes:

Property	Description
Username	The database user name. User name with permissions to access the database that supports the Type 4 JDBC driver.
Password	The password for the database user name.
Schema Name	Optional. The schema name to connect in the database. If you do not specify the schema name, all the schemas available in the database are listed.

JDBC Driver Class Name	Name of the JDBC driver class.
	The following list provides the driver class name that you can enter for the applicable database type:
	- JDBC driver class name for Azure SQL Database:
	`com.microsoft.sqlserver.jdbc.SQLServerDriver`
	- JDBC driver class name for Aurora PostgreSQL:
	`org.postgresql.Driver`
	- JDBC driver class name for SAP HANA Database:
	`com.sap.db.jdbc.Driver`
	For more information about which driver class to use with specific databases, see the third-party vendor documentation.

Connection String	Connection string to connect to the database.
	Use the following connection string:
	`jdbc:<subprotocol>:<subname>`
	The following list provides sample connection strings that you can enter for the applicable database type:
	- Connection string for Azure SQL Database JDBC driver:
	`jdbc:sqlserver://<host>:<port>;database=<database_name>`
	- Connection string for Aurora PostgreSQL JDBC driver:
	`jdbc:postgresql://<host>:<port>[/<database_name>]`
	- Connection string for SAP HANA Database driver:
	`jdbc:sap://<host>:<port>/?databaseName=<Database_Name>`
	For more information about the connection string to use with specific drivers, see the third-party vendor documentation.

Property	Description
Sub Type	The database type to which you want to connect. You can select from the following database types to connect: - **Azure SQL Database**. Connects to Azure SQL database. - **PostgreSQL**. Connects to Aurora PostgreSQL database. - **SAP HANA Database**. Connects to SAP HANA database. - **Others** . Connects to any database that supports the Type 4 JDBC driver.
Support Mixed-case Identifiers	Enable if the database uses case-sensitive identifiers. When enabled, the Data Integration Service encloses all identifiers within the character selected for the **SQL Identifier Character** property. For example, Aurora PostgreSQL database supports mixed-cased characters. You must enable this property to connect to the Aurora PostgreSQL database. When the **SQL Identifier Character** property is set to none, the **Support Mixed-case Identifiers** property is disabled.
SQL Identifier Character	Type of character that the database uses to enclose delimited identifiers in SQL queries. The available characters depend on the database type. Select (None) if the database uses regular identifiers. When the Data Integration Service generates SQL queries, the service does not place delimited characters around any identifiers. Select a character if the database uses delimited identifiers. When the Data Integration Service generates SQL queries, the service encloses delimited identifiers within this character. **Note:** Select **SQL Identifier Character** as **None** when you specify the SAP HANA Database subtype.

Kafka Connection Properties

The Kafka connection is a messaging connection. Use the Kafka connection to access Kafka as a message target. You can create and manage a Kafka connection in the Developer tool or through infacmd.

The following table describes the general connection properties for the Kafka connection:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: ~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	The string that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	The description of the connection. Enter a string that you can use to identify the connection. The description cannot exceed 4,000 characters.
Location	The domain where you want to create the connection.
Type	The connection type.

The following table describes the Kafka broker properties for the Kafka connection:

Property	Description
Kafka Broker List	Comma-separated list of Kafka brokers which maintains the configuration of the Kafka messaging broker. To specify a Kafka broker, use the following format: <IP Address>:<port>
ZooKeeper Host Port List	Optional. Comma-separated list of Apache ZooKeeper which maintains the configuration of the Kafka messaging broker. To specify the ZooKeeper, use the following format: <IP Address>:<port>
Retry Timeout	Number of seconds the Integration Service attempts to reconnect to the Kafka broker to write data. If the source or target is not available for the time you specify, the mapping execution stops to avoid any data loss.
Kafka Broker Version	Configure the Kafka messaging broker version to 0.10.1.x-2.0.0.

General Properties

The following table describes the general connection properties for the Kafka connection:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: ~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	The string that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	The description of the connection. Enter a string that you can use to identify the connection. The description cannot exceed 4,000 characters.
Location	The domain where you want to create the connection. Select the domain name.
Type	The connection type. Select Messaging/Kafka.

Kafka Broker Properties

The following table describes the Kafka broker properties for the Kafka connection:

Property	Description
Kafka Broker List	Comma-separated list of Kafka brokers which maintain the configuration of the Kafka messaging broker. To specify a Kafka broker, use the following format: `<IP Address>:<port>`
Retry Timeout	Number of seconds after which the Integration Service attempts to reconnect to the Kafka broker to read or write data. If the source or target is not available for the time you specify, the mapping execution stops to avoid any data loss.
Kafka Broker Version	Configure the Kafka messaging broker version to Apache 0.10.1.1 & above.
Additional Connection Properties	Optional. Comma-separated list of connection properties to connect to the Kafka broker. For example, you can use the following syntax: `request.timeout.ms=<value>,session.timeout.ms=<value>, fetch.max.wait.ms=<value>,heartbeat.interval.ms=<value>, security.protocol=SASL_PLAINTEXT,sasl.kerberos. service.name=<kerberos name>,sasl.mechanism=GSSAPI, sasl.jaas.config=com.sun.security.auth.module. Krb5Login Modulerequired useKeyTab=true doNotPrompt=true storeKey=true client=true keyTab="<Keytab Location>" principal="<principal>";` To reduce the time taken to connect to the Kafka broker, ensure that you set the following properties: - `request.timeout.ms` - `session.timeout.ms` - `fetch.max.wait.ms` - `heartbeat.interval.ms` To connect to the Kafka broker in a secured way, ensure that you set one of the following values for the `security.protocol` property: - SASL_SSL - SSL The default value of `security.protocol` property is SASL_PLAINTEXT. **Technical Preview:** The Additional Connection Properties is available for technical preview. Technical preview functionality is supported but is unwarranted and is not production-ready. Informatica recommends that you use in non-production environments only. For more information about the connection properties, see https://kafka.apache.org/documentation/.

SSL Properties

The following table describes the SSL properties for the Kafka connection:

Property	Description
SSL Mode	Required. SSL mode indicates the encryption type to use for the connection. You can choose a mode from the following SSL modes: - Disabled - One way - Two way
SSL TrustStore File Path	Required when **One way** SSL mode is selected. Absolute path and file name of the SSL truststore file that contains certificates of the trusted SSL server.
SSL TrustStore Password	Required when **One way** SSL mode is selected. Password for the SSL truststore.
SSL KeyStore File Path	Required when **Two way** SSL mode is selected. Absolute path and file name of the SSL keystore file that contains private keys and certificates for the SSL server.
SSL KeyStore Password	Required when **Two way** SSL mode is selected. Password for the SSL keystore.

Creating a Kafka Connection Using infacmd

You can use the infacmd command line program to create a Kafka connection.

To create a Kafka connection on UNIX, run the following command:

```
sh infacmd.sh createConnection -dn <domain name> -un <domain user> -pd <domain password> -cn
<connection name> -cid <connection id> -ct Kafka -o
kfkBrkList=<host1:port1>,<host2:port2>,<host3:port3> kafkabrokerversion=<version>
additionalConnectionProperties=<additional properties>
```

For more information about the CreateConnection command, see the *Informatica Command Reference*.

Kudu Connection Properties

Use a Kudu connection to access Kudu.

Note: The order of the connection properties might vary depending on the tool where you view them.

You can create and manage a Kudu connection in the Administrator tool or the Developer tool. The following table describes the Kudu connection properties:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: ~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	The description of the connection. The description cannot exceed 4,000 characters.
Location	The domain where you want to create the connection.
Type	The connection type. Select Kudu.

The following table describes the properties for metadata access:

Property	Description
Kudu Master URLs	The URLs of the Kudu master tables.
Kudu Library Version	The version number of the Kudu library.
Cluster Configuration	The Hadoop cluster that you use for the connection.

Microsoft Azure Blob Storage Connection Properties

Use a Microsoft Azure SQL Blob Storage connection to access a Microsoft Azure Blob Storage.

Note: The order of the connection properties might vary depending on the tool where you view them.

You can create and manage a Microsoft Azure Blob Storage connection in the Administrator tool or the Developer tool. The following table describes the Microsoft Azure Blob Storage connection properties:

Property	Description
Name	Name of the Microsoft Azure Blob Storage connection.
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	Description of the connection.

Property	Description
Location	The domain where you want to create the connection.
Type	Type of connection. Select Azure Blob Storage.

The **Connection Details** tab contains the connection attributes of the Microsoft Azure Blob Storage connection. The following table describes the connection attributes:

Property	Description
Account Name	Name of the Microsoft Azure Storage account.
Authorization Type	Authorization type. You can select any of the following authorization mechanisms: - Shared Key Authorization - Shared Access Signatures
Account Key	Microsoft Azure Storage access key. Applies when you select shared key authorization.
SAS Token	SAS URI with SAS token that you generate on Microsoft Azure portal for your account. Applies when you select shared access signature authorization type. **Note:** You must provide a valid SAS URI with a valid SAS token.
Container Name	The root container or sub-folders with the absolute path. **Note:** To import complex files, specify only the root container.
Endpoint Suffix	Type of Microsoft Azure end-points. You can select any of the following end-points: - `core.windows.net`: Default - `core.usgovcloudapi.net`: To select the US government Microsoft Azure end-points - `core.chinacloudapi.cn`: Not applicable

Microsoft Azure Cosmos DB SQL API Connection Properties

Use a Microsoft Azure Cosmos DB connection to connect to the Cosmos DB database. When you create a Microsoft Azure Cosmos DB connection, you enter information for metadata and data access.

The following table describes the Microsoft Azure Cosmos DB connection properties:

Property	Description
Name	Name of the Cosmos DB connection.
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.

Property	Description
Description	Description of the connection. The description cannot exceed 765 characters.
Location	The project or folder in the Model repository where you want to store the Cosmos DB connection.
Type	Select Microsoft Azure Cosmos DB SQL API.
Cosmos DB URI	The URI of Microsoft Azure Cosmos DB account.
Key	The primary and secondary key to which provides you complete administrative access to the resources within Microsoft Azure Cosmos DB account.
Database	Name of the database that contains the collections from which you want to read or write JSON documents.

Note: You can find the Cosmos DB URI and Key values in the **Keys** settings on Azure portal. Contact your Azure administrator for more details.

Microsoft Azure Data Lake Storage Gen1 Connection Properties

Use a Microsoft Azure Data Lake Storage Gen1 connection to access a Microsoft Azure Data Lake Storage Gen1.

Note: The order of the connection properties might vary depending on the tool where you view them.

You can create and manage a Microsoft Azure Data Lake Storage Gen1 connection in the Administrator tool or the Developer tool. The following table describes the Microsoft Azure Data Lake Storage Gen1 connection properties:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: ~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	The description of the connection. The description cannot exceed 4,000 characters.
Location	The domain where you want to create the connection.
Type	The connection type. Select Microsoft Azure Data Lake Storage Gen1.

The following table describes the properties for metadata access:

Property	Description
ADLS Account Name	The name of the Microsoft Azure Data Lake Storage Gen1.
ClientID	The ID of your application to complete the OAuth Authentication in the Active Directory.
Client Secret	The client secret key to complete the OAuth Authentication in the Active Directory.
Directory	The Microsoft Azure Data Lake Storage Gen1 directory that you use to read data or write data. The default is root directory.
AuthEndpoint	The OAuth 2.0 token endpoint from where access code is generated based on based on the Client ID and Client secret is completed.

For more information about creating a client ID, client secret, and auth end point, contact the Azure administrator or see Microsoft Azure Data Lake Storage Gen1 documentation.

Microsoft Azure Data Lake Storage Gen2 Connection Properties

Use a Microsoft Azure Data Lake Storage Gen2 connection to access a Microsoft Azure Data Lake Storage Gen2.

Note: The order of the connection properties might vary depending on the tool where you view them.

You can create and manage a Microsoft Azure Data Lake Storage Gen2 connection in the Administrator tool or the Developer tool. The following table describes the Microsoft Azure Data Lake Storage Gen2 connection properties:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: ~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	The description of the connection. The description cannot exceed 4,000 characters.
Location	The domain where you want to create the connection.
Type	The connection type. Select Microsoft Azure Data Lake Storage Gen2.

The following table describes the properties for metadata access:

Property	Description
Account Name	The Microsoft Azure Data Lake Storage Gen2 account name or the service name.
Client ID	The ID of your application to complete the OAuth Authentication in the Azure Active Directory (AD).
Client Secret	The client secret key to complete the OAuth Authentication in the Azure AD.
Tenant ID	The Directory ID of the Azure AD.
File System Name	The name of an existing file system in the Microsoft Azure Data Lake Storage Gen2.
Directory Path	The path of an existing directory without the file system name. There is no default directory. You can select one of the following syntax: - `/` for root directory. - `/dir1` - `dir1/dir2`
Adls Gen2 End-point	Type of Microsoft Azure endpoints. You can select any of the following endpoints: - `core.windows.net`: Default - `core.usgovcloudapi.net`: To select the Azure Government endpoints

For more information about creating a client ID, client secret, tenant ID, and file system name, contact the Azure administrator or see Microsoft Azure Data Lake Storage Gen2 documentation.

Microsoft Azure SQL Data Warehouse Connection Properties

Use a Microsoft Azure SQL Data Warehouse connection to access a Microsoft Azure SQL Data Warehouse.

Note: The order of the connection properties might vary depending on the tool where you view them.

You can create and manage a Microsoft Azure SQL Data Warehouse connection in the Administrator tool or the Developer tool. The following table describes the Microsoft Azure SQL Data Warehouse connection properties:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters: ~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. It must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	The description of the connection. The description cannot exceed 4,000 characters.

Property	Description
Location	The domain where you want to create the connection.
Type	The connection type. Select Azure SQL Data Warehouse.

The following table describes the properties for metadata access:

Property	Description
Azure DW JDBC URL	Microsoft Azure Data Warehouse JDBC connection string. For example, you can enter the following connection string: jdbc:sqlserver:// <Server>.database.windows.net: 1433;database=<Database>. The Administrator can download the URL from Microsoft Azure portal.
Azure DW JDBC Username	User name to connect to the Microsoft Azure SQL Data Warehouse account. You must have permission to read, write, and truncate data in Microsoft Azure SQL Data Warehouse.
Azure DW JDBC Password	Password to connect to the Microsoft Azure SQL Data Warehouse account.
Azure DW Schema Name	Name of the schema in Microsoft Azure SQL Data Warehouse.
Azure Storage Type	Type of Azure storage to stage the files. You can select any of the following storage type: - Azure Blob. Default. To use Microsoft Azure Blob Storage to stage the files. - ADLS Gen2. To use Microsoft Azure Data Lake Storage Gen2 as storage to stage the files.
Azure Blob Account Name	Name of the Microsoft Azure Storage account to stage the files.
Azure Blob Account Key	The key that authenticates the access to the Blob storage account.
ADLS Gen2 Storage Account Name	Name of the Microsoft Azure Data Lake Storage Gen2 account to stage the files.
ADLS Gen2 Account Key	Microsoft Azure Data Lake Storage Gen2 access key to stage the files.
Blob End-point	Type of Microsoft Azure end-points. You can select any of the following end-points: - `core.windows.net`: Default - `core.usgovcloudapi.net`: To select the US government Microsoft Azure end-points You can configure the US government Microsoft Azure end-points when a mapping runs in the native environment and on the Spark engine.
VNet Rule	Enable to connect to a Microsoft Azure SQL Data Warehouse endpoint residing in a virtual network (VNet).

Snowflake Connection Properties

When you set up a Snowflake connection, you must configure the connection properties.

Note: The order of the connection properties might vary depending on the tool where you view them.

The following table describes the Snowflake connection properties:

Property	Description
Name	The name of the connection. The name is not case sensitive and must be unique within the domain. You can change this property after you create the connection. The name cannot exceed 128 characters, contain spaces, or contain the following special characters:~ ` ! $ % ^ & * ( ) - + = { [}] | \ : ; " ' < , > . ? /
ID	String that the Data Integration Service uses to identify the connection. The ID is not case sensitive. The ID must be 255 characters or less and must be unique in the domain. You cannot change this property after you create the connection. Default value is the connection name.
Description	Optional. The description of the connection. The description cannot exceed 4,000 characters.
Location	The domain where you want to create the connection.
Type	The connection type. Select SnowFlake.
Username	The user name to connect to the Snowflake account.
Password	The password to connect to the Snowflake account.
Account	The name of the Snowflake account.
Warehouse	The Snowflake warehouse name.
Role	The Snowflake role assigned to the user.

Additional JDBC URL Parameters	Enter one or more JDBC connection parameters in the following format:
	`<param1>=<value>&<param2>=<value>&<param3>=<value>....`
	For example:
	`user=jon&warehouse=mywh&db=mydb&schema=public`
	To access Snowflake through Okta SSO authentication, enter the web-based IdP implementing SAML 2.0 protocol in the following format:
	`authenticator=https://<Your_Okta_Account_Name>.okta.com`
	Note: Microsoft ADFS is not supported.
	For more information about configuring Okta authentication, see the following website:
	https://docs.snowflake.net/manuals/user-guide/admin-security-fed-auth-configure-snowflake.html#configuring-snowflake-to-use-federated-authentication

Creating a Connection to Access Sources or Targets

Create connections before you import data objects, preview data, and profile data.

1. Within the Administrator tool click **Manage** > **Connections**.
2. Select **Actions** > **New** > **Connection**.
3. Select the type of connection that you want to create:
   - To select an HBase connection, select **NoSQL** > **HBase**.
   - To select an HDFS connection, select **File Systems** > **Hadoop File System**.
   - To select a Hive connection, select **Database** > **Hive**.
   - To select a JDBC connection, select **Database** > **JDBC**.
4. Click **OK**.
5. Enter a connection name, ID, and optional description.
6. Configure the connection properties. For a Hive connection, you must choose the **Access Hive as a source or target** option to use Hive as a source or a target.
7. Click **Test Connection** to verify the connection.
8. Click **Finish**.

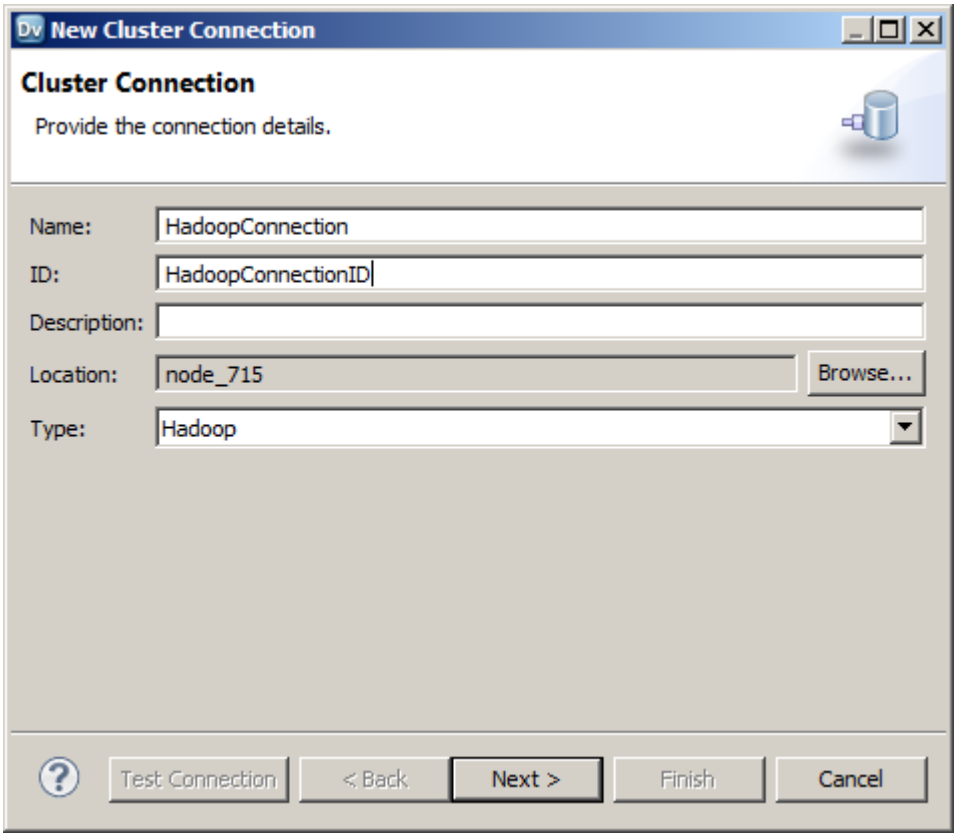
Creating a Hadoop Connection

Create a Hadoop connection before you run a mapping in the Hadoop environment.

1. Click **Window** > **Preferences**.
2. Select **Informatica** > **Connections**.
3. Expand the domain in the **Available Connections** list.
4. Select the **Cluster** connection type in the **Available Connections** list and click **Add**.

   The **New Cluster Connection** dialog box appears.

5. Enter the general properties for the connection.



6. Click **Next**.
7. Enter the Hadoop cluster properties, common properties, and the reject directory properties.
8. Click **Next**.
9. Click **Next**.

   Effective in version 10.2.2, Informatica dropped support for the Hive engine. Do not enter Hive configuration properties.
10. Enter configuration properties for the Blaze engine and click **Next**.
11. Enter configuration properties for the Spark engine and click **Finish**.

Configuring Hadoop Connection Properties

When you create a Hadoop connection, default values are assigned to cluster environment variables, cluster path properties, and advanced properties. You can add or edit values for these properties. You can also reset to default values.

You can configure the following Hadoop connection properties based on the cluster environment and functionality that you use:

- Cluster Environment Variables
- Cluster Library Path

- Common Advanced Properties
- Blaze Engine Advanced Properties
- Spark Engine Advanced Properties

Note: Informatica does not recommend changing these property values before you consult with third-party documentation, Informatica documentation, or Informatica Global Customer Support. If you change a value without knowledge of the property, you might experience performance degradation or other unexpected results.

To reset to default values, delete the property values. For example, if you delete the values of an edited Cluster Library Path property, the value resets to the default $DEFAULT_CLUSTER_LIBRARY_PATH.

Cluster Environment Variables

Cluster Environment Variables property lists the environment variables that the cluster uses. Each environment variable contains a name and a value. You can add environment variables or edit environment variables.

To edit the property in the text box, use the following format with &: to separate each name-value pair:

```
<name1>=<value1>[&:<name2>=<value2>…&:<nameN>=<valueN>]
```

Configure the following environment variables in the **Cluster Environment Variables** property:

HADOOP_NODE_JDK_HOME

Represents the directory from which you run the cluster services and the JDK version that the cluster nodes use. Required to run the Java transformation in the Hadoop environment and Sqoop mappings on the Blaze engine. Default is /usr/java/default. The JDK version that the Data Integration Service uses must be compatible with the JDK version on the cluster.

Set to <cluster JDK home>/jdk<version>.

For example, `HADOOP_NODE_JDK_HOME=<cluster JDK home>/jdk<version>`.

Cluster Library Path

Cluster Library Path property is a list of path variables for shared libraries on the cluster. You can add or edit library path variables.

To edit the property in the text box, use the following format with : to separate each path variable:

```
<variable1>[:<variable2>…:<variableN]
```

Configure the library path variables in the **Cluster Library Path** property.

Common Advanced Properties

Common advanced properties are a list of advanced or custom properties that are unique to the Hadoop environment. The properties are common to the Blaze and Spark engines. Each property contains a name and a value. You can add or edit advanced properties.

To edit the property in the text box, use the following format with &: to separate each name-value pair:

```
<name1>=<value1>[&:<name2>=<value2>…&:<nameN>=<valueN>]
```

Configure the following property in the **Advanced Properties** of the common properties section:

infapdo.java.opts

List of Java options to customize the Java run-time environment. The property contains default values.

If mappings in a MapR environment contain a Consolidation transformation or a Match transformation, change the following value:

- -Xmx512M. Specifies the maximum size for the Java virtual memory. Default is 512 MB. Increase the value to at least 700 MB.
  For example, `infapdo.java.opts=-Xmx700M`

Blaze Engine Advanced Properties

Blaze advanced properties are a list of advanced or custom properties that are unique to the Blaze engine. Each property contains a name and a value. You can add or edit advanced properties.

To edit the property in the text box, use the following format with &: to separate each name-value pair:

```
<name1>=<value1>[&:<name2>=<value2>…&:<nameN>=<valueN>]
```

Configure the following properties in the **Advanced Properties** of the Blaze configuration section:

infagrid.cadi.namespace

Namespace for the Data Integration Service to use. Required to set up multiple Blaze instances.

Set to <unique namespace>.

For example, `infagrid.cadi.namespace=TestUser1_namespace`

infagrid.blaze.console.jsfport

JSF port for the Blaze engine console. Use a port number that no other cluster processes use. Required to set up multiple Blaze instances.

Set to <unique JSF port value>.

For example, `infagrid.blaze.console.jsfport=9090`

infagrid.blaze.console.httpport

HTTP port for the Blaze engine console. Use a port number that no other cluster processes use. Required to set up multiple Blaze instances.

Set to <unique HTTP port value>.

For example, `infagrid.blaze.console.httpport=9091`

infagrid.node.local.root.log.dir

Path for the Blaze service logs. Default is /tmp/infa/logs/blaze. Required to set up multiple Blaze instances.

Verify that all blaze users have write permission on /tmp.

Set to <local Blaze services log directory>.

For example, `infagrid.node.local.root.log.dir=<directory path>`

infacal.hadoop.logs.directory

Path in HDFS for the persistent Blaze logs. Default is /var/log/hadoop-yarn/apps/informatica. Required to set up multiple Blaze instances.

Set to <persistent log directory path>.

For example, `infacal.hadoop.logs.directory=<directory path>`

infagrid.node.hadoop.local.root.log.dir

Path in the Hadoop connection for the service log directory.

Set to <service log directory path>.

For example, `infagrid.node.local.root.log.dir=$HADOOP_NODE_INFA_HOME/blazeLogs`

Spark Advanced Properties

Spark advanced properties are a list of advanced or custom properties that are unique to the Spark engine. Each property contains a name and a value. You can add or edit advanced properties. Each property contains a name and a value. You can add or edit advanced properties.

Configure the following properties in the **Advanced Properties** of the Spark configuration section:

To edit the property in the text box, use the following format with &: to separate each name-value pair:

```
<name1>=<value1>[&:<name2>=<value2>…&:<nameN>=<valueN>]
```

infasjs.env.spark.context-settings.passthrough.spark.dynamicAllocation.executorIdleTimeout

Maximum time that an Spark Jobserver executor node can be idle before it is removed. Increase the value to assist in debugging data preview jobs that use the Spark engine.

You can specify the time in seconds, minutes, or hours using the suffix *s*, *m*, or *h*, respectively. If you do not specify a time unit, the property uses milliseconds.

If you disable dynamic resource allocation, this property is not used.

Default is 120s.

infasjs.env.spark.jobserver.max-jobs-per-context

Maximum number of Spark jobs that can run simultaneously on a Spark context. If you increase the value of this property, you might need to allocate more resources by increasing *spark.executor.cores* and *spark.executor.memory*.

Default is 10.

infasjs.env.spark.jobserver.sparkJobTimeoutInMinutes

Maximum time in minutes that a Spark job can run on a Spark context before the Spark Jobserver cancels the job. Increase the value to assist in debugging data preview jobs that use the Spark engine.

Default is 15.

infaspark.class.log.level.map

Logging level for specific classes in the Spark driver or executor. When you configure this property, it overrides the tracing level you set for the mapping.

Set the value of this property to a JSON string in the following format: {"<fully qualified class name":"<log level>"}

Join multiple class logging level statements with a comma. You can use the following logging levels: FATAL, WARN, INFO, DEBUG, ALL.

For example, set to:

```
infaspark.class.log.level.map={"org.apache.spark.deploy.yarn.ApplicationMaster":"TRAC
E","org.apache.spark.deploy.security.HadoopFSDelegationTokenProvider":"DEBUG"}
```

infaspark.driver.cluster.mode.extraJavaOptions

List of extra Java options for the Spark driver that runs inside the cluster. Required for streaming mappings to read from or write to a Kafka cluster that uses Kerberos authentication.

For example, set to:

```
infaspark.driver.cluster.mode.extraJavaOptions=
-Djava.security.egd=file:/dev/./urandom
-XX:MaxMetaspaceSize=256M -Djavax.security.auth.useSubjectCredsOnly=true
-Djava.security.krb5.conf=/<path to keytab file>/krb5.conf
-Djava.security.auth.login.config=<path to jaas config>/kafka_client_jaas.config
```

To configure the property for a specific user, you can include the following lines of code:

```
infaspark.driver.cluster.mode.extraJavaOptions =
-Djava.security.egd=file:/dev/./urandom
-XX:MaxMetaspaceSize=256M -XX:+UseG1GC -XX:MaxGCPauseMillis=500
-Djava.security.krb5.conf=/etc/krb5.conf
```

infaspark.driver.log.level

Logging level for the Spark driver logs. When you configure this property, it overrides the tracing level you set for the mapping.

Set the value to one of the following levels: FATAL, WARN, INFO, DEBUG, ALL.

infaspark.executor.extraJavaOptions

List of extra Java options for the Spark executor. Required for streaming mappings to read from or write to a Kafka cluster that uses Kerberos authentication.

For example, set to:

```
infaspark.executor.extraJavaOptions=
-Djava.security.egd=file:/dev/./urandom
-XX:MaxMetaspaceSize=256M -Djavax.security.auth.useSubjectCredsOnly=true
-Djava.security.krb5.conf=/<path to krb5.conf file>/krb5.conf
-Djava.security.auth.login.config=/<path to jAAS config>/kafka_client_jaas.config
```

To configure the property for a specific user, you can include the following lines of code:

```
infaspark.executor.extraJavaOptions =
-Djava.security.egd=file:/dev/./urandom
-XX:MaxMetaspaceSize=256M -XX:+UseG1GC -XX:MaxGCPauseMillis=500
-Djava.security.krb5.conf=/etc/krb5.conf
```

infaspark.executor.log.level

Logging level for the Spark executor logs. When you configure this property, it overrides the tracing level you set for the mapping.

Set the value to one of the following levels: FATAL, WARN, INFO, DEBUG, ALL.

infaspark.flatfile.writer.nullValue

When the Databricks Spark engine writes to a target, it converts null values to empty strings (" "). For example, 12, AB,"",23p09udj.

The Databricks Spark engine can write the empty strings to string columns, but when it tries to write an empty string to a non-string column, the mapping fails with a type mismatch.

To allow the Databricks Spark engine to convert the empty strings back to null values and write to the target, configure the property in the Databricks Spark connection.

Set to: TRUE

infaspark.json.parser.mode

Specifies the parser how to handle corrupt JSON records. You can set the value to one of the following modes:

- DROPMALFORMED. The parser ignores all corrupted records. Default mode.

- PERMISSIVE. The parser accepts non-standard fields as nulls in corrupted records.
- FAILFAST. The parser generates an exception when it encounters a corrupted record and the Spark application goes down.

infaspark.json.parser.multiLine

Specifies whether the parser can read a multiline record in a JSON file. You can set the value to true or false. Default is false. Applies only to non-native distributions that use Spark version 2.2.x and above.

infaspark.pythontx.exec

Required to run a Python transformation on the Spark engine for Data Engineering Integration. The location of the Python executable binary on the worker nodes in the Hadoop cluster.

For example, set to:

```
infaspark.pythontx.exec=/usr/bin/python3.4
```

If you use the installation of Python on the Data Integration Service machine, set the value to the Python executable binary in the Informatica installation directory on the Data Integration Service machine.

For example, set to:

```
infaspark.pythontx.exec=INFA_HOME/services/shared/spark/python/lib/python3.4
```

infaspark.pythontx.executorEnv.LD_PRELOAD

Required to run a Python transformation on the Spark engine for Data Engineering Streaming. The location of the Python shared library in the Python installation folder on the Data Integration Service machine.

For example, set to:

```
infaspark.pythontx.executorEnv.LD_PRELOAD=
INFA_HOME/services/shared/spark/python/lib/libpython3.6m.so
```

infaspark.pythontx.executorEnv.PYTHONHOME

Required to run a Python transformation on the Spark engine for Data Engineering Integration and Data Engineering Streaming. The location of the Python installation directory on the worker nodes in the Hadoop cluster.

For example, set to:

```
infaspark.pythontx.executorEnv.PYTHONHOME=/usr
```

If you use the installation of Python on the Data Integration Service machine, use the location of the Python installation directory on the Data Integration Service machine.

For example, set to:

```
infaspark.pythontx.executorEnv.PYTHONHOME=
INFA_HOME/services/shared/spark/python/
```

infaspark.pythontx.submit.lib.JEP_HOME

Required to run a Python transformation on the Spark engine for Data Engineering Streaming. The location of the Jep package in the Python installation folder on the Data Integration Service machine.

For example, set to:

```
infaspark.pythontx.submit.lib.JEP_HOME=
INFA_HOME/services/shared/spark/python/lib/python3.6/site-packages/jep/
```

infaspark.useHiveWarehouseAPI

Enables the Hive Warehouse Connector. Set to TRUE.

For example, `infaspark.useHiveWarehouseAPI=true`.

spark.authenticate

Enables authentication for the Spark service on Hadoop. Required for Spark encryption.

Set to TRUE.

For example, `spark.authenticate=TRUE`

spark.authenticate.enableSaslEncryption

Enables encrypted communication when SASL authentication is enabled. Required if Spark encryption uses SASL authentication.

Set to TRUE.

For example, `spark.authenticate.enableSaslEncryption=TRUE`

spark.datasource.hive.warehouse.load.staging.dir

Directory for the temporary HDFS files used for batch writes to Hive. Required when you enable the Hive Warehouse Connector.

For example, set to `/tmp`

spark.datasource.hive.warehouse.metastoreUri

URI for the Hive metastore. Required when you enable the Hive Warehouse Connector. Use the value for *hive.metastore.uris* from the hive_site_xml cluster configuration properties.

For example, set the value to `thrift://mycluster-1.com:9083 .`

spark.driver.cores

Indicates the number of cores that each driver uses uses to run jobs on the Spark engine.

Set to: `spark.driver.cores=1`

spark.driver.extraJavaOptions

List of extra Java options for the Spark driver.

When you write date/time data within a complex data type to a Hive target using a Hortonworks HDP 3.1 cluster, append the following value to the property: `-Duser.timezone=UTC`

spark.driver.memory

Indicates the amount of driver process memory that the Spark engine uses to run jobs.

Recommended value: Allocate at least 256 MB for every data source.

Set to: `spark.driver.memory=3G`

spark.executor.cores

Indicates the number of cores that each executor process uses to run tasklets on the Spark engine.

Set to: `spark.executor.cores=1`

spark.executor.extraJavaOptions

List of extra Java options for the Spark executor.

When you write date/time data within a complex data type to a Hive target using a Hortonworks HDP 3.1 cluster, append the following value to the property: `-Duser.timezone=UTC`

spark.executor.instances

Indicates the number of instances that each executor process uses to run tasklets on the Spark engine.

Set to: `spark.executor.instances=1`

spark.executor.memory

Indicates the amount of memory that each executor process uses to run tasklets on the Spark engine.

Set to: `spark.executor.memory=3G`

spark.hadoop.hive.llap.daemon.service.hosts

Application name for the LLAP service. Required when you enable the Hive Warehouse Connector. Use the value for *hive.llap.daemon.service.hosts* from the hive_site_xml cluster configuration properties.

spark.hadoop.hive.zookeeper.quorum

Zookeeper hosts used by Hive LLAP. Required when you enable the Hive Warehouse Connector. Use the value for *hive.zookeeper.quorum* from the hive_site_xml cluster configuration properties.

spark.hadoop.validateOutputSpecs

Validates if the HBase table exists. Required for streaming mappings to write to a HBase target in an Amazon EMR cluster. Set the value to false.

spark.scheduler.maxRegisteredResourcesWaitingTime

The number of milliseconds to wait for resources to register before scheduling a task. Default is 30000. Decrease the value to reduce delays before starting the Spark job execution. Required to improve performance for mappings on the Spark engine.

Set to 15000.

For example, `spark.scheduler.maxRegisteredResourcesWaitingTime=15000`

spark.scheduler.minRegisteredResourcesRatio

The minimum ratio of registered resources to acquire before task scheduling begins. Default is 0.8. Decrease the value to reduce any delay before starting the Spark job execution. Required to improve performance for mappings on the Spark engine.

Set to: 0.5

For example, `spark.scheduler.minRegisteredResourcesRatio=0.5`

spark.shuffle.encryption.enabled

Enables encrypted communication when authentication is enabled. Required for Spark encryption.

Set to TRUE.

For example, `spark.shuffle.encryption.enabled=TRUE`

spark.sql.hive.hiveserver2.jdbc.url

URL for HiveServer2 Interactive. Required to use the Hive Warehouse Connector. Use the value in Ambari for HiveServer2 JDBC URL.

spark.yarn.access.hadoopFileSystems

Comma-separated list of external file systems that the Spark service can access. By default, the Spark service has access to the file systems listed in fs.defaultFS in the core-site.xml configuration set of the cluster configuration. Set this property to give the Spark service access to additional file systems.

If you run a mapping on a Cloudera CDP Public Cloud cluster and you use an HDFS on a Cloudera Data Lake cluster, you must allow access to that file system. Append the value for the property *fs.defaultFS* found in `core-site.xml` on the Data Lake cluster. For example:

```
spark.yarn.access.hadoopFileSystems=hdfs://infarndcdppamdl-master1.infarndc.src9-
ltfl.cloudera.site:8020
```

APPENDIX B

Data Type Reference

This appendix includes the following topics:

- Data Type Reference Overview, 380
- Transformation Data Type Support in a Non-native Environment, 381
- Complex File and Transformation Data Types, 382
- Flat File and Transformation Data Types, 388
- Hive Data Types and Transformation Data Types, 389
- Sqoop Data Types, 391

Data Type Reference Overview

Informatica Developer uses the following data types in mappings that run in a non-native environment:

Native data types

Native data types are specific to the sources and targets used as a physical data object. Native data types appear in the physical data object column properties.

Transformation data types

Transformation data types are set of data types that appear in the transformations. They are internal data types based on ANSI SQL-92 generic data types, which the Data Integration Service uses to move data across platforms. Transformation data types appear in all transformations in a mapping.

Transformation data types include the following data types:

- Primitive data type. Represents a single data value in a single column position.
- Complex data type. Represents multiple data values in a single column position. Use complex data types in mappings that run on the Spark engine to process hierarchical data in complex files.

When the Data Integration Service reads source data, it converts the native data types to the comparable transformation data types before transforming the data. When the Data Integration Service writes to a target, it converts the transformation data types to the comparable native data types.

Transformation Data Type Support in a Non-native Environment

The following table shows the Informatica transformation data type support on non-native run-time engines:

Transformation Data Type	Engine
Array	- Spark - Databricks Spark
Bigint	- Blaze - Spark - Databricks Spark
Binary	- Blaze - Spark
Date/Time	- Blaze - Spark - Databricks Spark
Decimal	- Blaze - Spark - Databricks Spark
Double	- Blaze - Spark - Databricks Spark
Integer	- Blaze - Spark - Databricks Spark
Map	- Spark - Databricks Spark
String	- Blaze - Spark - Databricks Spark
Struct	- Spark - Databricks Spark
Text	- Blaze - Spark - Databricks Spark
timestampWithTZ	Not supported

Complex File and Transformation Data Types

You can use complex data types in mappings to process hierarchical data in complex files.

You can use complex data types in the following complex files in mappings that run on the Spark and Databricks Spark engines:

- Avro
- JavaScript Object Notation (JSON)
- Optimized Row Columnar (ORC)
- Parquet

Avro Data Types and Transformation Data Types

Avro data types map to transformation data types that the Data Integration Service uses to move data across platforms.

The following table compares the Avro data types that the Data Integration Service supports and the corresponding transformation data types:

Avro Data Type	Transformation Data Type	Range
Array	Array	Unlimited number of characters.
Boolean	Integer	TRUE (1) or FALSE (0).
Bytes	Binary	Precision 4000.
Date	Date/Time	January 1, 0001 to December 31, 9999.
Decimal	Decimal	Decimal value with declared precision and scale. Scale must be less than or equal to precision. For transformations that support precision up to 38 digits, the precision is 1 to 38 digits, and the scale is 0 to 38. For transformations that support precision up to 28 digits, the precision is 1 to 28 digits, and the scale is 0 to 28. If you specify the precision greater than the maximum number of digits, the Data Integration Service converts decimal values to double in high precision mode.
Double	Double	Precision 15.
Fixed	Binary	1 to 104,857,600 bytes.
Float	Double	Precision 15.
Int	Integer	-2,147,483,648 to 2,147,483,647 Precision 10 and scale 0.
Long	Bigint	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Precision 19 and scale 0.
Map	Map	Unlimited number of characters.

Avro Data Type	Transformation Data Type	Range
Record	Struct	Unlimited number of characters.
String	String	1 to 104,857,600 characters.
Time	Date/Time	Time of the day. Precision to microsecond.
Timestamp	Date/Time	January 1, 0001 00:00:00 to December 31, 9999 23:59:59.997. Precision to microsecond.
Union	Corresponding data type in a union of ["primitive_type|complex_type", "null"] or ["null", "primitive_type|complex_type"].	Dependent on primitive or complex data type.

Avro Union Data Type

A union indicates that a field might have more than one data type. For example, a union might indicate that a field can be a string or a null. A union is represented as a JSON array containing the data types.
The Developer tool only interprets a union of ["primitive_type|complex_type", "null"] or ["null", "primitive_type|complex_type"]. The Avro data type converts to the corresponding transformation data type.

Avro Timestamp Data Type Support

The following table lists the Timestamp data type support for Avro file formats:

Timestamp Data type	Native	Spark
Timestamp_micros	Yes	Yes
Timestamp_millis	Yes	No
Time_millis	Yes	No
Time_micros	Yes	No

Unsupported Avro Data Types

The Developer tool does not support the following Avro data types:

- Enum
- Null
- Timestamp_tz

JSON Data Types and Transformation Data Types

JSON data types map to transformation data types that the Data Integration Service uses to move data across platforms.

The following table compares the JSON data types that the Data Integration Service supports and the corresponding transformation data types:

JSON	Transformation	Range
Array	Array	Unlimited number of characters.
Double	Double	Precision of 15 digits.
Integer	Integer	-2,147,483,648 to 2,147,483,647. Precision of 10, scale of 0.
Object	Struct	Unlimited number of characters.
String	String	1 to 104,857,600 characters.

Unsupported JSON Data Types

The Developer tool does not support the following JSON data types:

- Date
- Decimal
- Timestamp
- Enum
- Union

ORC Data Types and Transformation Data Types

ORC file data types map to transformation data types that the Data Integration Service uses to move data across platforms.

The following table lists the ORC file data types that the Data Integration Service supports and the corresponding transformation data types:

ORC File Data Type	Transformation Data Type	Range and Description
BigInt	BigInt	-9223372036854775808 to 9,223,372,036,854,775,807.
Boolean	Integer	TRUE (1) or FALSE (0).
Char	String	1 to 104,857,600 characters.
Date	Date/Time	January 1, 0001 to December 31, 9999.
Double	Double	Precision of 15 digits.

ORC File Data Type	Transformation Data Type	Range and Description
Float	Double	Precision of 15 digits.
Integer	Integer	-2,147,483,648 to 2,147,483,647.
SmallInt	Integer	-32,768 to 32,767.
String	String	1 to 104,857,600 characters.
Timestamp	Date/Time	January 1, 0001 00:00:00 to December 31, 9999 23:59:59.997. Precision to microsecond.
TinyInt	Integer	-128 to 127.
Varchar	String	1 to 104,857,600 characters.

When you run a mapping on the Spark or Databricks Spark engine to write an ORC file to a target, the Data Integration Service writes the data of the Char and Varchar data types as String.

Note: You can use ORC data types to read and write complex file objects in mappings that run on the Spark engine only.

Unsupported ORC Data Types

The Developer tool does not support the following ORC data types:

- Map
- List
- Struct
- Union

Parquet Data Types and Transformation Data Types

Parquet data types map to transformation data types that the Data Integration Service uses to move data across platforms.

The following table compares the Parquet data types that the Data Integration Service supports and the corresponding transformation data types:

Parquet	Transformation	Range
Binary	Binary	1 to 104,857,600 bytes
Binary (UTF8)	String	1 to 104,857,600 characters
Boolean	Integer	-2,147,483,648 to 2,147,483,647 Precision of 10, scale of 0
Date	Date/Time	January 1, 0001 to December 31, 9999.

Parquet	Transformation	Range
Decimal	Decimal	Decimal value with declared precision and scale. Scale must be less than or equal to precision. For transformations that support precision up to 38 digits, the precision is 1 to 38 digits, and the scale is 0 to 38. For transformations that support precision up to 28 digits, the precision is 1 to 28 digits, and the scale is 0 to 28. If you specify the precision greater than the maximum number of digits, the Data Integration Service converts decimal values to double in high precision mode.
Double	Double	Precision of 15 digits.
Float	Double	Precision of 15 digits.
Int32	Integer	-2,147,483,648 to 2,147,483,647 Precision of 10, scale of 0
Int64	Bigint	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 Precision of 19, scale of 0
Map	Map	Unlimited number of characters.
Struct	Struct	Unlimited number of characters.
Time	Date/Time	Time of the day. Precision to microsecond.
Timestamp	Date/Time	January 1, 0001 00:00:00 to December 31, 9999 23:59:59.997. Precision to microsecond.
group (LIST)	Array	Unlimited number of characters.

The Parquet schema that you specify to read or write a Parquet file must be in smaller case. Parquet does not support case-sensitive schema.

Parquet Timestamp Data Type Support

The following table lists the Timestamp data type support for Parquet file formats:

Timestamp Data type	Native	Spark
Timestamp_micros	Yes	No
Timestamp_millis	Yes	No
Time_millis	Yes	No

Timestamp Data type	Native	Spark
Time_micros	Yes	No
int96	Yes	Yes

Unsupported Parquet Data Types

The Developer tool does not support the following Parquet data types:

- Timestamp_nanos
- Time_nanos
- Timestamp_tz

Rules and Guidelines for Data Types

Consider the following rules and guidelines for data types:

- Avro data types support:
  - Date, Decimal, and Timestamp data types are applicable when you run a mapping in the native environment or on the Spark engine in Cloudera CDH 6.3 distribution.
  - Time data type is applicable when you run a mapping in the native environment in Cloudera CDH 6.3 distribution.
- JSON data types support:
  - For PowerExchange for Microsoft Azure Data Lake Storage Gen2, you can read and write complex file objects in JSON format in mappings that run in the native environment, Spark engine, and Databricks Spark engine.
    For other file-based adapters, you can read and write complex file objects in JSON format in mappings that run on the Spark engine only.
- Parquet data types support:
  - Before you create and run a new mapping on the Databricks engine to read a parquet file with hierarchical data types, you must set the `-DINFA_HADOOP_DIST_DIR=hadoop\Databricks_7.2` option in the `developerCore.ini` file.
  - When you set the `-DINFA_HADOOP_DIST_DIR=hadoop\<Distro>` option in the `developerCore.ini` file and import a Parquet file, the format of the imported metadata differs based on the distribution. For Cloudera CDP 7.1, the metadata is imported as string and for other supported distributions, the metadata is imported as UTF8.
  - Date, Time, and Timestamp data types till microseconds are applicable when you run a mapping in the native environment , Blaze, and Spark engine in the Hortonworks HDP 3.1, Azure HDInsight HDI 4.0, and Cloudera CDP 7.1 distributions.
  - Date, Time_Millis, and Timestamp_Millis data types are applicable when you run a mapping in the native environment or Spark engine in MapR 6.1.
  - Decimal data types are applicable when you run a mapping in the native environment and Spark engine in Cloudera CDH 6.3, Hortonworks HDP 3.1, Amazon EMR 5.20, MapR 6.1, and Azure HDInsight HDI 4.0 distributions.
  - Date, Time, Timestamp, and Decimal data types are applicable when you run a mapping on the Databricks Spark engine.

- When you run a mapping and use Date data type that does not have a time value, the Data Integration Service adds the time value, based on the time zone, to the date in the target.
  For example, Date data type used in the source:

  ```
  1980-01-09
  ```

  Value generated in the target:

  ```
  1980-01-09 00:00:00
  ```

- When you run a mapping in the native environment and use Time data type in the source, the Data Integration Service writes incorrect date value to the target.
  For example, Time data type used in the source:

  ```
  1980-01-09 06:56:01.365235000
  ```

  Incorrect Date value is generated in the target:

  ```
  1899-12-31 06:56:01.365235000
  ```

- When you run a mapping in the native environment and use Date data type in the source, the Data Integration Service writes incorrect time value to the target.
  For example, Date data type used in the source:

  ```
  1980-01-09 00:00:00
  ```

  Incorrect Time value generated in the target:

  ```
  1980-01-09 05:30:00
  ```

- To run a mapping that reads and writes Date, Time, Timestamp, and Decimal data types, update the `-DINFA_HADOOP_DIST_DIR` option to the `developerCore.ini` file. The `developerCore.ini` file is located in the following directory:
  `<Client installation directory>\clients\DeveloperClient\`

  Add the following path to the `developerCore.ini` file:

  ```
  -DINFA_HADOOP_DIST_DIR=hadoop\<Hadoop distribution>_<version>
  ```

  For example: `-DINFA_HADOOP_DIST_DIR=hadoop\CDH_6.3`

- To use precision up to 38 digits for Decimal data type in the native environment, set the `EnableSDKDecimal38` custom property to `true` for the Data Integration Service. The `EnableSDKDecimal38` custom property is applicable to all file-based PowerExchange adapters except PowerExchange for HDFS.

Flat File and Transformation Data Types

Flat file data types map to transformation data types that the Data Integration Service uses to move data across platforms.

The following table compares flat file data types to transformation data types:

Flat File Data type	Transformation Data type	Range
Bigint	Bigint	Precision of 19 digits, scale of 0
Datetime	Date/Time	Jan 1, 0001 A.D. to Dec 31, 9999 A.D. (precision to the nanosecond)
Double	Double	Precision of 15 digits

Flat File Data type	Transformation Data type	Range
Int	Integer	-2,147,483,648 to 2,147,483,647
Nstring	String	1 to 104,857,600 characters
Number	Decimal	For transformations that support precision up to 38 digits, the precision is 1 to 38 digits, and the scale is 0 to 38. For transformations that support precision up to 28 digits, the precision is 1 to 28 digits, and the scale is 0 to 28. If you specify the precision greater than the maximum number of digits, the Data Integration Service converts decimal values to double in high precision mode. If the precision is greater than 15, the Data Integration Service converts decimal values to double in low-precision mode.
String	String	1 to 104,857,600 characters
TimestampWithTZ	TimestampWithTZ	Aug. 1, 1947 A.D to Dec. 31, 2040 A.D. -12:00 to +14:00 Precision of 36 and scale of 9. (precision to the nanosecond) Timestamp with Time Zone data type does not support the following time zone regions: - AFRICA_CAIRO - AFRICA_MONROVIA - EGYPT - AMERICA_MONTREAL **Note:** TimestampWithTZ is applicable only to the native environment.

When the Data Integration Service reads non-numeric data in a numeric column from a flat file, it drops the row and writes a message in the log. Also, when the Data Integration Service reads non-datetime data in a datetime column from a flat file, it drops the row and writes a message in the log.

Hive Data Types and Transformation Data Types

The following table lists the Hive data types that Data Integration Service supports and the corresponding transformation data types:

Hive Data Type	Transformation Data Type	Range and Description
Binary	Binary	1 to 104,857,600 bytes. You can read and write data of Binary data type in a Hadoop environment. You can use the user-defined functions to transform the binary data type.
Tiny Int	Integer	-32,768 to 32,767

Hive Data Type	Transformation Data Type	Range and Description
Integer	Integer	-2,147,483,648 to 2,147,483,647 Precision 10, scale 0
Bigint	Bigint	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 Precision 19, scale 0
Decimal	Decimal	For transformations that support precision up to 38 digits, the precision is 1 to 38 digits, and the scale is 0 to 38. For transformations that support precision up to 28 digits, the precision is 1 to 28 digits, and the scale is 0 to 28. If a mapping is not enabled for high precision, the Data Integration Service converts all decimal values to double values. If a mapping is enabled for high precision, the Data Integration Service converts decimal values with precision greater than 38 digits to double values.
Double	Double	Precision 15
Float	Double	Precision 15
String	String	1 to 104,857,600 characters
Boolean	Integer	TRUE (1) or FALSE (0). The default transformation type for boolean is integer. You can also set this to string data type with values of True and False.
Array	Array	Unlimited number of characters.
Struct	Struct	Unlimited number of characters.
Map	Map	Unlimited number of characters.
Timestamp	datetime	January 1, 0001 00:00:00 to December 31, 9999 23:59:59.997. Precision to microsecond.
Date	datetime	January 1, 0001 to December 31, 9999.
Char	String	1 to 255 characters
Varchar	String	1 to 65355 characters

Hive Complex Data Types

Hive complex data types such as arrays, maps, and structs are a composite of primitive or complex data types. Informatica Developer represents complex data types with the string data type and uses delimiters to separate the elements of the complex data type.

You can read and write hierarchical data from Hive tables in a mapping that runs on the Spark engine.

The following table lists the complex data types:

Complex Data Type	Description
Array	An array contains an ordered collection of elements of same data type. The elements in an array are delimited by commas. For example, an array of `fruits` is represented as `[apple,banana,orange]`.
Map	A map contains an unordered collection of key-value pairs and are represented as pairs of strings and integers delimited by the = character. String and integer pairs are delimited by commas. For example, a map of `fruits` is represented as `[1=apple,2=banana,3=orange]`.
Struct	A struct contains a collection of elements of different data types delimited by the : character. String and integer pairs are delimited by commas. For example, a struct is represented as `struct {1:"apple" [, "apple":"red", ...]} .`

Sqoop Data Types

When you use Sqoop, some variations apply in the processing. Sqoop supports a subset of data types that database vendors support.

Aurora Data Types

Informatica supports the following Aurora data types when you use Sqoop:

- Binary
- Bit
- Blob (supported only for import)
- Char
- Date
- Datetime
- Decimal
- Double
- Enum
- Float
- Integer
- Numeric
- Real
- Set
- Text
- Time
- Timestamp
- Varbinary
- Varchar

IBM DB2 and DB2 for z/OS Data Types

Informatica supports the following IBM DB2 and DB2 for z/OS data types when you use Sqoop:

- Bigint
- Blob (supported only for import)
- Char
- Clob
- Date
- DBClob
- DecFloat (supported only for import)
- Decimal
- Double (supported only for DB2 for z/OS)
- Float (supported only for DB2)
- Graphic
- Integer
- LongVargraphic (supported only for DB2)
- Numeric
- Real
- Smallint
- Time
- Timestamp
- Varchar
- Vargraphic
- XML (supported only for import)

Greenplum Data Types

Informatica supports the following Greenplum data types when you use Sqoop:

- Bigint
- Bigserial
- Bytea
- Date
- Decimal
- Double
- Integer
- Nchar
- Numeric
- Nvarchar
- Real
- Serial
- Smallint

Text

Time

Timestamp

Microsoft SQL Server Data Types

Informatica supports the following Microsoft SQL Server data types when you use Sqoop:

Bigint

Bit

Char

Datetime

Decimal

Float

INT

Money

Numeric

Real

Smalldatetime

Smallint

Smallmoney

Text

Time

Tinyint

Varchar

Rules and Guidelines for Sqoop Microsoft SQL Server Data Types

Consider the following rules and guidelines when you configure Microsoft SQL Server data types in a Sqoop mapping:

- If you create or replace the target table at run time and run the mapping on the Blaze or Spark engine to export Bigint data, the mapping fails.
- If you run a Sqoop mapping to export time data, Sqoop does not export milliseconds.

Netezza Data Types

Informatica supports the following Netezza data types when you use Sqoop:

Bigint

Blob (supported only for import)

Byteint

Char

Date

Double

Float4

Float8

Number

Timestamp

Varchar

Oracle Data Types

Informatica supports the following Oracle data types when you use Sqoop:

Blob (supported only for import)

Char

Date

Float

Long

Nchar (supported if you configure OraOop)

Nvarchar (supported if you configure OraOop)

Number(P,S)

Timestamp

Varchar

Varchar2

Rules and Guidelines for Sqoop Oracle Data Types

Consider the following rules and guidelines when you configure Oracle data types in a Sqoop mapping:

- If you run a Sqoop mapping on the Blaze engine to export Oracle float data, Sqoop truncates the data.
- If you run a Sqoop mapping on the Blaze engine to export Oracle timestamp data with nanoseconds, Sqoop writes only three digits to the target.
- If you configure OraOop and run a Sqoop mapping on the Spark engine to export Oracle timestamp data, Sqoop writes only three digits to the target.

Teradata Data Types

Informatica supports the following Teradata data types when you use Sqoop:

Bigint (supported only for import)

Blob (supported only for import)

Byteint

Char

Clob

Date

Decimal

Double

Float

Integer

Number

- Numeric
- Real
- Smallint
- Time
- Timestamp
- Varchar

Note: If you run a Sqoop mapping to export data of the Time data type, Sqoop does not export milliseconds.

Teradata Data Types with TDCH Specialized Connectors for Sqoop

Informatica supports the following Teradata data types when you use the Cloudera Connector Powered by Teradata, Hortonworks Connector for Teradata, and MapR Connector for Teradata with Sqoop:

- Bigint
- Byte (supported only by Hortonworks Connector for Teradata and MapR Connector for Teradata)
- Byteint
- Character
- Date
- Decimal
- Double Precision/Float/Real
- Integer
- Number(P,S)
- Numeric
- Smallint
- Time (supported only by Cloudera Connector Powered by Teradata and Hortonworks Connector for Teradata)
- Timestamp
- Varchar
- Varbyte (supported only by Hortonworks Connector for Teradata and MapR Connector for Teradata)

Vertica Data Types

Informatica supports the following Vertica data types when you use Sqoop:

- Boolean (supported only on the Spark engine)
- Bigint
- Char
- Date
- Datetime
- Decimal
- Double precision
- Float
- Float (n)

Float 8

Int

Int8

Integer

Long Varchar

Money

Number

Numeric

Real

Smalldatetime

Smallint

Time

Timestamp

Tinyint

Varchar

APPENDIX C

Function Reference

This appendix includes the following topics:

- Function Support in a Non-native Environment, 397
- Function and Data Type Processing, 399

Function Support in a Non-native Environment

Some Informatica transformation language functions that are valid in the native environment might be restricted or unsupported in a non-native environment, such as Hadoop or Databricks. Functions not listed in this table are supported on all engines without restrictions.

Important: When you push a mapping to a non-native environment, the engine that processes the mapping uses a set of rules different from the Data Integration Service. As a result, the mapping results can vary based on the rules that the engine uses.

The following table lists functions and levels of support for functions on different engines in the non-native environments.

Function	Blaze Engine	Spark Engine	Databricks Spark
ABORT	Not supported	Not supported	Not supported
ANY	Supported	Supported	Supported
ARRAY	Not supported	Supported	Supported
AVG	Supported	Supported	Supported
CAST	Not supported	Supported	Supported
COLLECT_LIST	Not supported	Supported	Supported
COLLECT_MAP	Not supported	Supported	Supported
CONCAT_ARRAY	Not supported	Supported	Supported
COUNT	Supported	Supported	Supported
CREATE_TIMESTAMP_TZ	Supported	Not supported	Not supported

Function	Blaze Engine	Spark Engine	Databricks Spark
CUME	Not supported	Not supported	Not supported
DECOMPRESS	Supported	Supported	Supported
ERROR	Not supported	Not supported	Not supported
EXTRACT_STRUCT	Not supported	Supported	Supported
FIRST	Not supported	Not supported	Not supported
GET_TIMEZONE	Supported	Not supported	Not supported
GET_TIMESTAMP	Supported	Not supported	Not supported
IS_DATE	Not supported	Supported	Supported
LAST	Not supported	Not supported	Not supported
LAG	Not supported	Supported	Supported
LEAD	Not supported	Supported	Supported
MAP	Not supported	Supported	Supported
MAP_FROM_ARRAYS	Not supported	Supported	Supported
MAP_KEYS	Not supported	Supported	Supported
MAP_VALUES	Not supported	Supported	Supported
MAX (Dates)	Supported	Supported	Supported
MAX (Numbers)	Supported	Supported	Supported
MAX (String)	Supported	Supported	Supported
METAPHONE	Supported	Supported	Supported
MIN (Dates)	Supported	Supported	Supported
MIN (Numbers)	Supported	Supported	Supported
MIN (String)	Supported	Supported	Supported
MOVINGAVG	Not supported	Not supported	Not supported
MOVINGSUM	Not supported	Not supported	Not supported
PARSE_JSON	Not supported	Supported	Not supported
PARSE_XML	Not supported	Supported	Not supported
PERCENTILE	Supported	Supported	Supported

Function	Blaze Engine	Spark Engine	Databricks Spark
RESPEC	Not supported	Supported	Supported
SIZE	Not supported	Supported	Supported
STDDEV	Supported	Supported	Supported
STRUCT	Not supported	Supported	Supported
STRUCT_AS	Not supported	Supported	Supported
SUM	Supported	Supported	Supported
SYSTIMESTAMP	Supported	Supported	Supported
TO_DATE	Supported	Supported	Supported
TO_DECIMAL	Supported	Supported	Supported
TO_DECIMAL38	Supported	Supported	Supported
TO_TIMESTAMP_TZ	Supported	Not supported	Not supported
UUID4	Supported	Supported	Supported
UUID_UNPARSE	Supported	Supported	Supported
VARIANCE	Supported	Supported	Supported

For more information, see the *Informatica Transformation Language Reference*.

Function and Data Type Processing

When you run a mapping in a Hadoop environment, the Hadoop engine might process Informatica functions and data types differently from the Data Integration Service. Also, each of the run-time engines in the Hadoop environment can process Informatica functions and data types differently. Therefore, some variations apply in the processing and validity of functions and data types, and mapping results can vary.

INDEX

A

accessing elements
  array 238
  map 238
  struct 238
Address Validator transformation
  non-native environment 120
ADLS
  complex file sources 85, 92
  complex file targets 102, 106
aggregate window function
  example 300
  nesting 302
  offsets 302
Aggregator transformation
  Blaze engine 121
  Databricks Spark engine 122
  non-native environment 121
  Spark engine 122
Amazon AWS 313
Amazon Redshift connection
  properties 322
Amazon S3
  complex file sources 88
  complex file targets 105
Amazon S3 connection
  properties 324
architecture
  Data Engineering Integration 18
  Data Engineering with Databricks 24
  Hadoop environment 19
array
  complex data type 223
  accessing elements 238
  dimension 223
  example 223
  extracting element 239
  format 223
  generating 240
  index 223
  multi-dimensional 223
  one-dimensional 223
array functions
  ARRAY 240
  COLLECT_LIST 240
  CONCAT_ARRAY 240
  SIZE 240
array port
  type configuration 234
audit
  condition 50
  configuring storage 50
  overview 47
  results 52
  rule 48
audit *(continued)*
  rules and guidelines 52
  running an audit 51
AutoDeploy 185
Avro data types
  transformation data types 382
Azure
  configuration 316
Azure Blob
  complex file sources 91
  complex file targets 108
Azure Blob Storage
  complex file sources 86
  complex file targets 103
Azure Data Lake Storage Gen2
  complex file sources 93
  complex file targets 103, 107
AzureData Lake Storage Gen2
  complex file sources 86

B

Blaze engine
  Blaze engine architecture 22
  connection properties 338
  mapping properties 208
  monitoring 204, 206, 207
  segment time 207
  summary report 207–210
  tasklet execution time 209
  tasklet information 210
Blaze execution plan
  monitoring 201
Blaze Job Monitor
  logging 211
blockchain
  blockchain data object 284–286
  blockchain data object operation 288, 291
  blockchain data object read operation 288, 291
  blockchain data object write operation 290, 291
  connection properties 326
  example 292
  overview 283
  Response_200 284
  responseCode 284
  use case 292

C

Case Converter transformation
  non-native environment 123
Cassandra connections
  properties 328

CLAIRE
insights 73
recommendations 69
recommendations and insights 68
Classifier transformation
non-native environment 123
cloud platform clusters 169
cloud provisioning configuration
Databricks properties 321
Amazon AWS properties 313
Microsoft Azure properties 316
cluster workflow
cloud provisioning connection 312
cluster workflows
Azure job monitoring 186
components 170, 185, 186
Delete Cluster task 186
external RDS as the Hive metastore db 184
mappings in cluster workflows 185
monitoring 186
Overview 169
running 186
Running mappings on Blaze engine 183
sample 170
workflow tasks 186
Comparison transformation
non-native environment 123
complex data type
array 222
map 222
struct 222
complex data type definition
creating 232
example 229
importing 232
nested data type definition 231
recursive data type definition 231
struct 229
type definition library 229
complex data types
array 223
map 225
struct 225
complex file formats
overview 382
complex file sources
ADLS 85, 92
Amazon S3 88
Azure Blob 91
Azure Blob Storage 86
Azure Data Lake Storage Gen2 86, 93
Google Cloud Storage 89
HDFS 89
Hive 90
MapR-FS 90
complex file targets
ADLS 102, 106
Amazon S3 105
Azure Blob 108
Azure Blob Storage 103
Azure Data Lake Storage Gen2 103, 107
Databricks environment 102
Google Cloud Storage 106
HDFS 109
Hive 110
MapR-FS 109
complex functions
ARRAY 240
complex functions *(continued)*
CAST 240
COLLECT_LIST 240
COLLECT_MAP 240
CONCAT_ARRAY 240
MAP 240
MAP_FROM_ARRAYS 240
MAP_KEYS 240
MAP_VALUES 240
RESPEC 240
SIZE 240
STRUCT 240
STRUCT_AS 240
complex operator
dot operator 238
example 238
subscript operator 238
complex port
array 227
creating 229
map 227
nested 227
Read transformation 228
struct 227
transformations 228
type configuration 227, 234
Write transformation 228
component architecture
clients and tools 18
configure a mapping
non-native 46
configuring
array port properties 234
map port properties 236
struct port
properties 237
Confluent Kafka connection
Confluent Kafka broker properties 330
create using infacmd 331
general properties 330
Connection
details 363
properties 363
connection properties
Databricks 331
blockchain 326
connections
properties 312, 338
Google PubSub 338
HBase 312
HDFS 312
Hive 312
JDBC 312
Consolidation transformation
non-native environment 124
conversion
hierarchical to relational 246
Cosmos DB connection
creating 364
Create Cluster task
properties 173
creating
Cosmos DB connection 364
Python transformation 155
creating a column profile
profiles 191

D

data engineering
  application services 18
  data engineering process 25
data engineering integration
  repositories 19
data engineering process
  collect your data 25
Data Masking transformation
  non-native environment 125
data object
  processing data with an intelligent structure 277
data object profiles
  creating a single profile 189
  enterprise discovery 190
data preview
  data preview process 162
  data preview interface 164
  data viewer 165
  hierarchical type panel 166
  overview 161
  previewing hierarchical data 163
  rules and guidelines for data preview 168
Data Processor transformation
  non-native environment 126
data type
  complex data type 222
  nested data type 222
  primitive data type 222
data type reference
  flat files 388
data types
  complex files 382
  Hive 389
  Hive complex data types 390
  processing in a Hadoop environment 399
  support 381
Databricks
  cloud provisioning configuration 321
  create cluster task 179, 181
  warm pool support 169
  warm pools, configuring 179, 181
Databricks connection properties 331
Databricks environment
  flat file sources 87
  flat file targets 104
  complex file targets 102
  Delta Lake sources 87
  sources 85
Databricks integration
  overview 21
Databricks Spark
  execution plans 56
Decision transformation
  non-native environment 127
Delete Cluster task 186
Delta Lake
  connection 357
  transformation support 117
dot operator
  extracting struct element 239
dynamic complex port
  input rules 267

E

enterprise discovery
  running in Informatica Analyst 193
ephemeral clusters
  cloud provisioning connection 312
  Overview 169
example
  aggregate window function 300
  array 223
  complex data type definition 229
  complex operator 238
  dot operator 238
  map 225
  nested data type definition 231
  partition and order keys 297
  struct 225
  subscript operator 238
  windowing 304, 306
execution plan
  Databricks Spark engine 56
  Spark engine 56
Expression transformation
  Blaze engine 128
  Databricks Spark engine 128
  non-native environment 127
  Spark engine 128
external RDS as the Hive metastore 184

F

file sources
  Hadoop environment 88
file targets
  Hadoop environment 105
Filter transformation
  Blaze engine 129
  non-native environment 128
flat file source 94
flat file sources
  in Databricks environment 87
  in Hadoop environment 94
flat file targets
  in Databricks environment 104
functions
  processing in a Hadoop environment 399

G

Google Analytics connections
  properties 333
Google BigQuery connection
  properties 334
Google Cloud Spanner connection
  properties 336
Google Cloud Storage
  complex file sources 89
  complex file targets 106
Google Cloud Storage connections
  properties 337
Google PubSub
  connection properties 338
grid
  description 64

H

Hadoop 312
Hadoop connections
  creating 370
Hadoop environment
  file sources 88
  file targets 105
  flat file limitations 94
  flat file targets 110
  Hive targets 112
  logs 198
  optimization 67
  relational sources 95
  Sqoop sources restrictions 98
  Sqoop targets restrictions 115
hadoop utilities
  Sqoop 20
HBase connections
  MapR-DB properties 346
  properties 346
HDFS
  complex file sources 89
  complex file targets 109
HDFS connections
  creating 370
  properties 344
HDFS mappings
  data extraction example 61
  description 60
hierarchical conversion
  generate nested struct 249
  generate struct 247
  hierarchical to hierarchical 257
  hierarchical to relational 257, 259
  relational to hierarchical 247, 249
hierarchical data
  complex files 219
  converting 246
  extracting elements 257
  flattening elements 259
  how to process 220
  modifying 257, 259
  processing 219
  processing semi-structured data 275
  processing unstructured data 275
  Spark engine 219
  use midstream parsing with 243
Hierarchical to Relational transformation
  non-native environment 129
high availability
  description 65
Hive
  complex file sources 90
  complex file targets 110
  target limitations 112
Hive connections
  creating 370
  properties 347
Hive execution plan
  monitoring 201
Hive mappings
  description 62
Hive metastore 184
Hive pushdown
  connection properties 338
Hive query plan
  viewing, for mapping 57
Hive query plan *(continued)*
  viewing, for profile 195
Hive sources
  postSQL 95
  preSQL 95
  with Blaze engine 96
  with Informatica mappings 95
Hive target
  truncate 113
Hive targets
  postSQL 112
  preSQL 112
  with Blaze engine 114
how to
  process hierarchical data 220
  use midstream parsing 243

I

Informatica Data Engineering Integration
  overview 17
Informatica engine
  Informatica engine execution plan 55
Informatica Intelligent Cloud Services account
  creating 281
input rules
  dynamic array 267
  dynamic complex port 267
  dynamic map 267
  dynamic struct 267
intelligent structure discovery
  process 276
Intelligent Structure Discovery
  license 281
intelligent structure mappings
  data conversion example 280
intelligent structure model
  creating 281
  exporting 282
  overview 275
intelligent structures
  use case 277

J

Java transformation
  Blaze engine 129
  Non-native environment 129
  Spark engine 130
JDBC
  sources 35
  targets 35
JDBC connection
  to Delta Lake resource 357
JDBC connections
  connection string 352
  properties 350
JDBC V2 connection
  properties 357
Joiner transformation
  Blaze engine 131
  Databricks Spark engine 132
  non-native environment 131
  Spark engine 131
JSON data types
  transformation data types 384

K

Kafka connection
create using infacmd 362
general properties 360
Kafka broker properties 361
Key Generator transformation
Non-native environment 132
Kudu connection
properties 362

L

Labeler transformation
Non-native environment 133
logging
mapping run on Hadoop 211
Spark engine 217
logs
Blaze engine 210
Hadoop environment 198
Spark engine 217
logs URL
YARN web user interface 217
Lookup transformation
Blaze engine 133
Databricks Spark engine 136, 142
Non-native environment 133
Spark engine 134

M

Macro transformation
non-native environment 137
map
complex data type 225
accessing elements 238
example 225
format 225
generating 240
key-value pair 225
map functions
COLLECT_MAP 240
MAP 240
MAP_FROM_ARRAYS 240
MAP_KEYS 240
MAP_VALUES 240
SIZE 240
map port
type configuration 236
mapping
audit 47
intelligent structure 277
mapping example
Hive 62
Twitter 63
mapping execution plans
overview 54
mapping optimization
insights 73
recommendations 69
recommendations and insights 68
mapping run on Hadoop
logging 200
monitoring 201
mapping run on non-native environment
overview 29
MapR-FS
complex file sources 90
complex file targets 109
Match transformation
Blaze engine 137
Non-native environment 137
Spark engine 138
Merge transformation
Non-native environment 138
Messaging connection
Confluent Kafka connection 329
Microsoft Azure 316
Microsoft Azure Data Lake Storage Gen1 connection
properties 365
Microsoft Azure Data Lake Storage Gen2 connection
properties 366
Microsoft Azure SQL Data Warehouse connection
properties 367
monitoring cluster workflow jobs 186
Monitoring URL
Blaze and Spark jobs 200
multiple matches
Lookup transformation 133

N

native environment
high availability 65
mappings 60
optimization 64
partitioning 64
nested data type definition
example 231
nested struct
generating 249
node labeling
Blaze 80
non-native environment
transformation support 117
valid sources 83
non-native execution plan
description, for mapping 29
overview 54
non-native mapping
run-time properties 30
non-native mappings
workflows 46
Normalizer transformation
Non-native environment 138

O

optimization
compress temporary staging tables 76
node labeling 79
queuing 80
scheduling 79
truncate partitions 77
ORC file data types
transformation data types 384
overview
processing hierarchical data 219

P

Parquet data types
  transformation data types 385
Parser transformation
  Non-native environment 139
partitioning
  description 64
  optimization 65
postSQL
  for Hive sources 95
  for Hive targets 112
  for JDBC targets 35
preSQL
  for Hive sources 95
  for Hive targets 112
  for JDBC sources 35
processing hierarchical data
  overview 219
processing with an intelligent structure
  in a mapping 277
  running on Spark 277
profile run on Blaze engine
  Overview 187
profile run on Hadoop
  monitoring 195
profile run on Hive
  Overview 187
profiles
  creating a column profile 191
python
  guidelines 154
  rules 154
Python
  advanced properties 151
  code 153
  components 152
  creating 154
  data type 150
  example 159
    input port
      data type 151
  input ports 151
    output port
      data type 151
  output ports 151
  pre-trained model 152
  resource file 152
  script 152
  use case 159
Python transformation
  active transformation 150
  creating 155
  example 156, 157
  nonpartitioned data 156
  overview 149
  passive transformation 150

R

Rank transformation
  Blaze engine 139
  Databricks Spark engine 140
  Non-native environment 139
  Spark engine 139
registration
  Informatica Intelligent Cloud Services account 281
relational to hierarchical 246
Relational to Hierarchical transformation
  non-native environment 140
Router transformation
  Non-native environment 141
Rule Specification transformation
  Non-native environment 141
rules and guidelines
  Spark engine 43
  window functions 304
  windowing properties 299
run-time properties
  non-native mapping 30, 31

S

semi-structured data
  processing 275
Sequence Generator transformation
  Blaze engine 141
  Non-native environment 141
  Spark engine 142
Snowflake connection
  properties 369
social media mappings
  description 63
Sorter transformation
  Blaze engine 142
  Databricks Spark engine 144
  Non-native environment 142
  Spark engine 143
source file name
  for flat file sources 94
sources
  in a non-native environment 83
  Databricks environment 85
  JDBC 35
Spark
  execution plans 56
Spark deploy mode
  Hadoop connection properties 338
Spark engine
  data type processing 43
  function processing 43
  rules and guidelines 43
  connection properties 338
  hierarchical data 219
  monitoring 212
Spark Event Log directory
  Hadoop connection properties 338
Spark execution parameters
  Hadoop connection properties 338
Spark HDFS staging directory
  Hadoop connection properties 338
Sqoop arguments
  parameterization 41
Sqoop configuration
  mapping properties 41
  profiling 188
Sqoop connection arguments
  -Dsqoop.connection.factories 354
  connect 354
  direct 354
  driver 354
Sqoop connectivity
  supported data types 391

Sqoop data types
Aurora 391
Greenplum 392
IBM DB2 392
IBM DB2 for z/OS 392
Microsoft SQL Server 393
Netezza 393
Oracle 394
Teradata 394
Teradata Data Types with TDCH Sqoop Specialized Connectors 395
Vertica 395
Sqoop incremental data extraction
--infa-incremental-key 39
--infa-incremental-type 39
--infa-incremental-value 39
--infa-incremental-value-format 39
configuring 39
Sqoop mapping arguments
batch 38
infaoptimize 38
infaownername 39
m 37
num-mappers 37
schema 39
split-by 37
verbose 39
Sqoop mappings
overview 36
supported engines 36
Sqoop sources
importing data from Vertica 98
in Hadoop environment 98
Sqoop targets
in Hadoop environment 115
Standardizer transformation
Non-native environment 144
stateful computing
overview 295
struct
complex data type 225
accessing elements 238
changing data type 240
changing schema 240
complex data type definition 229
example 225
extracting element 239
format 225
generating 240, 247
name-type pair 225
schema 225
struct functions
CAST 240
RESPEC 240
STRUCT 240
STRUCT_AS 240
struct port
complex data type definition 237
type configuration 237
subscript operator
extracting array element 239

T

targets
flat files in Hadoop mapping 110
JDBC 35
TDCH connection factory
-Dsqoop.connection.factories 354
third-party tools
hadoop cluster 20
transformations
in a non-native environment 117
type configuration
array 234
map 234
struct 234
struct port 237
type definition library
complex data type definition 229
mapping 229
mapplet 229
nested data type definition 231

U

Union transformation
Non-native environment 145
unstructured data
processing 275
Update Strategy transformation
Blaze engine 145
Non-native environment 145
Spark engine 146
utilities
hadoop cluster 20

V

validation environments
non-native mapping 31

W

warm pools 179, 181
Weighted Average transformation
Non-native environment 148
window functions
aggregate functions 300
LAG 300
LEAD 300
overview 299
rules and guidelines 304
windowing
example 304, 306, 308
overview 295
properties 296
windowing properties
frame 296, 302
order 296, 297
partition 296, 297
rules and guidelines 299
workflows
non-native mappings 46

Y

YARN
queues 79
scheduler
capacity 79

YARN *(continued)*
scheduler *(continued)*
fair 79
YARN web user interface
description 199