Informatica® Cloud Data Integration
February 2025

Transformations

Informatica Cloud Data Integration Transformations
February 2025

Table of Contents

Preface

Refer to *Transformations* for information about the transformations that you can include in mappings and mapplets. Learn how to transform your data when you move it from source to target.

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 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 Intelligent Cloud Services web site

You can access the Informatica Intelligent Cloud Services web site at http://www.informatica.com/cloud. This site contains information about Informatica Cloud integration services.

Informatica Intelligent Cloud Services Communities

Use the Informatica Intelligent Cloud Services Community to discuss and resolve technical issues. You can also find technical tips, documentation updates, and answers to frequently asked questions.

Access the Informatica Intelligent Cloud Services Community at:

https://network.informatica.com/community/informatica-network/products/cloud-integration

Developers can learn more and share tips at the Cloud Developer community:

https://network.informatica.com/community/informatica-network/products/cloud-integration/cloud-developers

Informatica Intelligent Cloud Services Marketplace

Visit the Informatica Marketplace to try and buy Data Integration Connectors, templates, and mapplets:

https://marketplace.informatica.com/

Data Integration connector documentation

You can access documentation for Data Integration Connectors at the Documentation Portal. To explore the Documentation Portal, visit https://docs.informatica.com.

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 Intelligent Cloud Services Trust Center

The Informatica Intelligent Cloud Services Trust Center provides information about Informatica security policies and real-time system availability.

You can access the trust center at https://www.informatica.com/trust-center.html.

Subscribe to the Informatica Intelligent Cloud Services Trust Center to receive upgrade, maintenance, and incident notifications. The Informatica Intelligent Cloud Services Status page displays the production status of all the Informatica cloud products. All maintenance updates are posted to this page, and during an outage, it will have the most current information. To ensure you are notified of updates and outages, you can subscribe to receive updates for a single component or all Informatica Intelligent Cloud Services components. Subscribing to all components is the best way to be certain you never miss an update.

To subscribe, on the Informatica Intelligent Cloud Services Status page, click **SUBSCRIBE TO UPDATES**. You can choose to receive notifications sent as emails, SMS text messages, webhooks, RSS feeds, or any combination of the four.

Informatica Global Customer Support

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

To find online support resources on the Informatica Network, click **Contact Support** in the Informatica Intelligent Cloud Services Help menu to go to the **Cloud Support** page. The **Cloud Support** page includes system status information and community discussions. Log in to Informatica Network and click **Need Help** to find additional resources and to contact Informatica Global Customer Support through email.

The telephone numbers for Informatica Global Customer Support are available from the Informatica web site at https://www.informatica.com/services-and-training/support-services/contact-us.html.

Transformations

Transformations are a part of a mapping that represent the operations that you want to perform on data. Transformations also define how data enters each transformation.

Each transformation performs a specific function. For example, a Source transformation reads data from a source, and an Expression transformation performs row-level calculations.

Active and passive transformations

A transformation can be active or passive.

An active transformation can change the number of rows that pass through the transformation. For example, the Filter transformation is active because it removes rows that do not meet the filter condition.

A passive transformation does not change the number of rows that pass through the transformation.

You can connect multiple branches to a downstream passive transformation when all transformations in the branches are passive.

You cannot connect multiple active transformations or an active and a passive transformation to the same downstream transformation or transformation input group. You might not be able to concatenate the rows. An active transformation changes the number of rows, so it might not match the number of rows from another transformation.

For example, one branch in a mapping contains an Expression transformation, which is passive, and another branch contains an Aggregator transformation, which is active. The Aggregator transformation performs aggregations on groups, such as sums, and reduces the number of rows. If you connect the branches, Data Integration cannot combine the rows from the Expression transformation with the different number of rows from the Aggregator transformation. Use a Joiner transformation to join the two branches.

Transformation types

After you add a transformation to a mapping, you can define transformation details. Each transformation type has a unique set of options that you can configure.

The following table provides a brief description of each transformation:

Transformation	Description
Source	Reads data from a source.
Target	Writes data to a target.
Access Policy	A passive transformation that applies access policies created in Data Access Management.
Aggregator	An active transformation that performs aggregate calculations on groups of data.
Chunking	An active transformation that splits large pieces of text into smaller segments, or chunks, to increase the content's relevance.
Cleanse	A passive transformation that adds a cleanse asset that you created in Data Quality to a mapping or mapplet. Use a cleanse asset to standardize the form and content of your data.
Data Masking	A passive transformation that masks sensitive data as realistic test data for nonproduction environments.
Data Services	An active transformation that invokes data services from the data services repository, including industry-standard services, such as HL7 and HIPAA, and customized services.
Deduplicate	An active transformation that adds a deduplicate asset that you created in Data Quality to a mapping or mapplet. Use a deduplicate asset to find instances of duplicate identities in a data set and optionally to consolidate the duplicates into a single record.
Expression	A passive transformation that performs calculations on individual rows of data.
Filter	An active transformation that filters data from the data flow.
Hierarchy Builder	An active transformation that converts relational input into hierarchical output.
Hierarchy Parser	A passive transformation that converts hierarchical input into relational output.
Hierarchy Processor	An active transformation that converts hierarchical input into relational output, or relational input into hierarchical output, or hierarchical output into hierarchical output of a different schema, or hierarchical input into denormalized flattened output.
Input	A passive transformation that passes data into a mapplet. Can be used in a mapplet, but not in a mapping.
Java	Executes user logic coded in Java. Can be active or passive.
Joiner	An active transformation that joins data from two sources.
Labeler	A passive transformation that adds a labeler asset that you created in Data Quality to a mapping or mapplet. Use a labeler asset to identify the types of information in an input field and to assign labels for each type to the data.

Transformation	Description
Lookup	Looks up data from a lookup object. Defines the lookup object and lookup connection. Also defines the lookup condition and the return values.
	A passive lookup transformation returns one row. An active lookup transformation returns more than one row.
Machine Learning	Runs a machine learning model and returns predictions to the mapping.
Mapplet	Inserts a mapplet into a mapping or another mapplet. A mapplet contains transformation logic that you can create and use to transform data before it is loaded into the target.
	Can be active or passive based on the transformation logic in the mapplet.
Normalizer	An active transformation that processes data with multiple-occurring fields and returns a row for each instance of the multiple-occurring data.
Output	A passive transformation that passes data from a mapplet to a downstream transformation. Can be used in a mapplet, but not in a mapping.
Parse	A passive transformation that adds a parse asset that you created in Data Quality to a mapping or mapplet. Use a parse asset to parse the words or strings in an input field into one or more discrete output fields based on the types of information that the words or strings contain.
Python	Runs Python code that defines transformation functionality. Can be active or passive.
Rank	An active transformation that limits records to a top or bottom range.
Router	An active transformation that you can use to apply a condition to incoming data.
Rule Specification	A passive transformation that adds a rule specification asset that you created in Data Quality to a mapping or mapplet. Use a rule specification asset to apply the data requirements of a business rule to a data set.
Sequence	A passive transformation that generates a sequence of values.
Sorter	A passive transformation that sorts data in ascending or descending order, according to a specified sort condition.
SQL	Calls a stored procedure or function or executes a query against a database.
	Passive when it calls a stored procedure or function. Can be active or passive when it processes a query.
Structure Parser	A passive transformation that analyzes unstructured data from a flat file source and writes the data in a structured format.
Transaction Control	An active transformation that commits or rolls back sets of rows during a mapping run.
Union	An active transformation that merges data from multiple input groups into a single output group.
Vector Embedding	A passive transformation that generates vector embeddings for input text, capturing the semantic meaning of the text in a vector format.
Velocity	A passive transformation that executes a Velocity script to convert JSON or XML hierarchal input from one format to another without flattening the data.

Transformation	Description
Verifier	A passive transformation that adds a verifier asset that you created in Data Quality to a mapping or mapplet. Use a verifier asset to verify and enhance postal address data.
Web Services	An active transformation that connects to a web service as a web service client to access, transform, or deliver data.

Transformations available in mappings

When you create a mapping, the transformation palette displays the available transformations. The transformations that are available differ depending on whether you are using advanced mode or SQL ELT mode.

When you copy a transformation into a mapping, the **Validation** panel shows validation errors if a transformation is available only in advanced mode. To resolve the errors, update the data logic in the mapping to use only the available transformations.

The following transformations are available:

Source

Target

Access Policy

Aggregator

Cleanse

Data Masking

Deduplicate

Expression

Filter

Hierarchy Builder

Hierarchy Parser

Input

Java

Joiner

Labeler

Lookup

Mapplet

Normalizer

Output

Parse

Rank

Router

Rule Specification

Sequence

Sorter

SQL

Structure Parser

Transaction Control

Union

Velocity

Web Services

Transformations available in advanced mode

In advanced mode, the transformation palette displays the transformations that enable advanced functionality. When you copy a mapping to advanced mode, you might need to update the data flow to use the available transformations.

When you copy a mapping to advanced mode, the **Validation** panel shows validation errors if a transformation is not available in advanced mode. To resolve the errors, update the data logic in the mapping to use only the transformations that are available in advanced mode. Transformations in advanced mode might behave differently.

The following transformations are available in advanced mode:

Source

Target

Aggregator

Chunking

Cleanse

Deduplicate

Expression

Filter

Hierarchy Processor

Input

Java

Joiner

Labeler

Lookup

Machine Learning

Mapplet

Normalizer

Output

Parse

Python

Rank

Router

Rule Specification

Sequence

Sorter

Structure Parser

Union

Vector Embedding

Transformations available in SQL ELT mode

In SQL ELT mode, the transformation palette displays the transformations that the target cloud data warehouse can process.

When you copy a transformation into a mapping in SQL ELT mode, the **Validation** panel shows validation errors if a transformation is not available in SQL ELT mode. To resolve the errors, update the data logic in the mapping to use only the transformations that are available in SQL ELT mode.

The following transformations are available in SQL ELT mode:

Source

Target

Aggregator

Expression

Filter

Joiner

Lookup

Normalizer

Rank

Router

Sequence

Sorter

Union

Licensed transformations

The transformations that you can use in a mapping vary based on your organization's licenses.

In the Mapping Designer, you can use the licensed transformations icon to show all transformations available in Data Integration or only those that your organization has licenses for. By default, the transformation palette shows the licensed transformations. To see all transformations, click the licensed transformations icon.

The licensed transformations icon appears at the bottom of the transformation palette as shown in the following image:
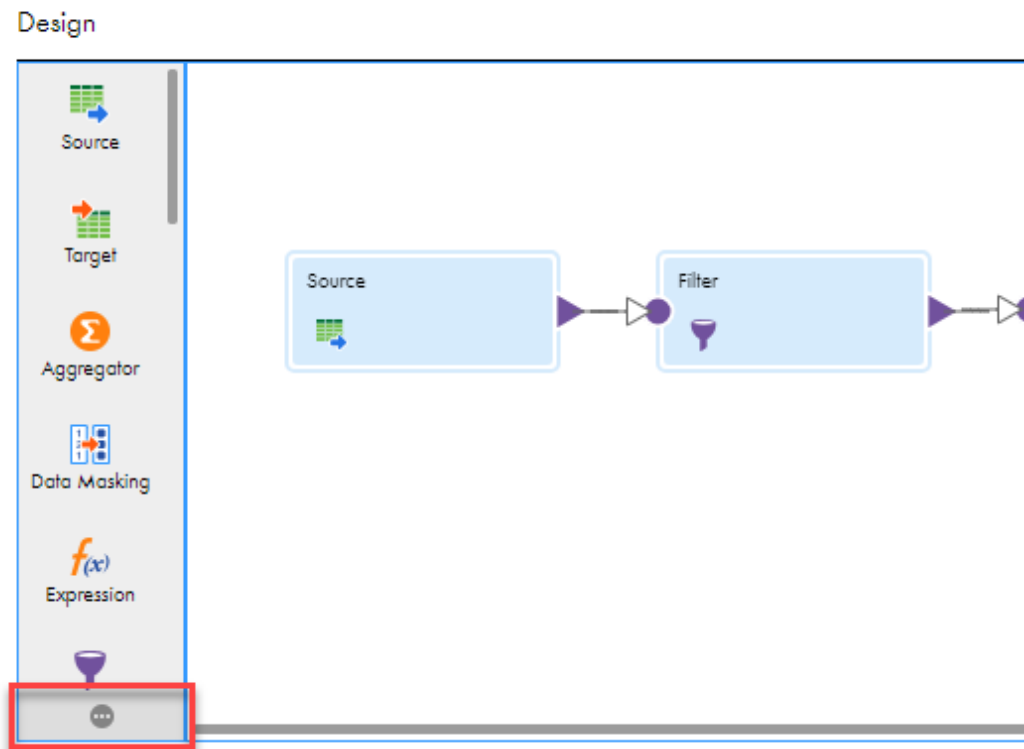
You can select from the following options:

- **Show Licensed**. The palette only displays transformations available for your organization.
- **Show All**. The palette displays all transformations available in Data Integration. Unlicensed transformations are disabled and cannot be used the mapping.

If a transformation license expires, you must renew the license to validate, run, or import any mapping or task that contains the transformation.

Incoming fields

An incoming field is a field that enters a transformation from an upstream transformation.

By default, a transformation inherits all incoming fields from an upstream transformation. However, you might want to change the default. For example, you might not need all of the fields from an upstream transformation, or you might need to rename fields from an upstream transformation.

A field rule defines how data enters a transformation from an upstream transformation. You can create field rules to specify which incoming fields to include or exclude and to rename incoming fields as required.

A field name conflict occurs when fields come from multiple transformations and have the same name. To resolve a field name conflict caused by fields from an upstream transformation, you can create a field name conflict resolution to rename incoming fields in bulk.

The following list shows the order of events that take place as fields enter and move through a transformation:

1. Field name conflict resolution rules run, if any are present.

2. Field rules run as fields from upstream transformations enter a transformation.

3. Depending on the transformation type, new fields might be added to a transformation. For example, in a Lookup transformation, fields can enter the transformation from a lookup object.

Field name conflicts

The Mapping Designer generates a field name conflict error when you validate a mapping that has fields with matching names from different transformations. When a field name conflict occurs, you need to ensure that each field has a unique name.

To resolve a field name conflict, you can create a field rule to rename fields. If you create a field rule to resolve a field name conflict, you create the field rule in the upstream transformation.

Alternatively, field name conflict error messages contain a link that you can use to create a field name conflict rule to resolve the field name conflict. A field name conflict rule renames all of the fields from the upstream transformation, not just the fields that cause a conflict.

Field name conflict rules take effect before field rules take effect. Field name conflict rules are only applicable to incoming fields from upstream transformations. Field name conflicts that occur after incoming fields first enter a transformation cannot be corrected by field name conflict rules. For example, you cannot use field name conflict rules to correct field name conflicts that occur due to field rules or activities such as lookup fields. Instead, modify the field rules or transformations that cause the conflict.

Creating a field name conflict resolution

You can resolve a field name conflict by renaming all of the fields coming from an upstream transformation in bulk using the **Resolve Field Name Conflict** dialog box, which you access from a field name conflict error message.

1. Click the link in the error message to access the **Resolve Field Name Conflict** dialog box.

2. Select the upstream transformation that contains the fields you want to rename in bulk.

3. In the **Bulk Rename Options** column, specify whether you want to rename by adding a prefix or by adding a suffix.

4. Enter the text to add to the field names, then click **OK**.

Field rules

Configure a field rule based on incoming fields from an upstream transformation. Then configure the field selection criteria and naming convention for the fields.

When you configure a field rule, you perform the following steps:

1. Choose the incoming fields that you want to include or exclude. To improve processing time and keep a clean set of data, you can include only the incoming fields that you need.

2. Configure the field selection criteria to determine which incoming fields apply to the rule. If you use the Named Fields selection criteria, you can use a parameter for the incoming fields.

3. Optionally, choose to rename the fields. To distinguish fields that come from different sources or to avoid field name conflicts, you can rename incoming fields. If you use the pattern option, you can create a parameter to rename fields in bulk.

4. Verify the order of execution. If you configure multiple rules, you can change the order in which the mapping task applies them.

Note: You cannot configure field rules on Source transformations or Mapplet transformations that contain sources.

Step 1. Choose incoming fields

When you configure a field rule, you indicate whether the rule includes or excludes the incoming fields that you specify.

The include/exclude operator works in conjunction with field selection criteria to determine which incoming fields a field rule affects.

For example, you want a transformation to exclude all binary fields. You select the exclude operator to indicate that the incoming fields that meet the field selection criteria do not pass into the current transformation. Then you specify the binary data type for the field selection criteria.

Step 2. Configure field selection criteria

When you configure a field rule, you specify the field selection criteria to determine which incoming fields apply to the field rule.

You can choose one of the following field selection criteria:

All Fields

Includes all of the incoming fields. You can rename the incoming fields in bulk when you use this option in combination with the **Includes** operator.

Named Fields

Includes or excludes the incoming fields that you specify. Use the **Named Fields** selection criteria to specify individual incoming fields to rename or to include or exclude from the incoming transformation. When you enter the field selection criteria details, you can review all of the incoming fields and select the fields to include or exclude. You can add a field that exists in the source if it does not display in the list. You can also create a parameter to represent a field to include or exclude.

Fields by Data Type

Includes or excludes incoming fields with the data types that you specify. When you enter the field selection criteria details, you can select the data types that you want to include or exclude.

Fields by Text or Pattern

Includes or excludes incoming fields by prefix, suffix, or pattern. You can use this option to select fields that you renamed earlier in the data flow. When you enter the field selection criteria details, you can select a prefix, suffix, or pattern, and define the rule to use.

When you select the prefix option or suffix option, you enter the text to use as the prefix or suffix. For example, to find all fields that start with the string, "Cust," enter `Cust` as the prefix.

When you select the pattern option, you can enter a regular expression or you can use a parameter for the pattern. The expression must use perl compatible regular expression syntax. For example, to find all fields that start with the strings "Cust" or "Addr," enter the pattern `Cust.*|Addr.*`. To find all fields that contain the string "Cust" or "CUST" anywhere in the field name, enter the pattern `.*Cust.*|.*CUST.*`. For more information about perl compatible regular expression syntax, see the help for the REG_EXTRACT function in *Function Reference*.

The following image shows the selection of the **Fields by Data Types** field selection criteria:
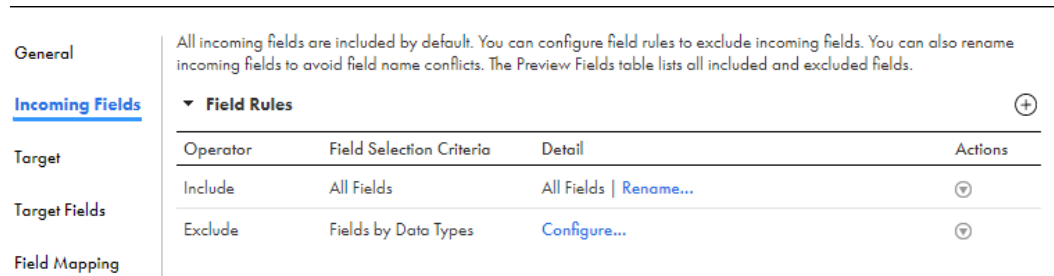
The following image shows the selection of the **date/time** data type for the field selection criteria details:
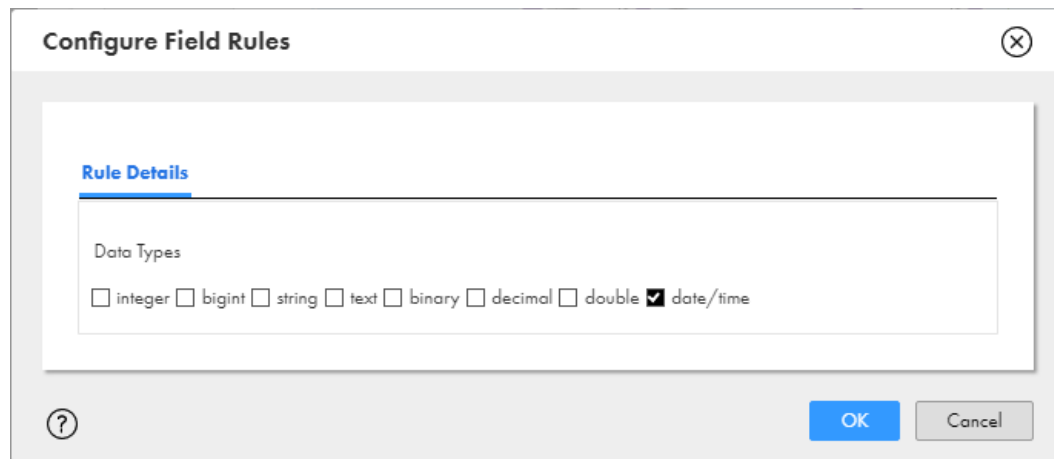
Step 3. Rename fields

Rename fields to avoid field name conflicts or to clarify field origins in complex mappings. You can rename fields as part of a field rule in a transformation. After you specify the field selection criteria for a field rule, you specify how to rename the selected fields.

You can rename fields individually or in bulk. When you rename fields individually, you select the fields you want to rename from a list of incoming fields. Then you specify the name for each of the selected fields.

When you rename in bulk, you can rename all fields by adding a prefix, suffix, or pattern. When you rename fields with a prefix or suffix, you enter the text string to use as a prefix or suffix. For example, you can specify to rename all fields as `FF_<field name>`.

When you rename fields by pattern, you enter a regular expression to represent the pattern or use a parameter to define the pattern in the task. You can create a simple expression to add a prefix or suffix to all field names or you can create an expression to replace a particular pattern with particular text.

To replace a pattern with text use a regular expression in the following syntax, where a forward slash separates the pattern to match and the text the pattern will be replaced with:

<pattern to match>/<replacement text>

The following table provides a few examples of using regular expressions when you rename fields in bulk:

Goal	Expression
Replace all occurrences of Inc with LLC.	`Inc/LLC`
Replace occurrences of Inc that occur at the end of a field name with LLC.	`Inc$/LLC`
Replace occurrences of Loc that occur at the beginning of a field name with Branch.	`^Loc/Branch`
Remove all occurrences of A/C.	`A\/C(.*)/$1` **Note:** When a character in a field name is a regular expression metacharacter, escape the character with a backslash to show that it is a literal. In this example, the forward slash is a metacharacter.
Add a prefix of FF and a suffix of _in to all fields.	`FF_$0_in`

The following image shows the **Configure Field Rules** dialog box with the Pattern bulk renaming option selected and a pattern specified to use:
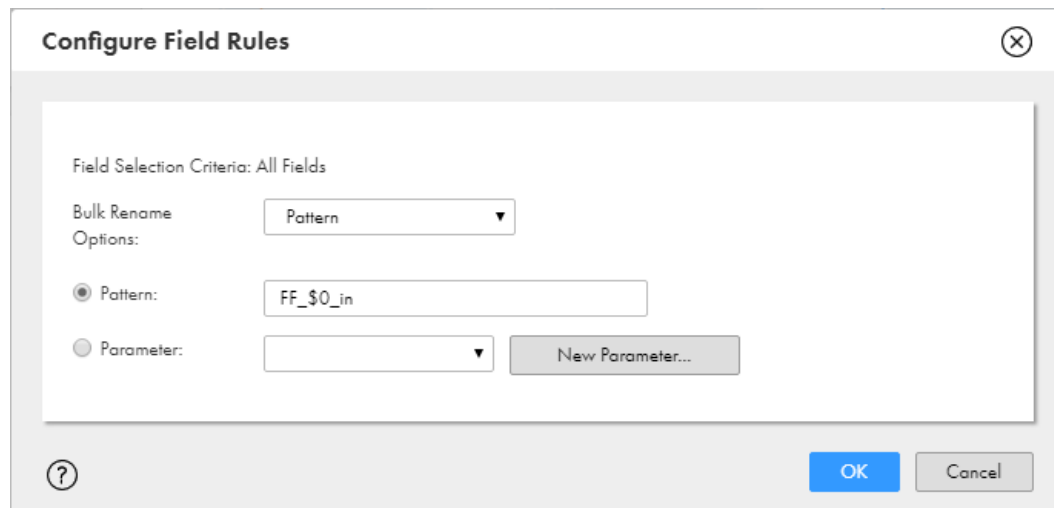
Carefully construct field renaming rules to ensure that the rules do not introduce issues such as field name conflicts. If a field renaming rule causes field name conflicts, you can edit the rule.

Tip: If the upstream transformation is a source where you cannot rename in bulk, you can add an Expression transformation to rename the fields.

Step 4. Verify order of rule execution

If you create multiple field rules, confirm that the rules run in a logical order.

To review the order in which the rules run, you can view the rules in the **Field Rules** area. The mapping task runs the rules in the order in which the rules appear. If the order of the field rules is incorrect, you can rearrange the order.

You also can preview the incoming fields for the transformation based on the rules that you have created in the Preview Fields table. The Preview Fields table lists all included and excluded fields. For example, if you

create a field rule that excludes binary fields, the **Excluded Fields** list shows the binary fields as excluded from the transformation.

If the Source transformation in the mapping uses a connection parameter or a data object parameter, the Preview Fields table does not display the transformation incoming fields.

The following image shows the Preview Fields table:

Field rule configuration examples

The following examples show how you can configure field rules in a transformation.

Changing incoming field names

You need to create a mapping to gather revenue data from multiple sales locations. You learn that multiple fields from the upstream transformation have the same names as fields in a source transformation. To avoid field name conflicts, you change the field names for all incoming fields so that the source is distinguishable throughout the mapping. To change the names of incoming fields, you create a field rule to rename all fields with the SalesForce_ prefix.

To increase performance, you want to ensure that the data set only includes required data. You determine that information regarding transaction dates is not required, so you remove the data fields from the mapping. You create a rule to exclude fields that are not required. To exclude date fields, you create a rule to exclude fields with a date/time data type.

You review the order of the rules. You want the rule to rename the fields to run after the rule to exclude the date/time fields. You move the rule to remove date/time fields so that it appears before the renaming rule.

Removing patterns in field names

You can use parentheses to group different patterns and use a reference to replace the pattern. In a regular expression, parentheses group the patterns to be matched. You can use a `($)` reference to select a group matched in the input string.

You might need to change patterns in fields between transformations. If fields in an upstream transformation contain the suffix `_out`, you can remove the suffix from the field names in the current transformation. Use the following expression to remove the `_out` suffix: `(.*)_out/$1`. The `(.*)` part of the expression matches all the field name characters, `_out` matches the suffix, and `$1` references the input string specified by the matched `(.*)` field name characters.

The following expression shows another example:

```
(.*)(_)out/$1
```

In a matched string of `string_out`, for example, the matches are `$1` for `string`, `$2` for `_`, and `$3` for `out`. The `string` value is referenced by `$1` because `string` is the first group in the match.

Creating a field rule

Configure field rules on the **Incoming Fields** tab of the **Properties** panel in the Mapping Designer.

1. On the **Incoming Fields** tab, in the **Field Rules** area, insert a row for the rule based on the order in which the rules must run. In the **Actions** column for a rule that you want to run before or after the new rule, select either **Insert above** or **Insert below**.

2. To specify whether the rule includes or excludes fields, from the **Operator** column, choose either **Include** or **Exclude**.

3. In the **Field Selection Criteria** column, choose one of the following methods:

    *   To rename all of the incoming fields in bulk, select **All Fields**.

    *   To apply the rule to the fields that you specify, select **Named Fields**.

    *   To apply the rule based on the data type of each field, select **Fields by Data Types**.

    *   To apply the rule to all fields that contain a specific prefix, suffix, or pattern, select **Fields by Text or Pattern**.

4. To provide the field selection details, in the **Detail** column, click the **Configure** or **Rename** link. The **Rename** link appears if the field selection criteria is **All Fields**.

5. In the **Configure Field Rules** dialog box, select the fields to apply to the rule, based on the chosen field selection criteria. Alternatively, click **Parameters** to add a parameter so fields can be selected in the mapping task.

6. To rename fields, click the **Rename Fields** tab and choose to rename fields individually or in bulk.

    If you want to rename all fields, you must rename in bulk. If you want to rename fields in bulk by pattern, you can create a parameter to specify the pattern in the mapping task.

7. To ensure that the field rules run in a logical order, in the **Field Rules** area, review the order in which the rules display. In the **Included Fields** and **Excluded Fields** lists, review the results of the rules. Move field rules to the appropriate location if required.

8. To delete a rule, in the **Actions** column, select **Delete**.

Data object preview

When you add a Source, Target, or Lookup transformation to a mapping and you select a single object as the source, target, or lookup object, you can preview the data.

This data preview feature is different from mapping data preview. A mapping data preview lets you see data that changed as a result of processing the mapping logic. For information on mapping data preview, see *Mappings*.

To preview the data object, open the **Source**, **Target**, or **Lookup Object** tab of the **Properties** panel, and click **Preview Data**.

When you preview data, Data Integration displays the following information:

*   The first 10 rows with the fields in native order.

*   The rows with their fields in alphabetical order if you enable the **Display source fields in alphabetical order** option.

If the source, target, or lookup object is a flat file, you can also configure the formatting options. The following table describes the formatting options for flat files:

Property	Description
Flat File Type	File type, either delimited or fixed-width.
Delimiter	Delimiter character for delimited files.
Treat multiple characters as a single delimiter	Treats the specified set of delimiters as one delimiter. For example, a source file contains the following record: abc~def|ghi~|~|jkl|~mno. If you specify the delimiter as (~ |), Data Integration reads the record as three columns separated by two delimiters: abc~def|ghi, NULL, jkl|~mno. If you disable this option, Data Integration reads the record as nine columns separated by eight delimiters: abc, def, ghi, NULL, NULL, NULL, jkl, NULL, mno.
Text Qualifier	Character to qualify text for delimited files.
Escape Character	Escape character for delimited files.
Field Labels	For delimited files, determines whether the task generates the field labels or imports them from the source file. If you import them from the source file, enter the row number that contains the field labels.
Fixed Width File Format	File format to use for fixed-width files. If the list includes multiple fixed-width file formats with the same name, use the project and folder location that's appended to the name to determine the appropriate file format to use. If there are no available fixed-width file formats, select **New** > **Components** > **Fixed-Width File Format** to create one.

Note: Other formatting options might be available based on the connection type. For more information, see the help for the appropriate connector.

Variable fields

A variable field defines calculations and stores data temporarily. You can use variable fields in the Expression and Aggregator transformations.

You might use variables to perform the following tasks:

- Temporarily store data.
- Simplify complex expressions.
- Store values from prior rows.
- Compare values.

For example, you want to generate a mailing list by concatenating first and last names, and then merging the name with the address data. To do this, you might create a variable field, FullName, that concatenates the First and Last fields. Then, you create an expression field, NameAddress, to concatenate the FullName variable field with the Address field.

Mappings don't pass the results of a variable field to the data flow. To use data from a variable field in the data flow, create an expression field for the variable field output. In the preceding example, to pass the concatenated first and last name to the data flow, create a FullName_out expression field. And then, use the FullName variable field as the expression for the field.

You can't use variables to store values from a prior row in a mapping in advanced mode. To compare the current row with a previous row for stateful computation, use a window function in the expression.

Transformation caches

Data Integration allocates cache memory for Aggregator, Joiner, Lookup, Rank, and Sorter transformations in a mapping.

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

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

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

By default, Data Integration automatically calculates the memory requirements at run time based on the maximum amount of memory that it can allocate. After you run a mapping in auto cache mode, you can tune the cache sizes for each transformation.

Cache types

Aggregator, Joiner, Lookup, and Rank transformations require an index cache and a data cache. Sorter transformations require one cache.

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

Transformation	Cache types
Aggregator	Index. Stores group values as configured in the group by fields. Data. Stores calculations based on the group by fields.
Joiner	Index. Stores all master rows in the join condition that have unique keys. Data. Stores master source rows.
Lookup	Index. Stores lookup condition information. Data. Stores lookup data that is not stored in the index cache.
Rank	Index. Stores group values as configured in the group by fields. Data. Stores row data based on the group by fields.
Sorter	Sorter. Stores sort keys and data.

Cache files

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

For Aggregator, Joiner, Lookup, and Rank transformations, Data Integration creates index and data cache files to run the transformation. For Sorter transformations, Data Integration creates one sorter cache file. By default, Data Integration stores cache files in the directory entered in the Secure Agent $PMCacheDir property for the Data Integration Server. You can change the cache directory on the **Advanced** tab of the transformation properties. If you change the cache directory, verify that the directory exists and contains enough disk space for the cache files.

Cache size

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

Auto cache

By default, a transformation cache size is set to Auto. Data Integration automatically calculates the cache memory requirements at run time. You can also define the maximum amount of memory that Data Integration can allocate in the advanced session properties when you configure the task.

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

In transformations that use a data and an index cache, Data Integration also allocates more memory to the data cache than to the index cache. It allocates all of the memory for the Sorter transformation to the sorter cache.

Specific cache size

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

You can use session logs to determine the optimal cache size. When you configure the cache size to use the value specified in the session log, you can ensure that no allocated memory is wasted. However, the optimal cache size varies based on the size of the source data. Review the mapping logs after subsequent mapping runs to monitor changes to the cache size.

To define specific cache sizes, enter the cache size values on the **Advanced** tab in the transformation properties.

Optimizing the cache size

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

1. On the **Advanced** tab of the transformation properties, set the tracing level to verbose initialization.
2. Run the task in auto cache mode.
3. Analyze the transformation statistics in the session log to determine the cache sizes required for optimal performance.

For example, you have a Joiner transformation called "Joiner." The session log contains the following text:

```
CMN_1795 [2023-01-06 16:16:59.026] The index cache size that would hold [10005]
input rows from the master for [Joiner], in memory, is [8437760] bytes
CMN_1794 [2023-01-06 16:16:59.026] The data cache size that would hold [10005] input
rows from the master for [Joiner], in memory, is [103891920] bytes
```

The log shows that the index cache size requires 8,437,760 bytes and the data cache requires 103,891,920 bytes.

4. On the **Advanced** tab of the transformation properties, enter the value in bytes that the session log recommends for the cache sizes.

Expression macros

An expression macro is a macro that you use to create repetitive or complex expressions in mappings.

You can use an expression macro to perform calculations across a set of fields or constants. For example, you might use an expression macro to replace null values in a set of fields or to label items based on a set of sales ranges.

In an expression macro, one or more input fields represent source data for the macro. An expression represents the calculations that you want to perform. And an output field represents the results of the calculations.

At run time, the task expands the expression to include all of the input fields and constants, and then writes the results to the output fields.

You can create expression macros in Expression and Aggregator transformations but you cannot combine an expression macro and an in-out parameter in an Expression transformation.

Macro types

You can create the following types of macros:

Vertical

A vertical macro expands an expression vertically. The vertical macro generates a set of similar expressions to perform the same calculation on multiple incoming fields.

Horizontal

A horizontal macro expands an expression horizontally. The horizontal macro generates one extended expression that includes a set of fields or constants.

Hybrid

A hybrid macro expands an expression both vertically and horizontally. A hybrid macro generates a set of vertical expressions that also expand horizontally.

Macro input fields

A macro input field is a field that represents input that you want to use in the expression macro. The input can be fields or constants. All expression macros require a macro input field.

A macro input field in a vertical macro represents a set of incoming fields.

A macro input field in a horizontal macro can represent a set of incoming fields or a set of constants. You can create a multiple macro input fields in a horizontal macro to define multiple sets of constants.

Macro input fields use the following naming convention: %<macro_field_name>%.

For example, you want to apply an expression to a set of address fields. You create a macro input field named %AddressFields% and define a field rule to indicate the incoming fields to use. When you configure the expression, you use %AddressFields% to represent the incoming fields.

Vertical macros

Use a vertical macro to apply a macro expression to a set of incoming fields.

The macro input field in a vertical macro represents the incoming fields. The expression represents the calculations that you want to perform on all incoming fields. And the macro output field represents a set of output fields that passes the results of the calculations to the rest of the mapping. You configure the macro expression in the macro output field.

The macro output field represents the output fields of the macro, but the names of the output fields are not explicitly defined in the mapping. To include the results of a vertical macro in the mapping, configure a field rule in the downstream transformation to include the output fields that the macro generates.

To write the results of a vertical macro to the target, link the output fields to target fields in the Target transformation.

When the task runs, the task generates multiple expressions to perform calculations on each field that the macro input field represents. The task also replaces the macro output field with actual output fields, and then uses the output fields to pass the results of the calculations to the rest of the mapping.

Note: The macro output field does not pass any data.

Example

The following vertical macro expression trims leading and trailing spaces from fields that the %Addresses% macro input field represents:

```
LTRIM(RTRIM(%Addresses%))
```

At run time, the task generates the following set of expressions to trim spaces from the fields that %Address % represents:

```
LTRIM(RTRIM(Street))
LTRIM(RTRIM(City))
LTRIM(RTRIM(State))
LTRIM(RTRIM(ZipCode))
```

Configuring a vertical macro

You can configure a vertical macro on the **Expression** tab of the Expression transformation or the **Aggregate** tab of the Aggregator transformation.

1. Create a macro input field to define the incoming fields to use.
2. Create a macro output field to define the datatype and naming convention for the output fields.
3. In the macro output field, configure the macro expression. Include the macro input field in the macro expression.
4. In the downstream transformation, configure a field rule to include the results of the macro in the mapping.

Macro input fields for vertical macros

Use a macro input field to represent the incoming fields that you want to use in a vertical macro.

When you create a macro input field, define a name for the macro input field, and then use field rules to define the incoming fields that you want to use. At run time, the macro input field expands to represent the selected fields.

You can use the following field rules when you configure a macro input field:

- All Fields
- Named Fields
- Fields by Text or Pattern

The following image shows a Named Fields field rule that includes the Q1 to Q4 fields:
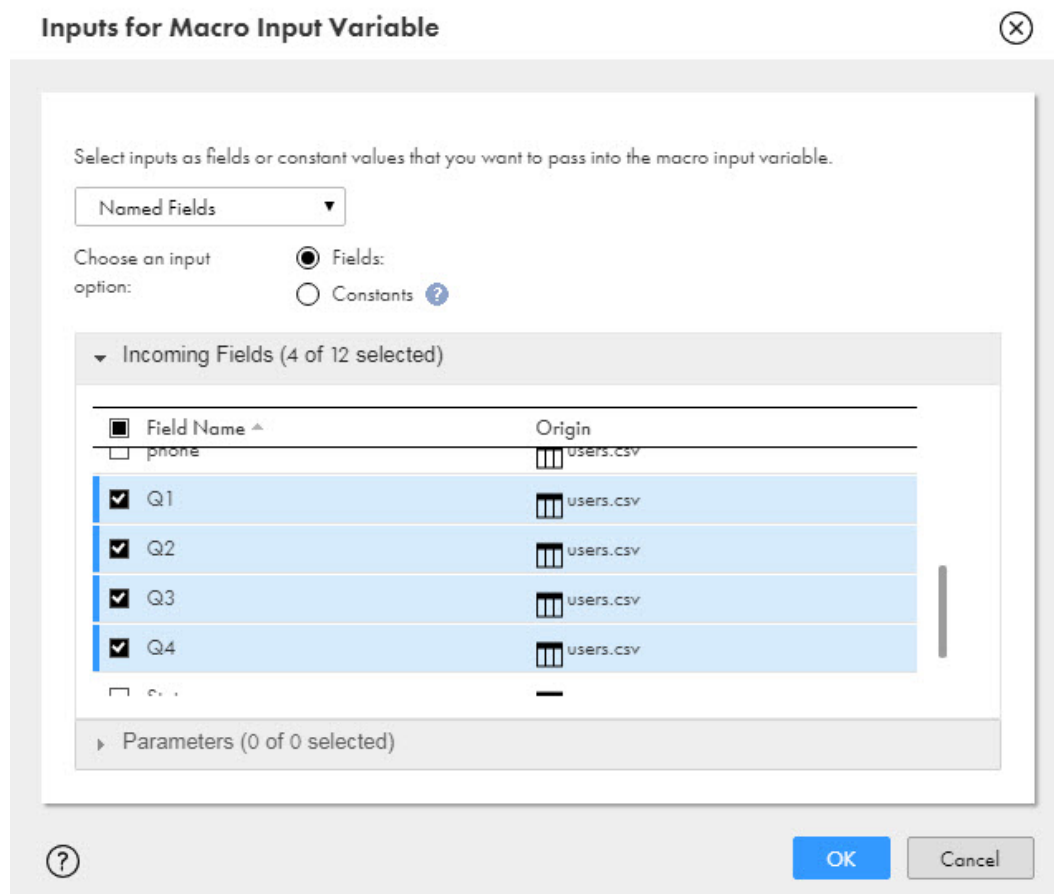
Macro output fields for vertical macros

A macro output field represents the output fields that the task generates at run time for a vertical macro. You also define the expression that you want to use in the macro output field.

When you configure a macro output field, you select the macro input field to use and define a naming convention for the output fields. You can customize a prefix or suffix for the naming convention. By default, the macro output field uses the following naming convention for output fields: <macro_input_field>_out.

You can define the data type, precision, and scale of the output fields. Or, you can configure the macro output field to use the datatype, precision, and scale of the incoming fields. Use the datatype of incoming fields

when the incoming fields include more than one datatype and when the expression does not change the datatype of incoming data.

At run time, the task generates output fields based on the macro output field configuration. The task creates an output field for each incoming field that the macro input field represents, and then writes the results of the expression to the output fields.

For example, the following image shows a macro output field that creates output fields based on the incoming fields that %QuarterlyData% represents:

If the %QuarterlyData% macro input field represents the Q1 to Q4 fields, then the task creates the following output fields at run time: Q1_out, Q2_out, Q3_out, Q4_out. The output fields have the same datatype as the incoming fields.

Note that you cannot define the precision and scale after you select the Input Field Type datatype.

Field rules for vertical macro output fields

To use the results of a vertical macro in a mapping, configure a field rule to include the output fields in the downstream transformation or parameterize the target field mapping.

Because an expression macro represents fields that are not explicitly defined until run time, you need to configure the downstream transformation to include the output fields of a vertical macro. There are two ways to do this:

- Create named fields in the downstream transformation. On the **Incoming Fields** tab, create a named field rule and create a new incoming field for each output field of the vertical macro. You can use these fields in downstream transformations.
- Alternatively, if your Target transformation is directly downstream from the macro, completely parameterize the target field mapping. When you configure the mapping task, Data Integration creates the macro output fields in the target. Map the incoming fields to the target fields.

Example

A macro input field named %InputDates% represents the following source fields for a macro that converts the data to the Date data type:

```
OrderDate
ShipDate
PaymentReceived
```

The macro output field uses the default naming convention: <macro input field>_out. To use the Date fields that the macro generates, create a Named Field rule in the downstream transformation. Create the following fields:

```
OrderDate_out
ShipDate_out
PaymentReceived_out
```

Configure the field rule to include the fields that you create.

After you create the field rule, you can use the fields in expressions and field mappings in downstream transformations.

Vertical macro example

To find the annual sum of quarterly data for each store, you might use a vertical expression macro in an Aggregator transformation.

The Aggregator transformation uses the store ID field as the group by field. A %QuarterlyData% macro input field reads sales data from the following input fields: Q1, Q2, Q3, and Q4.

A %QuarterlyData%_out macro output field is based on the %QuarterlyData% macro input field. To find the sum of sales for each quarter, the macro output field includes the following expression: `SUM(%QuarterlyData %)`.

In the Target transformation, a field rule includes the following output fields in the incoming fields list: Q1_out, Q2_out, Q3_out, Q4_out. In the target field mapping, the Qx_out fields are mapped to the target.

The following image shows the vertical expression macro in an Aggregator transformation:

When the task runs, the expression expands vertically, as follows:

```
SUM(Q1)
SUM(Q2)
```

```
SUM(Q3)
SUM(Q4)
```

The task groups the data by store when it performs the aggregation and writes the results to the target.

Horizontal macros

Use a horizontal macro to generate a single complex expression that includes a set of incoming fields or a set of constants.

In a horizontal macro, a macro input field can represent a set of incoming fields or a set of constants.

In a horizontal macro, the expression represents calculations that you want to perform with the incoming fields or constants. The expression must include a horizontal expansion function.

A horizontal macro expression produces one result, so a transformation output field passes the results to the rest of the mapping. You configure the horizontal macro expression in the transformation output field.

The results of the expression pass to the downstream transformation with the default field rule. You do not need additional field rules to include the results of a horizontal macro in the mapping.

To write the results of a horizontal macro to the target, connect the transformation output field to a target field in the Target transformation.

Example

For example, a horizontal macro can check for null values in the fields represented by the %AllFields% macro input field. When a field is null, it returns 1. And then, the %OPR_SUM% horizontal expansion function returns the total number of null fields.

The following expression represents the calculations in the macro:

```
%OPR_SUM[ IIF(ISNULL(%AllFields%), 1, 0) ]%
```

At run time, the application expands the expression horizontally as follows to include the fields that %AllFields% represents:

```
IIF(ISNULL (AccountID, 1,0)+IIF(ISNULL(AccountName, 1, 0)+IIF(ISNULL(ContactName, 1,
0)+IIF(ISNULL(Phone, 1, 0)+IIF(ISNULL(Email, 1, 0)...
```

Horizontal expansion functions

Use a horizontal expansion function to create an expression in an expression macro.

Horizontal expansion functions use the following naming convention: %OPR_<function_type>%. Horizontal expansion functions use square brackets ([ ]) instead of parentheses.

In the Field Expression dialog box, the functions appear in the Horizontal Expansion group of the functions list.

You can use the following horizontal expansion functions:
%OPR_CONCAT%

Uses the CONCAT function and expands an expression in an expression macro to concatenate multiple fields. %OPR_CONCAT% creates calculations similar to the following expression:

```
FieldA || FieldB || FieldC...
```

%OPR_CONCATDELIM%

Uses the CONCAT function and expands an expression in an expression macro to concatenate multiple fields, and adds a comma delimiter. %OPR_CONCATDELIM% creates calculations similar to the following expression:

```
FieldA || ", " || FieldB || ", " || FieldC...
```

%OPR_IIF%

Uses the IIF function and expands an expression in an expression macro to evaluate a set of IIF statements. %OPR_IIF% creates calculations similar to the following expression:

```
IIF(<field> >= <constantA>, <constant1>,
  IIF(<field> >= <constantB>, <constant2>,
    IIF(<field> >= <constantC>, <constant3>, 'out of range')))
```

%OPR_SUM%

Uses the SUM function and expands an expression in an expression macro to return the sum of all fields. %OPR_SUM% creates calculations similar to the following expression:

```
FieldA + FieldB + FieldC...
```

For more information about horizontal expansion functions, see *Function Reference*.

Configuring a horizontal macro

You can configure a horizontal macro on the **Expression** tab of the Expression transformation or the **Aggregate** tab of the Aggregator transformation.

Configure a horizontal macro based on whether you want to use incoming fields or constants in the macro expression.

1. Create one or more macro input fields:

    - To use incoming fields, create a macro input field to define the incoming fields to use.

    - To use constants, create a macro input field for each set of constants that you want to use.

2. Create a transformation output field.

3. In the transformation output field, configure the macro expression. Use a horizontal expansion function and include the macro input fields.

4. To include the results of a horizontal macro in the mapping, use the default field rule in the downstream transformation. You can use any field rule that includes the transformation output field.

5. To write the results of a horizontal macro to the target, connect the transformation output field to a target field in the Target transformation.

Macro input fields for incoming fields in horizontal macros

You can use a macro input field to represent the incoming fields that you want to use in a horizontal macro.

When you create a macro input field, define a name for the macro input field, and then use field rules to define the incoming fields that you want to use. At run time, the macro input field expands to represent the selected fields.

You can use the following field rules when you configure a macro input field:

- All Fields

- Named Fields

- Fields by Text or Pattern

The following image shows a Named Fields field rule that includes the Q1 to Q4 fields:
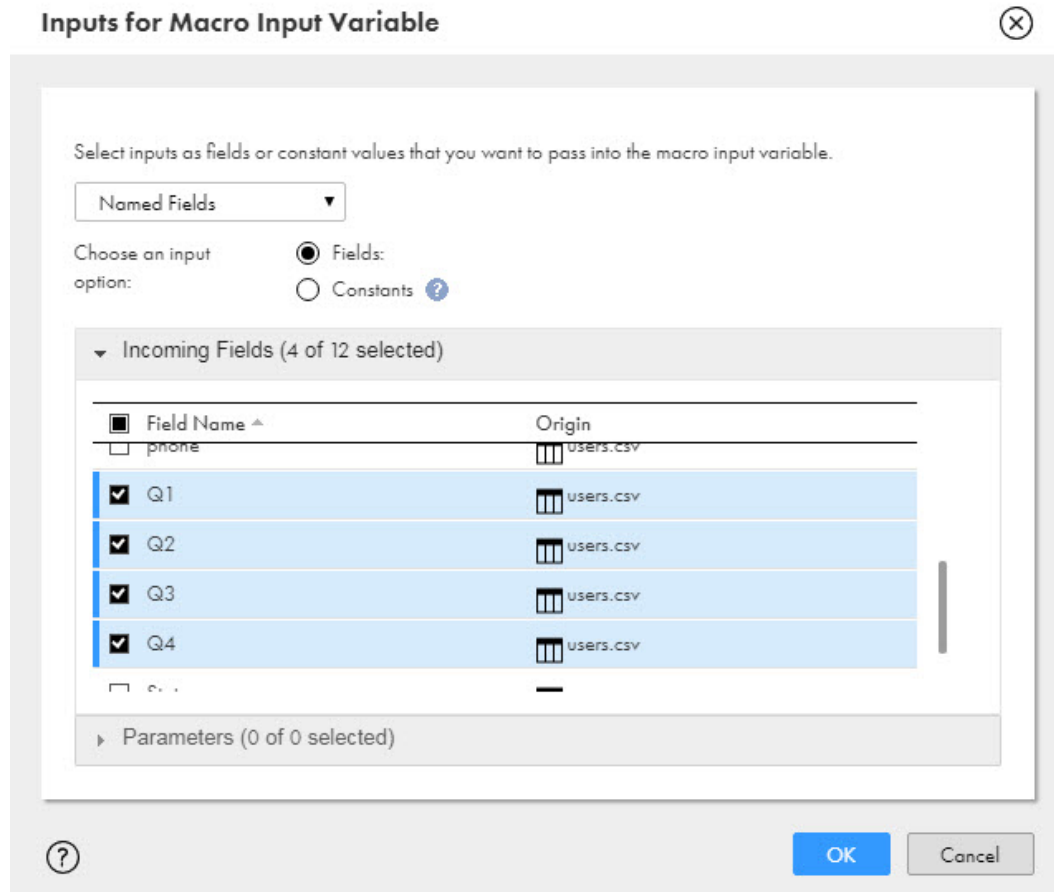
Macro input fields for constants in horizontal macros

You can use a macro input field to represent the constants that you want to use in a horizontal macro. You can also create multiple macro input fields to represent corresponding sets of constants.

When you create a macro input field, define a name for the macro input field, and then define the constants that you want to use. At run time, the macro input field expands to represent the constants and uses them in the listed order.

When you create multiple macro input fields with corresponding sets of constants, the task evaluates each set of constants in the listed order.

The following image shows a macro input field that represents constants:

Inputs for Macro Input Variable   ⊗

Select inputs as fields or constant values that you want to pass into the macro input variable.

All Incoming Fields   ▼

Choose an input   ○ Fields:
option: ◉ Constants ❓

Constants   ➕

Value

50000

100000

150000

                                                    OK   Cancel

At run time, the macro input field expands and uses the constants in the following order: 50000, 100000, 150000.

Transformation output field configuration for horizontal macros

Use a transformation output field to define the expression for a horizontal macro and to pass the results to the rest of the mapping.

When you create a transformation output field, you define the name and datatype for the field. You also configure the expression for the macro. In the expression, include a horizontal expansion function and any macro input fields that you want to use.

The default field rule passes the transformation output field to the downstream transformation. You can use any field rule that includes the transformation output field to pass the results of a horizontal macro to the mapping.

Horizontal macro example

To create categories for employees based on salary ranges, you might create a horizontal macro that defines the minimum and maximum values for each range and corresponding job categories.

In an Expression transformation, macro input fields define the constants to use in the expression. %IncomeMin% defines the low end of each salary range and %IncomeMax% defines the high end of each salary range. %EmployeeType% lists the job category that corresponds to each range.

The EmployeeStatus transformation output field passes the results to the mapping and includes the following horizontal macro expression:

```
%OPR_IIF[ (EMP_SALARY>=%IncomeMin%) AND (EMP_SALARY<%IncomeMax%), %EmployeeType%,
'unknown' ]%
```

In the Target transformation, the default field rule includes the EmployeeStatus transformation output field in the incoming fields list. In the target field mapping, the EmployeeStatus is mapped to the target.

The following image shows the horizontal macro in an Expression transformation:
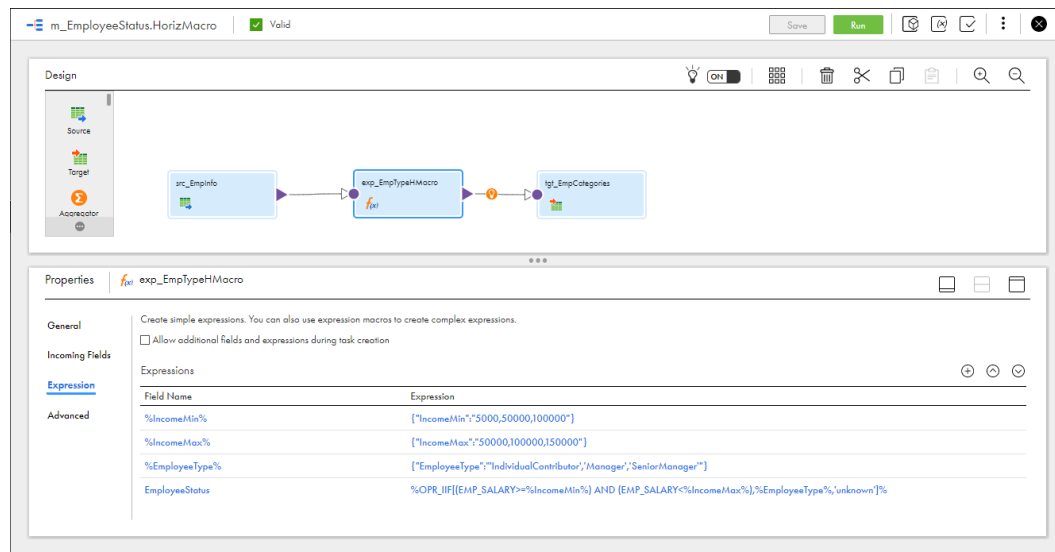
The horizontal macro expression expands as follows when you run the task:

```
IIF(Salary>=5000 AND Salary<50000), 'IndividualContributor',
  IIF (Salary>=50000 AND Salary<100000), 'Manager',
    IIF (Salary>=100000 AND Salary<150000), 'SeniorManager', 'unknown')))
```

Note that the expression uses the first value of each macro input field in the first IIF expression and continues with each subsequent set of constants.

Hybrid macros

A hybrid macro expands an expression both vertically and horizontally. A hybrid macro generates a set of vertical expressions that also expand horizontally.

Configure a hybrid macro based on your business requirements. Use the configuration guidelines for vertical and horizontal macros to create a hybrid macro.

Example

For example, the following expression uses the %OPR_IIF% horizontal expansion function to convert the format of the date fields represented by the %dateports% macro input field to the 'mm-dd-yyyy' format:

```
%OPR_IIF[IS_DATE(%dateports%,%fromdateformat%),TO_STRING(TO_DATE(%dateports
%,%fromdateformat%),'mm-dd-yyyy'),%dateports%]%
```

The %fromdateformat% macro input field defines the different date formats used in the date fields: mm/dd/yy and mm/dd/yyyy.

At run time, the application expands the expression vertically and horizontally, as follows:

```
IIF(IS_DATE(StartDate,'mm/dd/yy'),TO_STRING(TO_DATE(StartDate,'mm/dd/yy'),'mm-dd-
yyyy'),
```

```
    IIF(IS_DATE(StartDate,'mm/dd/yyyy'),TO_STRING(TO_DATE(StartDate,'mm/dd/yyyy'),'mm-dd-
yyyy'), StartDate))

    IIF(IS_DATE(EndDate,'mm/dd/yy'),TO_STRING(To_Date(EndDate,'mm/dd/yy'),'mm-dd-yyyy'),
    IIF(IS_DATE(END _DT,'mm/dd/yyyy'),TO_STRING(TO_DATE(EndDate,'mm/dd/yyyy'),'mm-dd-
yyyy'), EndDate))
```

The expression expands vertically to create an expression for the StartDate and EndDate fields that %dateports% represents. The expression also expands horizontally to use the constants that %fromdateformat% represents to evaluate the incoming fields.

File lists

You can use a file list as a source for flat file connections. A file list is a file that contains the names and directories of each source file that you want to use in a mapping. Use a file list to enable a task to read multiple source files for one source object in a mapping.

For example, you might want to use a file list if your organization collects data for multiple locations that you want to process through the same mapping.

You configure a source object such as a Source transformation or Lookup transformation to read the file list. You can also write the source file name to each target row. When you run a mapping that uses a file list, the task reads rows of data from the different source files in the file list.

Use the following rules and guidelines when you create a file list:

- Each file in the list must use the user-defined code page that is configured in the connection.
- Each file in the list must share the same file properties as configured in the connection.
- If you do not specify a path for a file, the task assumes that the file is in the same directory as the file list.
- Each path must be local to the Secure Agent.

You can create a file list manually or you can use a command to create the file list.

Manually created file lists

To create a file list manually, create the list in a text editor and save it as a text file. Place the text file in a directory that is local to the Secure Agent.

When you create the file list, enter one file name or one file path and file name on each line. Data Integration extracts the field names from the first file in the file list.

The following example shows a valid Windows file list:

```
customers_canada.dat
C:\Customer_Accounts\customers_us.dat
C:\Customer_Accounts\customers_uk.dat
C:\Customer_Accounts\customers_india.dat
```

In the previous example, Data Integration extracts the field names from the customers_canada.dat file which resides in the same folder as the file list.

File list commands

You can use a command to generate a list of source files for a mapping. You can use a valid DOS or UNIX command, batch file, or shell script. Data Integration reads each file in the list when the task runs.

Use a command to generate a file list when the list of source files changes often or when you want to generate a file list based on specific conditions. For example, you can use a command to generate a file list from all files in a directory or based on the file names.

Use the following guidelines when you generate a file list through a command:

- You must enter Windows commands that use parameters such as "/b" in a batch file.
- You must enter fully qualified file paths in each command, batch file, and shell script.
- You cannot use an in-out parameter for the file list command.

UNIX Example with Shell Script

You need to extract data from parts lists that are stored on a Linux machine. The parts lists are text files that are stored in the `/home/dsmith/flatfile/parts` directory.

The following table shows the command that you enter in the Source transformation and the contents of the corresponding shell script:

Command	Shell Script (parts.sh)
`/home/dsmith/flatfile/parts/parts.sh`	`cd /home/dsmith/flatfile/parts` `ls *.txt`

Windows Example with Batch File

You need to extract data from sales records that are stored on a Windows machine. The sales record files are stored in the `C:\SalesRecords` directory and use the naming convention `SalesRec_??-??-2017.txt`.

The following table shows the command that you enter in the Source transformation and the corresponding batch file contents:

Command	Batch File (SalesSrc.bat)

`C:\SalesRecords\SalesSrc.bat`	`@echo off`
	`cd C:\SalesRecords`
	`dir /b SalesRec_??-??-2017.txt`

Example without Shell Script or Batch File

You can also generate a file list through a command instead of through a batch file or shell script. For example, the following command generates a file list that contains one file named source.csv:

```
echo C:\sources\source.csv
```

Command sample file

When you generate a file list through a command, you select a sample file that Data Integration uses to extract the field names. Data Integration does not extract data from the sample file unless the sample file is included in the generated file list.

If a file in the generated file list does not contain all fields in the sample file, Data Integration sets the record values for the missing fields to null. If a file in the file list contains fields that are not in the sample file, Data Integration ignores the extra fields.

For example, the sample file that you select contains the fields CustID, NameLast, and NameFirst. One file in the generated file list does not contain the NameFirst field. When Data Integration reads data from the file, it sets the first names for each record in the file to null.

Another file in the generated file list contains the fields CustID, NameLast, NameFirst, and PhoneNo. Data Integration does not import records for the PhoneNo field because the field is not in the sample file. If you want to import the phone numbers, either select a sample file that contains the PhoneNo field, or add a field for the phone numbers in the transformation.

Using a file list in a Source transformation

To use a file list in a Source transformation, create the text file, batch file, or shell script that creates the file list. Then configure the Source transformation to use the file list.

1. Create the text file, batch file, or shell script that creates the file list and install it locally to the Secure Agent.
2. In the Mapping Designer, select the Source transformation.
3. On the **Source** tab, select a flat file connection.
4. To use a manually created file list, perform the following steps:
   a. In the **Source Type** field, select **File List**.
   b. In the **Object** field, select the text file that contains the file list.
   c. On the **Fields** tab, verify the incoming fields for the Source transformation.

      Data Integrationextracts source fields from the first file in the file list. If the source fields are not correct, you can add or remove fields.
5. To use a file list that is generated from a command, perform the following steps:
   a. In the **Source Type** field, select **Command**.
   b. In the **Sample Object** field, select the sample file from which Data Integration extracts source fields.

      You can use one of the files you use to generate the file list as the sample file or select a different file.
   c. In the **Command** field, enter the command that you use to generate the file list, for example, `/home/dsmith/flatfile/parts/parts.sh`.
   d. On the **Fields** tab, verify the incoming fields for the Source transformation.

      If the source fields are not correct, you can add or remove fields, or click the **Source** tab and select a different sample file.
6. Optionally, to write the source file name to each target row, click the **Fields** tab, and enable the **Add Currently Processed Filename field** option.

   The CurrentlyProcessedFileName field is added to the fields table.

Using a file list in a Lookup transformation

To use a file list in a Lookup transformation, create the text file, batch file, or shell script that creates the file list. Then configure the Lookup transformation to use the file list.

1. Create the text file, batch file, or shell script that creates the file list and install it locally to the Secure Agent.
2. In the Mapping Designer, select the Lookup transformation.
3. On the **Lookup Object** tab, select a flat file connection.
4. To use a manually created file list, perform the following steps:
    a. In the **Source Type** field, select **File List**.
    b. In the **Lookup Object** field, select the text file that contains the file list.
    c. On the **Return Fields** tab, verify the return fields for the Lookup transformation.

        Data Integration extracts the return fields from the first file in the file list. If the return fields are not correct, you can add or remove fields.
5. To use a file list that is generated from a command, perform the following steps:
    a. In the **Source Type** field, select **Command**.
    b. In the **Lookup Object** field, select the sample file from which Data Integration extracts return fields.

        You can use one of the files you use to generate the file list as the sample file or select a different file.
    c. In the **Command** field, enter the command that you use to generate the file list, for example, `/home/dsmith/flatfile/parts/parts.sh`.
    d. On the **Return Fields** tab, verify the return fields for the Lookup transformation.

        If the return fields are not correct, you can add or remove fields, or click the **Lookup Object** tab and select a different sample file.

Configuration for multibyte hierarchical data

If a mapping includes a transformation that processes hierarchical data and the data uses multibyte characters, configure the Secure Agent machine to use UTF-8.

On Windows, create the INFA_CODEPAGENAME=UTF-8 environment variable in Windows System Properties.

On Linux, set the LC_LOCALE variable to UTF-8.

CHAPTER 2

Source transformation

A Source transformation extracts data from a source. When you add a Source transformation to a mapping, you define the source connection, source objects, and source properties related to the connection type. For some connection types, you can use multiple source objects within a Source transformation.

You can use a Source transformation to read data from the following source types:

- File. The Source transformation can read data from a single source file or a file list.
- Database. The Source transformation can read data from a single source table or multiple source tables.
- Web service. The Source transformation can read data from a single web service operation.
- Data Integration connectors. The Source transformation can read data from a single source object, a multi-group source object, or multiple source objects based on the connection type.

    For more information about sources for individual connectors, see the **Connectors** section of the online help.

You can use one or more Source transformations in a mapping. If you use two Source transformations in a mapping, you can use a Joiner transformation to join the data. If you use multiple Source transformations with the same structure, you can use a Union transformation to merge the data into a single pipeline.

In a Source transformation, the source properties that appear vary based on the connection type. For example, when you select a Salesforce connection, you can use multiple related source objects and configure the Salesforce API advanced source property. In contrast, when you select a flat file connection, you specify the file type and configure the file formatting options.

Source object

Select the source object for the Source transformation on the **Source** tab of the Properties panel.

The properties that you configure for the source object vary based on the connection type and the mapping type. Your organization's licenses can also determine the source properties that appear when the Source transformation is part of a mapplet.

The following image shows the **Source** tab for a relational source:
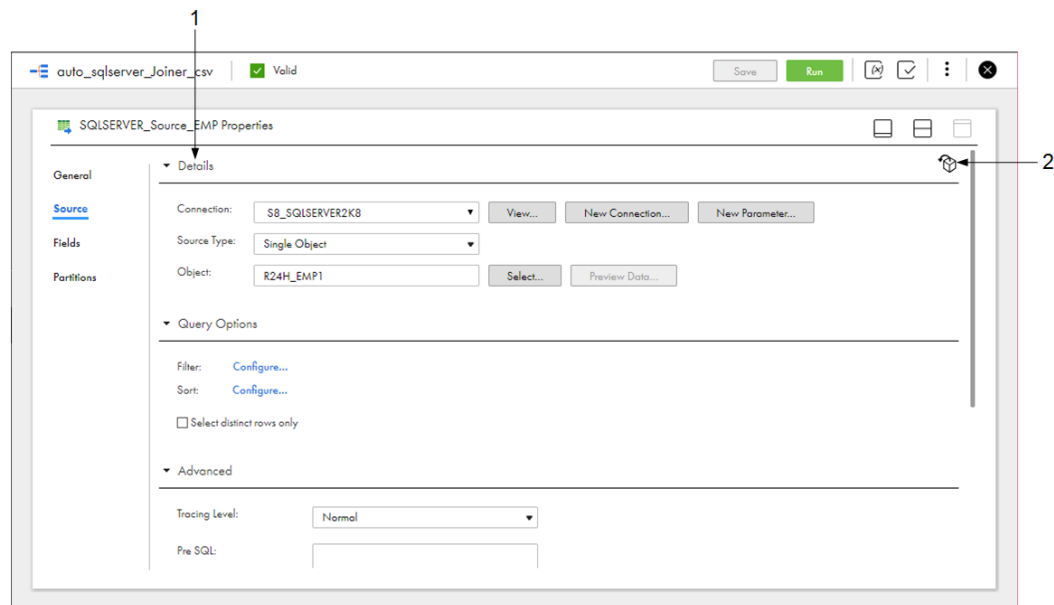
1. Source details where you configure the source connection, source type, and source object.
2. Select the source object from the mapping inventory.

You can select a source in the following ways:

Select the connection and source object.

In the **Details** area, select the source connection, source type, and source object. For some source types, you can select multiple source objects. You can also create a new connection.

The source type varies based on the connection type. For example, for relational sources you can select a single object, multiple related objects, or an SQL query. For flat file sources, you can select a single object, file list, or file list command.

Select the source object from the mapping inventory.

If your organization administrator has configured Enterprise Data Catalog integration properties, and you have added objects to the mapping from the **Data Catalog** page, you can select the source object from the **Inventory** panel. If your organization administrator has not configured Enterprise Data Catalog integration properties or you have not performed data catalog discovery, the **Inventory** panel is empty. For more information about data catalog discovery, see *Mappings*.

The **Inventory** panel isn't available in mappings in SQL ELT mode.

Use a parameter.

You can use input parameters to define the source connection and source object when you run the mapping task. For more information about parameters, see *Mappings*.

File sources

File sources include flat files and FTP/SFTP files. When you configure a file source, you specify the connection, source type, and formatting options. Configure file source properties on the **Source** tab of the **Properties** panel.

The following table describes the file source properties:

Property	Description
Connection	Name of the source connection.
Source Type	Source type. The source type can be single object, file list, command, or parameter.
Object	If the source type is a single object, this property specifies the file source, for example, Customers.csv. If the source property is a file list, this property specifies the text file that contains the file list, for example, SourceList.txt. If the source type is a command, this property specifies the sample file from which Data Integration imports the source fields. In advanced mode, the object name cannot contain the dollar sign character, $. The dollar sign is a reserved character for parameters.
Command	If the source type is a command, this property specifies the command that generates the source file list, for example, ItemSourcesCmd.bat.

Property	Description
Parameter	If the source type is a parameter, this property specifies the source parameter.
Formatting Options	Flat file format options. Opens the **Formatting Options** dialog box to define the format of the file. You can choose either a delimited or fixed-width file type. Default is delimited. For a delimited flat file type, configure the following file format options: - Delimiter. Delimiter character. Can be a comma, tab character, colon, semicolon, nonprintable control character, or a single-byte or multibyte character that you specify. - Treat multiple characters as a single delimiter. Treats the specified set of delimiters as one delimiter. For example, a source file contains the following record: abc~def|ghi~|~|jkl|~mno. If you specify the delimiter as (~ |), Data Integration reads the record as three columns separated by two delimiters: abc~def|ghi, NULL, jkl|~mno. If you disable this option, Data Integration reads the record as nine columns separated by eight delimiters: abc, def, ghi, NULL, NULL, NULL, jkl, NULL, mno. - Treat consecutive delimiters as one. Treats one or more consecutive column delimiters as one. The default is to treat consecutive delimiters as a null value. - Row Delimiter. Line break character for data rows. Select a line break character from the list. The default is line-feed, \012 LF (\n). - Text Qualifier. Character to qualify text. - Escape character. Escape character. - Field labels. Determines if the task generates field names or imports them from the source file. By default, the transformation generates field names using the first line of the file. When you select **Use first data row to generate header**, Source or Lookup transformation field names are generated using the specified **First Data Row**. When the line specified as the **First Data Row** is empty, the next nonempty line is used to generate field names. - First data row. The first line of data. The task starts the read at the row number that you enter. You can use a tab, space, or any printable special character as a delimiter. The delimiter can have a maximum of 10 characters. The delimiter must be different from the escape character and text qualifier. For a fixed-width flat file type, select the fixed-width file format to use. If the list includes multiple fixed-width file formats with the same name, use the project and folder location that's appended to the name to determine the appropriate file format to use. If you do not have a fixed-width file format, go to **New** > **Components** > **Fixed-Width File Format** to create one.

For more information about file lists and commands, see "File lists" on page 45. For more information about parameters and file formats, see *Mappings*.

The following table lists the advanced properties for file sources:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Thousand Separator	Thousand separator character. Can be none, comma, or period. Cannot be the same as the decimal separator or the delimiter character. Field type must be Number. You might also need to update the field precision and scale. Default is None.
Decimal Separator	Decimal character. Can be a comma or period. Cannot be the same as the thousand separator or delimiter character. Field type must be Number. You might also need to update the field precision and scale. Default is Period.

Property	Description
Datetime Format	Optional. Overrides the date format specified in the flat file connection. Enter the Date/Time format to override. For example, YYYYMMDD.
Source File Directory	For flat file sources, name of the source directory. For FTP sources, name and path of the local source file directory used to stage the source data. By default, the mapping task reads source files from the source connection directory. You can also use an input parameter to specify the file directory. If you use the service process variable directory $PMSourceFileDir, the task writes target files to the configured path for the system variable. To find the configured path of a system variable, see the pmrdtm.cfg file located at the following directory: `<Secure Agent installation directory>\apps\Data_Integration_Server\<Data Integration Server version>\ICS\main\bin\rdtm` You can also find the configured path for the $PMSourceFileDir variable in the Data Integration Server system configuration details in Administrator.
Source File Name	For flat file sources, file name, or file name and path of the source file. You can also use an input parameter to specify the file name. For FTP sources, name of the local source file used to stage the source data.
Remote File Name	For FTP sources, file name, or file name and path of the remote file. You can also use an input parameter to specify the remote file name.
File Reader Truncate String Null	For flat file sources, truncates string field values from the first null character. Enable when the source file contains null characters. Do not enable when you use the FileRdrTreatNullCharAs custom property. Using both properties creates conflicting settings for how Data Integration handles null characters in a flat file source, and the task fails. Default is disabled.

Database sources

Database sources include relational sources such as Oracle, MySQL, and Microsoft SQL Server. When you configure a Source transformation for a database source, you can use a single source table or multiple source tables. If you use multiple tables as a source, you can select related tables or create a relationship between tables.

To configure a Source transformation for a database source, perform the following tasks:

- Configure the source properties.
- If the source includes multiple tables, configure the Source transformation to join the tables. You can join related tables or specify a custom relationship.
- Optionally, configure a custom SQL query to extract source data.
- Optionally, configure the Source transformation to filter or sort source data.
- Ensure that the table and column names do not exceed 74 characters.

Database source properties

Configure properties for database sources such as the database connection, source type, and source objects. You can also specify filter and sort conditions, pre- and post-SQL commands, and whether the output is deterministic or repeatable.

The following table describes the database source properties:

Property	Description
Connection	Name of the source connection.
Source Type	Source type.
Object	Source object for a single source.
Add Source Object	Primary source object for multiple sources.
Add Related Objects	For multiple sources. Displays objects related to the selected source object. Select an object with an existing relationship or click **Custom Relationship** to create a custom relationship with another object.
Filter	Adds conditions to filter records. Configure a simple or an advanced filter.
Sort	Adds conditions to sort records.
Select Distinct Rows Only	Reads unique rows from the source. Adds SELECT DISTINCT to the SQL query.
Define Query	For a custom query. Displays the **Edit Custom Query** dialog box. Enter a valid custom query and click **OK**.
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Pre SQL	SQL command to run against the source before reading data from the source. You can enter a command of up to 5000 characters.
Post SQL	SQL command to run against the source after writing data to the target. You can enter a command of up to 5000 characters.
SQL Query	SQL query to override the default query that Data Integration uses to read data from the source. You can enter an SQL statement supported by the source database.
Output is deterministic	Relational source or transformation output that does not change between mapping runs when the input data is consistent between runs. When you configure this property, the Secure Agent does not stage source data for recovery if transformations in the pipeline always produce repeatable data.
Output is repeatable	Relational source or transformation output that is in the same order between session runs when the order of the input data is consistent. When output is deterministic and output is repeatable, the Secure Agent does not stage source data for recovery.

Related objects

You can configure a Source transformation to join related objects. You can join related objects based on existing relationships or custom relationships. The types of relationships that you can create are based on the connection type.

Use the following relationships to join related objects:

Existing relationships

You can use relationships defined in the source system to join related objects. You can join objects with existing relationships for the following connection types:

- Database
- Salesforce
- Some Data Integration connectors

To join related objects, you select a primary object. Then you select a related object from a list of related objects.

For example, after you add Opportunity as a primary Salesforce source object, you can add any related objects, such as Account.

The following image shows a list of Salesforce objects with existing relationships with the Opportunity object:
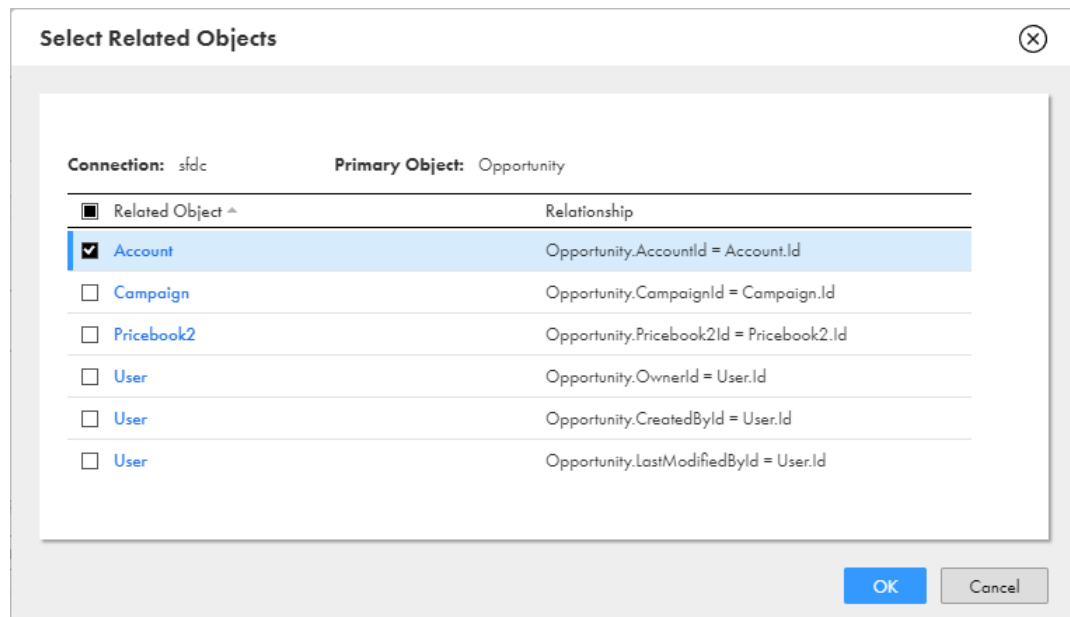
Custom relationships

You can create custom relationships to join objects in the same source system. To create a custom relationship, select a primary object, select another object from the source system, and then select a field from each source to use in the join condition. You must also specify the join type and join operator.

You can select one of the following join types:

Inner

> Performs a normal join. Includes rows with matching join conditions. Discards all rows that do not match, based on the condition.

Left

> Performs a left outer join. Includes all rows for the source to the left of the join syntax and the rows from both tables that meet the join condition. Discards the unmatched rows from the right source.

Right

> Performs a right outer join. Includes all rows for the source to the right of the join syntax and the rows from both tables that meet the join condition. Discards the unmatched rows from the left source.

For example, the following image shows a custom relationship that uses an inner join to join the EMPLOYEE and MANAGER database tables when the EMPLOYEE.E_MANAGERID and MANAGER.M_ID fields match:
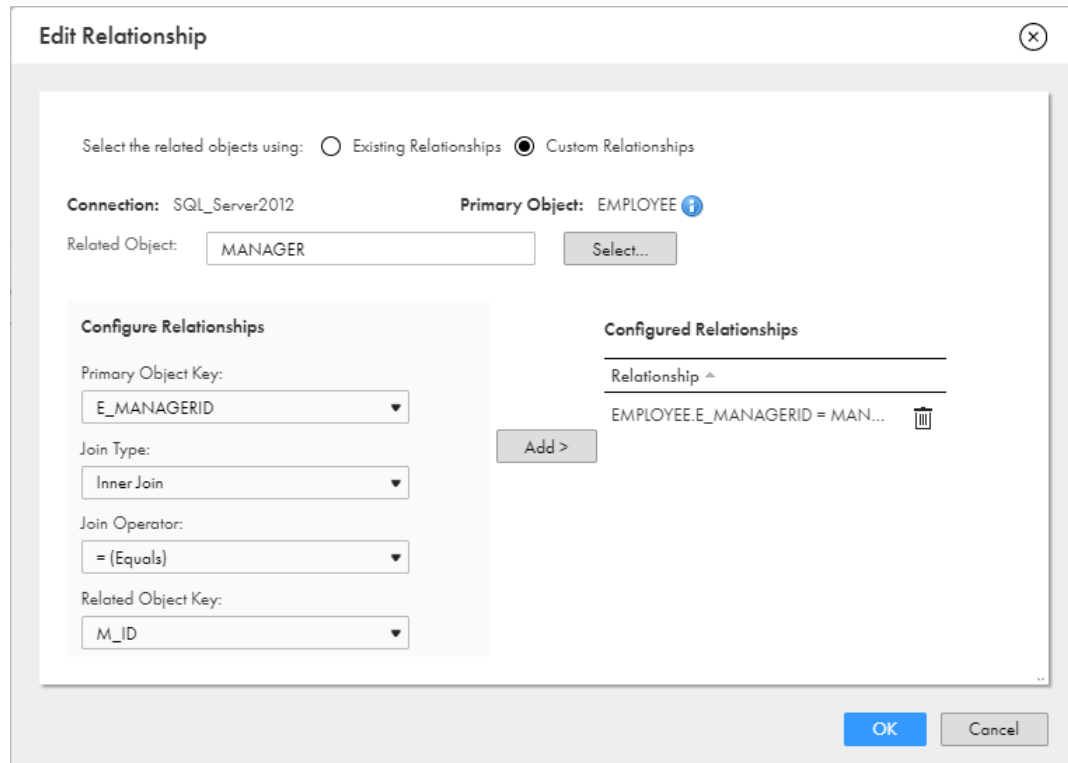
Joining related objects

You can join related objects in the Source transformation. You can join related objects based on a relationship defined in the source system or a custom relationship defined in the Source transformation.

1. On the **Source** tab, select **Multiple Objects** as the source type.

2. In the **Objects and Relationships** table, select **Add Source Object** from the **Actions** menu.

3. In the **Select Source Object** dialog box, select a source object.

    The source object appears in the **Objects and Relationships** table.

4. From the **Actions** menu, select **Add Related Objects**.

5. To join an object based on relationships in the source system, select **Existing Relationships**, select the related object, and then click **OK**.

6. To join an object based on a custom relationship, select **Custom Relationships** and perform the following steps:

    a. Select the **Related Object** to use in the join.

b. Configure the **Primary Object Key** by selecting the primary object field to use in the join.

c. Configure the join type.

    You can configure an inner, left outer, or right outer join.

d. Configure the join operator.

e. Configure the **Related Object Key** by selecting the related object field to use in the join.

f. Click **Add**.

    The relationship appears in the **Configured Relationships** table.

g. Click **OK**.

7. To join additional sources, select the source to act as the primary source and then repeat steps 4 through 6.

Advanced relationships

You can create an advanced relationship for database sources when the source object in the mapping is configured for multiple sources.

To create an advanced relationship, you add the primary source object in the **Objects and Relationships** table. Then you select fields and write the SQL statement that you want to use. Use an SQL statement that is valid for the source database. You can also add additional objects from the source.

You can also convert a custom relationship to an advanced relationship. To do this, create a custom relationship, and then select **Advanced Relationship** from the menu above the **Objects and Relationships** table. You can edit the relationship that Data Integration creates.

When you create an advanced relationship, the wizard converts any relationships that you defined to an SQL statement that you can edit.

Custom queries

Create a custom query when you want to use a database source that you cannot configure using the single- or multiple-object source options. You might create a custom query to perform a complicated join of multiple tables or to reduce the number of fields that enter the data flow in a very large source.

To use a custom query as a source, select **Query** as the source type, and then click **Define Query**. When you define the query, use SQL that is valid for the source database. You can use database-specific functions in the query.

You can also use a custom query as a lookup source. For information about using a custom query in a Lookup transformation, see .

When you create a custom query, enter an SQL SELECT statement to select the source columns you want to use. Data Integration uses the SQL statement to retrieve source column information. You can edit the datatype, precision, or scale of each column before you save the custom query.

For example, you might create a custom query based on a TRANSACTIONS table that includes transactions from 2016 with the following SQL statement:

```
SELECT TRANSACTION_ID, TRANSACTION_TOTAL, TRANSACTION_TIMESTAMP from dbo.TRANSACTIONS WHERE
TRANSACTION_TIMESTAMP>'0:0:0:0 01/01/2016'
```

Data Integration ensures that custom query column names are unique. If an SQL statement returns a duplicate column name, Data Integration adds a number to the duplicate column name as follows:

```
<column_name><number>
```

When you change a custom query in a saved mapping, at design time Data Integration replaces the field metadata with metadata using the revised query. Typically, this is the desired behavior. However, if the mapping uses a relational source and you want to retain the original metadata, use the **Retain existing field metadata** option. When you use this option, Data Integration doesn't refresh the field metadata during design time. Data Integration maps the existing fields with the fields from the revised query at run time. Fields that can't be mapped will cause run time failure.

Tip: Test the SQL statement you want to use on the source database before you create a custom query. Data Integration does not display specific error messages for invalid SQL statements.

Source filtering and sorting

You can configure the Source transformation to filter or sort data before the data enters the data flow. Use the source query options to filter or sort source data.

Configure the query options on the **Source** tab of the Source transformation. Expand the **Query Options** section, and configure the filter and sort conditions.

You can use the following source query options:

Filter

Filter source data to limit the amount of source data that enters the data flow. You can create the following types of filters:

- Non-parameterized. Select the source field and configure the operator and value to use in the filter. When you configure more than one filter, the task applies the filter expressions in the listed order with an AND operator between the filters.

- Completely parameterized. Use a parameter for a filter expression and define the filter expression in the task.

- Advanced. Create complex expressions that use AND, OR, or nested conditions. The expression that you enter becomes the WHERE clause in the query used to retrieve records from the source. You can use source fields, input and in-out parameters, or system variables in the expression.
  For example, you can use an input parameter for one of the fields, and select it when the task runs. You can reuse the same parameter in an Expression transformation to create a field expression and also in the Target transformation. Or, you can use an in-out parameter in the expression to retrieve rows that have been updated since the last run.

  For more information about system variables, see *Function Reference*. For more information about parameters, see *Mappings*.

  You can convert simple non-parameterized data filters to an advanced data filter, but you cannot convert an advanced data filter to simple data filters.

Sort

You can sort source data to provide sorted data to the mapping. For example, you can improve task performance when you provide sorted data to an Aggregator transformation that uses sorted data.

When you sort data, you select one or more source fields. When you select more than one source field, the task sorts the fields in the listed order.

Data in each field is sorted in ascending order. If you want to sort in descending order, you can use the Sorter transformation.

You can use parameters for the sort fields and define the sort fields in the task.

Web service sources

You can use a web service connector for Source transformations, such as Workday v2.

Data that comes from a web service typically has a hierarchical structure. For example, when you use a Workday v2 source connection, the data passes as XML with a hierarchical structure.

When you select a web service connection for a Source transformation, you perform the following steps to configure the transformation:

1.   Select the web service operation.

2.   Customize the request message to filter unneeded data out of the data flow.

3.   Map the hierarchical data structure to a relational structure.

For example, you want to include worker information from Workday in a mapping with a relational database target. You create a Source transformation and select the Workday connection. You select the Get_Workers operation, which pulls the worker data in a defined XML structure. You define an advanced filter so that only name and contact information enters the data flow. You define a relational structure for the worker data and then map the fields to fields in the target database.

Web service operations for sources
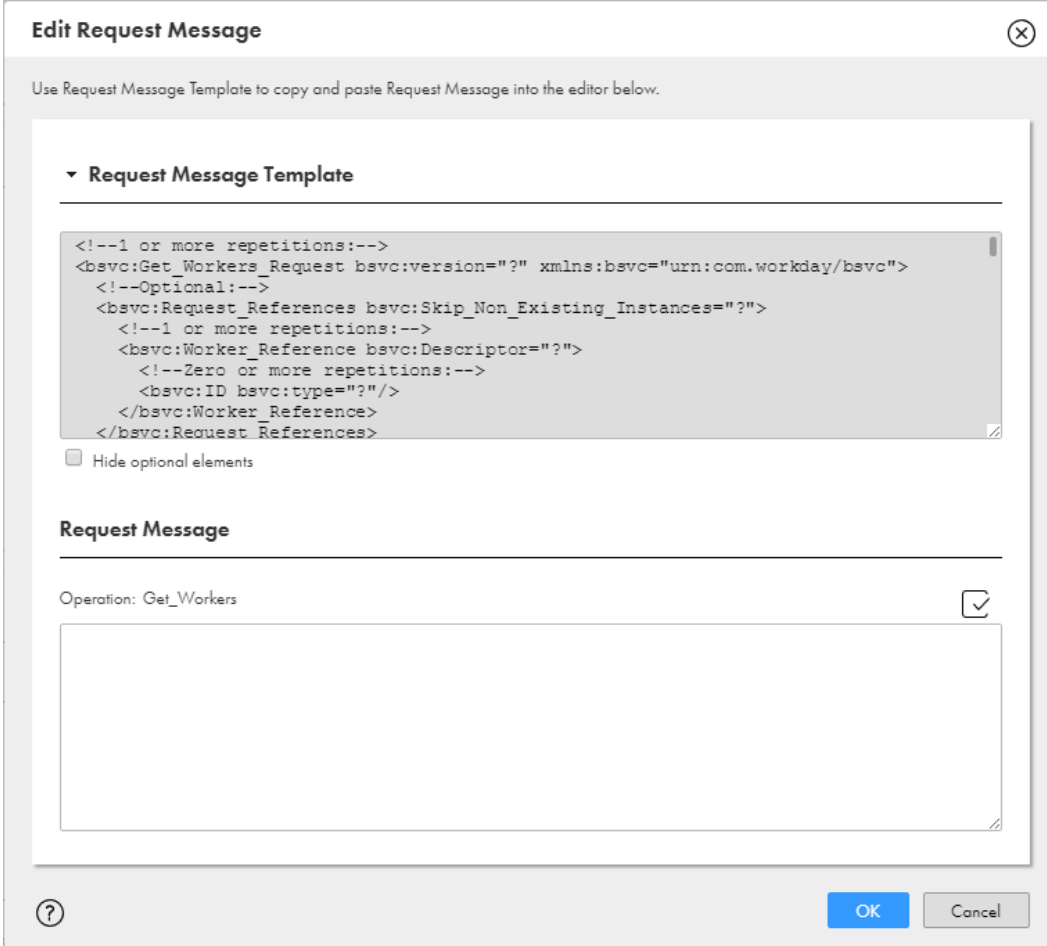
An operation determines the set of data to pass from a web service connection and defines the hierarchical data structure.

When you define the source properties for a web service connection, you select the web service operation. The available operations are determined by the connection. For example, for a Workday connection, Get_Workers is an operation.

Request messages

Web service operations often include data that you do not want to use in your mapping. You can use the default request message for the operation or customize the request message to specify the data that you want to enter the data flow.

The request message is in XML format. To customize the request message, you can begin with a template that includes the necessary formatting for the message. The request message template shows the contents for the selected operation.

Copy and paste the template into the Request Message editor pane and then revise the message.

For example, if you want to include transactions occurring between January 1, 2016 and March 31, 2016, you can enter the Effective_From and Effective_Through values in the request message, as shown in the following image:


```
Edit Request Message                                                    ⊗

Use Request Message Template to copy and paste Request Message into the editor below.


  ▼ Request Message Template
  ─────────────────────────────────────────────────────────────────

  <!--1 or more repetitions:-->
  <bsvc:Get_Workers_Request bsvc:version="?" xmlns:bsvc="urn:com.workday/bsvc">
    <!--Optional:-->
    <bsvc:Request_References bsvc:Skip_Non_Existing_Instances="?">
      <!--1 or more repetitions:-->
      <bsvc:Worker_Reference bsvc:Descriptor="?">
        <!--Zero or more repetitions:-->
        <bsvc:ID bsvc:type="?"/>
      </bsvc:Worker_Reference>
    </bsvc:Request_References>

  ☐ Hide optional elements


  Request Message
  ─────────────────────────────────────────────────────────────────

  Operation: Get_Workers                                              ☑

  <bsvc:Get_Workers_Request bsvc:version="v25.0" xmlns:bsvc="urn:com.workday/bsvc">
    <bsvc:Request_Criteria>
      <bsvc:Transaction_Log_Criteria_Data>
        <bsvc:Transaction_Date_Range_Data>
          <bsvc:Effective_From>2016-01-01</bsvc:Effective_From>
          <bsvc:Effective_Through>2016-03-31</bsvc:Effective_Through>
        </bsvc:Transaction_Date_Range_Data>
        <bsvc:Transaction_Type_References>
          <bsvc:Transaction_Type_Reference>
            <bsvc:ID bsvc:type="Business Process Type">Hire Employee</bsvc:ID>

  ?                                                          OK    Cancel
```
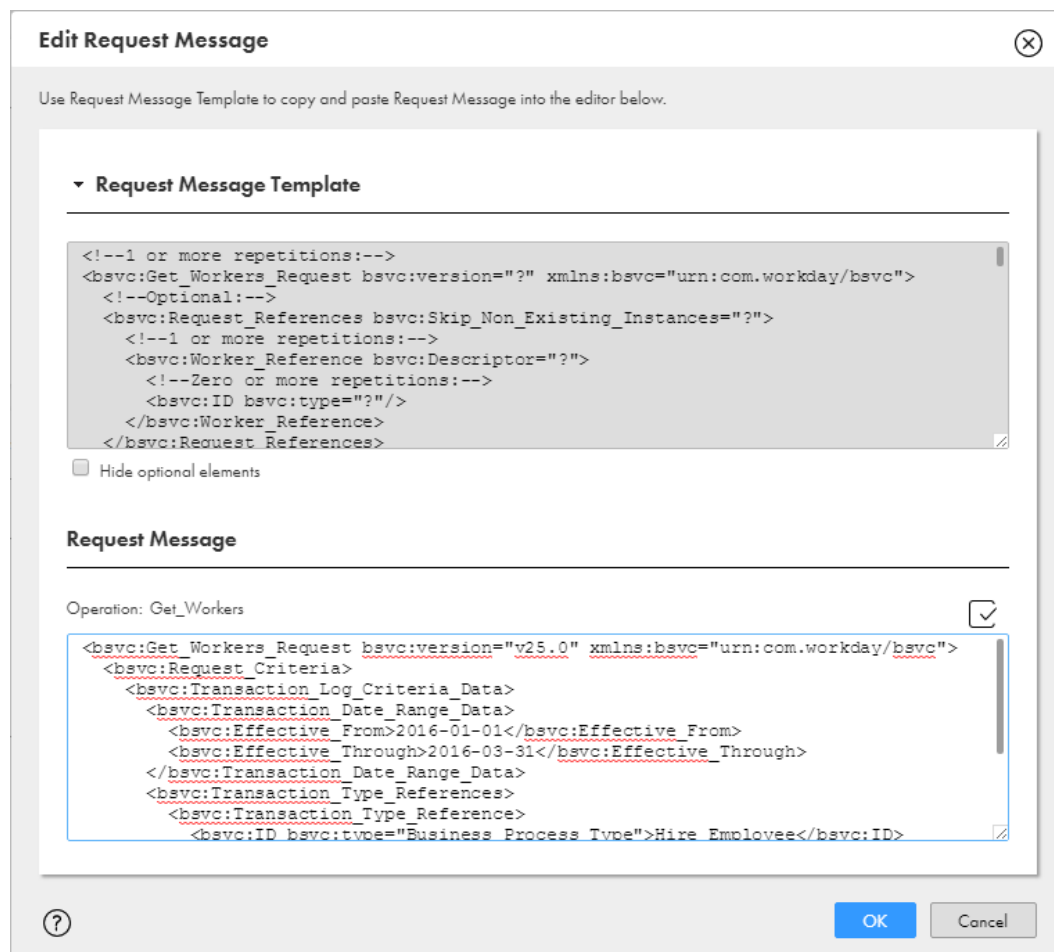
You can parameterize the request message using in-out parameters. For example, instead of using specific Effective_From and Effective_Through dates in the message, you can use $$Effective_From and $$Effective_Through parameters. You need to create the in-out parameters in the Parameters panel before you can use them in the request message.

For more information about in-out parameters, see the "Parameters" section in *Mappings*.

Be sure you use well-formed XML formatting in the request message. You can validate the message to be sure that the XML matches the structure expected by operation.

Field mapping for web service sources

You can map response fields into a relational structure of output groups and fields for output.

The response shown in the Field Mapping tab shows the hierarchical structure of the data that comes from the source.

When you select fields in the Response area, the fields appear in the Output Fields area in a relational structure with generated primary keys and foreign keys. For example, in the Response Fields area you select First_Name and Last_Name, and then you select Email_Address, which is located under a different parent in

the hierarchy. In the Output Fields area, the structure is relational with assigned primary and foreign keys, as shown in the following image:
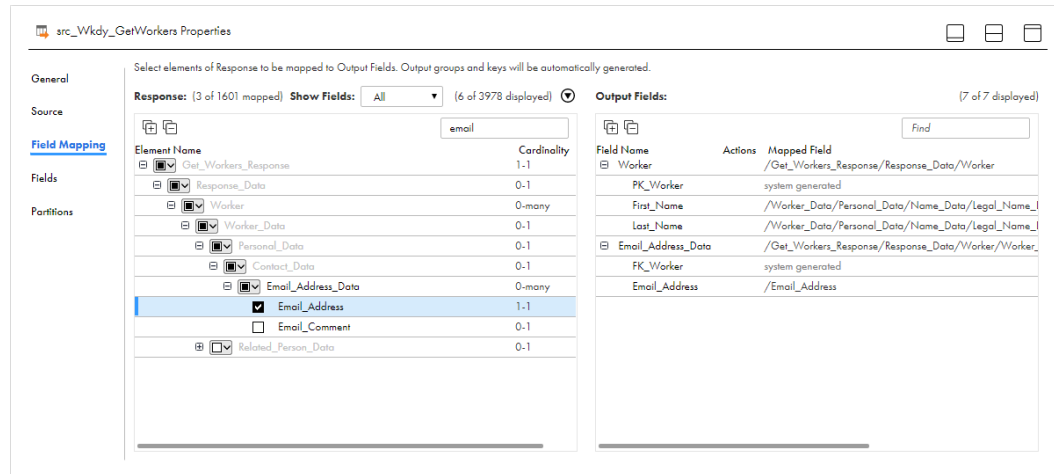
Consider cardinality when you map response fields to the relational structure. Cardinality imposes constraints on the number of times a field or group can occur at a specific point in the XML structure. A cardinality of 0-many means the field or group can have zero to many occurrences. A cardinality of 1-1 means a field or group is required and can only occur once.

If you map a field with 0-1 or 1-1 cardinality, the first parent node that has 0 to more than 1 cardinality is also mapped. If a parent group with 0 to more than 1 cardinality does not exist, the system creates a group. For example, if you map Email_Comment, which has cardinality of 0-1, the Email_Address_Data group, which has cardinality of 0-many, is automatically mapped.

Packed fields

You can pack fields to reduce the number of output groups for a request message. You can mark fields to be packed when you configure field mapping. When you run the mapping task, the task packs the element and its children into a single XML string.

Fields can come from the source already marked for packing. The Pack icon displays next to elements marked to be packed. To pack a field, click the Pack icon, as shown in the following image:
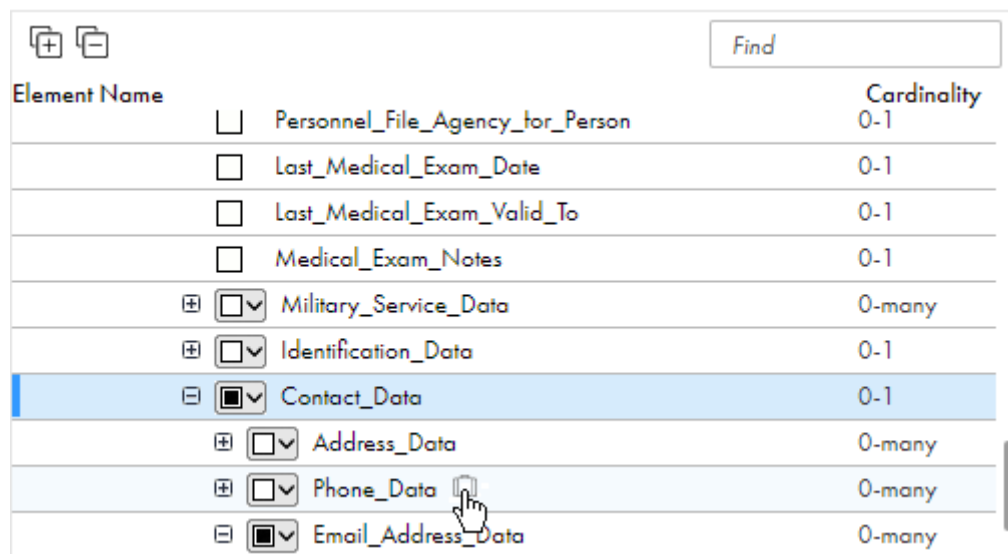
You cannot pack fields that are children of packed fields.

In the following image, you can see that the Phone_Data element is packed:
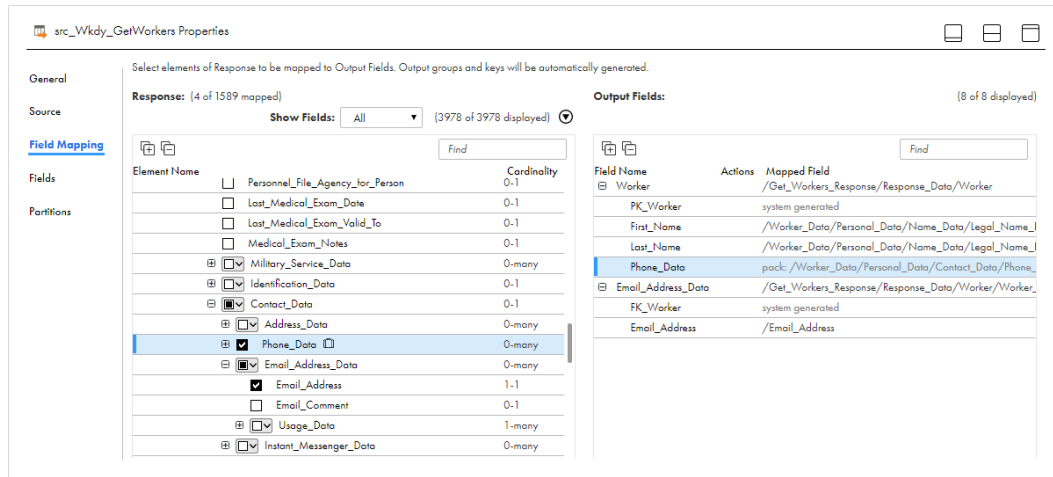
You can use XPath expressions to mark multiple fields for packing or unpacking. In the Response Fields area, click the arrow and select Mark Packed Structures, as shown in the following image:
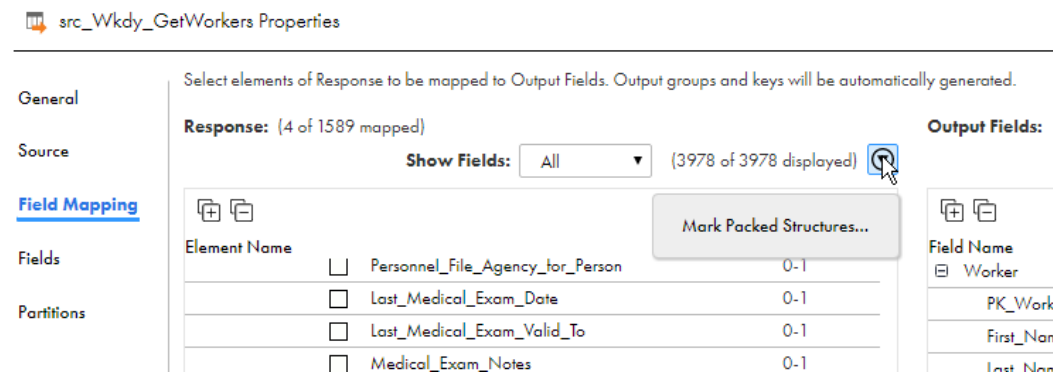
In the following image, all fields that have ID as a child are marked to pack:
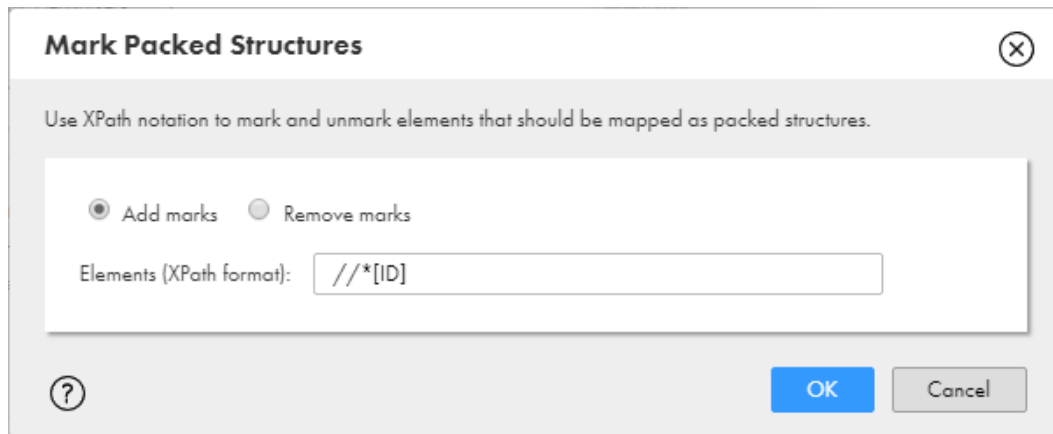
Partitions

You can use partitions to optimize performance for mapping tasks.

If a mapping task processes large data sets or includes transformations that perform complicated calculations, the task can take a long time to process. When you use multiple partitions, the mapping task divides data into partitions and processes the partitions concurrently, which can optimize performance. Not all source types support partitioning.

Enable partitioning when you configure the Source transformation in the Mapping Designer. When you configure partitions in the Source transformation, partitioning occurs throughout the mapping.

Each Source transformation across all data flows in the mapping must contain the same number of partitions.

To enable partitioning for a source, select a partitioning method on the **Partitions** tab. The partitioning methods that you can select vary based on the source type. For more information about partitioning different types of sources, see the help for the appropriate connector.

You can select one of the following partitioning methods based on the source type:

None

> The mapping task processes all data in a single partition. This is the default option.

Fixed

> The mapping task distributes rows of data based on the number of partitions that you specify. You can specify up to 64 partitions.
>
> Use this method for a source type that does not allow key range partitioning such as a flat file source, or when the mapping includes a transformation that does not support key range partitioning.
>
> Consider the number of records to be passed in the mapping to determine an appropriate number of partitions for the mapping. For a small number of records, partitioning might not be advantageous.
>
> If the mapping includes multiple sources, specify the same number of partitions for each source.

Key range

> The mapping task distributes rows of data based on a field that you define as a partition key. You select one field in the source as the partition key, and then you define a range of values for the partition key.
>
> You can use this method for tabular sources such as relational, Google BigQuery, and JDBC V2 sources.
>
> Key ranges can be of the following data types:
>
> - String
> - Any type of number data type. However, you cannot use decimals in key range values.
> - Date/time type. Use the following format: `MM/DD/YYYY HH24:MI:SS`
>
> If the mapping includes multiple sources, use the same number of key ranges for each source.

Pass through

> The mapping task processes data without redistributing rows among partitions. All rows in a single partition stay in the partition. Choose pass-through partitioning when you want to create additional partitions to improve performance, but do not want to change the distribution of data across partitions.
>
> You can use this method for sources such as Amazon S3, Netezza, and Teradata.

Dynamic

> The mapping task determines the optimal number of partitions to create at runtime based on the source size.

You cannot partition a mapping in the following situations:

- The mapping includes a Web Services or Hierarchy Parser transformation.
- The mapping includes multiple sources that use custom relationships or advanced relationships.
- The mapping is a mapping in SQL ELT mode.

When you configure partitions, be sure to save and run the mapping in the Mapping Designer to validate the partition settings.

Partitioning rules and guidelines

Before you partition a mapping, note the following rules and guidelines:

- Consider the types of transformations in the mapping and the order in which transformations appear so that you do not get unexpected results. You can partition a mapping if the mapping task can maintain data consistency when it processes the partitioned data.
- For flat file partitioning, session performance is optimal with large source files. The load may be unbalanced if the amount of input data is small.
- When a Sequence Generator transformation is in a mapping with partitioning enabled, ensure that you set up caching in the Sequence Generator transformation. Otherwise, the sequence numbers the task generates for each partition are not consecutive.
- Sequence numbers generated by Normalizer and Sequence Generator transformations might not be sequential for a partitioned source, but they are unique.
- When a Sorter transformation is in a mapping with partitioning enabled, the task sorts data in each partition separately.
- A Sorter transformation must be placed before any Joiner transformation or Aggregator transformation that is configured to use sorted data.
- You cannot use in-out parameters for key range values.
- If your mapping has more than eight partitions, mapping task performance might degrade. You can configure the Buffer Block Size and DTM Buffer Size advanced properties in the mapping task to improve performance.
- On Linux, if a target table name includes a unicode character, you need to set the default locale to UTF-8 to support multibyte data. To set the default locale to UTF-8, see the following examples:
  - For bash and related UNIX shells:

        export LC_ALL=en_US.UTF-8
  - For csh and related UNIX shells:

        setenv LC_ALL en_US.UTF-8
- If you use key range partitioning on a source in a mapping in advanced mode, the mapping fails if all of the following conditions are true:
  - The Data Integration Server processes the source.
  - An advanced cluster processes a midstream transformation.
  - The advanced cluster runs in a Google Cloud or Microsoft Azure environment.

Partitioning examples

The following examples show how you can configure partitioning in a mapping.

Partitioning with a Flat File Source

You have a mapping task that uses a large, 1GB flat file source. You want to specify two partitions in the Source transformation to optimize performance.

On the **Partitions** tab for the Source transformation, you select fixed partitioning and enter the number of partitions, as shown in the following image:

Properties Preview src_AllCustomers		
General	Partitioning type:	Fixed
Source	Enter the number of partitions to process data in parallel.	
Fields	Number of partitions:	2
Partitions		
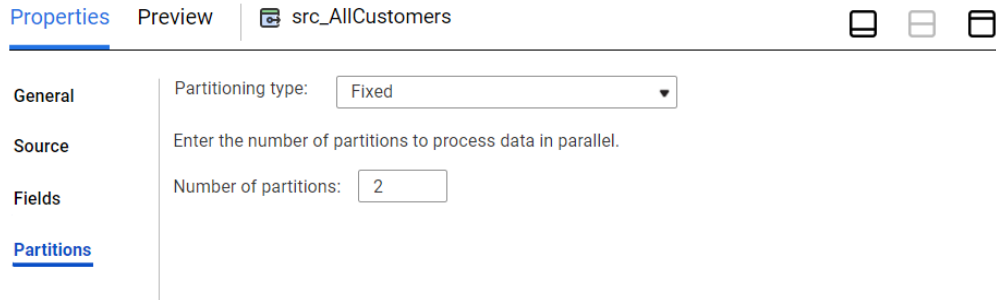
Key Range Partitioning with a Relational Database Source

You have customer names, addresses, and purchasing history in a relational database source. You decide to partition the source data into three partitions based on postal codes, using the following ranges:

- First partition: Minimum value to 30000
- Second partition: 30000 to 50000
- Third partition: 50000 to maximum value

On the **Partitions** tab for the Source transformation, you select key range partitioning and choose the BILLINGPOSTALCODE field as the partition key. You add three key ranges to create three partitions, as shown in the following image:

Properties   Preview   src_Customers

General — Partitioning type: Key Range

Source — Set up key ranges to process data in parallel. Select the partition key and then specify the range for each partition. Use a blank value for the start range to indicate the minimum value. Use a blank value for the end range to indicate the maximum value.

Fields — Partition key: BILLINGPOSTALCODE

Partitions

Key Ranges +

Partition	Start range	End range	
#1		30000	⊙
#2	30000	50000	⊙
#3	50000		⊙

Note that for the first partition, you leave the start value blank for the minimum value. In the last partition, you leave the end value blank for the maximum value.

Using these values, records with a postal code of 0 up to 30000 are processed in partition #1, records with a postal code of 30000 to 50000 are processed in partition #2, and records with a postal code of 50000 or higher are processed in partition #3.

After you configure the mapping, you save and run the mapping to validate the partitions.

Reading hierarchical data in advanced mode

You can use a Source transformation in advanced mode to read hierarchical data from complex files, such as Avro, JSON, and Parquet files. Advanced mode represents the data as an array, map, or struct.

To read hierarchical data, set the format on the **Source** tab to a hierarchical format, such as JSON, or to **Discover Structure**. Use **Discover Structure** when you want to use an intelligent structure model to define the structure of your data. For more information, see *Components*.

Downstream in the mapping, you can use the hierarchical fields as pass-through fields to convert data from one complex file format to another. For example, you can read hierarchical data from an Avro source and write the data to a JSON target. You can also use the hierarchical fields and their child fields in expressions and conditions in downstream transformations. For information about accessing child fields, see the *Function Reference*.

You can pass hierarchical fields to the following transformations:

- Target
- Aggregator
- Expression
- Filter
- Hierarchy Processor
- Joiner
- Rank
- Router
- Sequence Generator
- Sorter

Rules and guidelines for reading hierarchical data

Consider the following guidelines when you read hierarchical data:

- You must use an Amazon S3 V2 or Azure Data Lake Storage Gen2 connection to read hierarchical data. For more information, see the help for the appropriate connector.
- To read data from an XML source, use an intelligent structure model in the Source transformation. For information about intelligent structure models, see *Components*.
- You cannot use a parameter for the source connection or the source object.
- If hierarchical fields contain child fields with decimal data types, the mapping runs using low precision.
- The transformation sets the precision and scale based on the values in the first row of data. Note that this first row is sometimes referred to as row 0.
- To avoid data truncation, increase the precision and scale in the first row of data. Also ensure that the first row does not include null values.

Reading documents in advanced mode

In advanced mode, the Source transformation can read text from PDF files.

The Source transformation extracts the full structure of the document including text, tables, headings, and metadata. You can extract text from documents that have different document structures, such as invoices and reports, while preserving the order of the text.

To read a PDF, use the **Source** tab and select **Document**. Data Integration sets the input type to PDF automatically.

To read a directory of PDFs, change the **Source Type** in the advanced properties to **Directory**. For the **File Name Override**, enter `*.pdf`.

The **Fields** tab displays fields to store the text, file path, file type, and file name for each PDF.

You can pass the text to downstream Chunking and Vector Embedding transformations to build a RAG ingestion pipeline, or you can process the text, create structured data from it, and write it to a JSON file.

Configuration for multibyte hierarchical data

If a mapping includes a transformation that processes hierarchical data and the data uses multibyte characters, configure the Secure Agent machine to use UTF-8.

On Windows, create the INFA_CODEPAGENAME=UTF-8 environment variable in Windows System Properties.

On Linux, set the LC_LOCALE variable to UTF-8.

Source fields

You can configure the source fields that you want to use in the data flow. Configure source fields on the **Fields** tab of the **Properties** panel.

Configuration options vary based on the connection type. For most connection types, you can add and remove source fields, configure how the fields are displayed, edit field metadata, and restore original fields from the source object. For some connection types, you can restore original fields from the source object only. For more information about configuring source fields, see the help for the appropriate connector.

You can configure source fields in the following ways:

Add the source file name to each row.

> If you are using a file list as the source, and you want to identify the source for each row, add the source file name to the field list. You can pass this information to the target table.
>
> To add the source file name to each row, enable the **Add Currently Processed Filename Field** option. The **Add Currently Processed Filename Field** option is visible for file sources.
>
> When you enable or disable this option, Data Integration adds or removes the CurrentlyProcessedFileName field but it doesn't synchronize the fields with the source object. To synchronize with the source object, click the **Refresh** icon. You can synchronize all fields, synchronize new fields only, or skip the synchronization.

Retain existing fields at runtime.

If field metadata changes after a mapping is saved, Data Integration uses the updated field metadata when you run the mapping. Typically, this is the desired behavior. However, if the mapping uses a native flat file connection and you want to retain the metadata used at design time, enable the **Retain existing fields at runtime** option. When you enable this option, Data Integration mapping tasks will use the field metadata that was used when you created the mapping.

Add and remove fields.

You can add fields to a mapping source. Add a field to retrieve a field from the source object that is not displayed in the list. To add a field, click **Add Field**, and then enter the field name, type, precision, and scale.

You can also remove fields that you do not want to use in the mapping. To remove fields, select the fields that you want to remove, and then click **Delete**.

Change the sort order.

You can display source fields in native order, ascending order, or descending order. To change the sort order, click **Sort**, and select the appropriate sort order.

Use technical field names or labels.

You can display field names by label or technical field name.

To change the display option for field names, select **Options** > **Use Technical Field Names** or **Options** > **Use Labels**.

Edit field metadata.

You can edit the metadata for a field. You might edit metadata to change information that is incorrectly inferred. When you edit metadata, you can change the name, native type, native precision, and native scale, if applicable for the data type. For some source types, you can also change the transformation data type in the **Type** column.

To edit the name or metadata for one or more fields, click **Options** > **Edit Metadata**. When you edit metadata, you can also display native names by label or technical field name. To change the display option for native names, select **Options** > **Show Technical Field Names** or **Options** > **Show Labels**.

When you change the metadata for a field, avoid making changes that can cause errors when you run the task. For example, you can usually increase the native precision or native scale of a field without causing errors. But if you reduce the precision of a field, you might cause the truncation of data.

You can't edit field metadata in mappings in SQL ELT mode.

Restore original fields from the source object.

To restore the original fields from the source object, enable the **Synchronize** option. When you synchronize fields, Data Integration restores deleted source fields and adds fields that are new to the source. Data Integration removes any added fields that do not have corresponding fields in the source object.

Data Integration updates the metadata for existing source fields based on whether you synchronize all fields or synchronize new fields only. When you synchronize all fields, Data Integration replaces any field metadata that you edited with the field metadata from the source object. When you synchronize new fields only, Data Loader retains the metadata for any existing source field. Data Integration does not revert changes that you made to the **Name** field.

Editing native data types in complex file sources

Data Integration processes native data types in complex file sources differently on an advanced cluster and on the Data Integration Server.

On an advanced cluster, hierarchical data types such as array, map, and struct are assigned those native types. For example, a map field in an Amazon S3 source might have the native data type "map (string_integer)." You cannot edit the metadata for array, map, or struct fields.

On the Data Integration Server, Data Integration flattens complex hierarchical data types into native string datatypes with precision up to 4000 characters. Some native data types come from the connector, and others come from the parser that Data Integration uses when it reads the source data. Parser data types are prefixed with the format type. For example, in an Amazon S3 source with the Avro format, a map field that comes from the parser has the native data type avro_string. You can change the native data type for the connector and parser fields.

To change the native data type, edit the metadata for the source, and select the appropriate data type in the **Native Type** column.

When you change the native data type, you cannot change a non-parser data type to a parser data type. For example, in an Amazon S3 source, Data Integration sets the native data type for the FileName field to string. You can change the native data type to nstring but not to avro_string. Similarly, you cannot change a parser data type to a non-parser data type.

For more information about editing native data types in complex file sources, see the help for the appropriate connector.

Editing transformation data types

When Data Integration reads source data, it converts the native data types to comparable transformation data types before it transforms the data. When Data Integration writes to a target, it converts the transformation data types to the comparable native data types. When you edit source metadata, you can sometimes change the transformation data type for a field.

You can change the transformation data type for connectors in which the native data type has multiple corresponding transformation data types. For example, in a Kafka source, you can map the native data type binary to the transformation data type binary or string.

To change the transformation data type, edit the metadata for the source, and select the appropriate transformation data type in the **Type** column.

When you edit the transformation data type for a field, Data Integration updates the data type for the field in the downstream transformations. It also updates the data type for the field in the target if the target is created at runtime. If the mapping contains an existing target, you might need to edit the field metadata in the target to ensure that the data types are compatible.

For more information about editing transformation data types for different source types, see the help for the appropriate connector.

CHAPTER 3

Target transformation

Use the Target transformation to define the target connection and target object for the mapping. You can use one or more Target transformations in a mapping.

Based on the connection type, you can define advanced target options, specify to use a new or existing target object, or configure update columns for the target. The target options that appear also depend on the connection type that you select. For example, when you select a Salesforce connection, you can configure success and error log details.

You can use file, database, and Data Integration connections in the Target transformation.

The following properties are available for the Target transformation:

- General. Defines the transformation name and a description.
- Incoming Fields. Includes the field rules that define the data written to the target. Allows a preview of target fields.
- Target. Defines the target connection, target object, and advanced options. Based on the connection type, you can create a new target, use an existing target, or configure update columns.
- Target Fields. Lists the fields in the target objects. Optionally add or remove fields. You can also edit target field metadata.
- Field Mapping. Defines the field mappings from the upstream transformation to the target. Field mapping is only applicable when using an existing target object.

Target example

You might work with a flat file target in a mapping that reads Salesforce user account data but excludes user preference data. A Source transformation reads data from the Account object and the related User object.

The Target transformation uses a flat file connection that writes to the following directory: `C:\UserAccountData`. The default All Fields rule includes all incoming fields. You create a Named Field rule to exclude the unnecessary user preferences fields.

When you select **Create New Target at Runtime**, you enter the following name for the target file: SF_UserAccount_%d%m%y.csv.

The mapping task creates a target file named SF_UserAccount_291116.csv in the `C:\UserAccountData` directory when the task runs on November 29, 2016. The target file includes all fields from the Salesforce Account and User objects except for the user preferences fields specified in the Named Fields rule.

Target object

Select the target object for the Target transformation on the **Target** tab of the Properties panel.

The properties that you configure for the target object vary based on the connection type and the mapping type. Your organization's licenses can also determine the target properties that appear when the Target transformation is part of a mapplet.

The following image shows the **Target** tab for a flat file target:
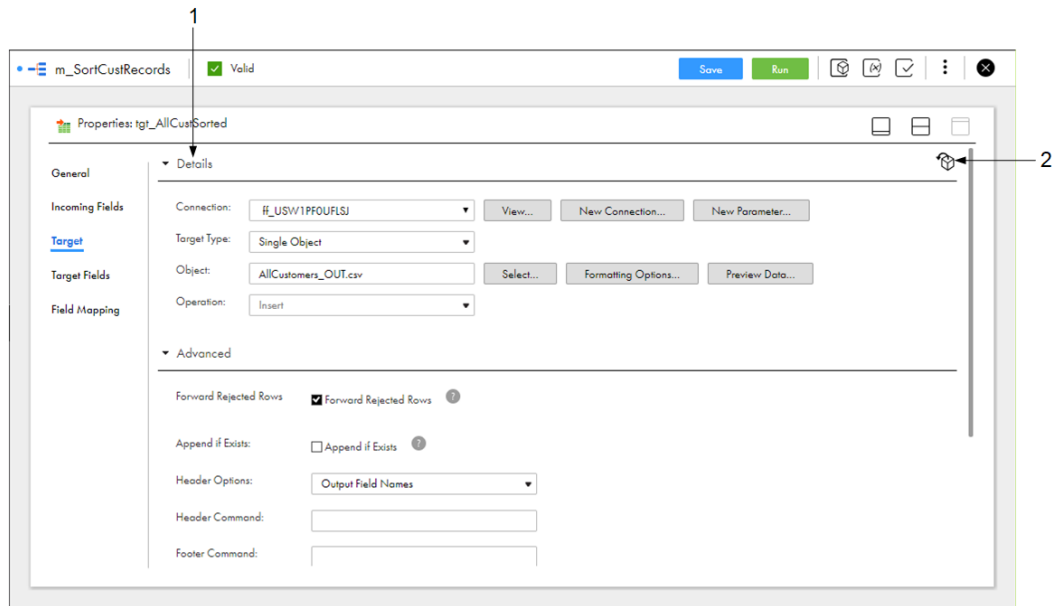
1. Target details where you configure the target connection, target type, target object, and target operation.
2. Select the target object from the mapping inventory.

You can select a target object in the following ways:

Select the connection and target object.

In the **Details** area, select the target connection, target type, and target object. You can create a new connection. For flat file and relational targets, you can also create the target object at run time. For mappings in SQL ELT mode, you configure the target connection in the mapping properties instead of in the Target transformation.

If you use an existing target object, select the target from a list of target objects and link the target to the upstream transformation. If the target table changes, you must update the Target transformation to match it. If the physical target and the Target transformation do not match, the mapping fails.

If you use an existing target object for a flat file target, the existing target is overwritten when you run the mapping task.

Select the target object from the mapping inventory.

If your organization administrator has configured Enterprise Data Catalog integration properties, and you have added objects to the mapping from the **Data Catalog** page, you can select the target object from the **Inventory** panel. If your organization administrator has not configured Enterprise Data Catalog integration properties or you have not performed data catalog discovery, the **Inventory** panel is empty. For more information about data catalog discovery, see *Mappings*.

The **Inventory** panel isn't available in mappings in SQL ELT mode.

Use a parameter.

You can use input parameters to define the target connection and target object when you run the mapping task. For more information about parameters, see *Mappings*.

You must also define the target operation. The available target operations vary based on the connection type.

Target file creation on advanced clusters

When you create a mapping that runs on an advanced cluster and contains a complex file target, the Target transformation creates a folder with the file name you enter in the File Name property in the Advanced area. It does not create a single file. This is necessary because the output may be divided into a series of files.

The folder name might also be appended with a unique identifier. You do not need to know the identifier, because the mapping recognizes the file name and all the contents within the folder.

The part files within the folder are named: `part-<unique identifier>.<format>`, where the suffix can be csv, txt, avro, parquet, json, orc.

File targets

File targets include flat files and FTP/SFTP files. When you configure a file target, you specify the connection, target type, and target object.

For FTP/SFTP targets, you can select an existing target object. For flat file targets, you can select an existing target object or create a new target at run time.

If you create a flat file target at run time, you can specify a static or dynamic file name.

File target properties

You configure file target properties on the **Target** tab of the Properties panel.

The following table describes the file target details:

Property	Description
Connection	Name of the target connection.
Target Type	Target type, either single object or parameter.
Object	Name of the target object. In advanced mode, the object name cannot contain the dollar sign character, `$`. The dollar sign is a reserved character for parameters.

Property	Description
Formatting Options	Flat file format options. Opens the **Formatting Options** dialog box to define the format of the file. You can choose either a delimited or fixed-width file type. Default is delimited. To write to a delimited flat file type, configure the following file format options: - Delimiter. Delimiter character. Can be a comma, tab character, colon, semicolon, nonprintable control character, or a single-byte or multibyte character that you specify. - Treat multiple characters as a single delimiter. Treats the specified set of delimiters as one delimiter. For example, a source file contains the following record: abc~def|ghi~|~|jkl|~mno. If you specify the delimiter as (~ |), Data Integration reads the record as three columns separated by two delimiters: abc~def|ghi, NULL, jkl|~mno. If you disable this option, Data Integration reads the record as nine columns separated by eight delimiters: abc, def, ghi, NULL, NULL, NULL, jkl, NULL, mno. - Treat consecutive delimiters as one. Treats one or more consecutive column delimiters as one. The default is to treat consecutive delimiters as a null value. - Row Delimiter. Line break character. Select a line break character from the list. The default is line-feed, \012 LF. **Note:** Linux machines write "\n" as the line-feed character. Windows machines write "\r\n" as the line-feed character. - Text Qualifier. Character to qualify text. - Escape character. Escape character. - Field labels. Determines if the mapping task generates field labels or imports labels from the source file. - First data row. The first row of data. The task starts the read at the row number that you enter. You can use a tab, space, or any printable special character as a delimiter. The delimiter can have a maximum of 10 characters. The delimiter must be different from the escape character and text qualifier. To write to a fixed-width flat file type, select the fixed-width file format to use. If the list includes multiple fixed-width file formats with the same name, use the project and folder location that's appended to the name to determine the appropriate file format to use. If you do not have a fixed-width file format, click **New** > **Components** > **Fixed Width File Format** to create one.
Operation	For file targets, the operation is always Insert.

The following table describes the advanced properties for flat file targets:

Property	Description
Forward Rejected Rows	Indicates whether the mapping task forwards rejected rows to the reject file. If you enable row error handling, the mapping task writes the rejected rows and the dropped rows to the row error logs instead of generating a reject file. If you want to write the dropped rows to the session log in addition to the row error logs, you can enable verbose data tracing. If you don't forward rejected rows, the mapping task drops rejected rows and writes them to the session log.
Thousand Separator	Thousand separator character. Can be none, comma, or period. Cannot be the same as the decimal separator or the delimiter character. Field type must be Number. You might also need to update the field precision and scale. Default is None.
Decimal Separator	Decimal character. Can be a comma or period. Cannot be the same as the thousand separator or delimiter character. Field type must be Number. You might also need to update the field precision and scale. Default is Period.

Property	Description
Datetime Format	Optional. Overrides the datetime format specified in the flat file connection. Enter the datetimeformat to override. For example, YYYYMMDD.
Append if Exists	Appends the output data to the target files and reject files for each partition. You cannot use this option for FTP/SFTP target files. If you do not select this option, the mapping task truncates each target file before writing the output data to the target file. If the file does not exist, the mapping task creates it.
Create Target Directory	Creates the target directory if it doesn't exist as specified in the Output file directory field.
Header Options	Creates a header row in the file target. You can choose the following options: - No Header. Do not create a header row in the flat file target. - Output Field Names. Create a header row in the file target with the output field names. - Use header command output. Use the command in the Header Command field to generate a header row. For example, you can use a command to add the date to a header row for the file target. Default is No Header.
Header Command	Command used to generate the header row in the file target. For example, you can use a command to add the date to a header row for the file target.
Footer Command	Command used to generate the footer row in the file target.
Output Type	Type of target for the task. Select **File** to write the target data to a file target. Select **Command** to output data to a command. You cannot select **Command** for FTP/SFTP target connections.
Output File Name	File name or file name and path of the output file. By default, the mapping task names output files after the target object. Target objects must use one of the following file extensions: `csv, dat, txt`
Output File Directory	Name of the output directory for a flat file target. By default, the mapping task writes output files to the target connection directory. You can also use an input parameter to specify the target file directory. If you use the service process variable directory $PMTargetFileDir, the task writes target files to the configured path for the system variable. To find the configured path of a system variable, see the pmrdtm.cfg file located at the following directory: `<Secure Agent installation directory>\apps\Data_Integration_Server\<Data Integration Server version>\ICS\main\bin\rdtm` You can also find the configured path for the $PMTargetFileDir variable in the Data Integration Server system configuration details in Administrator.

Property	Description
Reject File Directory	Directory path to write the reject file. By default, the mapping task writes all reject files to the following service process variable directory: `$PMBadFileDir/<federated task ID>` If you specify both the directory and file name in the **Reject File Name** field, clear this field. The mapping task concatenates this field with the **Reject File Name** field when it runs the task.
Reject File Name	File name, or file name and path of the reject file. By default, the mapping task names the reject file after the target object name: <target name>.bad. The mapping task concatenates this field with the **Reject File Directory** field when it runs the task. For example, if you have `C:\reject_file\` in the **Reject File Directory** field, and enter `filename.bad` in the **Reject File Name** field, the mapping task writes rejected rows to `C:\reject_file\filename.bad`.

The following table describes the advanced properties when you write a message to a file target:

Property	Description
Input Type	Type of input to write to the target. Select one of the following options: - Buffer - File The available options depend on the incoming data from upstream transformations. Default is buffer.
Acknowledgement Options	Options to save the acknowledgement. Select one of the following options: - **Save to session log** to write the acknowledgement to the session log. - **Save to file** to save the acknowledgement to a file. - **Discard** to discard the acknowledgement.
Acknowledgement File Path	File path to write the acknowledgement if you save the acknowledgement to a file.
Operation if File Exists	Operation to perform if the acknowledgement file already exists. Select one of the following operations: - **Rename** to rename the acknowledgement file. - **Append** to append the acknowledgement to the file. - **Overwrite** to overwrite the acknowledgement file.
Fail the job if acknowledgement is not AA/CA	Fails the job if the acknowledgement code is not AA or CA.
Retry the job if acknowledgement is AR/CR	Tries to run the job again if the acknowledgement code is AR or CR.
Message Retry Attempts	Number of attempts to try writing the message.
Message Retry Interval	Time interval in seconds to try writing the message again.

Property	Description
Validate Message	Indicates whether Data Integration validates the message.
Forward Rejected Rows	Indicates whether the mapping task forwards rejected rows to the reject file. If you enable row error handling, the mapping task writes the rejected rows and the dropped rows to the row error logs instead of generating a reject file. If you want to write the dropped rows to the session log in addition to the row error logs, you can enable verbose data tracing. If you don't forward rejected rows, the mapping task drops rejected rows and writes them to the session log.

Flat file targets created at run time

If a mapping includes a flat file target, you can select an existing target file or create the target at run time. When you create a target at run time, Data Integration automatically discovers the target object metadata for data type, precision, and scale, based on the data source.

If you need to edit target object metadata, you can edit it in the Source transformation.

You cannot link the target fields to the upstream transformation. If you want to reduce the number of unused fields in the target, configure field rules in the Target transformation or in the upstream transformations.

When you create a flat file target at run time, the mapping task creates the physical target the first time the mapping runs based on the fields from the upstream transformation. In subsequent runs, if the target file name does not change, the mapping task overwrites the target file. If the file name changes between mapping runs, the mapping task creates a new target. Data Integration creates the target in the default connection directory.

You can configure a static or dynamic file name for the target file. A static file name can include a time stamp. A dynamic file name uses an expression to generate the file name when the mapping task runs.

Data Integration can write to a delimited or fixed-width target file. By default, when Data Integration creates a target at run time using a fixed-width file format, it rounds data that uses the Decimal or Double data types. You can choose to skip rounding the data for Decimal and Double data types. Data Integration doesn't round delimited data by default.

Flat file targets with static file names

If you create a flat file target at run time, you can specify a static file name. The file name can include a time stamp that indicates when the file was created.

To specify a static file name, in the **Target Object** dialog box, enter the file name in the **Static File Name** field. To include a time stamp, enable **Handle Special Characters** and add the time stamp characters to the file name. For example, the file name MyTarget_%d-%m.csv includes the day and month in which the mapping ran.

The following image shows the **Target Object** dialog box:
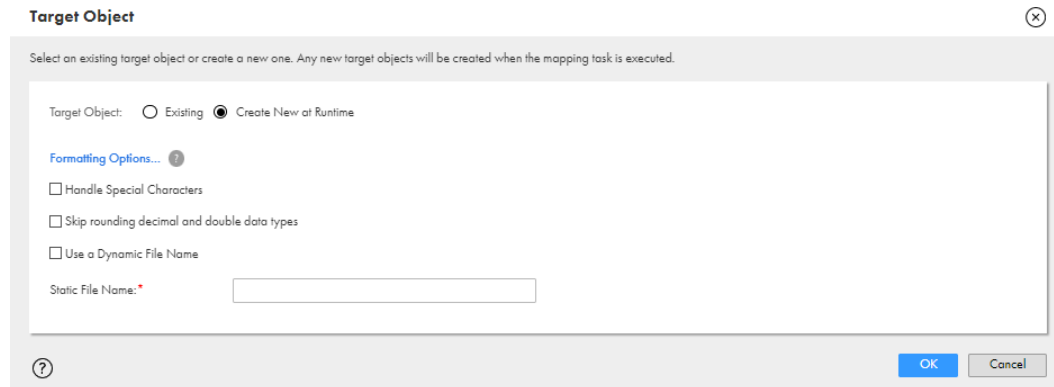
If you do not include a time stamp, the mapping task creates the target file the first time the task runs and overwrites the file during subsequent runs.

If you append a time stamp to the target file name, the mapping task writes data to a new file when the time stamp changes. For example, you enable special character handling, enter static file name MyTarget_%d-%m.csv, and run the mapping task on January 15 and January 16. The mapping task creates the target files MyTarget_15-01.csv and MyTarget_16-01.csv.

Flat file target time stamps

When you create a flat file target at run time, you can append time stamp information to the file name to show when the file is created.

When you specify the file name for the target file, you include special characters based on Linux STRFTIME function formats that the mapping task uses to include the time stamp information in the file name. The time stamp is based on the organization's time zone.

The following table describes some common STRFTIME function formats that you might use:

Special Character	Description
%d	Day as a two-decimal number, with a range of 01-31.
%m	Month as a two-decimal number, with a range of 01-12.
%y	Year as a two-decimal number without the century, with range of 00-99.
%Y	Year including the century, for example 2015.
%T	Time in 24-hour notation, equivalent to %H:%:M:%S.
%H	Hour in 24-hour clock notation, with a range of 00-24.
%I	Hour in 12-hour clock notation, with a range of 01-12.
%M	Minute as a decimal, with a range of 00-59.
%S	Second as a decimal, with a range of 00-60.
%p	Either AM or PM.

Flat file targets with dynamic file names

If you create a flat file target at run time, you can specify a dynamic file name. A dynamic file name uses an expression to generate the file name.

You can use a dynamic file name to create a new target file every time the mapping task runs. For example, the following expression creates a file called "OrdersOut_<system_timestamp_with_second_precision>.csv" each time the mapping task runs:

```
'OrdersOut_'||To_Char(SYSDATE, 'YYYYMMDDHH24MISS')||'.csv'
```

You can also use a dynamic file name in a mapping that contains a Transaction Control transformation to write data to a different target file each time a transaction boundary changes. For example, the following expression can be used in a target that is downstream of the Transaction Control transformation to commit data to a different target file every time the DEPT_ID field changes:

```
'Results_Dept_'||To_Char(DEPT_ID)||'.dat'
```

To specify a dynamic file name, in the **Target Object** dialog box, select **Use a Dynamic File Name** and enter the file name expression in the expression editor.

The following image shows the **Target Object** dialog box with the **Use a Dynamic File Name** option enabled:
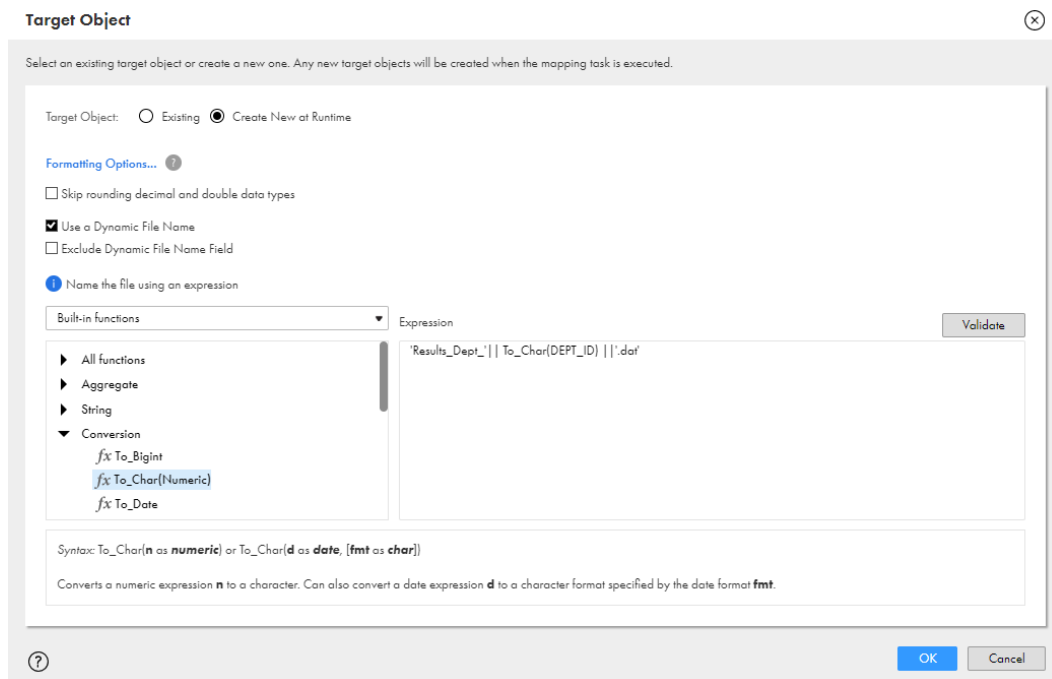
You can include incoming field names, constants, operators, built-in functions, and user-defined functions in the target file name expression.

If you use an incoming field name in the file name expression, you can choose to exclude the field from the target. When you enable the **Exclude Dynamic File Name Field** option, Data Integration does not write the incoming field used in the expression to the target. Include only one incoming field in the expression. If you include more than one incoming field, the expression is invalid.

To use more than one incoming field, add an Expression transformation directly before the Target transformation. In the Expression transformation, configure a field to hold the expression that you want to use as the file name. In the Target transformation, use this field as the expression in the dynamic file name.

For more information about creating expressions, see the *Function Reference*.

Creating a flat file target at run time

To create a flat file target at run time, select **Create New at Runtime** in the **Target Object** dialog box. Then configure a static or dynamic file name for the target file.

1. On the **Target** tab of the Target transformation, select a flat file connection.
2. Set the target type to **Single Object**.
3. Click **Select** to select the target object.
4. In the **Target Object** dialog box, select **Create New at Runtime**.
5. Configure the target file name:
   - To enter a static file name without a time stamp, enter the file name in the **Static File Name** field.
   - To enter a static file name that contains a time stamp, enable **Handle Special Characters** and enter the file name, including the time stamp characters, in the **Static File Name** field.
   - To enter a dynamic file name, enable **Use a Dynamic File Name** and configure the expression to create the file name.

     You can use fields, system variables, parameters, built-in functions, and user-defined functions in the file name expression.
6. Optionally, if you're using a fixed-width flat file format that uses the Decimal or Double data type, and you don't want Data Integration to round the data, select **Skip rounding decimal and double data types**.
7. Click **OK**.

Database targets

Database targets include relational sources such as Oracle, MySQL, and Microsoft SQL Server.

When you configure a Target transformation for a database target, you can write data to a single target table. You can select an existing table or create the table at run time.

Ensure that the table and column names do not exceed 74 characters.

Database target properties

You configure database target properties on the **Target** tab of the Properties panel.

The following table describes the database target properties:

Property	Description
Connection	Name of the target connection. Alternatively, you can define a parameter, and then specify the connection in the mapping task.
Target Type	Target type, either single object or parameter.
Object	Name of the target object. If you select a single object, you can also preview the data.
Operation	Target operation, either insert, update, upsert, delete, or data driven.

Property	Description
Truncate Target	Truncates the target object before inserting new rows. Applies to insert and data driven operations.
Enable Target Bulk Load	Uses the database bulk API to perform an insert operation. Use the bulk API to write large amounts of data to the database with a minimal number of API calls. Loading in bulk mode can improve performance, but it limits the ability to recover because no database logging occurs. Applies to insert operations.
Update Columns	The fields to use as temporary primary key columns when you update, upsert, or delete target data. When you select more than one update column, the mapping task uses the AND operator with the update columns to identify matching rows. Applies to update, upsert, delete and data driven operations.
Data Driven Condition	Enables you to define expressions that flag rows for an insert, update, delete, or reject operation. For example, the following IIF statement flags a row for reject if the ID field is null. Otherwise, it flags the row for update: `IIF (ISNULL(ID), DD_REJECT, DD_UPDATE )` Applies to the data driven operation.
Forward Rejected Rows	Causes the mapping task to forward rejected rows to the reject file. If you do not forward rejected rows, the mapping task drops rejected rows and writes them to the session log. If you enable row error handling, the mapping task writes the rejected rows and the dropped rows to the row error logs instead of generating a reject file. If you want to write the dropped rows to the session log in addition to the row error logs, you can enable verbose data tracing.
Pre SQL	SQL command to run against the target before reading data from the source. You can enter a command of up to 5000 characters.
Post SQL	SQL command to run against the target after writing data to the target. You can enter a command of up to 5000 characters.
Update Override	Overrides the default UPDATE statement for the target. Enter the update statement. Alternatively, click **Configure** to generate the default UPDATE statement, and then modify the default statement. The UPDATE statement that you enter overrides the default UPDATE statement that Data Integration uses to update targets based on key columns. You can define an override UPDATE statement to update target tables based on non-key columns.
Reject File Directory	Directory path to write the reject file. By default, the mapping task writes all reject files to the following service process variable directory: `$PMBadFileDir/<federated task ID>` If you specify both the directory and file name in the **Reject File Name** field, clear this field. The mapping task concatenates this field with the **Reject File Name** field when it runs the task.
Reject File Name	File name, or file name and path of the reject file. By default, the mapping task names the reject file after the target object name: <target name>.bad. The mapping task concatenates this field with the **Reject File Directory** field when it runs the task. For example, if you have `C:\reject_file\` in the **Reject File Directory** field, and enter `filename.bad` in the **Reject File Name** field, the mapping task writes rejected rows to `C:\reject_file\filename.bad`.

For more information about database target properties, see the help for the appropriate connector.

Vector database target properties

In advanced mode, you can write vectors to a vector database such as Pinecone. You can configure the target properties, advanced properties, and target fields.

Target properties

The following table describes the vector database target properties:

Property	Description
Connection	Name of the target connection. Alternatively, you can define a parameter, and then specify the connection in the mapping task.
Target Type	Target type, either single object or parameter.
Object	Name of the target object.
Operation	Target operation. You can use only upsert.
Update Columns	The fields to use as temporary primary key columns when you upsert target data. When you select more than one update column, the mapping task uses the AND operator with the update columns to identify matching rows. If you generate an ID within the mapping, the ID is dynamic and might not be consistent with existing IDs in the vector database. If a generated ID matches an existing ID in the vector database, the Target transformation replaces the row in the vector database, but the vector might not correspond to the same data.

For more information about the target properties, see the help for the appropriate connector.

Advanced properties

The following table describes the advanced properties:

Property	Description
Batch Size	Number of operations for the vector database to process at the same time.
Namespace	Namespace in the vector database where you want to store the vectors.

Target fields

The following table describes the target fields that a Target transformation might write to a vector database:

Property	Description
Vector ID	Vector identifier to store in the vector database. The ID helps to quickly access the vector representation for efficient storage, indexing, and retrieval operations. To create vector IDs, you can use the UUID_STRING function with no arguments in an Expression transformation or you can use a Sequence Generator transformation that uses a shared sequence across all mappings that load data to the same index in the vector database.
Vector	Array of doubles that represents the vector embedding. You can use a Vector Embedding transformation to create vector embeddings.
Metadata	Metadata to write to the vector database as a struct. Includes all incoming fields except for the vector and vector ID fields. The metadata field contains a list of key-value pairs in JSON format.

The target fields can differ based on the vector database. For more information about target fields, see the help for the appropriate connector.

Database targets created at run time

If a mapping includes a database target, you can select an existing target table or create the target table at run time. When you create a database target at run time, Data Integration automatically discovers the target object metadata for data type, precision, and scale, based on the data source.

If you need to edit target object metadata, you can edit it in the Source transformation.

You cannot link the target fields to the upstream transformation. If you want to reduce the number of unused fields in the target, configure field rules in the Target transformation or in the upstream transformations.

When you create a database target at run time, the mapping task creates the database table the first time the mapping runs based on the fields from the upstream transformation.

In subsequent runs, the mapping task replaces the data in the target table that was created in the initial run. Consequently, if you change the mapping after the initial run, in subsequent runs the target will not reflect changes to the number of target fields and its metadata. To see the changes, you can either delete the existing target before you run the mapping or change the name of the target.

If you create a relational target at run time, the target operation is always insert. You can choose to truncate the target.

Note: In mappings created before the Spring 2020 September release, Data Integration converts Bigint data from a parameterized source to Int data in database targets created at runtime. To write Bigint data to the target without conversion, edit the mapping in the Mapping Designer and enable the option in the mapping advanced properties.

Data Integration does not convert Bigint data in mappings created after the Spring 2020 September release.

Creating a database target at run time

To create a database target at run time, select **Create New at Runtime** in the **Target Object** dialog box and enter the target table name.

1.   On the **Target** tab of the Target transformation, select a database connection.

2.   Set the target type to **Single Object**.

3.   Click **Select** to select the target object.

4.   In the **Target Object** dialog box, select **Create New at Runtime**.

5.   Enter the target table name.

6.   Click **OK**.

Update columns for relational targets

You can configure one or more fields as update columns in relational targets. Update columns are columns that uniquely identify rows in the target table. The mapping task uses them to update, upsert, or delete data in the target.

Configure update columns when the target table does not contain a primary key and the mapping uses an update, upsert, or delete operation. When you select more than one update column, the mapping uses the AND operator with the update columns to identify matching rows.

When you run the mapping, it uses the field mapping to match rows in the upstream transformations to the target table. If the mapping task matches an incoming row to multiple target rows, it performs the specified task operation on all matched target rows.

When you use a parameter for the target connection or target object, you can configure update columns in the mapping task.

Configuring update columns

You can configure update columns when you use the update or upsert operation to update data in a relational target.

1.   In the Properties panel, click the **Target** tab.

2.   Select a relational connection.

     You can also use a connection parameter for a relational database connection type.

3.   Select the target type that you want to use.

4.   Select a target object.

5.   Select the update or upsert operation.

6.   To select update columns, click **Add**.

     The **Update Columns** window displays all target columns.

7.   Move the fields that you want to use from the **Target Columns** list to the **Update Columns** list.

8.   Click **OK**.

Target update override

By default, Data Integration updates target tables based on key values. However, you can override the default UPDATE statement for each target in a mapping. You might want to update the target based on non-key columns.

You can enter a target update override for relational and ODBC connections. For information on the specific steps to override the default target UPDATE statement, see the help for the appropriate connector.

Override the UPDATE statement in the Target transformation advanced properties. Enter the target UPDATE statement in the **Update Override** field. In some relational connections, you can generate the default UPDATE statement. Click **Configure** to generate the default UPDATE statement and then modify the statement.

Because the target fields must match the target column names, the update statement includes the keyword :TU to specify the fields in the target transformation. If you modify the UPDATE portion of the statement, you must use :TU to specify fields.

When you override the default UPDATE statement, you must enter an SQL statement that is valid for the database. Data Integration does not validate the syntax.

Example

A mapping passes the total sales for each salesperson to the T_SALES table.

Data Integration generates the following default UPDATE statement for the target T_SALES:

```
UPDATE
  T_SALES
SET
  EMP_NAME = :TU.EMP_NAME,
  DATE_SHIPPED = :TU.DATE_SHIPPED,
  TOTAL_SALES = :TU.TOTAL_SALES
WHERE
  EMP_ID = :TU.EMP_ID
```

You want to override the WHERE clause to update records for employees named Mike Smith only. To do this, you edit the WHERE clause as follows:

```
UPDATE
  T_SALES
SET
  DATE_SHIPPED = :TU.DATE_SHIPPED,
  TOTAL_SALES = :TU.TOTAL_SALES
WHERE
  :TU.EMP_NAME = EMP_NAME AND EMP_NAME = 'MIKE SMITH'
```

Guidelines for configuring the target update override

Use the following guidelines when you enter target update queries:

- If you use target update override, you must manually put all database reserved words in quotes.
- You cannot override the default UPDATE statement if a target column name contains any of the following characters:

  ' , ( ) < > = + - * / \ t \ n \ 0 <space>

- If you update an individual row in the target table more than once, the database only has data from the last update. If the mapping does not define an order for the result data, different runs of the mapping on identical input data may result in different data in the target table.
- A WHERE clause that does not contain any column references updates all rows in the target table, or no rows in the target table, depending on the WHERE clause and the data from the mapping. For example, the following query sets the EMP_NAME to "MIKE SMITH" for all rows in the target table if any row of the transformation has EMP_ID > 100:

  ```
  UPDATE T_SALES SET EMP_NAME = 'MIKE SMITH' WHERE :TU.EMP_ID > 100
  ```

- If the WHERE clause contains no field references, the mapping updates the same set of rows for each row of the mapping. For example, the following query updates all employees with EMP_ID > 100 to have the EMP_NAME from the last row in the mapping:

  ```
  UPDATE T_SALES SET EMP_NAME = :TU.EMP_NAME WHERE EMP_ID > 100
  ```

- If you set the target operation to update or upsert, configure the mapping task to treat source rows as update in the advanced session properties.

Web service targets

You can use a web service connector such as Workday V2 for Target transformations.

You can map fields that are in a relational structure to request fields in a web service target to generate hierarchical output.

When you select a web service connection for a Target transformation, you perform the following steps to configure the transformation:

1. Select the web service operation.

2. Map the incoming fields to the web service request structure.

For example, you want to update contact information in Workday. You create a Target transformation and select the Workday connection. You select the Maintain_Contact_Info operation. You map the incoming fields to the request of the Workday operation.

Web service operations for targets

In a Target transformation, the operation determines the set of data that you can pass to a web service.

When you define the target properties for a web service connection, you select the web service operation. The available operations are determined by the connection. For example, for a Workday connection, Update_Job_Posting is an operation.

Field mapping for web service targets

You can map fields that are in a relational structure to the hierarchical structure used by a web service target.
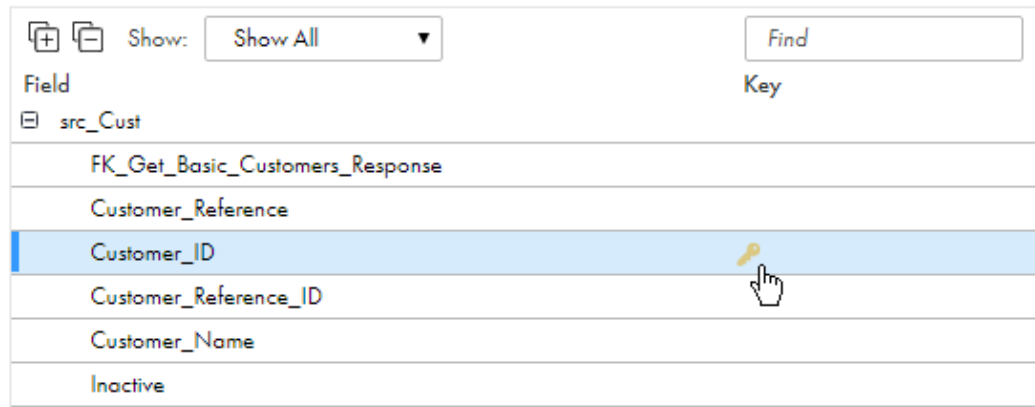
On the **Field Mapping** tab for the Target transformation, the fields shown in the **Target Fields** area are shown in a hierarchical structure. The target fields are determined by the request message structure of the operation selected for the Target transformation.

Each source object displays as a group in the **Input Fields** area. You can select fields in the **Input Fields** area to map the fields to the web service request. If the input fields include multiple input groups, map the groups to the corresponding nodes in the web service request. You need to map all of the fields that are required in the web service request.

If you map multiple source objects, you must assign primary and foreign keys to at least two groups. Ensure that the source data is sorted on the primary key for the parent object, and sorted on the foreign key and primary key for child objects.
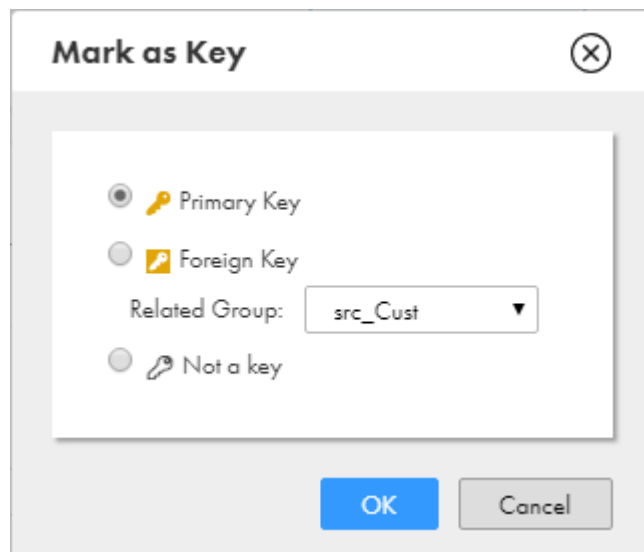
To assign keys, click the key icon next to a field that you want to be a primary or foreign key, as shown in the following image:

In the **Mark as Key** dialog box, you can assign a field as primary or foreign key and select the related group, as shown in the following image:

Keys must be of data type Bigint or String.

Partitions

In advanced mode, when you load data to some types of partitioned targets that you create at runtime, you can configure the partition key fields. For some target types, you can use partitions to optimize loading data to the target.

You can configure partition key fields and the partitioning method on the **Partitions** tab. The **Partitions** tab is displayed for targets in advanced mode.

Partition key fields

When you load data to certain types of partitioned targets that you create at runtime, you can configure the fields to be used as partition keys. You might need to configure partition key fields when you write data to complex file targets.

For example, you can create a mapping that loads data to an Amazon S3 V2 target that you create at runtime. The target is a partitioned Hive table that is backed by Avro data files. You want to write the data files in directories that are partitioned based on the columns YEAR, MONTH, and DAY. Configure the fields YEAR, MONTH, and DAY as partition keys.

Configure the fields to be used as partition keys in the Partition Fields area on the **Partitions** tab. You can add, delete, and change the order of the partition key fields.

For more information about configuring partition key fields for different target types, see the help for the appropriate connector.

Partitioning methods

If a mapping task loads large data sets, the task can take a long time to load data. When you use multiple partitions, the mapping task divides data into partitions and loads the data in each partition concurrently, which can optimize performance. Not all target types support partitioning.

If a target in advanced mode supports partitioning, you can select the partitioning method in the Parallel Processing area on the **Partitions** tab. The partitioning methods that you can select vary based on the target type. For more information about partitioning different types of targets, see the help for the appropriate connector.

You can select one of the following partitioning methods based on the target type:

None

> The mapping task loads all data in a single partition. This is the default option.

Fixed

> The mapping task distributes rows of data based on the number of partitions that you specify. You can specify up to 64 partitions.

> Consider the number of records to be passed to the target to determine an appropriate number of target partitions. For a small number of records, partitioning might not be advantageous.

Pass through

> The mapping task processes data without redistributing rows among partitions. All rows in a single partition stay in the partition. Choose pass-through partitioning when you want to create additional partitions to improve performance, but do not want to change the distribution of data across partitions.

Dynamic

> The mapping task determines the optimal number of partitions to create at runtime.

Writing hierarchical data in advanced mode

You can use a Target transformation in advanced mode to write hierarchical data to complex files, such as Avro, JSON, and Parquet files.

Consider the following guidelines when you write hierarchical data:

- You must use an Amazon S3 V2 or Azure Data Lake Storage Gen2 connection to write hierarchical data. For more information, see the help for the appropriate connector.
- You must create a new target at run time.
- Do not use a parameter for the target connection or the target object.

Configuration for multibyte hierarchical data

If a mapping includes a transformation that processes hierarchical data and the data uses multibyte characters, configure the Secure Agent machine to use UTF-8.

On Windows, create the INFA_CODEPAGENAME=UTF-8 environment variable in Windows System Properties.

On Linux, set the LC_LOCALE variable to UTF-8.

Target fields

You can configure the target fields that you want to use in the data flow. You can add and remove target fields, configure how the fields are displayed, edit field metadata, and restore original fields from the target object.

Configure target fields on the **Target Fields** tab of the **Properties** panel.

You can configure target fields in the following ways:

Add and remove fields.

You can add fields to a mapping target. To add a field, click **Add Field**, and then enter the field name, type, precision, and scale.

You can also remove fields that you do not want to use in the mapping. To remove fields, select the fields that you want to remove, and then click **Delete**.

Change the sort order.

You can display target fields in native order, ascending order, or descending order. To change the sort order, click **Sort**, and select the appropriate sort order.

Use technical field names or labels.

You can display field names by label or technical field name.

To change the display option for field names, select **Options** > **Use Technical Field Names** or **Options** > **Use Labels**.

Edit field metadata.

You can edit the metadata for a field. You might edit metadata to change information that is incorrectly inferred. When you edit metadata, you can change the name, native type, native precision, and native scale, if applicable for the data type.

To edit the name or metadata for one or more fields, click **Options** > **Edit Metadata**. When you edit metadata, you can also display native names by label or technical field name. To change the display option for native names, select **Options** > **Show Technical Field Names** or **Options** > **Show Labels**.

When you change the metadata for a field, avoid making changes that can cause errors when you run the task. For example, you can usually increase the native precision or native scale of a field without causing errors. But if you reduce the precision of a field, you might cause the truncation of data.

You can't edit field metadata in mappings in SQL ELT mode.

Restore original fields from the target object.

To restore the original fields from the target object, use the **Synchronize** option. When you synchronize fields, Data Integration restores deleted target fields, reverts data type and precision changes, and adds fields that are new to the target. Data Integration removes any added fields that do not have corresponding fields in the target object.

For existing target fields, Data Integration replaces any metadata that you edited with the field metadata from the target object. Data Integration does not revert changes that you made to the **Name** field.

Target transformation field mappings

Configure field mappings in a Target transformation to define how data moves from the data flow to the target object.

Configure field mappings on the **Field Mappings** tab.

The **Field Mappings** tab includes a list of incoming fields and a list of target fields.

You can configure the following field mapping options:

Target Field Group

The target field group that lists the tables for which you want to configure field mappings. This option appears when the transformation target type is **Multi Group**.

Field Map Options

Method of mapping incoming fields to target fields. Select one of the following options:

- Manual. Manually link incoming fields to target fields. Selecting this option removes links for automatically mapped fields. To map fields manually, drag a field from the incoming fields list and position it next to the appropriate field in the target fields list. Or, you can map selected fields, unmap selected fields, or clear all of the mappings using the **Actions** menu.

- Automatic. Automatically link fields with the same name. Use when all of the fields that you want to link share the same name. You cannot manually link fields with this option.

- Completely Parameterized. Use a parameter to represent the field mapping. In the task, you can configure all field mappings.

- Partially Parameterized. Configure links in the mapping that you want to enforce and use a parameter to allow other fields to be mapped in the mapping task. Or, use a parameter to configure links in the mapping, and allow all fields and links to display in the task for configuration.
To see more information on field mapping parameters, see *Mappings*.

Options

You can configure how the fields display and which fields to display. To do so, click **Options** and select from the following display options:

- Display fields using field labels or technical field names.

- Show mapped, unmapped, or all fields.

Automap

If you want Data Integration to automatically link fields with the same name and you also want to manually map fields, select the **Manual** option and open the **Automap** menu.

You can map fields in the following ways:

- Exact Field Name. Data Integration matches fields of the same name.

- Smart Map. Data Integration matches fields with similar names. For example, if you have an incoming field `Cust_Name` and a target field `Customer_Name`, Data Integration automatically links the `Cust_Name` field with the `Customer_Name` field.

You can use both Exact Field Name and Smart Map in the same field mapping. For example, use Exact Field Name to match fields with the same name and then use Smart Map to map fields with similar names.

You can undo all automapped field mappings by clicking **Automap** > **Undo Automap**. To unmap a single field, select the field to unmap and click **Actions** > **Unmap**.

Data Integration highlights newly mapped fields. For example, when you use Exact Field Name, Data Integration highlights the mapped fields. If you then use Smart Map, Data Integration only highlights the fields mapped using Smart Map.

Configuring a Target transformation

You can use an existing target or create a target to hold the results of a mapping. If you choose to create the target, the Secure Agent creates the target when you run the task.

1. Select the Target transformation in the mapping.
2. On the **General** tab, enter a target name and optional description
3. On the **Incoming Fields** tab, configure field rules that specify the fields to include in the target

   For more information about field rules, see .
4. On the **Target** tab, select the target connection and specify the target type, for example, **Single Object** or **Parameter**.

   For mappings in SQL ELT mode, you select the target connection in the mapping properties.

   The option to select multiple objects is available for NetSuite connections only.
5. To use an input parameter for the target object, select an existing parameter, or click **New Parameter** and create a new parameter for the target object.
6. To use an existing target object, either enter the object name or click **Select** and select the target object.

7. To create a new target object, perform the following steps:

    a. Click **Select**.

    b. In the **Target Object** dialog box, select **Create New at Runtime**:
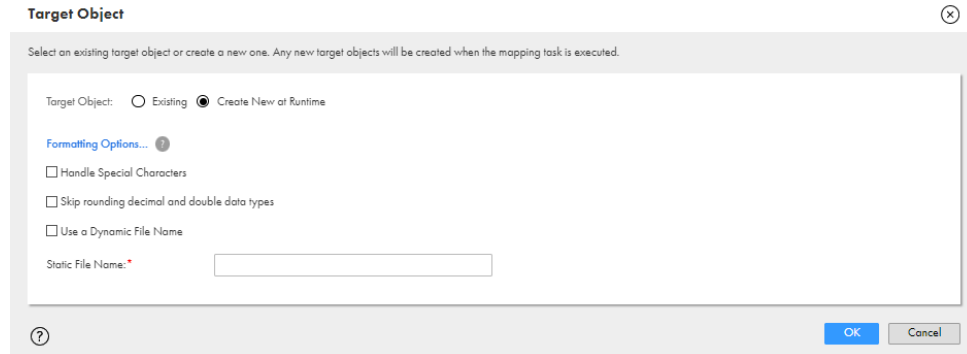
    c. For flat file targets, enter the name of the target file including the extension, for example, `Accounts.csv`.

        If you want the file name to include a time stamp, select **Handle Special Characters** and add special characters to the file name, for example, `Accounts_%d%m%y%T.csv`.

        If you want to use a dynamic file name, select **Use a Dynamic File Name** and configure the file name expression.

        If you're using a fixed-width flat file format that uses the Decimal or Double data type with precision and scale, and you don't want the data to be rounded in the target file, select **Skip rounding decimal and double data types**.

    d. For relational targets, enter the table name.

    e. For cloud data warehouse targets, optionally choose to use exact field names in the new target object, enter the object name, and enter other properties such as the table location, table type, and path, if required.

        For more information about the target properties for different connection types, see the help for the appropriate connector.

    f. Click **OK**.

8. To configure formatting options for flat file targets, click **Formatting Options**, and configure the formatting options such as the delimiter character and text qualifier.

9. For relational targets, select the target operation and related properties such as whether to truncate the table for Insert operations.

    If you create a relational target at run time, the target operation is always Insert.

10. Specify advanced properties for the target, if required.

    Advanced properties vary based on the connection type. For information about connector properties, see the help for the appropriate connector.

11. Configure the target fields on the **Target Fields** tab.

    You can edit field names and metadata, add fields, and delete unnecessary fields.

    You can't edit field metadata in mappings in SQL ELT mode. If you create the target at run time, you can't configure target fields.

12. Map incoming fields to target fields on the **Field Mapping** tab.

    If you create the target at run time, fields are mapped automatically.

    For more information about field mapping, see "Target transformation field mappings" on page 90.

CHAPTER 4

Access Policy transformation

The Access Policy transformation applies data de-identification policies and data filter policies created on the Data Access Management page in Data Governance and Catalog according to the properties of the Access Policy transformation. Data access policies are a set of policies and associated data de-identification rules and data filter rules that apply data protections and filters that transform and filter the data accordingly.

Data access policies can replace, transform, or redact values in a data set while maintaining the overall usefulness of the data. A data access policy can protect different values in different mappings, based on factors such as the intended user of the data and metadata classifications that users assign to the source data. Data access policies can help your organization comply with data privacy regulations such as the European General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA).

Data de-identification rules apply pre-defined data protections to data element classifications. A data element classification is a categorization applied to fields within data assets to indicate the category of data such as birth dates, national identifiers, and postal codes.

Rules in an access policy can apply multiple data de-identification techniques, including the following operations:

- Retaining data
- Redacting all values of a given type such as birth dates
- Replacing specified field values with NULL
- Truncating values such as redacting the first three characters of a postal code
- Replacing values with consistently tokenized values such as always replacing "Smith" with "Abcd" or "1234" with "5678"
- Generalizing date values to the month, year, or decade
- Replacing values with a constant text value such as replacing all passwords with five asterisks
- Substituting values with values stored in a file
- Hashing values with standard algorithms

Data filter policies limit, filter, or otherwise restrict user access to records within a data asset.

Data filter rules apply pre-defined filters that control access to rows or records of data.

Rules in data filter policies can apply multiple data filters based on the following attribute types:

- Asset term
- Data element term
- Data element classification
- Data entity classification
- Order usage context
- User group

An Access Policy transformation doesn't display the data access policies, since those are dynamically applied based on the data and metadata. Users with the appropriate permissions manage data access policies on the Data Access Management page in Data Governance and Catalog.

Note: In order to use Access Policy transformations in your mappings, your organization must have Data Governance and Catalog in use.

Using parameters in Access Policy transformations

You can use parameters as a placeholder for a value or values in an Access Policy transformation. You define the value of the parameter when you configure the mapping task. To learn more about parameters, see *Using parameters in a mapping*.

Use the following types of IDs when using parameters in an Access Policy transformation:

- The Consumer User field accepts IDs of users with the Data Marketplace User role.

- The Usage Context field accepts IDs of usage contexts available in Data Marketplace.
  For more information on how to fetch user IDs and usage context IDs from Data Marketplace, see "Manage requests, orders and consumer accesses" in the Data Marketplace help.

- The Data Asset field accepts IDs of Data Governance and Catalog data assets.
  For more information on how to fetch asset IDs from Data Governance and Catalog, see "List assets in the catalog" in the Data Governance and Catalog help.

1. Click the Access Policy tab in an Access Policy transformation.

   Alternatively, open the Parameters panel.
   For more information about the Parameters panel, see the "Parameters" section in *Mappings*.

2. Click **New Parameter**.

   The New Input Parameter window appears.

The following image shows the New Input Parameter window:

3. Enter a name for the parameter.

4. Optionally enter a display label.

5. Optionally enter a description.

6. Optionally select a type such as string, connection, expression, and so on.

7. Optionally enter a default value for the parameter.

8. Click **OK**.

The parameter name appears in the property selection.

Data filter policy best practices

To effectively use data filter policies in Access Policy transformations, use the following guidelines as best practices.

In order to provide flexibility for a variety of use cases, the Access Policy transformation appends a new field called access_policy_filter that indicates whether a row is affected by data filter policies. For most use cases, it is appropriate to filter out these rows and the access_policy_filter field from the output.

Use the following best practices when defining Access Policy transformations that include data filter policies:

- Add a Filter between the Access Policy transformation and the Target.
  - On the Incoming Fields tab, includes all fields.
  - On the Filter tab, add a simple filter condition for the field name access_policy_filter with a value of ACCESS_DENIED.
    The following image shows the Filter tab:
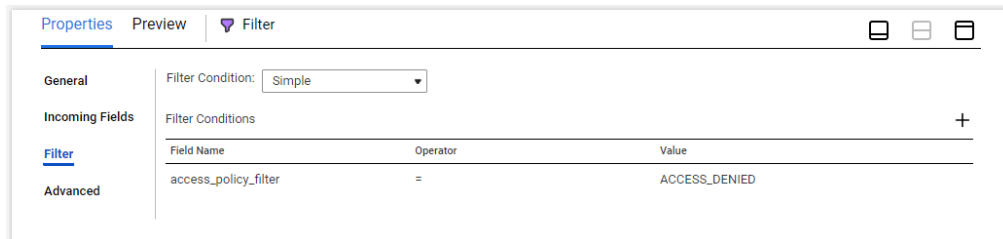
- Select the Target in your mapping.
  - On the Incoming Fields tab, exclude the field named access_policy_filter.
    The following image shows the Incoming Fields tab for the Target:
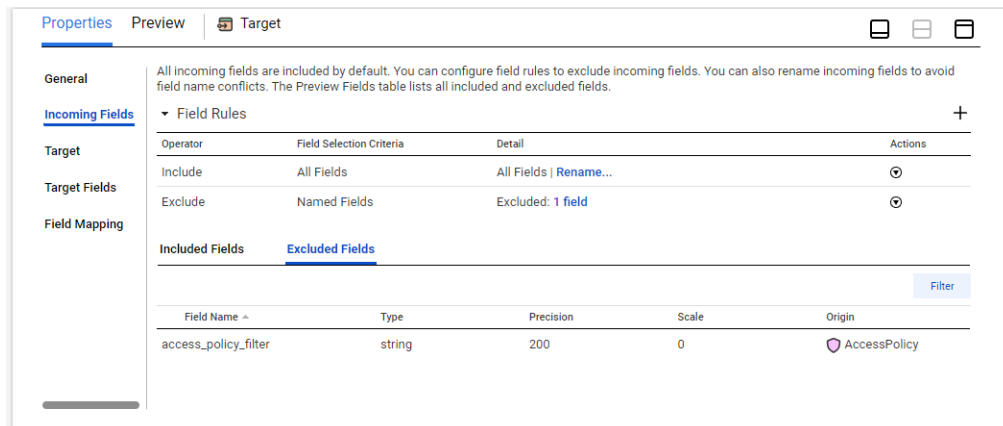
Access Policy transformation configuration

When you configure an Access Policy transformation, you select a data asset that associates policies that you create on the Data Access Management page in Data Governance and Catalog with the source data set.

A data asset is a data structure such as a table in a database that you have cataloged in Data Governance and Catalog. You select the data asset when you configure the properties on the Access Policy tab.

Use a single Source transformation with an Access Policy transformation. Connect the Access Policy transformation directly to the Source transformation.

The following image shows where to place an Access Policy transformation and its configuration options:

The following table describes the fields on the **Access Policy** tab:

Field	Description
Consumer	Identifies the user who will consume the data. Select a single user.
	Active users in your organization with the Data Marketplace User role appear in this list. Select one of those users. Note that the user you choose helps determine the set of rules that the transformation applies to the data.
Usage	Identifies the usage context such as for customer analysis or anti-money laundering research. Usage context is one type of metadata that helps determine the set of rules that the transformation applies to the data.
	Users create and manage usage contexts in Data Marketplace. To learn more, see *Set Up Data Marketplace* in the Data Marketplace help.
Data Asset	Identifies the data asset to which the transformation applies the rules.
	A data asset is a data structure such as a table in a database that you have cataloged in Data Governance and Catalog.
	By default, the source object that you selected when you configured the Source transformation appears when you click **Select**. Select the data asset in that source object.
	Alternatively, you can select a new connection and a new data asset.
	Select **Override Asset name** if there is a difference between the table name and the asset name in the data catalog.
Consistency Seed	Provides a value that represents the rule logic that the transformation applies to the data element classification.
	A consistency seed ensures that a rule's behavior for a data element classification will be consistent for the same consistency seed value, usage context, and user across mappings. For example, you can ensure that a rule always replaces the last name "Smith" with "Abcd."
	If you want consistent tokenization between different Access Policy transformations, insert the same consistency seed in other Access Policy transformations. If you don't want consistent tokenization between different Access Policy transformations, generate a unique seed.

Click **Synchronize Access Policy** and save the mapping to ensure that all required design time properties are configured. This ensures that the appropriate data access policy rules resolve correctly when the mapping runs.

Access Policy transformation example

You need to make customer data available to a colleague in your bank's marketing department for customer analysis. As part of their role and the needs of their project, your colleague in the marketing department has limited access rights to customer data.

Use an Access Policy transformation to ensure that your organization complies with data privacy regulations while allowing your colleague to gain valuable insights. At run time, the transformation hides or obscures the information for which your colleague does not have access rights based on metadata and the properties that you set on the Access Policy tab.

The following table shows part of a table called BANK_CUSTOMERS, which includes contact details, birth dates, and other items of personally identifiable information:

CUSTOMER_NAME	ADDRESS	POSTAL_CODE	PROFESSION	EMAIL_ADDRESS	CUSTOMER_TYPE	DAY_OF_BIRTH
Brayden Combs	7776 Proctor Drive	48089	Associate Marketing	brayden.combs@organization.com	1	03/14/1990
Osvaldo Patton	63 Rosewood Court	64051	Project Manager	osvaldo.patton@organization.com	5	01/13/1988
Alivia Oneal	64 East Thorne Court	30043	Business Analyst	alivia.oneal@organization.com	3	06/22/1956
Skyla Weeks	7556 South Shore Drive	48045	Operations Manager	skyla.weeks@organization.com	4	10/08/1961
Justice Cobb	9136 S. Lafayette St.	60187	Financial Controller	justice.cobb@organization.com	1	02/23/1963
Braxton Kramer	7 Hill Field St.	28376	Restaurant Manager	braxton.kramer@organization.com	5	11/01/1969
Erin Ryan	20 Lakeshore Street	20706	Project Manager	erin.ryan@organization.com	2	07/23/1959

To configure the mapping, complete the following tasks:

1. Add a Source transformation that reads the BANK_CUSTOMERS source table.
2. Add an Access Policy transformation to the mapping canvas, and connect it to the data flow.
3. On the **Access Policy** tab, take the following actions:
   a. In the **Consumer** field, select your marketing department colleague.
   b. In the **Usage** field, select "customer analysis" as the usage context.

Users create and manage usage context in Data Marketplace. To learn more, see *Working With Data Collections* in the Data Marketplace help.

    c.   In the **Data Asset** field, select the BANK_CUSTOMERS table as the source data asset.

    d.   To use consistent tokenization, enter a consistency seed from another Access Policy transformation. Otherwise, generate a new consistency seed.

    e.   Click **Synchronize Access Policy**.

4.   Add other transformations to the mapping as required.

5.   Add a Target transformation to the mapping and connect it to the upstream transformation.

6.   Click **Run**.

    The mapping task applies the policies to the data, protecting it.

The following table shows the protected data:

CUSTOMER_ NAME	ADDR ESS	POSTAL_C ODE	PROFESSIO N	EMAIL_ADDRE SS	CUSTOMER_T YPE	DAY_OF_BI RTH	acces s_poli cy_filt er
							ACCES S_DEN IED
Osva		64051	Project Manager	yojniw.sdfhihd @organization. com	5		
Aliv		30043	Business Analyst	25dxk7og.x9jn eyc@organizati on.com	3		
Sky		48045	Operations Manage	oeunx.ed7uncr @organization. com	4		
							ACCES S_DEN IED
Brax		28376	Restaurant Manager	zdkljvkr.xbei@o rganization.co m	5		
Erin		20706	Project Manager	0ick.39ft@orga nization.com	2		

The Access Policy transformation protects the data in the following ways:

- It retains only the first four characters of the customer names.
- It replaces the street address and day of birth data with null values.
- It assigns random characters to the customer email addresses, but it keeps the addresses in a valid email format.
- It removes the data in the first and fifth rows.

- It adds a new access_policy_filter field that indicates whether data filter policies have denied access to the data.
  To learn how to filter out that field, see "Data filter policy best practices."

The ZIP codes, professions, and customer types remain unaltered.

CHAPTER 5

Aggregator transformation

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

The Aggregator transformation behaves like the Expression transformation except you can configure the Aggregator transformation to perform calculations on a group of data. The Expression transformation returns results on a row-by-row basis.

For example, you can use the Aggregator transformation to calculate the average salary for employees in each department of an organization. In the Aggregator transformation, create a group for the department number and then configure an expression to calculate the average salary for the employees in each group.

Group by fields

Use group by fields to define how to group data for aggregate expressions. Configure group by fields on the **Group By** tab of the **Properties** panel.

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

When you group values, Data Integration produces one row for each group. If you do not group values, Data Integration returns one row for all input rows.

If the Aggregator transformation runs on an advanced cluster and the input and output fields aren't grouped by field, the transformation might not return the last row of each group with the result of the aggregation.

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

For example, you create aggregate fields called TOTAL_QTY and TOTAL_PRICE to store the total quantity and total price for each item by store. You define the following expressions for each field:

- TOTAL_QTY: `SUM (QTY)`
- TOTAL_PRICE: `SUM (QTY*PRICE)`

You define STORE_ID and ITEM as the group by fields.

The input rows contain the following data:

STORE_ID	ITEM	QTY	PRICE
101	'battery'	3	2.99

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

Data Integration performs the aggregate calculations on the following unique groups:

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

Data Integration returns the store ID, item name, total quantity for each item by store, and total price for each item by store:

STORE_ID	ITEM	TOTAL_QTY	TOTAL_PRICE
101	'AAA'	2	4.90
101	'battery'	6	17.34
201	'battery'	5	8.35
301	'battery'	1	2.45

Sorted data

To improve job performance, you can configure an Aggregator transformation to use sorted data. To configure the Aggregator transformation to process sorted data, on the **Advanced** tab, select **Sorted Input**.

When you configure an Aggregator transformation to use sorted data, you must sort data earlier in the data flow. If the Aggregator transformation processes data from a relational database, you must also ensure that the sort keys in the source are unique. If the data is not presorted correctly or the sort keys are not unique, you can receive unexpected results or errors when you run the mapping task.

When the mapping task performs aggregate calculations on sorted data, the task caches sequential rows of the same group. When the task reads data for different group, it performs aggregate calculations for the cached group, and then continues with the next group.

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

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

When you do not use sorted data, the mapping task performs aggregate calculations after it reads all data.

Aggregate fields

Use an aggregate field to define aggregate calculations.

When you configure an Aggregator transformation, create an aggregate field for the output of each calculation that you want to use in the data flow. You can use aggregate functions in aggregate fields. You can also use conditional clauses and nonaggregate functions.

Configure aggregate fields on the **Aggregate** tab of the **Properties** panel. When you configure an aggregate field, you define the field name, data type, precision, scale, and optional description. The description can contain up to 4000 characters. You also define the calculations that you want to perform.

When you configure aggregate fields, you can use variable fields for calculations that you want to use within the transformation. You can also include macros in aggregate and variable fields.

In advanced mode, the output is NULL if the Group by field returns a single row and the aggregate expression contains the STDDEV and VARIANCE functions. This is because Data Integration uses Spark 3.2. To get an output value of 0, set the `spark.sql.legacy.statisticalAggregate` session property to true in the mapping task.

Aggregate functions

You can use aggregate functions in expressions in aggregate fields.

For more information about aggregate functions, see *Function Reference*.

In mappings in SQL ELT mode, you use your cloud data warehouse's native aggregate functions. For more information, see the documentation for your cloud data warehouse.

Nested aggregate functions

A nested aggregate function is an aggregate function within another aggregate function.

For example, the following expression sums sales and returns the highest number:

```
MAX( SUM( SALES ))
```

You can include multiple single-level or multiple nested functions in different output fields in an Aggregator transformation. You cannot include both single-level and nested functions in an Aggregator transformation. You cannot nest aggregate functions in advanced mode.

When an Aggregator transformation contains a single-level function in any output field, you cannot use a nested function in any other field in that transformation. If you need to create both single-level and nested functions, create separate Aggregator transformations.

Conditional clauses

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

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

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

Advanced properties

You can configure advanced properties for an Aggregator transformation. The advanced properties control settings such as the tracing level for session log messages, whether the transformation uses sorted input, cache settings, and whether the transformation is optional or required.

The properties that are available vary based on the mapping mode.

You can configure the following properties:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Sorted Input	Indicates that input data is pre-sorted by groups. Select this option only if the mapping passes sorted data to the Aggregator transformation.
Cache Directory	Local directory where Data Integration creates the index and data cache files. By default, Data Integration uses the directory entered in the Secure Agent $PMCacheDir property for the Data Integration Server. If you enter a new directory, make sure that the directory exists and contains enough disk space for the aggregate caches.
Data Cache Size	Data cache size for the transformation. Select one of the following options: - Auto. Data Integration sets the cache size automatically. If you select Auto, you can also configure a maximum amount of memory for Data Integration to allocate to the cache. - Value. Enter the cache size in bytes. Default is Auto.
Index Cache Size	Index cache size for the transformation. Select one of the following options: - Auto. Data Integration sets the cache size automatically. If you select Auto, you can also configure a maximum amount of memory for Data Integration to allocate to the cache. - Value. Enter the cache size in bytes. Default is Auto.

Property	Description
Transformation Scope	Specifies how Data Integration applies the transformation logic to incoming data. Select one of the following options: - Transaction. Applies the transformation logic to all rows in a transaction. Choose Transaction when a row of data depends on all rows in the same transaction, but does not depend on rows in other transactions. - All Input. Applies the transformation logic on all incoming data. When you choose All Input, Data Integration drops incoming transaction boundaries. Choose All Input when a row of data depends on all rows in the source.
Optional	Determines whether the transformation is optional. If a transformation is optional and there are no incoming fields, the mapping task can run and the data can go through another branch in the data flow. If a transformation is required and there are no incoming fields, the task fails. For example, you configure a parameter for the source connection. In one branch of the data flow, you add a transformation with a field rule so that only Date/Time data enters the transformation, and you specify that the transformation is optional. When you configure the mapping task, you select a source that does not have Date/Time data. The mapping task ignores the branch with the optional transformation, and the data flow continues through another branch of the mapping.

Hierarchical data in advanced mode

In advanced mode, you can use hierarchical fields that represent an array, map, or struct.

You can use hierarchical fields as pass-through fields. You can also use complex operators to access a primitive child field and use the child field to perform an aggregate calculation. For example, you can use the dot operator to access an integer in a struct field and pass the integer as an argument to the SUM function. For more information about complex operators, see *Function Reference*.

Consider the following guidelines when you use hierarchical fields:

- You cannot use a hierarchical field as a group by field.
- The output of an aggregate calculation cannot be an array, map, or struct.

Aggregator transformation example

You need to retrieve the latest price for every stock in your change data capture (CDC) system.

To retrieve the most recent price for each stock, create a mapping and add an Aggregator transformation and Joiner transformation.

- Configure the Aggregator transformation to group all the stocks by stock ID and calculate the latest trading time for each stock ID.
- Configure the Joiner transformation to perform a self join with the stock ID and trading time, using the source data and Aggregator transformation output.

The Target transformation writes the stock ID and price to the target file.

Source data

The following table shows the source data:

STOCK_ID	PRICE	VOLUME	TRADETIME
100	1.12	125	2023.01.20.1245
105	3.45	56	2023.01.18.0821
102	2.29	25	2023.01.19.1118
101	4.56	87	2023.01.17.0901
102	2.35	38	2023.01.20.1649
100	0.99	94	2023.01.20.1355
104	1.88	67	2023.01.20.1730
103	2.11	41	2023.01.17.1211
105	3.40	90	2023.01.18.1525
104	3.40	90	2023.01.20.1634
103	3.40	90	2023.01.20.1526
105	3.40	90	2023.01.19.1745

Mapping configuration

The following image shows the mapping that you configure:
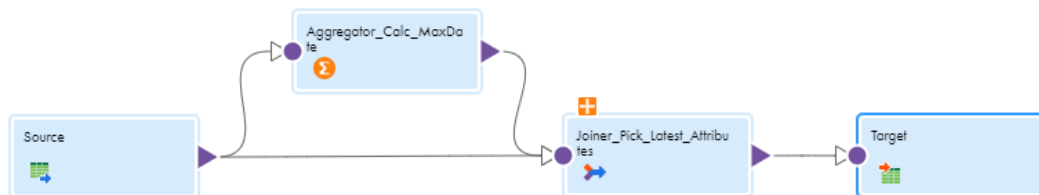
Configure the transformations in the following ways:

Aggregator transformation

The following table describes the properties to configure in the Aggregator transformation:

Properties	Configuration
Incoming Fields	Configure a field rule to prefix all incoming fields with "`agg_`" so that the aggregated fields have unique names.
Group By	Group by the field `agg_stock_id`.
Aggregate	Create one aggregate field with the following properties to return the latest trading time for each stock ID: - Field Type: Output Field - Name: agg_maxtime - Type: string - Precision: 10 Configure the following expression: `max(agg_tradetime)`

Joiner transformation

In the Joiner transformation, configure a normal join with the following join conditions:

- `agg_stock_id = STOCK_ID`
- `agg_maxtime = TRADETIME`

Target transformation

The following table describes the properties to configure in the Target transformation:

Properties	Configuration
Incoming Fields	Configure the following field rules: - Include all fields. - Exclude **Fields by Text or Pattern with the prefix "agg_"**. The list of included fields shows all the source fields: PRICE, STOCK_ID, TRADETIME, VOLUME
Target Fields	Add the following target fields: STOCK_ID and PRICE
Field Mapping	Perform an automap with **Exact Field Name**. This maps the incoming fields STOCK_ID and PRICE to the target fields STOCK_ID and PRICE.

Target Data

The following table shows the data that the mapping writes to the target file:

```
STOCK_ID          PRICE

100               0.99

101               4.56

102               2.35

103               3.40

104               3.40

105               3.40
```

Since the fields are grouped by STOCK_ID in the Aggregator transformation, each stock ID is listed in one row with its most recent price. For example, the most recently traded price for stock 105 is $3.40.

CHAPTER 6

B2B transformation

Use the B2B transformation that is available in Data Integration mappings to communicate with B2B Gateway and send commands such as updating an event status. With the **Update Event Status** command type you can select a dynamic event status name that is configured in the field mapping or select an event status name that is configured in B2B Gateway.

B2B Incoming Fields

An incoming field is a field that enters a B2B transformation from an upstream transformation.

For more information about incoming fields in transformations, see "Incoming fields" on page 25.

B2B settings

Configure B2B settings to either map a dynamic event status name based on the field names that are configured on the **Mapping Fields** tab or select a status that is defined in B2B Gateway.

To configure the transformation, complete the following tasks:

1. Connect the B2B transformation to a Source transformation.
2. On the **B2B Settings** tab, perform the following actions:
   - Select the type as **Update Event Status**.
   - In the **Event Status** field, you can either map a dynamic event status name based on the field names that are configured on the **Mapping Fields** tab or select a status that is defined in B2B Gateway.

Output fields

View the output fields on the **Output Fields** tab of the **Properties** panel. A B2B transformation creates output fields based on the fields that you map on the **Field Mapping** tab.

You can also define new fields and select from an input groups to be displayed in **Output Fields**. The tab displays the following fields:

- ERROR_CODE

- ERROR_MESSAGE
- EVENT_ID
- STATUS_NAME

You can only delete the ERROR_CODE and ERROR_MESSAGE fields from the list of default output fields. You can't delete the EVENT_ID and the STATUS_NAME entries.

When you define output fields, note the following:

- When you add an output field, you define the field name, data type, precision, scale, and an optional description. You can edit the data type and precision of the output fields. The description can contain up to 4,000 characters.
- If you connect the B2B transformation to another step that does not pass in any field, the output fields are not initialized.
- At run time, the mapping passes null values to output fields that are not in a field mapping.

Field mapping

Configure field mappings to define how the event statuses must move from the incoming transformation fields to the downstream trasnformation. Configure field mappings on the **Field Mapping** tab of the **Properties** panel.

The **Field Mapping** tab includes a list of incoming fields and B2B input fields. In B2B input fields, EVENT_ID will be the default field. The STATUS_ NAME field appears when you select **Pass the status as a parameter** in the B2B Settings tab. You can also include additional entries in the B2B input fields by adding a new field or by selecting from input group on the **Output fields** tab.

Field Map Options

Defines the method of mapping fields to the B2B transformation. Select one of the following options:

- Manual. Manually link incoming fields to the B2B transformation input fields.
- Automatic. Automatically link fields with the same name. Use when all of the fields that you want to link share the same name. You can't manually link fields with this option.

Automap

Links fields with matching names. Allows you to link matching fields, and then manually configure other field mappings.

You can map fields in the following ways:

- Exact Field Name. Data Integration matches fields of the same name.
- Smart Map. Data Integration matches fields with similar names. For example, if you have an incoming field `Cust_Name` and a target field `Customer_Name`, Data Integration automatically links the `Cust_Name` field with the `Customer_Name` field.

You can use both Exact Field Name and Smart Map in the same field mapping. For example, use Exact Field Name to match fields with the same name and then use Smart Map to map fields with similar names.

You can undo all automapped field mappings by clicking **Automap** > **Undo Automap**. To unmap a single field, select the field to unmap and click **Actions** > **Unmap**.

Data Integration highlights newly mapped fields. For example, when you use Exact Field Name, Data Integration highlights the mapped fields. If you then use Smart Map, Data Integration only highlights the fields mapped using Smart Map.

Options

Controls how fields are displayed in the **Incoming Fields** and **Output Fields** lists. Configure the following options:

- The fields that appear. You can show all fields, unmapped fields, or mapped fields.
- Field names. You can use technical field names or labels.

Actions menu

Provides additional field mapping options. You can select the following options:

- Map Selected. Links the selected incoming field with the selected B2B input field.
- Unmap Selected. Clears the link for the selected field.
- Clear Mapping. Clears all field mappings.

Advanced settings

You can configure advanced properties for a B2B transformation. The advanced properties control the tracing level for session log messages.

You can configure the following property:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Optional	Determines whether the transformation is optional. If a transformation is optional and there are no incoming fields, the mapping task can run and the data can go through another branch in the data flow. If a transformation is required and there are no incoming fields, the task fails.

CHAPTER 7

Chunking transformation

In advanced mode, the Chunking transformation splits large pieces of text into smaller segments, or chunks. This helps to increase the content's relevance before the Target transformation writes the embeddings and metedata to a vector database.

Pass output from a Chunking transformation to a Vector Embedding transformation to create vector embeddings for the text. For more information, see Chapter 38, "Vector Embedding transformation" on page 438.

When you configure a Chunking transformation, choose a chunking method based on the text you want to split.To select the method, consider the following factors:

- Whether the text comes from a long document or short messages
- The length and complexity of user queries to the large language model (LLM)
- The use case of the application that uses the LLM, such as semantic search, question answering, or summarization

Note: The Chunking transformation can't run in a serverless runtime environment or on GPUs. If the transformation runs on a GPU-enabled cluster, GPUs are disabled and the transformation consumes CPUs.

Chunking methods

The Chunking transformation offers different chunking methods that you can use to split text into chunks.

You can choose one of the following chunking methods to chunk one input string field at a time:

Sentence splitting

Split the text by sentences. Sentence splitting is useful for accurately analyzing and understanding text structure to summarize and analyze content. Sentence boundaries are detected by punctuation and through formal list items. For example, list items that begin with letters or numbers, such as *A)* or *1.*, are counted as individual sentences regardless of whether or not they have ending punctuation. Alternatively, list items that begin with bullet points or en dashes aren't automatically detected as individual sentences and are chunked based off of punctuation.

For example, the text "The quick brown fox jumps over the lazy dog. It is a sunny day." is split into the following sentence chunks:

```
"The quick brown fox jumps over the lazy dog.", "It is a sunny day."
```

Word splitting

Split the text based on a configured number of words. Word splitting is useful for preserving word meanings and relationships, like names or locations, without aligning to sentence boundaries. Set a word chunking limit that's likely to capture common phrases or full sentences.

For example, the text "The quick brown fox jumps over the lazy dog. It is a sunny day." is split into the following chunks of five words each:

```
"The quick brown fox jumps", "jumps over the lazy dog.", "dog. It is a sunny",
"sunny day."
```

Fixed length character splitting

Split the text based on a configured fixed number of characters. This method might split text in the middle of sentences, but it preserves whole words. Fixed length character splitting is useful for scenarios when you want to ensure near constant chunk sizes.

For example, the text "The quick brown fox jumps over the lazy dog. It is a sunny day." is split into the following chunks of 10 characters each:

```
"The quick ", " quick ", " brown fox", "fox jumped", "jumped ", " over the ", " lazy
dog.", "It is a ", " sunny day", "day."
```

Word splitting and fixed length character splitting create a 10 percent overlap between chunks to preserve whole words and sentences. The Chunking transformation rounds up when calculating the overlap. For example, 10 percent of a word chunking limit that's set to five words equals a 0.5 word overlap, so the transformation creates a one word overlap.

Chunking output fields

The Chunking transformation output fields include a **ChunkedText** field that stores the chunks.

As an active transformation, the Chunking transformation delivers an output that's usually equal to or more than the number of rows you input into the transformation. Empty or null rows aren't factored into the splitting methods and don't appear in the **ChunkedText** field.

You can pass the chunks to a Vector Embedding transformation to create vector embeddings.

CHAPTER 8

Cleanse transformation

The Cleanse transformation adds a cleanse asset that you created in Data Quality to a mapping. A cleanse asset is a set of data transformation operations that standardize the form and content of your data.

You add a single cleanse asset to a Cleanse transformation. You can map one or more input fields to a cleanse asset.

A cleanse asset can perform one or more of the following operations:

- Change the character case of the input data.
- Remove leading and trailing spaces from input data.
- Remove values from the input data.
- Find and replace values in the input data.
- Merge the cleansed data from two or more input fields into a single new output field.

You can configure multiple operations in a cleanse asset, and you can add any type of operation to the asset multiple times. The mapping performs the operations on an input data field in a sequence that you define, so that a single cleanse asset can specify multiple changes to the input field data.

A Cleanse transformation is similar to a Mapplet transformation, as it allows you to add data transformation logic that you designed elsewhere to a mapping. Like mapplets, cleanse assets are reusable assets.

A Cleanse transformation does not display the logic that the cleanse asset contains or allow you to edit the cleanse asset. To edit the cleanse asset, open it in Data Quality.

Cleanse transformation configuration

When you configure a Cleanse transformation in a mapping, you first select the cleanse asset that contains the logic to include in the transformation. Next, you map one or more incoming fields on the transformation to target fields that cleanse asset specifies.

The steps to configure the transformation depend on the number of inputs that the cleanse asset specifies.

To configure the transformation, complete the following tasks:

1. Connect the Cleanse transformation to a Source transformation or other upstream object.
2. On the **Cleanse** tab, select the cleanse asset that you want to include in the transformation.

   You can select a cleanse asset that contains a single input or multiple inputs.

   - If you include a cleanse asset that contains a single input, you can select one or more fields from upstream objects as inputs. To map the fields to the cleanse asset, click **Add**.

The following image shows the options that you use to select the cleanse asset and add the input fields:
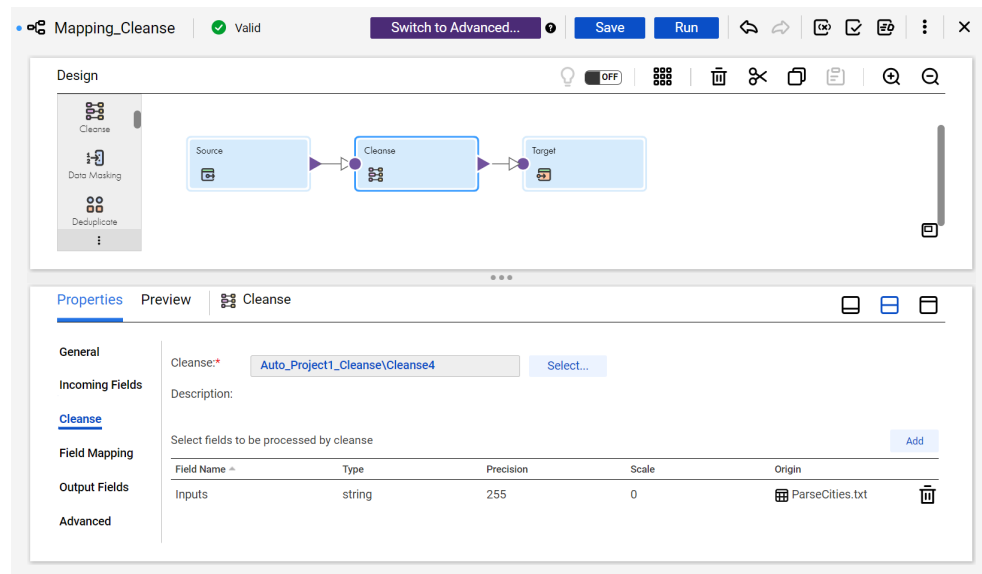
- If you include a cleanse asset that contains multiple inputs, the Add option does not appear on the Cleanse tab. Use the Field Mapping tab options to connect the transformation input fields to the asset.

3. On the **Incoming Fields** tab, verify the incoming fields.

   By default, the transformation inherits all incoming fields from any connected upstream object in the mapping. You can define a field rule to limit or rename the incoming fields.

4. On the **Field Mapping** tab, connect one or more input fields on the transformation to the asset. If the cleanse asset specifies a single input, Data Integration automatically links the incoming fields with the target field. If the cleanse asset specifies multiple inputs, link the incoming fields to the fields manually.

   The cleanse asset input names might reflect the names of the transformation input fields. If so, you can use the **Automap** options to connect the fields.

The following image shows the options that you can use to map the input fields in the transformation:
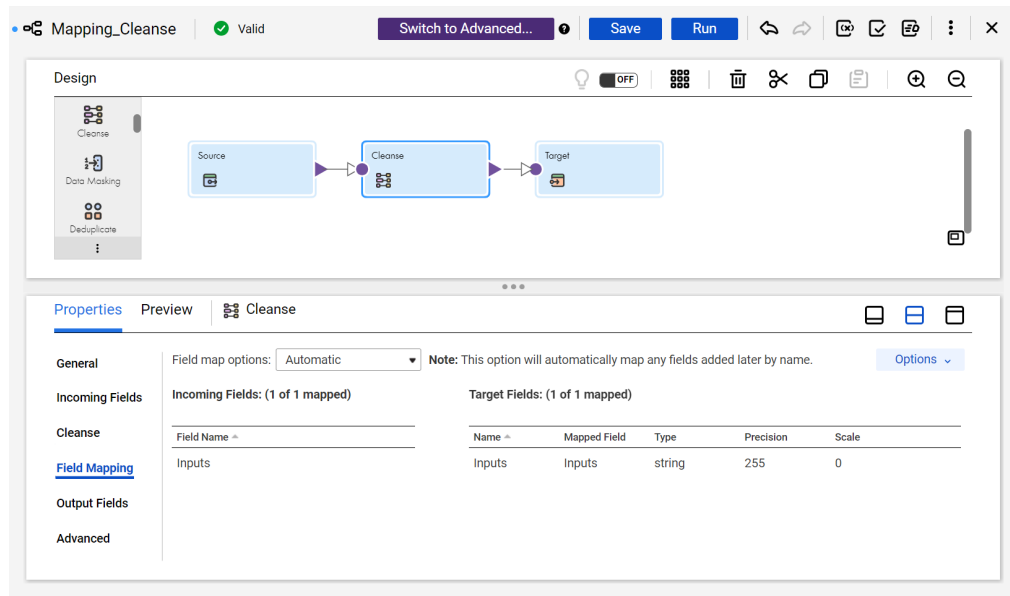
5. Verify the output field properties on the **Output Fields** tab.

6. You can optionally rename the Cleanse transformation and add a description on the **General** tab. You can also update the tracing level for the transformation on the **Advanced** tab. The default tracing level is Normal.

Cleanse asset considerations

A cleanse asset comprises one or more cleanse instances. Each instance represents one or more input fields for which the asset specifies a set of cleanse operations.

You can add multiple cleanse instances to an asset in Data Quality, so that a Cleanse transformation can apply multiple cleanse operations to different sets of input fields with a single asset.

If you select a cleanse asset that specifies more than one input field, you must understand the asset configuration and know the types of cleanse operation that the asset will perform on each input. The Cleanse transformation does not identify the instance on which each asset input originates. If you define multiple instances on the asset in Data Quality, make a record of the instances to which each input belongs. Use the record as a guide when you connect the asset inputs to the transformation input fields.

Synchronizing data quality assets

If you update an asset in Data Quality after you add it to a transformation, you may need to synchronize the asset version in the transformation with the latest version.

To synchronize the asset versions, open the transformation in the mapping and select the transformation name in the properties panel. For example, in a Cleanse transformation select **Cleanse** in the properties panel. If synchronization is necessary, Data Integration displays a message that prompts you to synchronize the assets.

When you synchronize the asset versions, Data Integration may prompt you to propagate the field attributes of the current asset to other assets in the mapping. Data Integration may display the prompt if the current asset originates in an earlier version of Data Quality.

You select the assets to which to propagate the field attributes. You can select multiple assets and propagate the attributes in a single operation.

Note: Field propagation occurs by default for assets that you create in the current version of Data Quality.

Cleanse transformation field mappings

Configure field mappings to define how data moves from the incoming transformation fields to the cleanse asset. Configure field mappings on the **Field Mapping** tab of the **Properties** panel.

You can configure the following field mapping options:

Field map options

Method of mapping fields to the transformation.

Select one of the following options:

- Manual. Manually link an incoming field to an asset input field. Removes links for any automatically mapped field. Manual is the default option when you select an asset that contains multiple inputs.
- Automatic. Automatically link fields with the same name. You cannot manually link fields with this option. Automatic is the default option when you select an asset that contains a single input.
- Completely Parameterized. Use a parameter to represent the field mapping.
  Choose the Completely Parameterized option when the cleanse asset in the transformation is parameterized or any upstream transformation in the mapping is parameterized.
- Partially Parameterized. Configure links in the mapping that you want to enforce and use a parameter to allow other fields to be mapped in the mapping task. Or, use a parameter to configure links in the mapping, and allow all fields and links to display in the task for configuration.

Parameter

Select the parameter to use for the field mapping, or create a new parameter. This option appears when you select Completely Parameterized or Partially Parameterized as the field map option. The parameter must be of type *field mapping*.

Do not use the same field mapping parameter in more than one Cleanse transformation in a single mapping.

Options

Controls how fields are displayed in the **Incoming Fields** and **Target Fields** lists.

Configure the following options:

- The fields that appear. You can show all fields, unmapped fields, or mapped fields.
- Field names. You can use technical field names or labels.

Automap

Links fields with matching names. Allows you to link matching fields and to manually configure other field mappings. The Automap options appear when you select the Manual or Partially Parameterized field map option.

You can map fields in the following ways:

- Exact Field Name. Data Integration matches fields of the same name.

- Smart Map. Data Integration matches fields with similar names. For example, if you have an incoming field `Cust_Name` and a target field `Customer_Name`, Data Integration automatically links the `Cust_Name` field with the `Customer_Name` field.

You can use both Exact Field Name and Smart Map in the same field mapping. For example, use Exact Field Name to match fields with the same name and then use Smart Map to map fields with similar names.

You can undo all automapped field mappings by clicking **Automap** > **Undo Automap**.

To unmap a single field, select the field to unmap and click **Actions** > **Unmap** on the context menu for the field. To unmap one or more fields that you selected, click **Unmap Selected** on the Target Fields context menu.

To clear all field mappings from the transformation, click **Clear Mapping** on the Target Fields context menu.

Cleanse transformation output fields

View the output fields on the **Output Fields** tab of the **Properties** panel. A Cleanse transformation creates output fields based on the fields that you map in the **Field Mapping** tab.

The **Output Fields** tab displays the name, type, precision, and scale for each output field.

The transformation creates output fields in the following manner:

Cleanse asset with a single output

The transformation creates the cleansed field.

The output field name is the name of the target field appended by the name of the Cleanse transformation. For example, if you have a target field `Person_name` and you entered the name for the Cleanse transformation as `Cleanse_TX`, the operation returns the output field name as `Person_name_Cleanse_TX`.

Cleanse asset with multiple outputs

The transformation creates the cleansed fields and additionally any merged output field that you or another user configured in the asset.

The transformation attempts to adds the suffix `_Cleansed` to the name of each target field. For example, if you have a target field `FirstName`, the operation returns the output field name as `FirstName_Cleansed`.

Note: If you created the cleanse asset in the current version of Data Quality, the transformation applies the suffix *_Cleansed* to all output fields. If you created the asset in an older version of Data Quality, the transformation may apply a different naming policy to the outputs. See *Rules and guidelines for output field names* for more information.

The merged output field names are the names of the merged fields that you configured in the asset.

You cannot edit the output field properties in the Cleanse transformation. To edit the properties, open the cleanse asset in Data Quality.

Rules and guidelines for output field names

You may create a cleanse asset in Data Quality and add outputs to the asset over time. This can impact the type of suffix that the Cleanse transformation applies to the output fields.

When you use a cleanse asset with multiple outputs in a transformation, verify that the field names meet your mapping requirements. You may need to map the output fields again to ensure that the transformation defines the field names in the manner that you expect.

Consider the following rules and guidelines when you review the output field names:

- Some older cleanse assets supported a single output field in a single cleanse instance. If you add outputs to such an asset and you do not update the original instance, the Cleanse transformation applies the _Cleansed suffix to the newer outputs only. The transformation does not apply any suffix to the output in the original instance.

  The Cleanse transformation applies this policy regardless of when you add the asset to the transformation. The transformation deletes a suffix from an older output field when you add an output in another instance and you do not change the original instance.

- If you add an output to the original instance in any cleanse asset, the Cleanse transformation applies the _Cleansed suffix to all output names.

  The Cleanse transformation applies this policy regardless of the age of the asset and regardless of when you add the asset to the transformation.

Advanced properties

You can configure advanced properties for a Cleanse transformation. The advanced properties control the tracing level for session log messages.

You can configure the following property:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

CHAPTER 9

Data Masking transformation

Use the Data Masking transformation to change sensitive production data to realistic test data for non-production environments. The Data Masking transformation modifies source data based on masking rules that you configure for each column.

Create masked data for software development, testing, training, and data mining. You can maintain data relationships in the masked data and maintain referential integrity between database tables. The Data Masking transformation is a passive transformation.

The Data Masking transformation provides masking rules based on the source data type and masking type you configure for a port. For strings, you can restrict the characters in a string to replace and the characters to apply in the mask. For numbers and dates, you can provide a range of numbers for the masked data. You can configure a range that is a fixed or percentage variance from the original number. The Integration Service replaces characters based on the locale that you configure with the masking rules.

Masking techniques

The masking technique is the type of data masking to apply to a selected column.

Select one of the following masking techniques:

Credit Card masking

Applies a credit card mask format to columns of string data type that contain credit card numbers.

Email masking

Applies an email mask format to columns of string data type that contain email addresses.

Advanced Email Masking

Masks an email address with a realistic email address from a first name, last name, and a domain name. You can mask the string data type.

IP Address masking

Applies an IP address mask format to columns of string data type that contain IP addresses.

Key masking

Produces deterministic results for the same source data and seed value. You can apply key masking to datetime, string, and numeric data types.

Phone masking

Applies a phone number mask format to columns of string data type that contain phone numbers.

Random masking

Produces random results for the same source data and mask format. You can apply random masking to datetime, string, and numeric data types.

SIN masking

Applies a Social Insurance number mask format to columns of string data type that contain Social Insurance numbers.

SSN masking

Applies a Social Security number mask format to columns of string data type that contain Social Security numbers.

Custom Substitution masking

Replaces a column of data with similar but unrelated data from a custom dictionary. You can apply custom substitution masking to columns with string data type.

Dependent masking

Replaces a field value with a value from a custom dictionary based on the values returned from the dictionary for another input column. You can mask the string data type.

Substitution masking

Replaces a column of data with similar but unrelated data from a default dictionary. You can apply substitution masking to columns with string data type.

URL masking

Applies a URL mask format to columns of string data type that contain URLs.

Configuration properties for masking techniques

You can define how a specific masking technique works by configuring different properties for a masking technique. The Data Masking transformation masks data based on the masking technique that you select and the specific configuration that you set.

The configuration properties that appear depend on the masking technique and the data type. For example, you cannot blur string data. You cannot select a seed value when you use the Random masking technique.

Repeatable output

Repeatable output is the consistent set of values that the Data Masking transformation returns.

Repeatable output returns deterministic values. For example, you configure repeatable output for a column of first names. The Data Masking transformation returns the same masked value every time the same name appears in the workflow.

You can configure repeatable masking when you use the Random masking technique, Substitution masking technique, or the special mask formats for string data type. Select **Repeatable** and enter the seed value to configure repeatable masking.

You cannot configure repeatable output for the Key masking technique.

Optimize dictionary usage

The **Optimize Dictionary Output** option increases the use of dictionary values for masking and reduces duplicate dictionary values in the target.

If you perform substitution masking or custom substitution masking, you can choose to optimize the dictionary usage. The workflow uses some values from the selected dictionary to mask source data. These dictionary values might be used for multiple entries so that all source data is masked in the target. The chances of using duplicate dictionary values reduces if you optimize dictionary usage. To optimize dictionary output, you must configure the masking rule for repeatable output.

Seed

The seed value is a starting point to generate masked values.

The Data Masking transformation creates a default seed value that is a random number from 1 through 999. You can enter a different seed value. Apply the same seed value to a column to return the same masked data values in different source data. For example, if you have the same Cust_ID column in four tables, and you want all of them to output the same masked values. You can set all four columns to the same seed value.

You can enter the seed value as a parameter. Seed value parameter names must begin with $$. You can include an underscore (_) in the name but you cannot include other special characters. Add the required parameter and value to the parameter file and specify the parameter file name at run time.

Note: If you enter the seed value as a parameter, you must run the mapping in a mapping task. If you run a mapping that includes a seed value parameter, the mapping uses an incorrect value because it cannot read the parameter value.

Unique substitution

Unique substitution masking ensures that each unique source value uses a unique dictionary value.

To mask a source value with a unique dictionary value, you can configure unique substitution masking. If a source value is masked with a specific dictionary value, then no other source value is masked with this dictionary value.

For example, the Name column in the source data contains multiple entries of `John`. If you configure repeatable masking, every entry of `John` takes the same dictionary value, such as `Xyza`. However, other source values might also be masked with the same dictionary value. A source entry `Jack` can also use the dictionary value `Xyza`. As a result, all entries of John and Jack use the same dictionary value. When you configure unique substitution masking, if all source values of `John` use the `Xyza` dictionary value, then no other source value uses the same dictionary value.

Unique substitution masking requires a storage connection for the storage tables. Storage tables contain the source to dictionary value mapping information required for unique substitution masking.

Note: If the source data contains more unique values than the dictionary, the masking fails because there are not enough unique dictionary values to mask all the source data.

Mask format

When you configure key or random masking for string data type, configure a mask format to limit each character in the output column to an alphabetic, numeric, or alphanumeric character.

If you do not define a mask format, the Data Masking transformation replaces each source character with any character. If the mask format is longer than the input string, the Data Masking transformation ignores the

extra characters in the mask format. If the mask format is shorter than the source string, the Data Masking transformation does not mask the characters at the end of the source string.

When you configure a mask format, configure the source filter characters or target filter characters that you want to use the mask format with.

The mask format contains uppercase characters. When you enter a lowercase mask character, the Data Masking transformation converts the character to uppercase.

The following table describes mask format characters:

Character	Description
A	Alphabetical characters. For example, ASCII characters a to z and A to Z.
D	Digits. From 0 through 9.
N	Alphanumeric characters. For example, ASCII characters a to z, A to Z, and 0-9.
X	Any character. For example, alphanumeric or symbol.
+	No masking.
R	Remaining characters. R specifies that the remaining characters in the string can be any character type. R must appear as the last character of the mask.

For example, a department name has the following format:

```
nnn-<department_name>
```

You can configure a mask to force the first three characters to be numeric, the department name to be alphabetic, and the dash to remain in the output. Configure the following mask format:

```
DDD+AAAAAAAAAAAAAAA
```

The Data Masking transformation replaces the first three characters with numeric characters. It does not replace the fourth character. The Data Masking transformation replaces the remaining characters with alphabetic characters.

Source filter characters

When you configure key or random masking for string data type, configure source filter characters to choose the characters that you want to mask.

When you set a character as a source filter character, the character is masked every time it occurs in the source data. The position of the characters in the source string does not matter, and you can configure any number of characters. If you do not configure source filter characters, the masking replaces all the source characters in the column.

The source filter characters are case-sensitive. The Data Masking transformation does not always return unique data if the number of source string characters is fewer than the number of result string characters.

Target filter characters

When you configure key or random masking for string data type, configure target filter characters to limit the characters that appear in a target column.

The Data Masking transformation replaces characters in the target with the target filter characters. For example, enter the following characters to configure each mask to contain all uppercase alphabetic characters: `ABCDEFGHIJKLMNOPQRSTUVWXYZ`.

To avoid generating the same output for different input values, configure a wide range of substitute characters or mask only a few source characters. The position of each character in the string does not matter.

Range

Define a range for numeric or datetime data. When you define a range for numeric or date values, the Data Masking transformation masks the source data with a value between the minimum and maximum values.

Numeric Range

Set the minimum and maximum values for a numeric column. The maximum value must be less than or equal to the field precision. The default range is from one to the field precision length.

Date Range

Set minimum and maximum values for a datetime value. The minimum and maximum fields contain the default minimum and maximum dates. The default datetime format is MM/DD/YYYY HH24:MI:SS. The maximum datetime must be later than the minimum datetime.

Blurring

Blurring creates an output value within a fixed or percent variance from the source data value. Configure blurring to return a random value that is close to the original value. You can blur numeric and date values.

Select a fixed or percent variance to blur a numeric source value. The low bound value is a variance below the source value. The high bound value is a variance above the source value. The low and high values must be greater than or equal to zero. When the Data Masking transformation returns masked data, the numeric data is within the range that you define.

You can mask a date as a variance of the source date by configuring blurring. Select a unit of the date to apply the variance to. You can select the year, month, day, hour, minute, or second. Enter the low and high bounds to define a variance above and below the unit in the source date. The Data Masking transformation applies the variance and returns a date that is within the variance.

For example, to restrict the masked date to a date within two years of the source date, select year as the unit. Enter two as the low and high bound. If a source date is February 2, 2006, the Data Masking transformation returns a date between February 2, 2004, and February 2, 2008.

Connections in a Data Masking transformation

To use a custom dictionary connection or storage connection in a masking technique, you must add the connection to the Data Masking transformation.

You can use custom flat file or relational dictionaries. Unique substitution masking techniques also require a storage connection for source- to dictionary-value mapping.

Add the connections on the **Masking Rules** tab of the data masking transformation. When you configure a masking technique for a column, you can use the dictionaries and storage connection that you specify on the Masking Rules tab.

The following connection fields appear on the Masking Rules tab:

- Relational Dictionary Connection
- Flat File Dictionary Connection
- Storage Connection

If you export a mapping created before the April 2022 release, the Data Masking transformation in the mapping might not include the dictionary and storage connection information. When you import the mapping, the fields appear blank. To avoid this issue when you import the mapping into an environment with the April 2022 release or later, open and save the mapping before you export the mapping. The exported mapping displays the dictionary and storage connection information when imported. The connections also appear on the **Uses** tab of the **Show Dependencies** page.

Note: You might need to make a change to the mapping to enable the **Save** button.

Credit card masking

Credit card masking applies a built-in mask format to mask credit card numbers. You can create a masked number in the format for a specific credit card issuer.

The Data Masking transformation generates a logically valid credit card number when it masks a valid credit card number. The length of the source credit card number must be from 13 through 19 digits. The input credit card number must have a valid checksum based on credit card industry rules.

The first six digits of a credit card number identify the credit card issuer. You can keep the original credit card issuer or you can select another credit card issuer to appear in the masking results.

Credit card parameters

The following table describes the parameters that you can configure for credit card masking:

Parameter	Description
Repeatable	Returns the same masked value when you run a task multiple times or when you generate masked values for a field that is in multiple tables.
Seed Value	A starting number to create repeatable output. Enter a number from 1 through 999. Default seed value is 190. You can enter the seed value as a parameter.
Keep Issuer	Returns the same credit card type for the masked credit card. For example, if the source credit card is a Visa card, generate a masked credit card number that is the Visa format.
Mask Issuer	Replaces the source credit card type with another credit card type. When you disable Keep Issuer, select which type of credit card to replace it with. You can choose credit cards such as AMEX, VISA, and MASTERCARD. Default is ANY.

Email masking

The Data Masking transformation returns an email address of random characters when it masks an email address.

For example, the Data Masking transformation can mask `Georgesmith@yahoo.com` as `KtrIupQAPyk@vdSKh.BICJ`.

When you use the email masking format, you must set the seed value. The seed value is a random number from 1 through 999 and is a starting point to generate masked values. You can enter a different seed value. Apply the same seed value to a column to return the same masked data values in different source data. For example, you have the same Cust_ID column in four tables. You want all of them to output the same masked values. Set all four columns to the same seed value.

Advanced email masking

You can create a realistic email address from a first name, last name, and domain name.

When you configure advanced email masking, you can configure parameters to mask the user name and the domain name in the email address. For example, a source table might contain columns called First_Name and Last_Name. You can configure the email address to contain the first character of First_Name and seven characters of the last name. Define a domain name for the email address. The Masking task creates an address with the following syntax:

`VSingh@mycompany.com`

The following table describes the parameters you can configure for advanced email masking:

Parameter	Description
Repeatable	Returns the same masked value when you run a task multiple times or when you generate masked values for a field that is in multiple tables.
Seed Value	A starting number to create repeatable output. Enter a number from 1 through 999. Default seed value is 190. You can enter the seed value as a parameter.
First Name	Name of the column to use as the first part of the email name. The email name contains the masked value of the column you choose.
First Name Length	The number of characters in the first name to include in the email address.
Delimiter	Delimiter, such as a dot, hyphen, or underscore, to separate the first name and last name in the email address. If you do not want to separate the first name and last name in the email address, leave the delimiter blank.
Last Name	Name of the masked column to use in the email name. The email name contains the masked value of the column you choose.
Last Name Length	The number of characters in the last name to include in the email address.
Domain Name	A string value that represents an Internet Protocol (IP) resource such as gmail.com.

IP address masking

The Data Masking transformation masks an IP address as another IP address by splitting it into four numbers that are separated by a period. The first number is the network. The Data Masking transformation masks the network number within the network range.

The Data Masking transformation masks a Class A IP address as a Class A IP Address and a 10.x.x.x address as a 10.x.x.x address. The Data Masking transformation does not mask the class and private network address. For example, the Data Masking transformation can mask 11.12.23.34 as 75.32.42.52. and 10.23.24.32 as 10.61.74.84.

Note: When you mask many IP addresses, the Data Masking transformation can return nonunique values because it does not mask the class or private network of the IP addresses.

Key masking

A column configured for key masking returns deterministic masked data each time the source value and seed value are the same. The Data Masking transformation returns unique values for the column.

When you configure a column for key masking, the Data Masking transformation creates a seed value for the column. You can change the seed value to produce repeatable data between different Data Masking transformations. For example, configure key masking to enforce referential integrity. Use the same seed value to mask a primary key in a table and the foreign key value in another table.

You can configure masking rules that affect the format of data that the Data Masking transformation returns. You can mask numeric, string, and datetime data types with key masking.

When you can configure key masking for datetime values, the Data Masking transformation requires a random number as a seed. You can change the seed to match the seed value for another column to return repeatable datetime values between the columns. The Data Masking transformation can mask dates between 1753 and 2400 with key masking. If the source year is in a leap year, the Data Masking transformation returns a year that is also a leap year. If the source month contains 31 days, the Data Masking transformation returns a month that has 31 days. If the source month is February, the Data Masking transformation returns February. The Data Masking transformation always generates valid dates.

Configure key masking for numeric source data to generate deterministic output. When you configure a column for numeric key masking, you assign a random seed value to the column. When the Data Masking transformation masks the source data, it applies a masking algorithm that requires the seed.

You can configure key masking for strings to generate repeatable output. Configure a mask format to define limitations for each character in the output string. To define a mask format, configure the Source Filter characters and the Target Filter characters. The source filter characters define the source characters to mask. The target filter characters define the characters to mask the source filter characters with.

Phone number masking

You can mask phone numbers with random numbers.

The Data Masking transformation masks a phone number without changing the format of the original phone number. For example, the Data Masking transformation can mask the phone number (408) 382-0658 as (607) 256-3106.

The source data can contain numbers, spaces, hyphens, and parentheses. The Data Masking transformation does not mask alphabetic and special characters.

You can configure repeatable output when you mask phone numbers. You must select Repeatable and enter a seed value.

Random masking

Random masking generates random nondeterministic masked data.

The Data Masking transformation returns different values when the same source value occurs in different rows. You can configure masking rules that affect the format of data that the Data Masking transformation returns.

You can mask datetime, numeric, and string values with random masking.

To mask date values with random masking, either configure a range of output dates or choose a variance. When you configure a variance, choose a part of the date to blur. Choose the year, month, day, hour, minute, or second. The Data Masking transformation returns a date that is within the range you configure.

When you mask numeric data, you can configure a range of output values for a column. The Data Masking transformation returns a value between the minimum and maximum values of the range based on field precision. To define the range, configure the minimum and maximum ranges or configure a blurring range based on a variance from the original source value.

Configure random masking to generate random output for string columns. To configure limitations for each character in the output string, configure a mask format. Configure source and target filter characters to define which source characters to mask and the characters to mask them with.

Social Insurance number masking

The Data Masking transformation masks a Social Insurance number that is nine digits. The digits can be delimited by any set of characters.

If the number contains no delimiters, the masked number contains no delimiters. Otherwise the masked number has the following format:

```
xxx-xxx-xxx
```

SIN start digit

You can define the first digit of the masked SIN.

To set a start digit, enable Start Digit and enter the digit. The Data Masking transformation creates masked Social Insurance numbers that start with the number that you enter.

Repeatable Social Insurance numbers

You can configure the Data Masking transformation to return repeatable SIN values. When you configure a field for repeatable SIN masking, the Data Masking transformation returns deterministic masked data each time the source SIN value and seed value are the same. To return repeatable SIN numbers, enable Repeatable Values and enter a seed number.

The Data Masking transformation returns unique values for each Social Insurance number.

Social Security number masking

Social Security number masking applies a built-in mask format to change Social Security numbers.

The Data Masking transformation generates a Social Security number that is not valid based on the latest High Group List from the Social Security Administration. The High Group List contains valid numbers that the Social Security Administration has issued. The Data Masking transformation accesses the latest High Group List from the following location:

```
<Installation directory>\Informatica Cloud Secure Agent\highgroup.txt
```

The Data Masking transformation generates Social Security numbers that are not on the High Group List. The Social Security Administration updates the High Group List every month. Download the latest version of the list from the following location: http://www.socialsecurity.gov/employer/ssns/highgroup.txt

Social Security number format

The Data Masking transformation accepts any SSN format that contains nine digits. The digits can be delimited by any set of characters. For example, the Data Masking transformation accepts the following format: `+=54-*9944$#789-,*()"`.

Area code requirement

The Data Masking transformation returns a Social Security number that is not valid with the same format as the source. The first three digits of the SSN define the area code. The Data Masking transformation does not mask the area code. It masks the group number and serial number. The source SSN must contain a valid area code. The Data Masking transformation finds the area code on the High Group List and determines a range of unused numbers that it can apply as masked data. If the SSN is not valid, the Data Masking transformation does not mask the source data.

Repeatable Social Security number masking

The Data Masking transformation returns deterministic Social Security numbers with repeatable masking. The Data Masking transformation cannot return all unique Social Security numbers because it cannot return valid Social Security numbers that the Social Security Administration has issued.

Custom substitution masking

Use custom substitution masking to replace production data with realistic test data from a flat file or relational dictionary that you create.

Substitution masking replaces a column of data with similar but unrelated data. For example, you can create a dictionary that contains male and female first names. Use the dictionary to perform substitution masking on a column that contains both male and female first names.

You can configure custom substitution masking to replace the target column with unique masked values for every unique source column value. Unique masking requires a storage connection for source- to dictionary-value mapping.

Before you can use a dictionary or storage connection in a masking rule assignment, you must add the dictionary and storage connection to the transformation. Add the connections to the transformation on the **Masking Rules** tab. For flat file dictionaries, create a connection to the flat file dictionary from the **Configure | Connections** view and add the connection to the transformation.

When you configure custom substitution masking, select the dictionary type and the dictionary connection. You can then select the column that you want to use from the dictionary. To support non-English characters, you can use different code pages from a flat file connection.

The flat file connection code page and the Secure Agent system code page must be compatible for the masking task to work.

You can substitute data with repeatable or nonrepeatable values. When you choose repeatable values, the Data Masking transformation produces deterministic results for the same source data and seed value. You must configure a seed value to substitute data with deterministic results. You can substitute more than one column of data with masked values from the same dictionary row.

Note: Before you run the mapping, verify that the dictionary file is present in the following location: `<Secure Agent installation directory>\apps\Data_Integration_Server\data`

Custom substitution parameters

The following table describes the parameters that you configure for custom substitution masking:

Parameter	Description
Flat File Dictionary Relational Dictionary	Choose the type of custom dictionary to use. The transformation must include the required dictionary connection. If you choose flat file, you must create a flat file connection with the directory that points to the dictionary files. To make a flat file dictionary available to all Secure Agents in a runtime environment, verify that the file is in the following location: `<Secure Agent installation directory>\apps\Data_Integration_Server\data`
Dictionary	The custom dictionary that you want to select. The list includes relational or flat file dictionaries based on what you choose.
Dictionary Column	The output column from the custom dictionary. For flat file dictionaries, you can select a dictionary column if the flat file contains column headers.
Order By	Applicable for relational dictionaries. The dictionary column on which you want to sort entries. Specify a sort column to generate deterministic results even if the order of entries in the dictionary changes. For example, if you move a relational dictionary and the order of entries changes, sort on the serial number column to consistently mask the data. **Note:** The column that you choose must contain unique values. Do not use columns that can contain duplicate values to sort the data.
Lookup Input Column	Optional. The source input column on which you perform a lookup operation with the dictionary.
Lookup Dictionary Column	Required if you enter a lookup Input Column value. The dictionary column to compare with the input port. The source is replaced with values from the dictionary rows where the Lookup Input and Lookup Dictionary values match.

Parameter	Description
Lookup Error Constant	Optional. A constant value that you can configure when there are no matching values for the lookup condition from the dictionary. Default is an empty string.
Repeatable	Returns the same masked value when you run a task multiple times or when you generate masked values for a field that is in multiple tables.
Seed Value	A starting number to create repeatable output. Enter a number from 1 through 999. Default seed value is 190. You can enter the seed value as a parameter.
Optimize Dictionary Usage	Increases the usage of masked values from the dictionary. Available if you choose the **Repeatable** option. The property is not applicable if you enable unique substitution.
Is Unique	Applicable for repeatable substitution. Replaces the target column with unique dictionary values for every unique source column value. If there are more unique values in the source than in the dictionary file, the data masking operation fails. Default is nonunique substitution.

Dependent masking

Dependent masking replaces a column of data with values from a custom dictionary that you use to mask data in another column. To use dependent masking, at least one other source column must be masked with a custom substitution rule.

For example, mask a Name column in the source data with a custom substitution rule. Configure the rule to mask the values with values from the Name column in a Personal_Information dictionary.

You can configure dependent masking on another column to mask the source with values from a corresponding column in the same dictionary. For example, apply dependent masking on the Age column. Choose the Name column as the dependent column. You can then select a corresponding column from the Personal_Information dictionary as the dependent output column. If you select the Age column from the dictionary, the masking rule uses the age value that corresponds to the name value.

Dependent masking parameters

To apply dependent masking on a source column, at least one column must be masked with a custom substitution rule.

The following table describes the parameters that you can configure for dependent masking:

Property	Description
Dependent Column	The input column configured for custom substitution masking that you want to relate to the source column. Choose a column from the list. Columns that you configure with substitution masking appear in the list.
Dependent Output Column	The dictionary column to use to mask the source data column. Lists the columns in the dictionary used to mask the dependent column. Choose the required column from the list of dictionary columns.

Substitution masking

Substitution masking replaces a column of data with similar but unrelated data. Use substitution masking to replace production data with realistic test data. When you configure substitution masking, select the type of substitution masking based on the data.

The following table describes the substitution masking types that are available:

Name	Description
Substitution Name	Substitutes source data with data from a dictionary file of names.
Substitution Female Name	Substitutes source data with data from a dictionary file of female names.
Substitution Male Name	Substitutes source data with data from a dictionary file of male names.
Substitution Last Name	Substitutes source data with data from a dictionary file of last names.
Substitution Position	Substitutes source data with data from a dictionary file of job positions.
Substitution US ZIP Code	Substitutes source data with data from a dictionary file of U.S. ZIP codes.
Substitution Street	Substitutes source data with data from a dictionary file of street names.
Substitution City	Substitutes source data with data from a dictionary file of U.S. city names.
Substitution State	Substitutes source data with data from a dictionary file of U.S. state names
Substitution Country	Substitutes source data with data from a dictionary file of country names.

The Data Masking transformation performs a lookup on the dictionary file and replaces source data with data from the dictionary. You download the dictionary files when you download the Secure Agent. The dictionary files are stored in the following location:

`<Secure Agent installation directory>\apps\Data_Integration_Server\data`

You can substitute data with repeatable or nonrepeatable values. When you choose repeatable values, the Data Masking transformation produces deterministic results for the same source data and seed value. You must configure a seed value to substitute data with deterministic results.

You can substitute more than one column of data with masked values from the same dictionary row.

URL address masking

The Data Masking transformation parses a URL by searching for the ':/' string and parsing the substring to the right of it. The source URL must contain the ':/' string. The source URL can contain numbers and alphabetic characters.

The Data Masking transformation does not mask the protocol of the URL. For example, if the URL is http://www.yahoo.com, the Data Masking transformation can return http://MgL.aHjCa.VsD/. The Data Masking transformation can generate a URL that is not valid.

Note: The Data Masking transformation always returns ASCII characters for a URL.

Mask rule parameter

A parameter is a placeholder for a value or values in a mapping. You can use a mask rule parameter when you configure a Data Masking transformation object in a mapping.

Use a mask rule parameter when you do not have the source connection information. You can also use a mask rule parameter when you want to assign masking techniques to source fields when you run the mapping task. For example, if the source transformation object in a mapping uses a parameter, you cannot assign masking techniques when you create the mapping. After you create a mapping with a source connection, you might add additional fields to the source that you want to mask. Use a mask rule parameter when you create the mapping. You can then create multiple tasks to mask different data based on the same mapping.

For more information about parameters, see *Mappings*.

Rules and guidelines for mask rule parameters

Read the following rules and guidelines before you use a mask rule parameter:

- You can add a single mask rule parameter to a data masking transformation.
- You can use a mask rule parameter only in the mapping that you create it in.
- If you add a mask rule parameter to a mapping, you select columns and apply masking techniques at runtime.
- If a mapping includes a source parameter, you must use a mask rule parameter. You can't apply masking techniques when you create the mapping because you don't know the source columns.
- At runtime, the **Input Parameters** tab appears only if you use a mask rule parameter in the mapping.

Mask rule parameter example

You can create a mask rule parameter to use a single mapping to mask data in multiple sources or to avoid updating a mapping each time you add more columns to the source data that you want to mask.

See the following examples for more information:

- You create a mapping with a specific source and target and add the masking techniques as well. Over time, you add on more columns to the source table and want to mask those columns as well. The mapping then needs to be edited before it can be run again. To avoid having to edit the mapping, you can use a mask rule parameter when you configure it. You can then select columns to mask at runtime. If you added any columns after the mapping was created, you can select them at runtime without updating the mapping.
- You might have source data in multiple databases. Rather than creating multiple mappings for each source and target combination, you might want to create a single mapping to mask your source data in any of the databases and load them to different target databases. To run a mapping with different source and target connections each time, you can use parameters for the source and target connections and objects. You then need to choose different masking techniques depending on the source connection. To do this, add a mask rule parameter to the data masking transformation in your mapping. You can then choose the source connection and object, the masking rules to apply to each column, and the target connection and object at runtime. You can run the same mapping multiple times after selecting different values for these fields depending on which source and target you want to use.

Perform the following high-level tasks to run a mapping task with source, target, and mask rule parameters:

1. Create a mapping with parameters.
2. Run the mapping. You can run a mapping directly or from a mapping task.

Create a mapping with parameters

Create a mapping with parameters if you want to assign masking techniques at runtime, or if you want to be able to update the rows to mask without editing the mapping.

1. Log in toInformatica Intelligent Cloud Services and open Data Integration.
2. Click **New** > **Mappings** > **Mapping** to open a new mapping in the mapping designer.
3. Configure the source transformation.
   Optionally, use a parameter for the connection. If you use a parameter, and the object names in each source differ or you don't know the object name, ensure that you use a parameter for the object. If the object name is the same in all sources that you plan to use this mapping with, you can choose the source type as single object and then enter the object name.
4. Configure the target transformation.
   Optionally, use a parameter for the connection. If the object names in each target differ or you don't know the object name, ensure that you use a parameter for the object. If the object name is the same in all targets that you plan to use this mapping with, you can choose the target type as single object and then enter the object name. The object must exist when the mapping runs. You can't create the target object at runtime.
5. Drag a Data Masking transformation from the transformations palette onto the mapping designer and connect it to the data flow.
6. Select the Data Masking transformation object in the mapping designer. The properties appear on the **Properties** tab.
7. On the **General** tab, enter a name and optional description for the transformation object.
8. On the **Masking Rules** tab, select **Add a parameter** to configure masking techniques at runtime.
9. Click **New Parameter** and create a masking parameter. Then select the parameter from the list.
   The following image shows a sample masking_rules parameter created and selected:

10. Select a relational dictionary connection or a flat file dictionary connection and storage connection from the lists.
    If you don't select values here, you can't add or update the values when you run the mapping.
11. Click **Save** to save the mapping.

Run the mapping

You can choose to run the mapping directly or to add the mapping to a mapping task.

Run the mapping directly

1. On the mapping page, click **Run**.
2. Enter values that you want to use for the source, target, and masking techniques in this run.
3. Click **Run**.

You can monitor the progress and status of the mapping job on the **My Jobs** tab. You can access and run the mapping again to run it with different values. Wait for the first job to complete before you run the mapping again.

Run a mapping task

1. From the **Actions** menu on the mapping page, click **New Mapping Task**.
2. On the **Mapping Task** page that opens, add a name and description, choose a runtime environment, select the mapping that you created, and click **Next**.
3. On the **Sources** tab, perform the following steps:
    a. Select the source connection on which you want to run the mapping task.
    b. If you used an object type parameter in the source transformation, select the source type **Single** and select an object. You can then preview the source data on the **Data Preview** tab.
4. On the **Targets** tab, perform the following steps:
    a. Select the target connection to which you want to load the masked data.
    b. If you used an object type parameter in the target transformation, select the object.
    c. Select the operation type and other target configuration options, and then click **Next**.

        Note: The object that you specify must already exist. You cannot create the target object at runtime.
5. Optional. On the **Input Parameters** tab, if you need to change the dictionaries or storage connection that you previously included, select the connections from the list. You can ignore the fields if you don't need to use the connections.
6. Click **Add** to add a field and then click **Configure** to select the masking technique that you want to apply to the field. Repeat this for each field that you want to mask and then click **Next**.
7. Optional. On the **Runtime Options** tab, choose scheduling and notification options as needed.
    If you want to update the parameter values in each run, skip the scheduling options. You can't change parameter values before an automated run.
8. Click **Save** to save the mapping task.
9. Click **Run** to run the mapping task. You can monitor the progress and status of the mapping task on the **My Jobs** tab.

You can run the masking task again with updated parameter values to mask different or new source data. Wait for the masking task to complete before you run it again.

Creating a Data Masking transformation

Create a Data Masking transformation to mask source data. You can assign masking techniques in the transformation object or enter a mask rule parameter to assign masking techniques when you run the mapping.

Use the Mapping Designer to create a Data Masking transformation. When you create a mapping, a Source transformation and a Target transformation are already on the canvas for you to configure. Configure the Source transformation to represent the source data that you want to mask. Configure the Target transformation to represent the target connection where you want to store the masked data.

1. Drag a Data Masking transformation from the transformations palette onto the mapping canvas.

2. Connect the Data Masking transformation object to the data flow.

3. Select the Data Masking transformation object in the mapping designer.

   The properties appear in the properties tab.

4. On the **General** tab, enter a name and optional description for the transformation object.

5. On the **Incoming Fields** tab, configure the field rules that define the data that you want to copy to the target.

   By default, the transformation includes all fields.

6. On the **Masking Rules** tab, you can configure the following properties:

   - Parameter. To assign masking techniques at run time, add or create another mask rule parameter.

   - Relational Dictionary Connection. To use a relational dictionary for custom substitution masking, choose the dictionary connection from the list of relational connections.

   - Flat File Dictionary Connection. To use a flat file dictionary for custom substitution masking, choose the dictionary connection from the list of flat file connections.

   - Storage Connection. To configure unique substitution masking, choose a storage connection from the list of connections.

   - Add. To configure masking techniques in the mapping, click **Add** and select the fields that you want to mask. Click **Configure** to select and configure the required masking technique. If you assign a masking technique in the mapping, you cannot edit it at run time.

7. Select the Target transformation and map the incoming fields to fields in the target.

   By default, the Data Masking transformation adds the prefix "out_ " to fields that you include from the Source transformation object. You cannot use automatic field mapping to map the fields in the Target transformation object because the field names do not match. Rename the incoming field names in the Target transformation if you want to use automatic field mapping. Select the pattern rename option and enter "out_/" as the pattern to delete the prefix.

8. Save and run the mapping.

   If you use a mask rule parameter in the mapping, you assign and configure the masking techniques when you run the mapping task.
   The following image shows the second step of a mapping task that uses a mask rule parameter:
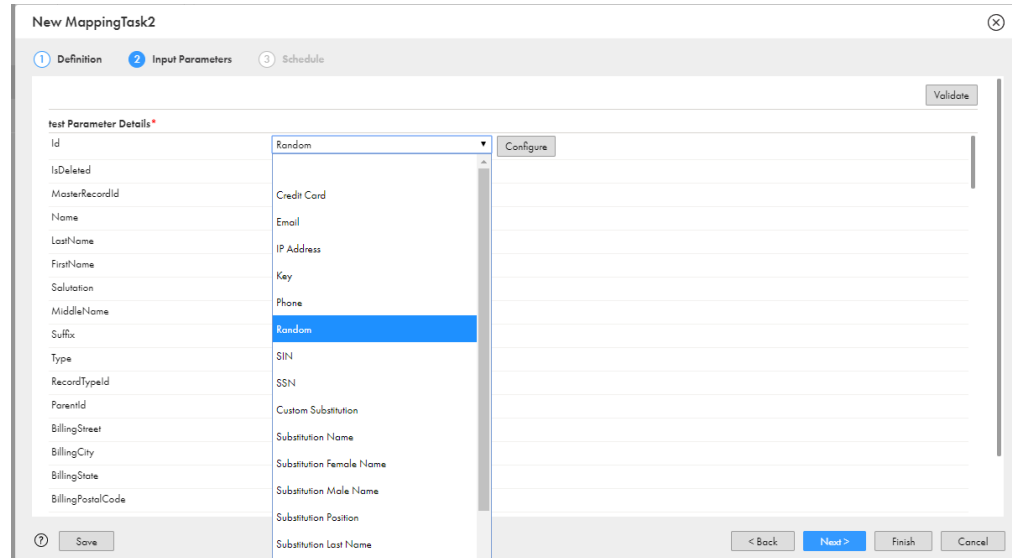
Note: If you assign a masking technique to a column in a mapping task and then change the assignment of the column in the mapping, the mapping task configuration takes precedence. If you unselect the rule assignment of the column in the mapping task, then the mapping task uses the masking technique assigned to the column in the mapping.

Consistent masked output

You might want to use different tools to mask the source data.

You can use the following tools to generate the same masked output from the same source data:

- Informatica Intelligent Cloud Services
  You can use the following tools in Informatica Intelligent Cloud Services:

  - Masking tasks on Informatica Intelligent Cloud Services

  - Mapping tasks that contain mappings with the data masking transformation on Informatica Intelligent Cloud Services

  - Mappings that contain the data masking transformation on Informatica Intelligent Cloud Services

- Test Data Management (on-premise)

- PowerCenter mappings that contain the data masking transformation

Rules and guidelines

Consider the following rules and guidelines before you run mappings, tasks, or workflows to generate consistent masked output:

- Use substitution masking rules to generate consistent masked output.

- The masking rules must use the same dictionaries.
- The Repeatable option must be set to ON.
- Use the same seed value.

Substitution masking rules use values from dictionaries to create masked output. The default dictionaries on Informatica Intelligent Cloud Services and on-premise Test Data Management are the same. When you use the same substitution rule, the workflow uses the same dictionary to substitute source data. The same seed value therefore ensures that the same substitute value is used for all rows provided the dictionaries are the same.

On Informatica Intelligent Cloud Services, the dictionary files are available at: `<Secure Agent installation directory>\apps\Data_Integration_Server\data`

In on-premise Test Data Management, the dictionary files are available at: `<Informatica installation directory>\server\infa_shared\LkpFiles`

The Repeatable option must be set to ON to ensure that the task or workflow repeats dictionary values for the same source value.

Example

Consider the following example:

The source data contains *First Name* and *Last Name* columns that you need to mask to ensure that you mask the full name in the target data.

You can use the following methods to generate the masked output:

- Run a masking task on Informatica Intelligent Cloud Services.
- Run a mapping task that contains a data masking transformation on Informatica Intelligent Cloud Services.
- Run a mapping that contains a data masking transformation on Informatica Intelligent Cloud Services
- Run a data masking plan in Test Data Management.
- Run a PowerCenter mapping that contains a data masking transformation.

Perform the following high-level tasks to generate the masked output:

1. Use the Substitution Name masking rule to mask the *First Name* column. Set the Repeatable option to ON. Enter a seed value.
2. Use the Substitution Last Name masking rule to mask the *Last Name* column. Set the Repeatable option to ON. Enter a seed value.
3. Use the default dictionaries available with the setup. Do not make changes to the dictionaries.

When you run the masking task, mapping, or mapping task on Informatica Intelligent Cloud Services, Test Data Management workflow, or PowerCenter mapping to generate output, you generate the same masked output for the same source data.

Data Masking transformation example

You need realistic data for testing in a nonproduction environment. You have production data that includes sensitive data columns. You cannot use the data without the risk of compromising sensitive data. Use the Data Masking transformation to mask sensitive data before you use the data in a test environment.

The production data includes a table Personnel_Information with the following data:

First Name	Surname	DOB	Address	State
Uma	Hilel	03/12/1985	24, Atkins Avenue	CA
John	Sen	07/15/1963	49, Wheeler Road	MN
Keiko	Burns	11/24/1989	13, Draker Drive	PA
Isadora	Buckley	08/16/1990	20, Fountain Center	CA

In the Mapping Designer, add *Personnel* as a source transformation for the table Personnel_Information. Add a target transformation Personnel_test.

Add the Data Masking transformation to the mapping canvas and connect it to the data flow.

You need to mask the Surname, DOB, and the State columns to ensure sensitive data is masked. You can use the Substitution Last Name masking technique to mask the Surname column. This masking technique replaces data in the column with data from the dictionary file on surnames. You can use the Random Date masking technique to mask the DOB column. Use the Substitution State masking technique to mask the State column. This masking technique replaces data in the column with data from the dictionary file on U.S. state names.
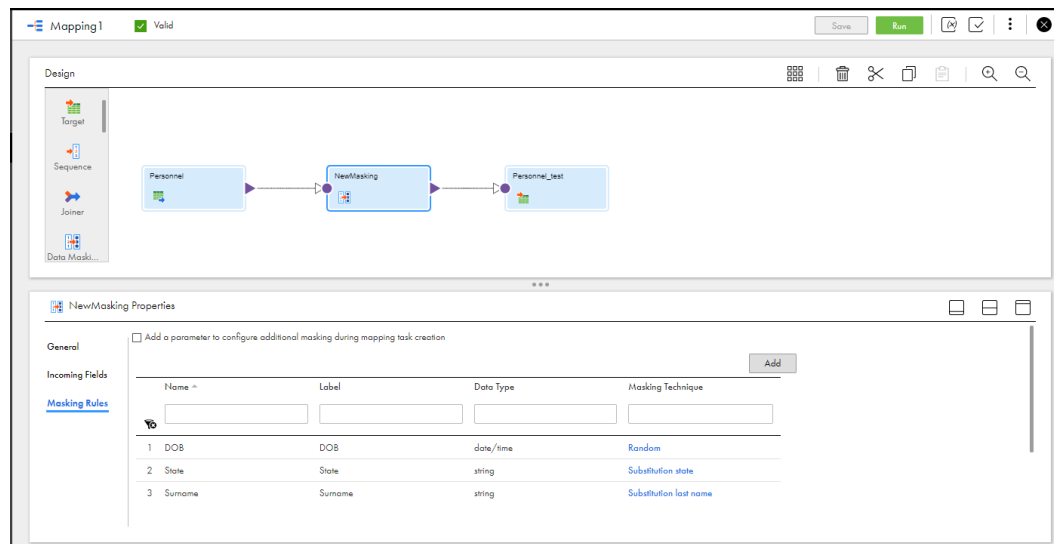
When the mapping run completes successfully, you can view the masked data in the Personnel_test output table :

First Name	Surname	DOB	Address	State
Uma	Acothley	05/12/1985	24, Atkins Avenue	Michigan
John	Mcgovern	04/15/1963	49, Wheeler Road	Oklahoma

First Name	Surname	DOB	Address	State
Keiko	Garsia	03/24/1990	13, Draker Drive	Montana
Isadora	Sonnier	01/16/1991	20, Fountain Center	South Carolina

You can now use the masked data in the table Personnel_test in the test environment.

CHAPTER 10

Data Services transformation

Use the Data Services transformation to invoke data services from the data services repository, including industry-standard services, such as HL7 and HIPAA, and customized services. For example, you can use the transformation to invoke data services to process HIPAA messages that you exchange with your healthcare partners.

By default, the data services repository includes out-of-the-box data services that process industry standards. You can create customized services in the data services repository and use the Data Services transformation to invoke them, as well. For more information about the data services and the data services repository, see *Data Services Repository* in the Administrator help.

When you add the transformation to a mapping, you define the data services that the transformation invokes. You can choose one of the following options:

- Use a single data service to process a single message type for a specific standard and a specific usage type. For HIPAA messages, you can also select a data service that sends messages and validation messages to your partners.

- Use a dynamic service name to process messages for multiple message types from different standards and usage types. When you use a dynamic service name, you pass the data service names to the transformation as parameters.

The Data Services transformation passes message data to the downstream transformation in a file or as a buffer, based on your selection. The transformation might create additional output fields, based on the status tracing level that you select for the transformation and on the data service usage type. You can add more Target transformations to the mapping and configure the Data Services transformation to send different output types to different targets. For example, send message data to one target and send status tracing reports to another target.

Dynamic service name

Use a dynamic service name with the Data Services transformation to process messages for message types from different standards and usage types. When you use a dynamic service name, you pass the data service names to the transformation as parameters.

The name of a data service indicates the industry standard, the version of the standard, the message type, and the service usage type, in the following syntax:

```
<industry standard>_<version>_<message type>_<usage type>
```

The usage type can be one of the following types:

- Parser. Use a parser service to receive messages from your partners.

- Serializer. Use a serializer service to send messages to your partners.

- Restricted serializer. Use a restricted serializer service to send messages to your partners, including validation messages. Applies to HIPAA messages.

For example, the name of the service you use to receive HIPAA message 270 version 5010A1 is HIPAA_5010A1_270_Parser.

To use a dynamic service name, complete the following tasks when you configure the mapping:

1. Configure the mapping to pass the data service names to the Data Services transformation as parameters. For example, add an Expression transformation to the mapping and configure the Expression transformation to extract the data service names from the messages and pass the names to the Data Services transformation.
2. When you configure Data Services properties, select the option to use a dynamic service name.
3. In the field mapping of the Data Services transformation, map the incoming field that contains the service name to the service_Name field.

Using the Message Discovery Service

When you use a dynamic service name, you can optionally enable the Message Discovery Service. It detects the name of a parser data service from the upstream transformation and invokes the parser data service from the data services repository.

To use the Message Discovery Service, complete the following tasks when you configure the mapping:

1. Add an Expression transformation to the mapping that receives incoming messages.
2. In the Expression transformation, create an expression field to pass an empty string to the Data Services transformation.
3. In the Data Services transformation, select the option to use a dynamic service name. Then, enable the Message Discovery Service and select the mode and type.
4. In the field mapping of the Data Services transformation, map the incoming expression field with the empty string to the service_Name field.

Note: The Message Discovery Service only detects the names of parser data services. It doesn't detect the names of serializers or restricted serializers.

Status tracing messages

You can configure the status tracing level of the Data Services transformation to write status messages to output fields. The Mapping Designer creates UDT_Status_Code and the UDT_Status_Message output fields in the Data Services transformation.

When you choose to write the description only, the transformation returns a status code and one of the following status messages:

Status code	Status message
1	Success
2	Warning
3	Failure

Status code	Status message
4	Error
5	Fatal Error

When you choose to write the full status, the transformation returns a status code and a detailed XML status message.

Data Services properties

Configure data services properties to define the input and output types, the data services that the transformation invokes, and the status tracing level on the **Data Services** tab.

Configure the following properties:

Property	Description
Input Type	Type of input that the Data Services transformation receives from the upstream transformation. Select one of the following options: - Buffer. The Data Services transformation receives source data in the inputBuffer input field. - File. The Data Services transformation receives a source file path in the inputBuffer input field. Default is Buffer.
Output Type	Type of output that the transformation passes on to the downstream transformation. Select one of the following options: - Buffer. The Data Services transformation passes the data to the outputBuffer output field. - File. The Data Services transformation writes the output to the outputBuffer output field as an output.xml file. Default is Buffer.
Service Type	Type of service that the Data Services transformation uses. Select one of the following options: - Data Services. The transformation uses the parser, serializer, and restricted serializer data services in the data services repository. - FHIR Validation. The transformation uses a built-in FHIR validation service to validate incoming FHIR bundles and messages in XML or JSON format. You can configure the FHIR version and the terminology server. - Hierarchical Mapper. The transformation uses a hierarchical mapper to convert hierarchical input to hierarchical output, such as the XML output from a parser data service to a target XSD file.
Service Name	Select a data service from the data services that reside in the data services repository. Choose this option for a mapping that processes a single message type for a specific standard and usage type. **Tip:** Click **Populate Fields** to automatically populate the input and output fields based on the data service. Applies when the service type is **Data Services** and you don't use a dynamic service name.
Dynamic Service Name	Select this option to pass service names to the downstream transformation as parameters. Choose this option for a mapping that processes messages for multiple message types from different standards and usage types. Applies when the service type is **Data Services**.

Property	Description
Message Discovery Service	Select this option to enable the Message Discovery Service to detect the name of a parser data service. Applies when **Dynamic Service Name** is enabled.
Mode	Select the environment that the data service runs in. The environment can be either development or production. Applies when **Message Discovery Service** is enabled.
Type	Select the data service type. The type can be either custom or out-of-the-box. Custom data services are services that you customize and upload to the data services repository. Out-of-the-box data services are services that are available in the data services repository by default. Applies when **Message Discovery Service** is enabled.
Operation	Operation that the data service performs. By default, the transformation uses a parser data service. Applies when **Message Discovery Service** is enabled.
Status Tracing Level	Detail level of job status that the Data Services transformation writes to output fields. Select one of the following options: - None. The transformation doesn't write the status to output fields. You can view the information in the job the session log. - Description only. The transformation writes a status code and a short description to the UDT_Status_Code output field. - Full status (XML). The transformation writes a status code to the UDT_Status_Code output field and a detailed XML status message to the UDT_Status_Mesage output field. Default is none. Applies when the service type is **Data Services** or **Hierarchical Mapper**.
FHIR Version	FHIR version that the FHIR validation service uses to validate FHIR bundles and messages. Applies when the service type is **FHIR Validation**.
Terminology Server	URL to the terminology server that the FHIR validation service uses to validate FHIR bundles and messages. If you leave the field blank or the value is n/a, the transformation doesn't use a terminology server to validate the resources. Applies when the service type is **FHIR Validation**.
Hierarchical Mapper	Hierarchical mapper that the transformation uses to convert hierarchical input to hierarchical output. Applies when the service type is **Hierarchical Mapper**.

Data Services transformation input fields

View and add input fields on the **Input Fields** tab of the **Properties** panel. The **Input Fields** tab displays the name, type, precision, and scale for each input field.

By default, the Data Services transformation creates an inputBuffer field. The Data Services transformation might create additional input fields depending on the transformation properties and data services type that you configure.

If you need to pass additional files, file names, or parameters to the data service, you can add additional input fields. You configure the field name, type, and precision for each field.

The following table describes the default input fields that Data Service transformation creates based on the transformation properties:

Field name	Description
inputBuffer	Receives the source data when the input type is buffer. Receives the source file name and path when the input type is file.
OutputFilename	Receives the output file name when the output type is file.
Service_Name	Receives data service names when you enable the Dynamic Service Name property.

The following table describes the additional input fields that you can create for a Data Services transformation:

Field type	Input/Output	Description
Service Parameter	Input	Receives an input parameter for a data service.
Additional Output (File)	Output	Receives an output file name.
Additional Input (Buffer)	Input	Receives input data to pass to the Data Services transformation.

Data Services transformation output fields

View the output fields on the **Output Fields** tab of the **Properties** panel. The **Output Fields** tab displays the name, type, precision, and scale for each output field.

A Data Services transformation creates an outputBuffer output field for the messages that it processes. The transformation might create additional output fields, based on the status tracing level that you select for the transformation and on the data service usage type. For example, for a HIPAA restricted serialize service, the transformation creates output fields for error reports and for a validation report. If you use the Data Services transformation to validate incoming FHIR resources, the transformation creates output fields only for error reports and for a validation report.

If you use custom messages, configure additional output fields for custom message elements. To pass parameters to the output, create pass-through fields with an Input/Output field type and map them to output fields.

When you add a Target transformation to the mapping, the Data Services transformation maps all the output fields to the Target transformation. To receive the output in different targets, add more Target transformations to the mapping and configure field rules to exclude output fields that you don't want to send to the target. For example, to receive messages and validation reports in two different targets, add two Target transformations to the mapping. Exclude the validation report output fields from the target that receives the messages and exclude the outputBuffer output field from the target that receives the validation reports.

Data Services transformation field mapping

View and configure the field mappings of the Data Services transformation on the **Fields Mappings** tab of the **Properties** panel.

By default, the Data Services transformation maps the input it receives from the upstream transformation to the inputBuffer field. When you use a dynamic service name, map the incoming field that contains the service name to the Service_Name field.

CHAPTER 11

Deduplicate transformation

The Deduplicate transformation adds a deduplicate asset that you created in Data Quality to a mapping.

Use a Deduplicate transformation to analyze the levels of duplication in a data set and optionally to consolidate sets of duplicate records into a single, preferred record. Deduplicate transformations analyze the *identity* information in the records. An identity is a group of data values in a record that identify a person or an organization.

Deduplication and consolidation are useful operations in the following types of data project:

- Customer Relationship Management. For example, a store designs a mail campaign and must check the customer database for duplicate customer records.

- Regulatory compliance initiatives. For example, a business operates under government or industry regulations that insist all data systems are free of duplicate records.

- Financial risk management. For example, a bank may want to search for relationships between account holders.

- Any project that must identify or eliminate records that store duplicate identity information.

Deduplication and consolidation operations

When you run a mapping, the Deduplicate transformation generates a temporary index from the input records that it reads. The transformation analyzes the index to find pairs of similar records.

The transformation calculates a series of percentage scores that represent the degrees of similarity between the pairs of records that it finds. If two records match each other with a score that exceeds a given threshold, the transformation considers the records to be duplicates.

The deduplicate asset that you add to the transformation specifies the comparison criteria for the deduplication operation, including the threshold score that duplicate records must satisfy.

Consolidation is an optional process that the deduplicate asset can specify for the transformation. During consolidation, the transformation evaluates the sets of matching records that the deduplication process identifies. The transformation selects or constructs a preferred version of the records in each matching set.

A Data Quality user configures the deduplication and consolidation processes in the deduplicate asset. For more information about the criteria that the asset defines, contact the Data Quality user.

Rules and guidelines for deduplication and consolidation

When you add a Deduplicate transformation to a mapping, consider the following rules and guidelines:

Mapping fields for identity analysis

The deduplicate asset that you add to the transformation specifies a type of identity, such as a person name or an organization name. The asset identifies the identity type as the *objective* of the deduplication operation. The type of identity defines the types of information that the transformation expects to find in the index.

You must map the appropriate input fields on the transformation to the target fields that the transformation indicates. You can optionally map additional input fields to other fields on the transformation.

Groups and sequence ID values

In duplicate analysis, a group is a set of records that contain identical values in a given field. At run time, the Deduplicate transformation analyzes the index data for records exclusively within each group and subsequently combines the results from each group into a single data set. Use the GroupKey field on the **Field Mapping** tab to define your groups. When you create groups on an appropriate field, you reduce the overall number of comparisons that the transformation must perform without any meaningful loss of accuracy in duplicate analysis.

The GroupKey field is mandatory. If you prefer not to sort your input data into groups, add a column to your data set that has the same value on every row and map the column to the GroupKey field.

Sequence ID values determine the order in which the transformation reads the input records. If your input records do not contain a field that can provide data to the SequenceId field, the transformation reads the records in the order in which they appear in the input data set. A SequenceId field is mandatory if you run a mapping in advanced mode.

Clusters and scores

When two or more records match each other, the transformation assigns them to the same matching set and adds an ID value to each record that identifies them as members of the set.

A set of matching records within a group is also known as a *cluster*, and the ID value that associates matching records together is the *cluster ID*.

The transformation represents the relationships between matching records with *link score* and *driver score* values in the output data set. The link score is the score between two records that identifies them as members of the same cluster. The driver score is the score between the first record added to a cluster and another record in the cluster.

Bear in mind that the transformation generates a single score for each pair of matching records that it finds. The link and driver scores define the types of relationship between different records and do not represent different calculations.

Metadata fields

On the **Output Fields** tab, the transformation adds fields that display the score values for pairs of matching records. The fields also identify the cluster to which each record belongs. If the deduplicate asset specifies a consolidation process, the metadata fields specify a preferred record for each cluster. The transformation identifies the preferred record as the *survivor* record.

Use the fields to understand the mapping results.

For more information about the metadata fields, see "Metadata fields on the Deduplicate transformation" on page 154 and "Link scores and driver scores" on page 155.

Identity population data

The deduplication process uses country-specific reference data files to evaluate the identity information in the input data. The reference data files are called *population files*. When you run a mapping with a Deduplicate transformation, the transformation compares the input data to the population file for the country that the asset specifies.

Data Quality downloads the population files to the Secure Agent host machine when you install the Secure Agent. You do not need to take any action to download the files.

For more information on population file properties, consult the *Deduplicate Guide* in the Data Quality online help.

Groups in duplicate analysis

A duplicate analysis mapping can take time to run because of the number of data comparisons that the Deduplicate transformation must perform. The number of comparisons relates to the number of data values on the fields that you select.

The following table shows the number of calculations that a mapping performs on a single field:

Number of data values	Number of comparisons
10,000	50 million
100,000	5,000 million
1 million	500,000 million

To reduce the time that the mapping takes to run, you configure the Deduplicate transformation to assign the input records to **groups**.

A group is a set of records that contain identical values on a field that you specify. When you perform duplicate analysis on grouped data, the Deduplicate transformation analyzes the record data exclusively within each group and combines the results from each group into a single output data set. The field on which you group the data is the **GroupKey** field. When you choose an appropriate group key, you reduce the overall number of comparisons that the Deduplicate transformation must perform without any meaningful loss of accuracy in the mapping analysis. Select the GroupKey field in the Deduplicate transformation.

The following table shows the number of calculations that a mapping performs on a single field that you sort into ten groups:

Number of data values	Number of groups	Group size	Total number of comparisons (all groups)
10,000	10	1,000	5 million
100,000	10	10,000	500 million
1 million	10	100,000	50,000 million

Consider the following rules and guidelines when you organize data into groups:

- The GroupKey field must contain a range of identical values, such as a city name or a state name in an address data set.

- Do not select a group key that contains information that is relevant to the duplicate analysis. For example, do not select the index key field as the GroupKey field. The goal in group creation is to organize the data according to values whose duplicate nature is not relevant to the objectives of the analysis.

- When you select a group key, consider whether the transformation can create groups that are a valid size relative to your input data. If the groups are too small, the match analysis might not find all of the duplicate records in the data set. If the groups are too large, the match analysis might return false duplicates.

- If your data does not contain a suitable field for group keys, create a data column that the transformation can use to can sort the records into the group sizes that you require. For example, for a data set that contains 1 million records, you might decide to create a column that repeats a series of values from 1 through 50. The records in each group will be distributed evenly in the data set and will allow the duplicate analysis to proceed on grouped data.

- If you do not want to sort the records into groups, specify a GroupKey field that contains the same value in every record. If a suitable field does not exist, create the field. For example, create a column of data in which every value is *Group1* and select the column as the GroupKey field. When the mapping runs, the Deduplicate transformation sorts the records by the GroupKey field values and therefore assigns every record to the same group.

- Groups do not reorder the position of the records in the mapping data set.

Example: Selecting a group key column

Let's say that a bank wants to search for duplicate bank account holders. The bank's customer data set includes columns for customer names and addresses, and the bank chooses **Contact** as the objective in the deduplicate asset. The bank decides to sort the input records into groups and to perform duplicate analysis on each group. The bank must select a column in the Deduplicate transformation on which to create the groups.

The following table shows a fragment of the data set:

Customer ID	Lastname	Firstname	Address1	City	State	Zip	Country
90999990	Armstrong	Al	6121 SUNSET BLVD.	LOS ANGELES	CA	90028	USA
90999907	Baldwin	Lynn	1600 EL CAMINO REAL, SUITE 1500	MENLO PARK	CA	94025	USA
90999917	Baldwyn	Linn	1600 EL CAMINO REAL, #1500	MENLO PK	CA	94025	USA
90999859	Belleperche	Carmen	9255 SUNSET BLVD.	LOS ANGELES	CA	90069	USA
90999876	Clark	Wick	777 S. FIGUEROA	LOS ANGELES	CA	90071	USA
90999859	Bachtin	Guy	30 S. WACKER	CHICAGO	IL	60606	USA
90999868	Dicintio	David	181 WEST MADISON ST	CHICAGO	IL	60602	USA
90999869	Ash	Pascal	335 WEST 16TH STREET	NEW YORK	NY	10011	USA

Customer ID	Lastname	Firstname	Address1	City	State	Zip	Country
90999996	Bachtin	David	1633 BROADWAY	NEW YORK	NY	10022	USA
90999994	Carpenter	Brad	30 BROAD ST	NEW YORK	NY	42304	USA
90999820	Dedmond	David	ONE FINANCIAL SQUARE	NEW YORK	NY	10008	USA
90999902	Backwell	Chris	901 SE OAK, WILLAMETTE PLZ	PORTLAND	OR	97214	USA
90999897	Askerup	Nancy	400 MARKET STREET	HOUSTON	TX	77027	USA
90999904	Choy	Shelley	1177 WEST LOOP SOUTH	HOUSTON	TX	77027	USA
90999886	Cote	Lian	530 E. SWEDESFORD RD.	HOUSTON	TX	77027	USA
90999999	Croteau	Paul	3829-55 GASKINS ROAD	HOUSTON	TX	77027	USA

In this scenario, you might identify the State column as the most suitable column on which to sort the records. You select the State column as the **GroupKey** field in the transformation.

When you select the State column as the GroupKey field, the deduplication operation enables the creation of five groups, one for each state. The likelihood that the bank has customers with the same contact information in different states is very low. Additionally, the data includes a Customer ID column that can add to the confidence of the deduplication process.

The Customer ID column is a poor candidate for group creation, as it is a primary key field. If you select the column as the GroupKey field, the deduplication operation creates a group for every unique ID and thus for every record.

The Country column is also a poor candidate for group creation, as the column contains the same value in every row. If you select the Country column as the GroupKey field, the deduplication operation adds all of the records to the same group. Your bank might have two or more genuine customers with the same name living across the country, and you do not want to deduplicate their entries.

Deduplicate transformation configuration

When you configure a Deduplicate transformation in a mapping, you first select the deduplicate asset that contains the logic to include in the transformation. Next, you map the appropriate incoming fields to the asset.

The deduplicate asset specifies the type of identity that the transformation searches for. The identity type determines the types of input field that you must connect to the transformation. For more information about the fields that the Deduplicate transformation expects, consult the Data Quality user who configured the asset.

To configure the transformation, complete the following tasks:

1. Connect the Deduplicate transformation to a Source transformation or other upstream object.

2. On the **Deduplicate** tab, select the deduplicate asset that you want to include in the transformation.

The following image shows the options that you use to select the deduplicate asset:
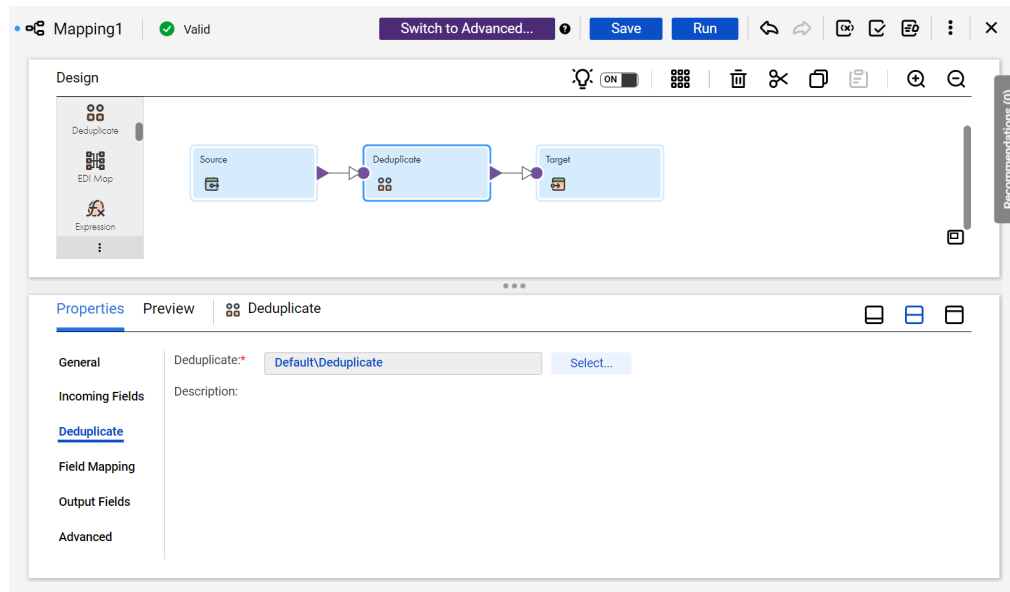
3. On the **Incoming Fields** tab, verify the fields that enter the transformation from upstream objects.

   By default, the transformation inherits all incoming fields from any connected upstream object in the mapping. You can define a field rule to limit or rename the incoming fields.

4. On the **Field Mapping** tab, connect one or more incoming fields to the fields on the deduplicate asset.

   The fields that you map must include data that can describe an identity of the type that the asset specifies. You must also map a GroupKey field that the transformation can use to create groups during duplicate analysis.

   The tab lists the fields from upstream objects in the **Incoming Fields** section and lists the fields that the asset specifies in the **Target Fields** section.

   The incoming field names might reflect the names of the target fields in the transformation. If so, you can use the **Automap** options to connect the fields.

5. Verify the output field properties on the **Output Fields** tab.

   The output fields include several metadata fields that contain important information about the results of the duplication and consolidation process. For more information about the metadata fields, see "Metadata fields on the Deduplicate transformation" on page 154.

6. You can optionally rename the Deduplicate transformation and add a description on the **General** tab. You can also update the tracing level for the transformation on the **Advanced** tab. The default tracing level is Normal.

Note: If you update an asset in Data Quality after you add it to a transformation, you may need to synchronize the asset version in the transformation with the latest version. For more information about data quality asset synchronization, see "Synchronizing data quality assets" on page 116.

Deduplicate transformation field mappings

Configure field mappings to define how data moves from the upstream transformation to the Deduplicate transformation and connects to the inputs on the deduplicate asset. Configure field mappings on the **Field Mapping** tab of the **Properties** panel.

Note: You map an incoming field to a single field on a deduplicate asset.

You can configure the following field mapping options:

Field map options

Method of mapping fields to the transformation.

Select one of the following options:

- Manual. Manually link an incoming field to a transformation input field. Removes links for any automatically mapped field.
- Automatic. Automatically link fields with the same name. You cannot manually link fields with this option.
- Completely Parameterized. Use a parameter to represent the field mapping.
  Choose the Completely Parameterized option when the deduplicate asset in the transformation is parameterized or any upstream transformation in the mapping is parameterized.
- Partially Parameterized. Configure links in the mapping that you want to enforce and use a parameter to allow other fields to be mapped in the mapping task. Or, use a parameter to configure links in the mapping, and allow all fields and links to display in the task for configuration.

Parameter

Select the parameter to use for the field mapping, or create a new parameter. This option appears when you select Completely Parameterized or Partially Parameterized as the field map option. The parameter must be of type *field mapping*.

Do not use the same field mapping parameter in more than one Deduplicate transformation in a single mapping.

Options

Controls how fields are displayed in the **Incoming Fields** and **Target Fields** lists.

Configure the following options:

- The fields that appear. You can show all fields, unmapped fields, or mapped fields.
- Field names. You can use technical field names or labels.

Automap

Links fields with matching names. Allows you to link matching fields and to manually configure other field mappings. The Automap options appear when you select the Manual or Partially Parameterized field map option.

You can map fields in the following ways:

- Exact Field Name. Data Integration matches fields of the same name.
- Smart Map. Data Integration matches fields with similar names. For example, if you have an incoming field `Cust_Name` and a target field `Customer_Name`, Data Integration automatically links the `Cust_Name` field with the `Customer_Name` field.

You can use both Exact Field Name and Smart Map in the same field mapping. For example, use Exact Field Name to match fields with the same name and then use Smart Map to map fields with similar names.

You can undo all automapped field mappings by clicking **Automap** > **Undo Automap**.

To unmap a single field, select the field to unmap and click **Actions** > **Unmap** on the context menu for the field. To unmap one or more fields that you selected, click **Unmap Selected** on the Target Fields context menu.

To clear all field mappings from the transformation, click **Clear Mapping** on the Target Fields context menu.

Metadata fields on the Deduplicate transformation

The Deduplicate transformation includes a set of predefined fields that contain metadata for the deduplication and consolidation processes. The transformation creates the fields by default and populates the fields when the mapping runs.

Metadata fields on the Field Mapping tab

The **Target Fields** list in the Field Mappings tab includes the following metadata fields:

GroupKey

Contains the data values that the transformation uses to sort input records into groups for duplicate analysis.

SequenceId

Contains a unique identifier for each record that enters the transformation.

The transformation uses the sequence ID values to identify records in the Out_DriverId and Out_LinkId data. If you do not map the SequenceId field, the transformation uses the values on the OutRowId field as unique identifiers for the records.

Metadata fields on the Output Fields tab

The Output Fields tab includes the following metadata fields:

Out_ClusterId

Contains the identifiers of the cluster to which each record belongs.

Note: In the deduplication process, a cluster is a set of records whose data values match each other to a degree that exceeds the duplicate threshold. Records in the same set are likely to identify the same identity. A set may contain a single record, as every unique record is a perfect match with itself.

Out_ClusterSize

Contains the number of records in the set to which the current record belongs. When a set contains a unique record, the cluster size is 1.

Out_DriverId

Contains the identifier of the driver record in each matching record set. The driver record is the record in the set with the lowest value on the SequenceId input field. If the transformation does not use the SequenceId field, the driver record is the record in the matching set with the lowest Out_RowId value.

Out_DriverScore

Contains the score that represents the degree of similarity between the current record and the driver record in the matching record set.

Out_IsSurvivor

Contains an identifier for the preferred record that a consolidation process specifies.

Out_LinkId

Contains the identifier of the record that matched with the current record and linked it to the matching record set.

Out_LinkScore

Contains the score between two records that results in the addition of a record to a matching record set. The Out_LinkId field identifies the record with which the current record shares the link score.

Out_RowId

Contains a unique identifier for each record in the mapping source data set.

The transformation uses the Out_RowId values to identify records if you do not map a field of unique identifiers to the SequenceId field.

Selecting metadata fields

The metadata fields can provide important information about the relationship between duplicate records. For example, the metadata includes the Out_LinkScore field, which represents the degree of similarity between two records as a numerical value. If you select the Out_LinkScore field, select the Out_LinkId field also. The Out_LinkId field identifies the other record in the pair of records that the Out_LinkScore value describes.

The Out_DriverId value provides a benchmark for all records in a matching record set. The Out_DriverId value is the score between the current record and the record in the set with the lowest sequence ID or row ID value. The record with the lowest ID is also the first record that the deduplication process added to the set.

Link scores and driver scores

The deduplication process adds link score and driver score data to the Deduplicate transformation output. You can use the data to better understand the relationship between duplicate records.

The link score is the score between two records that identifies them as members of the same matching set. The score represents a link between a given record and the first record that it matches with a score above the threshold value. The link ID identifies the records to which a link score applies.

The link score and link ID values do not imply that a pair of records are the best match in the input data. The purpose of the link score and link ID values is to describe the composition of the matching record set.

The driver score is the score between the first record added to a matching record set and another record in the same set. The transformation uses the sequence ID or row ID values to identify the first record in the set. Driver scores provide a means to assess all records in the set against a single record.

Note: Duplicate analysis generates a single set of scores for the input records. The driver scores and link scores represent the different relationships between the records and do not indicate different types of duplicate analysis. The driver score and link score assignments can depend on the order in which the records enter the transformation. A driver score for a given pair of records might be lower than the threshold value.

Example of link scores and driver scores

A Deduplicate transformation analyzes records with a column of surname data. The deduplicate asset defines a threshold value of 0.825 for duplicate records.

The following table shows the results that the transformation might return:

Surname	Sequence ID	ClusterId	ClusterSize	DriverId	DriverScore	LinkId	LinkScore
SMITH	1	1	2	1 - 6	1	1 - 1	1
SMYTH	2	2	2	1 - 3	0.83333	1 - 2	1
SMYTHE	3	2	2	1 - 3	1	1 - 2	0.83333
SMITT	4	3	1	1 - 4	1	1 - 4	1
SMITS	5	4	1	1 - 5	1	1 - 5	1
SMITH	6	1	2	1 - 6	1	1 - 1	1

The results provide the following information about the surname data:

- SMITT and SMITS do not match any other record with a score that meets the threshold. The transformation determines that the records are unique in the data set. The transformation can assign score values of 1 to the records because each record matches itself uniquely.

  SMITT and SMITS each have a ClusterSize value of 1, which indicates that they are the only record in their respective sets. To find unique records in the output, search for matching record sets that contain a single record.

- SMITH and SMITH have a link score of 1. The transformation determines that the records are identical. The transformation adds the records to a single matching record set. The ClusterId value indicates that the records belong to the same set.

- SMYTH and SMYTHE match with a score of 0.83333. The score exceeds the duplicate threshold. Therefore, the transformation adds the records to a single matching record set.

Deduplicate transformation output fields

A Deduplicate transformation displays the output fields from the Field Mapping tab and also displays a series of metadata fields that the transformation creates. View the fields on the **Output Fields** tab of the **Properties** panel.

The tab displays the name, type, precision, and scale for the output fields.

You cannot edit the output field properties in the Deduplicate transformation. To edit the properties, open the deduplicate asset in Data Quality.

For more information about the metadata fields that the transformation creates, see "Metadata fields on the Deduplicate transformation" on page 154.

Advanced properties

You can configure advanced properties for a Deduplicate transformation. The advanced properties control the tracing level for session log messages.

You can configure the following property:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

CHAPTER 12

Expression transformation

The Expression transformation calculates values within a single row. Use the Expression transformation to perform non-aggregate calculations.

For example, you might use an Expression transformation to adjust bonus percentages or to concatenate first and last names.

When you configure an Expression transformation, create an expression field for the output of each calculation that you want to use in the data flow. Create a variable field for calculations that you want to use within the transformation.
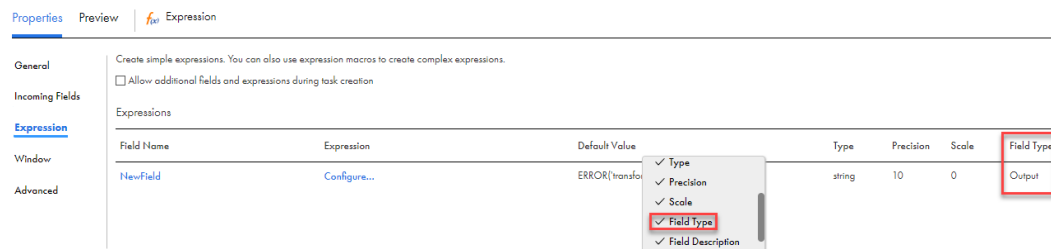
Expression fields

An expression field defines the calculations to perform on an incoming field and acts as the output field for the results. You can use as many expression fields as necessary to perform calculations on incoming fields.

When you configure an expression field, you define the field properties and the calculations that you want to perform. You configure expression fields in the **Expression** tab.

You can create the following types of expression fields:

- Output
- Variable
- Input Macro
- Output Macro

When you have a long list of types and want to see the type for each expression field, add the **Field Type** column in the Expression table as shown in the following image.

Output fields

Create an output field to define a calculation that does not require an expression macro.

The following table describes the properties you can define when you create an output field:

Properties	Description
Name	Name of the field.
Type	The data type of the field.
Precision	Total number of digits in a number. For example, the number 123.45 has a precision of 5. The precision must be greater than or equal to 1. You cannot specify a decimal field precision greater than 38.
Scale	Number of digits to the right of the decimal point of a number. For example, the number 123.45 has a scale of 2. Scale must be greater than or equal to 0. The scale of a number must be less than its precision. The scale may cause zero-filling or truncation with rounding. For example, a scale of 5 causes 123.45 to be expressed as "123.45000" while 123.456789 becomes "123.45679".
Default Value	The default value tells the mapping task what to do when the transformation encounters output errors. The default value is not available in mappings in advanced mode. You can enter one of the following values: - ERROR('transformation error'). When a transformation error occurs, the mapping task skips the row and writes the error to the session log or row error log. - A constant or constant expression. The mapping task replaces the error with the constant or constant expression. Nothing is written to the logs. - ABORT. Transformation aborts and the mapping task writes a message to the session log. Default is ERROR('transformation error'). Data Integration validates the output field default value when you save or validate the mapping. If you enter an invalid value, the Mapping Designer marks the mapping as not valid.
Description	Optional. Description of the field. The description can contain up to 4000 characters.

Variable fields

Create a variable field to define calculations that you want to use within the Expression transformation.

The following table describes the properties you can define when you create a variable field:

Property	Description
Name	Name of the field.
Type	The data type of the field.
Precision	Total number of digits in a number. For example, the number 123.45 has a precision of 5. The precision must be greater than or equal to 1. You cannot specify a decimal field precision greater than 38.

Property	Description
Scale	Number of digits to the right of the decimal point of a number. For example, the number 123.45 has a scale of 2. Scale must be greater than or equal to 0. The scale of a number must be less than its precision. The scale may cause zero-filling or truncation with rounding. For example, a scale of 5 causes 123.45 to be expressed as "123.45000" while 123.456789 becomes "123.45679".
Description	Optional. Description of the field. The description can contain up to 4000 characters.

Input macro field

Create an input macro field to represent the input that you want to use in an expression macro. Depending on the type of macro, the input can be fields or constants.

The following table describes the properties you can define when you create an input macro field:

Property	Description
Name	Name of the input macro field.
Type	The data type of the field.
Precision	Total number of digits in a number. For example, the number 123.45 has a precision of 5. The precision must be greater than or equal to 1. You cannot specify a decimal field precision greater than 38.
Scale	Number of digits to the right of the decimal point of a number. For example, the number 123.45 has a scale of 2. Scale must be greater than or equal to 0. The scale of a number must be less than its precision. The scale may cause zero-filling or truncation with rounding. For example, a scale of 5 causes 123.45 to be expressed as "123.45000" while 123.456789 becomes "123.45679". Not editable for all data types.
Description	Optional. Description of the field. The description can contain up to 4000 characters.

Output macro field

Create an output macro field to define the calculations that you want to perform on all incoming fields or constants. The macro input field represents the incoming fields or constants.

The following table describes the properties you can define when you create an output macro field.

| Property	Description
Input Macro Field	Name of the input macro field that represents the input of the expression macro.
Output Macro Field	Name of the output macro field in the following format: `<prefix>%<input mactiro field name>%<suffix>` The output macro field name must contain either a prefix, suffix, or both.

Property	Description
Suffix	Suffix of the output macro field name. Default is _out.
Prefix	Prefix of the output macro field name. Default is none.
Type	The data type of the field.
Precision	Total number of digits in a number. For example, the number 123.45 has a precision of 5. The precision must be greater than or equal to 1. You cannot specify a decimal field precision greater than 38.
Scale	Number of digits to the right of the decimal point of a number. For example, the number 123.45 has a scale of 2. Scale must be greater than or equal to 0. The scale of a number must be less than its precision. The scale may cause zero-filling or truncation with rounding. For example, a scale of 5 causes 123.45 to be expressed as "123.45000" while 123.456789 becomes "123.45679".
Description	Optional. Description of the field. The description can contain up to 4000 characters.

Expression editor

Use the expression editor to configure the expression field. The expression can contain constants, variables, built-in functions, and user-defined functions. You can also create a complex expression by nesting functions within functions.

To configure an expression, enter it in the expression editor.

The following image shows the expression editor and the actions you can perform:
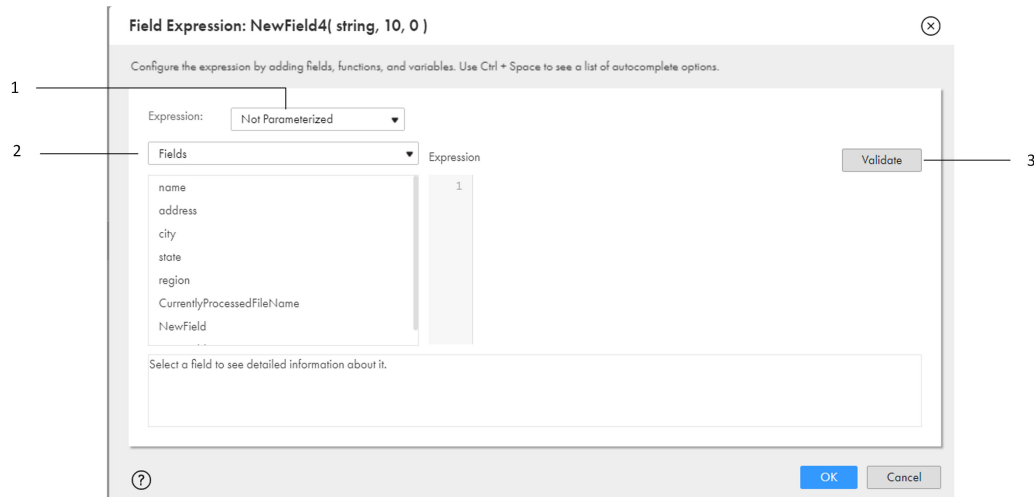
1. Parameterize the expression.
2. Switch between fields, system variables, parameters, built-in functions, and user defined functions.
3. Validate the expression.

You can add source fields, functions, and variables to the expression by clicking **Add** next to the object that you want to use. You can also type in the expression manually.

Alternatively, press the **Ctrl + Space** keys to see a list of recommended arguments and functions in-line. Data Integration provides recommendations based on the type of function arguments and keystrokes. In-line recommendations are not available for hierarchical source data.

To validate the expression, click **Validate**. Data Integration validates the expression.

In mappings in SQL ELT mode, the expression editor displays your cloud data warehouse's native functions instead of Informatica transformation language functions. Data Integration doesn't validate the expression. If you enter an invalid expression, the mapping fails at run time.

Transformation language components for expressions

The transformation language includes the following components to create simple or complex expressions:

- Fields. Use the name of a source field to refer to the value of the field.

- Literals. Use numeric or string literals to refer to specific values.

- Functions. Use these SQL-like functions to change data in a task.

- Operators. Use transformation operators to create expressions to perform mathematical computations, combine data, or compare data.

- Constants. Use the predefined constants to reference values that remain constant, such as TRUE.

In mappings in SQL ELT mode, you use your cloud data warehouse's native expression components to create expressions. You don't use the Informatica transformation language to create expressions.

Expression syntax

You can create a simple expression that only contains a field, such as ORDERS, or a numeric literal, such as 10. You can also write complex expressions that include functions nested within functions, or combine different fields using the transformation language operators.

Note: Although the transformation language is based on standard SQL, there are differences between the two languages.

String and numeric literals

You can include numeric or string literals.

Enclose string literals within single quotation marks. For example:

```
'Alice Davis'
```

String literals are case sensitive and can contain any character except a single quotation mark. For example, the following string is not allowed:

```
'Joan's car'
```

To return a string containing a single quotation mark, use the CHR function:

```
'Joan' || CHR(39) || 's car'
```

Do not use single quotation marks with numeric literals. Just enter the number you want to include. For example:

```
.05
```

or

```
$$Sales_Tax
```

Adding comments to expressions

You can use the following comment specifiers to insert comments in expressions:

- Two dashes:

  ```
  -- These are comments
  ```

- Two forward slashes:

  ```
  // These are comments
  ```

Data integration tasks ignore all text on a line preceded by comment specifiers. For example, to concatenate two strings, enter the following expression with comments in the middle of the expression:

```
  -- This expression concatenates first and last names for customers:
FIRST_NAME -- First names from the CUST table
|| // Concat symbol
LAST_NAME // Last names from the CUST table
// Joe Smith Aug 18 1998
```

Data integration tasks ignore the comments and evaluates the expression as follows:

```
FIRST_NAME || LAST_NAME
```

You cannot continue a comment to a new line:

```
-- This expression concatenates first and last names for customers:
FIRST_NAME -- First names from the CUST table
|| // Concat symbol
LAST_NAME // Last names from the CUST table
Joe Smith Aug 18 1998
```

In this case, data integration tasks do not validate the expression because the last line is not a valid expression.

Reserved words

Some keywords, such as constants, operators, and system variables, are reserved for specific functions. These include:

- :EXT
- :INFA
- :LKP
- :MCR
- :SD
- :SEQ
- :SP
- :TD
- AND
- DD_DELETE
- DD_INSERT
- DD_REJECT
- DD_UPDATE
- FALSE
- NOT
- NULL
- OR
- PROC_RESULT
- SPOUTPUT
- TRUE
- WORKFLOWSTARTTIME

The following words are reserved for Informatica Intelligent Cloud Services:

- ABORTED
- DISABLED
- FAILED
- NOTSTARTED
- STARTED

- STOPPED
- SUCCEEDED

Note: You cannot use a reserved word to name a field. Reserved words have predefined meanings in expressions.

Window functions

In advanced mode, you can use a window function to concisely express 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.

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, configure the following window properties on the **Window** 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.

Partition Keys

Separates the input rows into different partitions.

If you do not define partition keys, all rows belong to a single partition.

Order Keys

Defines how rows in a partition are ordered.

The fields 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.

You cannot parameterize an expression that contains a window function. If the expression is parameterized, you cannot specify a window function in the mapping task.

Frame

The frame determines which rows are included in the calculation for the current input row based on the rows' relative position to the current row. Configure a frame if you use an aggregate function as a window function.

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 describe 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 on the frame in the previous image, 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 describe a frame that includes six total rows, from the tenth to the fifteenth row after the current row.

Offsets of **All Preceding Rows** and **All Following Rows** represent the first row of the partition and the last row of the partition. For example, if the start offset is All Preceding Rows and the end offset is -1, the frame includes one row before the current row and all rows before that.

The following image shows a frame with a start offset of 0 and an end offset of All Following Rows:

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 processes only the rows within the partition. The aggregate function lists the skipped rows in the log file. The function returns one of the following responses for the skipped rows: a default value of ERROR('transformation error'), NULL, or a predefined constant.

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.

Consider the following rules and guidelines when you define a frame:

- LEAD and LAG use the frame that you specify in the function arguments and ignore the frame that you configure on the **Window** tab.

- The start offset must be less than or equal to the end offset.

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 group and order the rows in a window:

Partition keys

Configure partition keys to define partition boundaries rather than performing the calculation across all rows.

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 must also arrange the data in ascending or descending order. If you do not specify order keys, the rows in a partition are arranged randomly.

Consider the following rules and guidelines when you define window properties for partition and order keys:

- You cannot use hierarchical fields as partition or order keys.
- Define unique fields as partition and order keys.

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 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

You can run the MAX function within each partition to determine that the two best-selling coffees are espresso and Americano, and the two best-selling teas are white and black.

Example: Use a window to calculate expiration dates

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

To calculate expiration dates, complete the following tasks:

1. Define partition and order keys.
   You configure the following window properties to partition the data by customer code and order the data by ascending start date:

Property	Value	Description
Frame	Not specified.	The LEAD function will access rows based on the offset argument and ignore the frame.
Partition key	CustomerCode	Groups the rows according to customer code so that calculations are based on individual customers.
Order key	StartDate Ascending	Arranges the data chronologically by ascending start date.

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

CustomerCode	PlanCode	StartDate
C1	00004	2014-10-25
C2	00001	2014-10-01
C2	00002	2014-11-01

2. Define a window function.
   You define a LEAD function to access the subsequent row for every input.

   You configure an expression field that performs the following calculation:

   ```
   LEAD ( StartDate, 1, '01-Jan-2100' )
   ```

   For more information about the LEAD function, see *Function Reference*.

3. Define an ADD_TO_DATE function.
   You use an ADD_TO_DATE function to subtract one day from the date you accessed.

   You configure an expression field that performs the following calculation:

   ```
   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.

Example: Use a window to flag GPS pings

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.

Window properties

You define the following window properties on the **Window** tab:

Property	Value	Description
Frame	Not specified	Window functions access rows based on the offset argument and ignore the frame.
Partition key	`trip_id`	Groups the rows according to trip ID so that calculations are based on events from the same trip.
Order key	`_event_id Ascending`	Arranges the data chronologically by ascending event ID.

Window function

You define the following LAG function to get the event time from the previous row:

```
LAG ( _event_time, 1, NULL )
```

For more information about the LAG function, see *Function Reference*.

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' )
```

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 LAG function produces NULL values.

Example: Run an aggregate function on a window

You want to compare the salary of each employee with the average salary for the corresponding department.

The following table lists the department names, the employee identification number, and the employee 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
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 the salary of each employee and the average salary in the department.

Window Properties

You define the following window properties on the **Window** tab:

Property	Value	Description
Start offset	All Preceding Rows	Describes the number of rows that appear before the current input row.
End offset	All Following Rows	Describes the number of rows that appear after the current input row.
Order key	Salary Ascending	Arranges the data by increasing salary.
Partition key	Department	Groups the rows according to department.

When you select All Preceding Rows and All Following Rows, the function includes all partition rows. 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.

Window Function

Because you configure window properties in the Expression transformation, you can use an aggregate function as a window function.

You define the following aggregate function to calculate the difference between the salary of each employee and the average salary in the corresponding 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 within the same department. Based on this information, you can add other transformations to learn more about the data. For example, you might add a Rank transformation to produce a numerical rank for each employee within the same department.

Advanced properties

You can configure advanced properties for an Expression transformation. The advanced properties control settings such as the tracing level for session log messages and whether the transformation is optional or required.

The properties that are available vary based on the mapping mode.

You can configure the following properties:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Optional	Determines whether the transformation is optional. If a transformation is optional and there are no incoming fields, the mapping task can run and the data can go through another branch in the data flow. If a transformation is required and there are no incoming fields, the task fails.
	For example, you configure a parameter for the source connection. In one branch of the data flow, you add a transformation with a field rule so that only Date/Time data enters the transformation, and you specify that the transformation is optional. When you configure the mapping task, you select a source that does not have Date/Time data. The mapping task ignores the branch with the optional transformation, and the data flow continues through another branch of the mapping.

Hierarchical data in advanced mode

In advanced mode, you can use hierarchical fields that represent an array, map, or struct.

You can use hierarchical fields as pass-through fields. You can also use hierarchical fields in an expression or use a hierarchical field with a complex operator to access primitive child fields in the expression. For more information about complex operators, see *Function Reference*.

Consider the following guidelines when you use hierarchical fields in an expression:

- Do not use a parameter for the expression.
- The output of an expression cannot be an array, map, or struct.

CHAPTER 13

Filter transformation

The Filter transformation filters data out of the data flow based on a specified filter condition. To improve job performance, place the Filter transformation close to mapping sources to remove unnecessary data from the data flow.

A filter condition is an expression that returns TRUE or FALSE. When the filter condition returns TRUE for a row, the Filter transformation passes the row to the rest of the data flow. When the filter condition returns FALSE, the Filter transformation drops the row.

You can filter data based on one or more conditions. For example, to work with data within a data range, you can create conditions to remove data before and after specified dates.

Link a single transformation to the Filter transformation. You cannot merge transformations into the Filter transformation.

Filter conditions

The filter condition is an expression that returns TRUE or FALSE.

You can create one or more simple filter conditions. A simple filter condition includes a field name, operator, and value. For example, Sales > 0 retains rows where all sales values are greater than zero. In advanced mode, the filter condition must evaluate to a numeric result.

Filter conditions are case sensitive. You can use the following operators in a simple filter:

    = (equals)

    < (less than)

    > (greater than)

    < = (less than or equal to)

    > = (greater than or equal to)

    ! = (not equals)

When you define more than one simple filter condition, the mapping task evaluates the conditions in the order that you specify. The task evaluates the filter conditions using the AND logical operator to join the conditions. The task returns rows that match all of the filter conditions.

You can use an advanced filter condition to define a complex filter condition. When you configure an advanced filter condition, you can incorporate multiple conditions using the AND or OR logical operators. You can use a constant to represent condition results: 0 is the equivalent of FALSE, and any non-zero value is the equivalent of TRUE.

When you change the filter condition type from simple to advanced, the Mapping Designer includes configured simple filter conditions in the advanced filter condition. You can use or delete the simple filter conditions. The conversion does not include parameters.

To filter rows that contain null values, use the ISNULL function to test the value of the field. To filter rows that contain spaces, use IS_SPACES.

For example, if you want to filter out rows that contain a null value in the First_Name field, use the following condition: IIF(ISNULL(First_Name),FALSE,TRUE). The condition states that if the First_Name field is NULL, the return value is FALSE. The mapping task discards the row. Otherwise, the row passes through to the next transformation.

In mappings in SQL ELT mode, you need to enter a filter condition that is valid in your target cloud data warehouse.

Advanced properties

You can configure advanced properties for a Filter transformation. The advanced properties control settings such as the tracing level for session log messages and whether the transformation is optional or required.

The properties that are available vary based on the mapping mode.

You can configure the following properties:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Optional	Determines whether the transformation is optional. If a transformation is optional and there are no incoming fields, the mapping task can run and the data can go through another branch in the data flow. If a transformation is required and there are no incoming fields, the task fails.
	For example, you configure a parameter for the source connection. In one branch of the data flow, you add a transformation with a field rule so that only Date/Time data enters the transformation, and you specify that the transformation is optional. When you configure the mapping task, you select a source that does not have Date/Time data. The mapping task ignores the branch with the optional transformation, and the data flow continues through another branch of the mapping.

Hierarchical data in advanced mode

In advanced mode, you can use hierarchical fields that represent an array, map, or struct.

You can use hierarchical fields as pass-through fields. You can also use hierarchical fields in an advanced filter condition or use a hierarchical field with a complex operator to access primitive child fields in the filter condition. For more information about complex operators, see *Function Reference*.

Consider the following guidelines when you use hierarchical fields in a filter condition:

- You cannot use a hierarchical field in a simple filter condition.
- Do not use a parameter for the filter condition.

CHAPTER 14

Hierarchy Builder transformation

The Hierarchy Builder transformation converts relational input into hierarchical output.

It converts data based on a schema that defines the hierarchical structure of the output. The transformation can output hierarchical data in Avro, JSON, ORC, Parquet, or XML format.

To configure a Hierarchy Builder transformation, perform the following tasks:

1. Add the transformation to a mapping and configure the output settings.
2. Join incoming sources and map incoming fields with hierarchical elements to define the data conversion.
3. Optionally, configure advanced properties.
4. If needed, configure the Secure Agent machine for multibyte hierarchical data.

Note: If you use the Hierarchy Builder transformation in a mapplet, to prevent runtime errors, ensure that the combined Mapplet transformation name doesn't exceed 80 characters. For more information, see "Mapplet transformation names" on page 329.

Configure output settings

Configure properties for the output, including the schema and output format.

The following table describes the output properties:

Property	Description
Schema	Schema that defines the structure of the hierarchical output. To convert data to Avro, ORC, or Parquet format, click **Auto-generate from sample file** and create an intelligent structure model. To convert data to JSON or XML format, select an existing hierarchical schema or click **Create new schema**. When you create a hierarchical schema for the Hierarchy Builder transformation, refer to "Rules and guidelines for hierarchical schemas" on page 178. When you update the hierarchical schema, you can choose to retain the existing field mapping. For more information about intelligent structure models and hierarchical schemas, see *Components*.
Precision	Buffer size for the output.
Null Values	Determines whether the transformation keeps or omits null values in primitive JSON types. Applicable when you associate the transformation with a JSON-based schema.

Property	Description
Output Format	Format for the output data, either string or binary.
Write to file	Determines whether the transformation writes the data to a flat file. When you enable this option, enter the file path in the **File path** field. You can parameterize the file path only when you enable the **Optimize transformation** property. **Tip:** Write data to a flat file when the transformation processes a large amount of data and the output field size exceeds 100 MB.
Enable passthrough fields	Determines whether unmapped fields are passed to the downstream transformation. If there is more than one upstream transformation connected to the Hierarchy Builder transformation, the transformation only passes through fields from the upstream transformation that was connected first.
Generate single output	Determines if the transformation generates a single JSON or XML structure containing all the output elements. By default, the transformation generates a JSON or XML output structure for each output element. To generate a single output structure, map an incoming group to a recurring element in the hierarchy fields. For examples of the difference in ouput, see "Single output example" on page 178.
Optimize transformation	Optimizes the processing performance of converting data from relational to hierarchical output at run time. Default is enabled. For more information, see "Rules and guidelines for optimizing the transformation" on page 179.

Rules and guidelines for hierarchical schemas

Consider the following rules and guidelines when you create a hierarchical schema to associate with a Hierarchy Builder transformation:

- The schema must be smaller than 10,000 elements.

- The schema can contain up to 10,000 fields.

- The schema can contain up to 20,000 groups.

- The schema can contain up to 20 levels of data hierarchy.

- To optimize performance, create a schema with fewer than 5,000 ports and fewer than 10 groups.

Single output example

You can configure the Hierarchy Builder transformation to generate a single output structure or output structures for each output element. For example, you're creating an XML file for employee contact information.

If you choose to generate a single output structure, the transformation outputs the following data:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<EMPLOYEE_CONTACT xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <item>
        <PHONE>5551234567</PHONE>
    </item>
    <item>
        <PHONE>5553456789</PHONE>
    </item>
</EMPLOYEE_CONTACT>
```

If you don't generate a single output structure, the transformation outputs the following data:

```
<?xml version="1.0" encoding="UTF-8"?>
<EMPLOYEE_CONTACT xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <item>
        <PHONE>5551234567</PHONE>
    </item>
</EMPLOYEE_CONTACT>
<?xml version="1.0" encoding="UTF-8"?>
<EMPLOYEE_CONTACT xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <item>
        <PHONE>5553456789</PHONE>
    </item>
</EMPLOYEE_CONTACT>
```

Rules and guidelines for optimizing the transformation

Consider the following rules and guidelines when you optimize the processing performance in the output settings:

- When you generate a single output, ensure that you enable the **Write to file** option.

- When you write data to a file, you can parameterize the file path.
  To parameterize the file path, you must add $$ as a prefix and | as a suffix to the parameter name before the file path. You can define the parameter by adding the file path in the [global] section in the downloaded parameters file.

  The following image displays a file path that contains the parameter name along with its prefix and suffix:
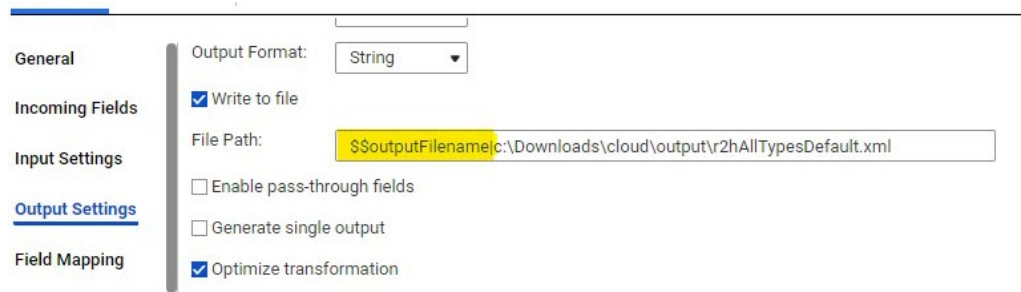
- When you pass through fields from the upstream transformation and the passthrough ports include the **FileName** port, the transformation reads from the port and writes each hierarchical output to the filename specified by this port.

Join and map fields for data conversion

To define the conversion of data from upstream transformations to the hierarchical output, join the incoming data and then map the relationships between fields.

If more than one upstream transformation is connected to the Hierarchy Builder transformation, set primary and foreign keys to join the incoming data before you map the fields. If only one upstream transformation is connected to the Hierarchy Builder transformation, the data will be treated as denormalized input, and you don't define primary and foreign keys. Then, map the input fields to the matching field in the hierarchy.

The **Field Mapping** tab displays the relational fields from the upstream transformations and the hierarchy fields from the schema. Use this tab to join incoming sources and map the relationships between fields. The following image shows the **Field Mapping** tab:
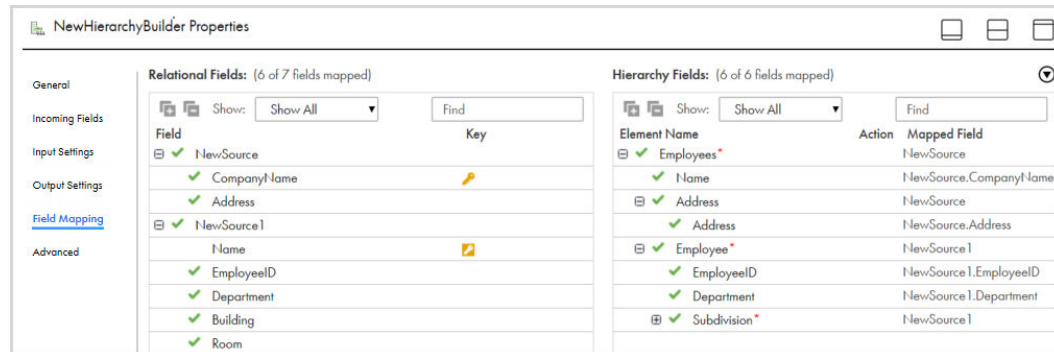
Joining incoming data

If more than one upstream transformation is connected to the Hierarchy Builder transformation, set primary and foreign keys to join the incoming data.

You can perform the following actions to set and remove keys:

- To set a field as a key, click the **Key** column of the field that you want to set, and then specify the field as either a primary or foreign key. When you set a foreign key, you also select the name of the source that contains the related primary key.

- To remove a primary or foreign key designation, click the **Key** column of the field and select **Not a key**.

- To clear all the keys, click the action menu at the top of the **Hierarchy Fields** panel, and then select **Clear Keys**.

Note: You can set only bigint and string fields as keys.

Mapping relational fields to hierarchy fields

Map relational input fields to hierarchy elements to define how the transformation converts the data.

You can perform the following actions to map and unmap fields:

- To map a relational input field to a schema element, drag the relational field to the schema element.

- To remove a mapped field from a schema element, click the delete icon in the **Action** column for the schema element.

- To clear all of the mapped fields, click the action menu at the top of the **Hierarchy Fields** panel and select **Clear Mapping**.

Configure advanced properties

You can configure advanced properties for a Hierarchy Builder transformation. The advanced properties control the tracing level for session log messages and the transformation scope.

You can configure the following properties:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Transformation Scope	Specifies how Data Integration applies the transformation logic to incoming data. Select one of the following options: - Transaction. Applies the transformation logic to all rows in a transaction. Choose Transaction when a row of data depends on all rows in the same transaction, but does not depend on rows in other transactions. - All Input. Applies the transformation logic on all incoming data. When you choose All Input, Data Integration drops incoming transaction boundaries. Choose All Input when a row of data depends on all rows in the source. Default is *All Input*.

Configuration for multibyte hierarchical data

If a mapping includes a transformation that processes hierarchical data and the data uses multibyte characters, configure the Secure Agent machine to use UTF-8.

On Windows, create the INFA_CODEPAGENAME=UTF-8 environment variable in Windows System Properties.

On Linux, set the LC_LOCALE variable to UTF-8.

Hierarchy Builder transformation example

You want to convert relational data to hierarchical data and write the data to a target file in a hierarchical format.

To use the Hierarchy Builder transformation to convert the data, complete the following steps:

1. Create a hierarchical schema.
2. Create a mapping.
3. Configure the Hierarchy Builder transformation.
4. Configure the target.
5. Run the mapping.

Step 1. Create a hierarchical schema

You need to configure a hierarchical schema that uses a schema file to define the hierarchy of the output data.

The following example shows the schema hierarchy that you want to use:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.itemfield.com"
targetNamespace="http://www.itemfield.com" elementFormDefault="qualified">
 <xs:element name="Employees">
  <xs:complexType>
   <xs:sequence>
     <xs:element name="Name" type="xs:string" minOccurs="0"/>
     <xs:element name="Address" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
     <xs:element name="Employee" minOccurs="1" maxOccurs="unbounded">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="EmployeeID" type="xs:string" minOccurs="0"/>
        <xs:element name="Department" type="xs:string" minOccurs="0"/>
        <xs:element name="Subdivision" maxOccurs="unbounded">
         <xs:complexType>
          <xs:sequence>
           <xs:element name="Building" type="xs:string" minOccurs="0"/>
           <xs:element name="Room" type="xs:string" minOccurs="0"
           maxOccurs="unbounded"/>
          </xs:sequence>
         </xs:complexType>
        </xs:element>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

The following example shows the first input file data that you want to use:

```
CompanyName,Address
First National Bank,874 Louis Road
Jackson Industry,13 Sydney Drive
```

The following example shows the second input file data that you want to use:

```
Name,EmployeeID,Department,Building,Room
First National Bank,122,Credit,6,1532
First National Bank,261,Credit,6,2251
First National Bank,431,Credit,6,5312
Jackson Industry,3875,Manufacture,C,673
Jackson Industry,2837,Manufacture,B,211
```

Create the hierarchical schema in Data Integration with the schema hierarchy that you want to use.

Step 2. Create a mapping

To parse the input data, use a Hierarchy Builder transformation in a mapping to transform the data from the hierarchical input.

In the Mapping Designer, you add two source objects that are flat files that contain the paths to the data files that you want to parse. The following image shows one of the Source transformations:
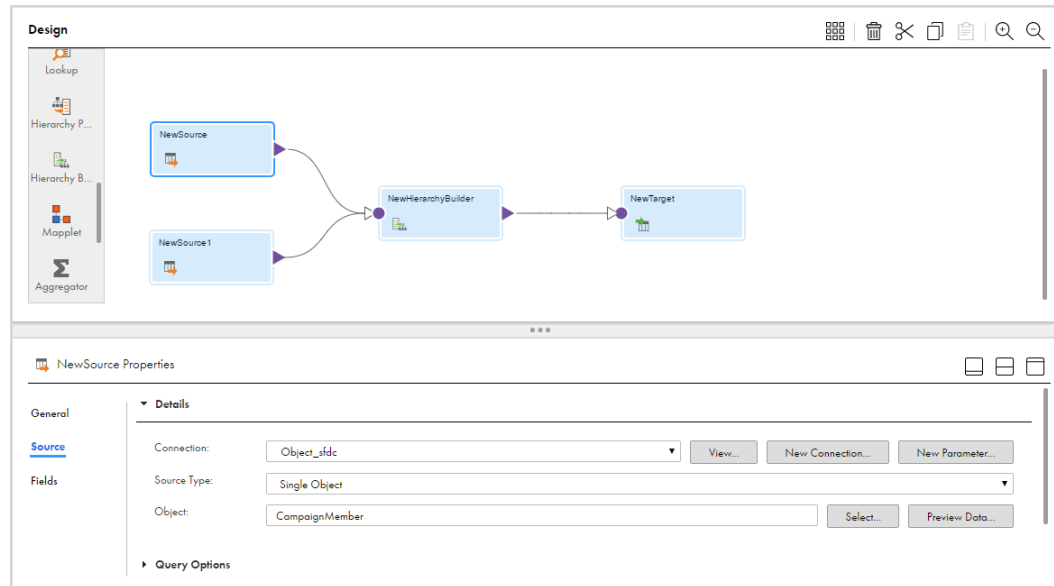
Step 3. Configure the Hierarchy Builder transformation

You add an Hierarchy Builder transformation and use the name NewHierarchyBuilder. Configure it to use the hierarchical schema that you created.

You connect the source objects to the NewHierarchyBuilder transformation. To map the incoming data to the fields of the transformation, select the NewHierarchyBuilder transformation. In the **Incoming Fields** tab, ensure that there are no field name conflicts. The following image shows the input field selection:
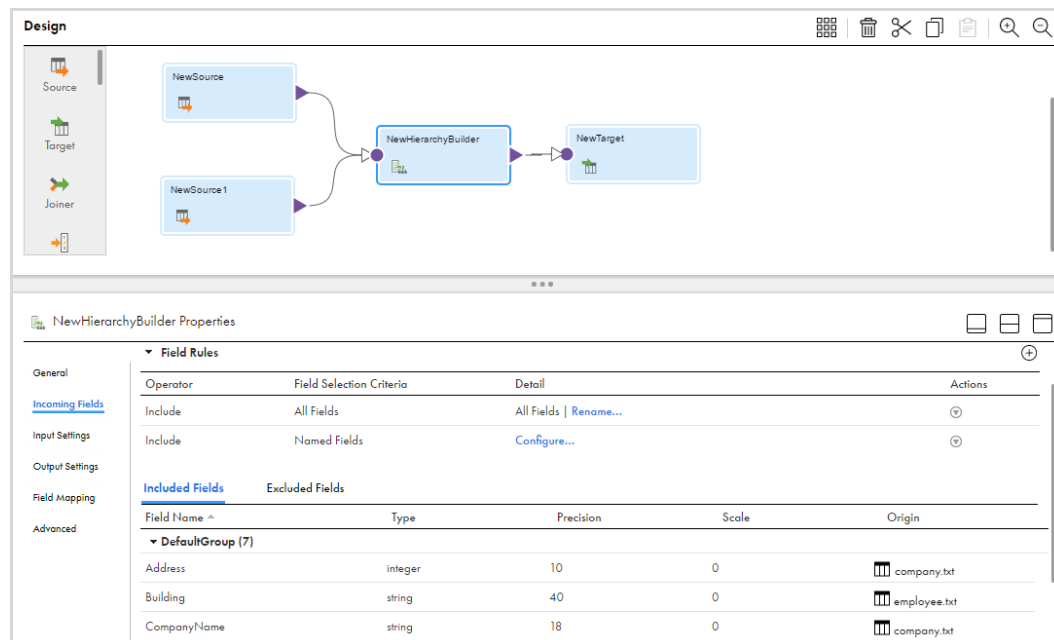
To map the relational fields to the hierarchical output, in the **Field Mapping** tab, select primary and foreign keys. Then select which relational fields are linked to schema elements for the hierarchical output.

The following image shows the field mapping selection:
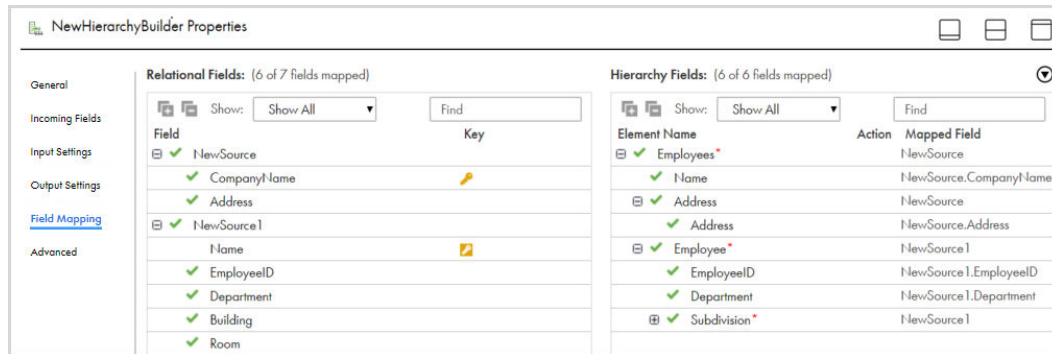
Step 4. Configure the target

Add a file target object for the fields.

The following image shows the final mapping:
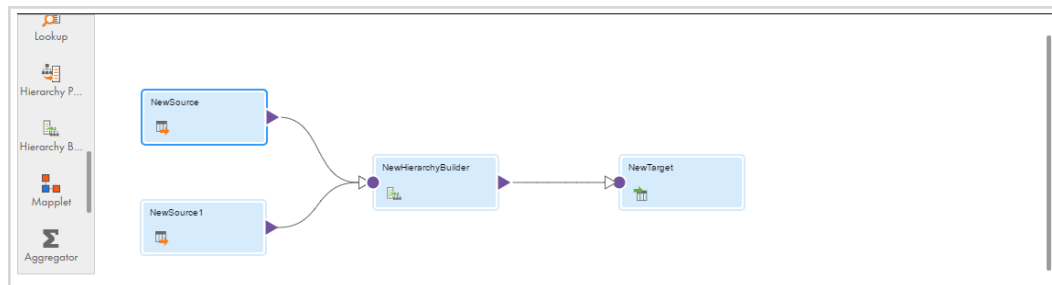
Step 5. Run the mapping

Run the mapping to write the data in a hierarchical format to the Target transformation.

The following example shows the hierarchical output:

```
<Employees>
<Name>First National Bank</Name>
<Address>874 Louis Road</Address>
<Employee>
<EmployeeID>122</EmployeeID>
<Department>Credit</Department>
<Subdivision>
<Building>6</Building>
<Room>1532</Room>
</Subdivision>
</Employee>
<Employee>
<EmployeeID>261</EmployeeID>
<Department>Credit</Department>
<Subdivision>
<Building>6</Building>
<Room>2251</Room>
</Subdivision>
</Employee>
<Employee>
<EmployeeID>431</EmployeeID>
<Department>Credit</Department>
<Subdivision>
<Building>6</Building>
<Room>5312</Room>
</Subdivision>
</Employee>
</Employees>
<Employees>
<Name>Jackson Industry</Name>
```

```
<Address>13 Sydney Drive</Address>
<Employee>
<EmployeeID>3875</EmployeeID>
<Department>Manufacture</Department>
<Subdivision>
<Building>C</Building>
<Room>673</Room>
</Subdivision>
</Employee>
<Employee>
<EmployeeID>2837</EmployeeID>
<Department>Manufacture</Department>
<Subdivision>
<Building>B</Building>
<Room>211</Room>
</Subdivision>
</Employee>
</Employees>
```

CHAPTER 15

Hierarchy Parser transformation

The Hierarchy Parser transformation converts hierarchical input into relational output. The transformation processes XML or JSON input from the upstream transformation and provides relational output to the downstream transformation.

You can configure a hierarchical schema that defines the expected hierarchy of the output data from a sample file or schema file. The Hierarchy Parser transformation converts hierarchical input based on the hierarchical schema that you associate with the transformation. You can use an existing hierarchical schema, or configure one.

Note: To create a JSON-based schema object use a JSON sample, not a JSON schema.

To parse complex hierarchical structures, consider using the Structure Parser transformation for a more comprehensive handling of hierarchical file inputs. For more information, see .

Using a Hierarchy Parser transformation

To use the Hierarchy Parser transformation in a mapping, perform the following steps:

1. Create a hierarchical schema.
2. Add a Hierarchy Parser transformation to the mapping.
3. Associate a hierarchical schema with the Hierarchy Parser transformation.
4. Configure the field mapping to select which schema elements provide relational output.

Hierarchy Parser rules and guidelines

When you use the Hierarchy Parser transformation, consider the following rules and guidelines.

Source file size

The Hierarchy Parser transformation can process source files that are up to 1.5 GB in size. To process files that are larger than 1.5 GB, use an intelligent structure model. For more information about intelligent structure models, see *Components*.

Handling of Boolean data types

The Hierarchy Parser transformation always returns 0 for Boolean data type input.

Hierarchical schema that is based on an XSD file

When you associate a hierarchical schema that is based on an XSD file with a Hierarchy Parser transformation, note that you can't use the Hierarchy Parser transformation with the following XSD component types:

- xsd:any
- xsd:type
- mixed
- xsi:type
- default values
- fixed values
- no type, for example `<xs:element name="A" maxOccurs="unbounded"/>`

To parse large or complex XSD files that contains more than 10,000 elements and attributes, recursive elements, or complex deep hierarchies, use the Structure Parser transformation. For more information, see Chapter 35, "Structure Parser transformation" on page 415.

Using a Hierarchy Parser transformation in a Mapplet transformation

When you use a Hierarchy Parser transformation in a mapplet, to prevent runtime errors, be sure that the combined Mapplet transformation name doesn't exceed 80 characters. For more information, see "Mapplet transformation names" on page 329.

Choosing a sample or schema file

When you create a hierarchical schema, you base it on either a sample file or a schema file. To improve accuracy, use a schema file instead of a sample file whenever possible.

Schema files

When you use a schema file, consider the following rules and guidelines:

- The schema can't contain recursive elements.
- Each schema file can contain a maximum of 10,000 elements. To use a schema that contains more than 10,000 elements, split it into multiple files.
- To use a schema that references other schemas, you can upload each file individually or upload them together in a ZIP file. If the ZIP file contains other files in addition to XSD files, Data Integration ignores the additional files and uses only the XSD files to generate the hierarchical schema.
- If you need to upload multiple schema files with the same name, upload them in a ZIP file. If you upload them individually, Data Integration renames the duplicate files.

Sample files

When you use a sample file, ensure that it represents the data that you expect to process, including all possible fields, all permutations of the values and data types, and values that represent the length of the data that the mapping will process.

Hierarchical schemas

A hierarchical schema is based on a schema file or sample file that you import to Data Integration. If you import a sample file, Data Integration generates a schema based on the structure of the sample file. The schema defines the expected hierarchy of the input data.

You can create a hierarchical schema in two ways. You can create a standalone hierarchical schema that can be associated with any transformation that you choose. Alternatively, you can create a hierarchical schema from within a specific transformation.

When you create a standalone hierarchical schema, you import a JSON sample file or XSD file as the basis of the schema.

You can associate the hierarchical schema with any transformation, whether you create it as a standalone hierarchical schema or as part of a specific transformation.

You can create, edit, or delete a hierarchical schema. You can edit a hierarchical schema and change the root or change the schema definition. However if you have used the hierarchical schema in a transformation, you cannot edit or delete it. Before you delete a hierarchical schema, verify that you did not use it in a transformation.

Rules and guidelines for hierarchical schemas

Consider the following rules and guidelines when you create a hierarchical schema to associate with a Hierarchy Parser transformation:

- The schema must be smaller than 10,000 elements.
- The schema can contain up to 10,000 fields.
- The schema can contain up to 500 groups.
- The schema can contain up to 20 levels of data hierarchy.
- Schemas that contain more than 5,000 ports and more than 100 groups might affect the processing time.

Creating a hierarchical schema

To create a hierarchical schema, define general properties and upload a sample or schema file.

1. Click **New** > **Components** > **Hierarchical Schema**.
2. On the **New Hierarchical Schema** page, specify a name and location for the hierarchical schema, or use the default values.
3. Optionally, enter a description of the hierarchical schema.
4. Click **Upload** to select a file as the basis of the hierarchical schema.
5. In the **Upload Schema/Sample File** dialog box, click **Choose File** to browse for a file, and then click **OK** to upload it.
6. If you upload an XSD file that references other XSD files, Data Integration displays a list of the referenced files. Click **Upload** to upload each referenced file.
7. If you upload an XSD file with multiple possible root elements, select a root element from the drop-down menu.
8. To save the hierarchical schema, click **OK**.

Input settings

The Hierarchy Parser transformation converts hierarchical input to relational data based on the hierarchical schema that you associate with the transformation.

The following table describes the input properties:

Property	Description
Input Type	Select the **Buffer** input type if the source transformation passes data to the Hierarchy Parser transformation. Select the **File** input type if the source transformation passes data by reference instead. If the source transformation passes a path to an input file, this applies.
Schema	Choose an existing hierarchical schema and add it to the transformation or create a hierarchical schema from an XSD file or a JSON sample file. When you update the hierarchical schema, you can choose to retain the existing field mapping.
Enable pass-through fields	Determines whether unmapped fields are passed to the downstream transformation. If there is more than one upstream transformation connected to the Hierarchy Parser transformation, the transformation passes through fields from the upstream transformation that was connected first.
Optimize transformation	Optimizes the processing performance of converting data from hierarchical to relational output at run time. Default is enabled.

Selecting a hierarchical schema

Choose the hierarchical schema for the Hierarchy Parser transformation.

1. In the **Properties** panel of the Hierarchy Parser transformation, click the **Input Settings** tab.
2. Click **Select**.

   The **Select Schema** dialog box appears.
3. Select a hierarchical schema from the list.
4. To search for a hierarchical schema, select the search criteria, enter the characters to search for in the Search field, and click **Search**.

   You can search for a hierarchical schema by name or description. You can sort the hierarchical schema by name, description, or date last modified.
5. Select the hierarchical schema to include in the Hierarchy Parser transformation and click **OK**.

   The selected hierarchical schema appears in the **Properties** panel.

Creating a hierarchical schema from sample

1. In the **Properties** panel of the Hierarchy Parser transformation, click the **Input Settings** tab.
2. To create a schema, click **New**.

   The **New Hierarchical Schema** page appears.
3. In the **New Hierarchical Schema** page, enter a name and description. You must provide a name for the hierarchical schema.

4. Browse to select a project location.

5. To select a schema or sample file, click **Upload**. Click **Choose File** and browse for an XSD file or select a sample JSON file, and then click **OK**.

   When you add a JSON sample file, Data Integration generates a schema from the sample.

6. If you selected an XSD file with multiple possible root elements, select a root from the drop-down menu.

7. If you selected a schema that refers to another schema, you must also upload the referenced schema. To upload the referenced schema, click **Upload**, browse for the referenced schema file and click **OK**.

8. To save the hierarchical schema, click **OK**.

Input field selection

Configure the input field selection in a Hierarchy Parser transformation to define which field in the upstream transformation to map to the input field in the Hierarchy Parser transformation.

Configure the **Input Field Selection** tab of the **Properties** panel.

You can configure the following input field selection options:

Automap

Links fields with matching names.

You can map fields in the following ways:

- Exact Field Name. Data Integration matches fields of the same name.
- Smart Map. Data Integration matches fields with similar names. For example, if you have an incoming field `Employee_Name` and an hierarchical schema input field `Emp_Name`, Data Integration automatically links the `Employee_Name` with the `Emp_Name` field.

You can use both Exact Field Name and Smart Map in the same field mapping. For example, use Exact Field Name to match fields with the same name and then use Smart Map to map fields with similar names.

You can undo all automapped field mappings by clicking **Automap** > **Undo Automap**. To unmap a single field, select the field to unmap and click **Actions** > **Unmap**.

Data Integration highlights newly mapped fields. For example, when you use Exact Field Name, Data Integration highlights the mapped fields. If you then use Smart Map, Data Integration only highlights the fields mapped using Smart Map.

Options

Controls the fields that appear in the Incoming Fields list. Show all fields, unmapped fields, or mapped fields. Determine how field names appear in the input field list. Use technical field names or labels.

Field mapping

Configure the field mapping in a Hierarchy Parser transformation to define which schema elements provide the relational output. Configure the field mapping on the **Field Mapping** tab of the **Properties** panel. Use the

field mapping editor to select or exclude schema elements. When you exclude schema elements, the editor removes the corresponding relational fields.

The following image shows the field mapping editor.
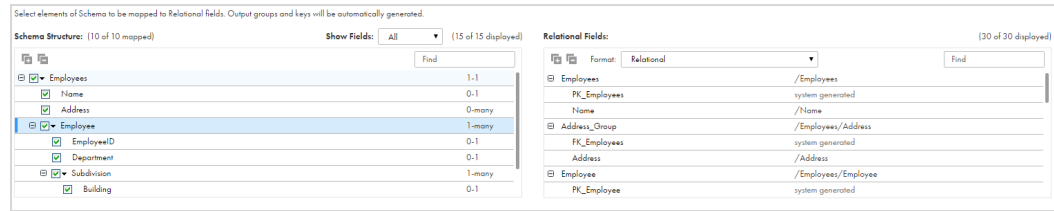
To the left, the field mapping editor shows the schema elements. To the right, the editor shows the relational fields. The transformation creates a separate output group for each multiple-occurring input element. The transformation also creates primary and foreign keys and assigns them to the groups. A primary key is signified by the prefix PK_ in the name of the field. A foreign key is signified by the prefix FK_.

For each relational field, the Relational Fields panel shows the XPath expression of the hierarchy element from which the relational field was mapped.

When you select to include schema elements, the editor displays corresponding relational fields to the right. When you exclude schema elements, the editor removes the corresponding relational fields.

You can select to include or exclude all child elements nested under an element. Alternatively, you can select to include or exclude immediate child elements that are one hierarchy level down.

You can select to denormalize relational output. When you denormalize relation output, all the hierarchy elements you select to map are mapped to relational fields in one group.

You can delete relational fields and groups. When you delete a group whose primary key serves as a foreign key for other groups, the editor updates or deletes the other groups. If the group you deleted was a reflection of a repeating element nested within another repeating element, the primary key for the higher level repeating element becomes the foreign key for the other groups. If the group you delete is a reflection of the highest level repeating element, the editor deletes all the groups.

Selecting the elements to convert

Select the schema elements to convert to relational output fields.

1. In the **Properties** panel of the Hierarchy Parser transformation, click the **Field Mapping** tab.

    The field mapping editor displays the transformation input elements and output fields. To the left, the editor shows the schema elements. To the right, the editor shows the relational output fields.

2. To view child elements, expand an element.

3. To search for an element, enter the characters to search for in the Search field, and click **Search**.

4. To denormalize the relational output, in the **Format** field select **Denormalized**, then select elements to include in the relational output. By default, the relational output is not denormalized, and the setting for the **Format** field is **Relational**.

5. To include an element without child elements in the relational output, select the element.

6. To exclude an element without child elements from the relational output, clear the element.

7. For an element with child elements, you can choose from the following options:

    -   To select to include all child elements in the relational output, right-click the element and select **Map all descendants**.

- To select to include immediate child elements in the relational output, right-click the element and select **Map immediate children**.

- To select to exclude all child elements in the relational output, right-click the element and select **Unmap all descendants**.

- To select to exclude immediate child elements in the relational output, right-click the element and select **Unmap immediate children**.

8. To search for a relational field, enter the characters to search for in the Search field, and click **Search**.

9. To exclude a relational output field from the output, click the field.

Output fields

When you select schema elements to use in the Hierarchy Parser transformation, the schema output fields appear on the **Output Fields** tab of the Properties panel.

The Mapping Designer displays the name, type, precision, scale, and origin for each output field in each output group.

To edit the precision of the output fields, click the precision and enter the precision you require.

You can edit the transformation output fields, except for primary key and foreign key fields.

Selecting an output group

When you link the transformation to a downstream transformation, you must select one output group to map to the downstream transformation.

1. Link the transformation to a downstream transformation.

   The **Select Output Group** dialog box appears.

2. To search for an output group, enter the characters to search for in the Search field, and click **Search**.

3. Select an output group.

4. Click **OK**.

Configuration for multibyte hierarchical data

If a mapping includes a transformation that processes hierarchical data and the data uses multibyte characters, configure the Secure Agent machine to use UTF-8.

On Windows, create the INFA_CODEPAGENAME=UTF-8 environment variable in Windows System Properties.

On Linux, set the LC_LOCALE variable to UTF-8.

Hierarchy Parser transformation example

You want to convert hierarchical data to relational data and write the data to a target file in a relational format.

You need to configure a hierarchical schema that uses a schema file to define the hierarchy of the input data.

The following example shows the schema hierarchy that you want to use:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.itemfield.com"
targetNamespace="http://www.itemfield.com" elementFormDefault="qualified">
<xs:element name="root">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Emp_Details" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="employee">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="Employeeid" type="xs:short"/>
          <xs:element name="Name">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="Firstname" type="xs:string"/>
             <xs:element name="Lastname" type="xs:string"/>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
          <xs:element name="Dependents" minOccurs="1" maxOccurs="unbounded"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element name="address" minOccurs="0">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="Addressid" type="xs:byte"/>
          <xs:element name="Employeeid" type="xs:short"/>
          <xs:element name="Line1" type="xs:string"/>
          <xs:element name="Line2" type="xs:string"/>
          <xs:element name="City" type="xs:string"/>
          <xs:element name="State" type="xs:string"/>
          <xs:element name="Zipcode" type="xs:int"/>
          <xs:element name="Fromdate" type="xs:date"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
      <xs:attribute name="EmpAttribute" type="xs:string"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

The following example shows the input file data that you want to use:

```
<itm:root xmlns:itm="http://www.itemfield.com">
  <!--Zero or more repetitions:-->
  <itm:Emp_Details EmpAttribute="string">
    <itm:employee>
      <itm:Employeeid>1</itm:Employeeid>
      <itm:Name>
        <itm:Firstname>Gina</itm:Firstname>
        <itm:Lastname>Aniston</itm:Lastname>
      </itm:Name>
      <!--1 or more repetitions:-->
      <itm:Dependents>anyType</itm:Dependents>
```

```
        </itm:employee>
        <!--Optional:-->
        <itm:address>
          <itm:Addressid>2</itm:Addressid>
          <itm:Employeeid>1</itm:Employeeid>
          <itm:Line1>Long Tech Park</itm:Line1>
          <itm:Line2>Industrial Zone</itm:Line2>
          <itm:City>Wichita</itm:City>
          <itm:State>KA</itm:State>
          <itm:Zipcode>773301</itm:Zipcode>
          <itm:Fromdate>2011-09-29</itm:Fromdate>
        </itm:address>
      </itm:Emp_Details>
    </itm:root>
```

Create the hierarchical schema in Data Integration with the schema hierarchy that you want to use.

To parse the input data, use a Hierarchy Parser transformation in a mapping to transform the data from the hierarchical input. In the Mapping Designer, you add a source object that is flat file that contains the path to the data that you want to parse.

The following image shows the selected Source transformation:
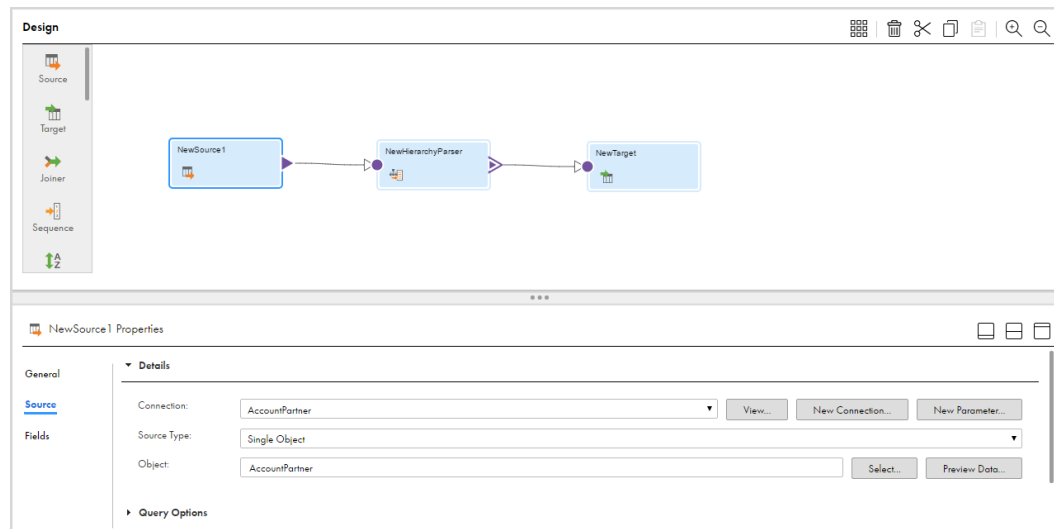
You add an Hierarchy Parser transformation and use the name NewHierarchyParser. Configure it to use the hierarchical schema that you created.

You connect the source object to the NewHierarchyParser transformation. To map the incoming data to the fields of the transformation, select the NewHierarchyParser transformation. In the **Input Field Selection** tab, map the selected incoming field from the source transformation to the NewHierarchyParser hierarchical schema input field.

To map the data to relational fields, in the **Field Mapping** tab, select which schema elements are reflected as relational fields to the output.
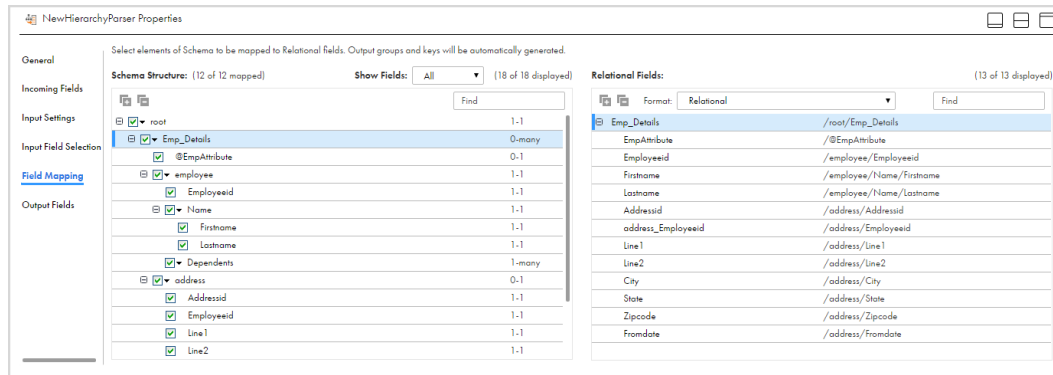
The following image shows the field mapping selection:

Add a file target object for the fields.

The following image shows the mapping:
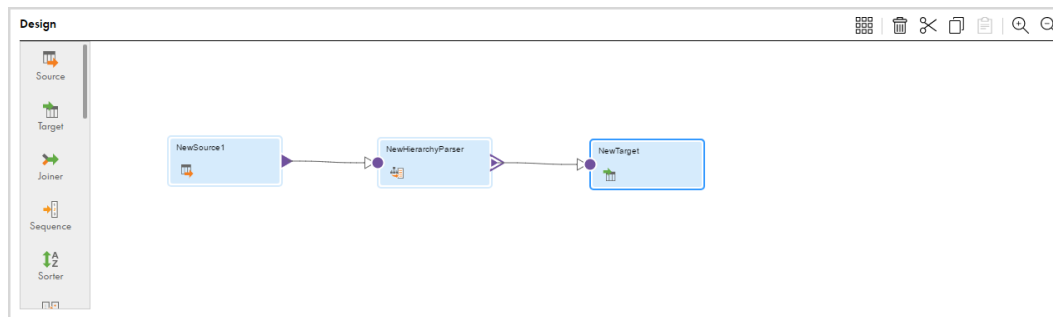
Run the mapping to write the data in a relational format to the Target transformation.

The following example shows the relational output:

EmpAttribute	Employeeid	Firstname	Lastname	Addressid	Employeeid1	Line1	Line2
string	1	Gina	Aniston	2	1	Long Tech Park	Industrial Zone

City	State	Zipcode	Fromdate
Wichita	KA	773301	9/29/2008 12:00:00 AM

CHAPTER 16

Hierarchy Processor transformation

In advanced mode, you can use the Hierarchy Processor transformation to process data from complex data sources. The transformation can read hierarchical or relational input and convert it to relational, hierarchical, or flattened denormalized output.

The Hierarchy Processor transformation is an active transformation that processes hierarchical fields that represent a struct or an array.

Hierarchy Processor transformation overview

You can choose from several different processing strategies to meet your needs, depending on the format of the source data and your desired output format.

The Hierarchy Processor transformation can operate in the following modes:

- Hierarchical to relational. Converts one hierarchical input group to multiple output groups, which can include delimited flat files or relational files.

- Relational to hierarchical. Converts up to five relational input groups to one hierarchical output group.

- Hierarchical to hierarchical. Converts one or more hierarchical input groups to one hierarchical output group with a different schema.

- Hierarchical to flattened. Converts one hierarchical input group to one flattened denormalized output group.

The default output format is relational.

The data that you pass to or from the Hierarchy Processor transformation requires a Microsoft Azure Data Lake Store V2 or an Amazon S3 V2 connection.

For more information about the Hierarchy Processor transformation, you can watch the following videos on YouTube:

- [Creating a Hierarchical to Relational Mapping Using the Hierarchy Processor Transformation](#)
- [Creating a Relational to Hierarchical Mapping Using the Hierarchy Processor Transformation](#)
- [Modifying the Schema of Hierarchical Data in an Elastic Mapping](#)

Hierarchical to relational data processing

In a mapping that converts hierarchical data to relational output, you can process one hierarchical input group and write the data to multiple relational output groups. The output data can be written as normalized relational data or to delimited flat files.

The following image shows an example mapping:
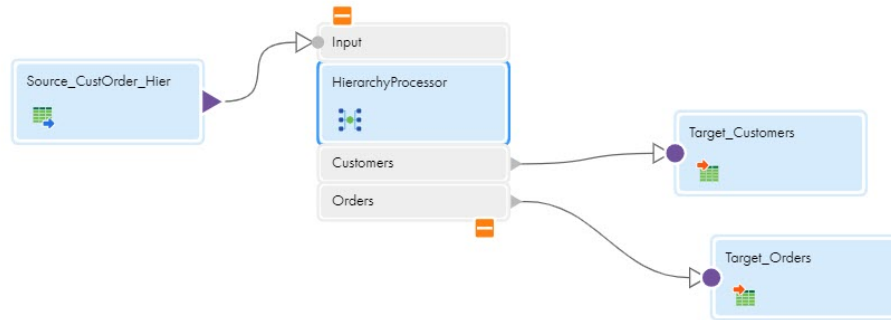
In this mapping, the data source is a complex file containing customer and order data. The data flows into two relational files: a file with customer data and a file with order data.

For more information, see "Defining relational output with the Hierarchy Processor transformation" on page 200.

Relational to hierarchical data processing

In a mapping that converts relational data to hierarchical output, you can have up to five relational input groups and write to one hierarchical output group. This transformation allows you to create structs and arrays. It also allows you to join data sources, group by and order by data fields, filter for specific information, and aggregate both the input and output data.

The following image shows an example mapping:
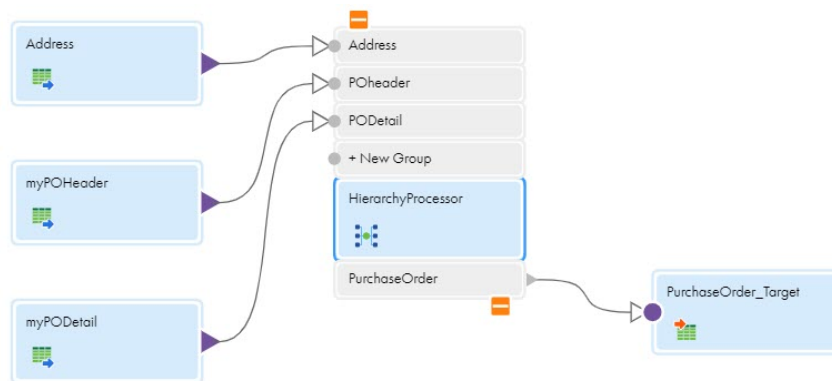
In this mapping, the source input includes three relational files: customer address data, purchase orders, and purchase order details. The data flows into one complex file that combines data from the three source files.

For more information, see "Defining hierarchical output with the Hierarchy Processor transformation" on page 210.

Hierarchical to hierarchical data processing

In a mapping that converts hierarchical data to hierarchical data, you can read from one or more hierarchical input groups and write to one hierarchical output group.

You can convert hierarchical input from one schema to a different schema. You can read data from primitive fields, structs, and arrays and arrange the data in a different structure.

You can also transform the data that you are converting. You can join data sources, configure group by and order by fields, filter for specific information, and aggregate incoming and output data.

The following image shows an example of a mapping that uses a Hierarchy Processor transformation to convert hierarchical data to hierarchical data of a different structure:

In this mapping, the data source is a JSON file that contains orders and items data. The data flows into a different JSON file that contains order information. The Hierarchy Processor transformation is selected, and the **Hierarchy Processor** tab shows the structure of the incoming and output data.

For more information, see "Defining hierarchical output with the Hierarchy Processor transformation" on page 210.

Hierarchical to flattened data processing

The Hierarchy Processor transformation includes a flattened option for output data. Use the flattened output format to convert hierarchical input into denormalized output.

In a mapping that converts hierarchical data to flattened data, you can read from one hierarchical input group and write to one flattened output group. You can read data from primitive fields, structs, and arrays and quickly create a fully denormalized output file. You can also flatten and denormalize only a portion of the incoming fields.

For data sources that contain sibling arrays, you can easily denormalize the output data without the need for complex joins. Select the check box next to the incoming fields you want. The Hierarchy Processor transformation adds the field to the output and creates the expression automatically.

The following image shows a mapping that uses a Hierarchy Processor transformation to convert hierarchical data to flattened data:
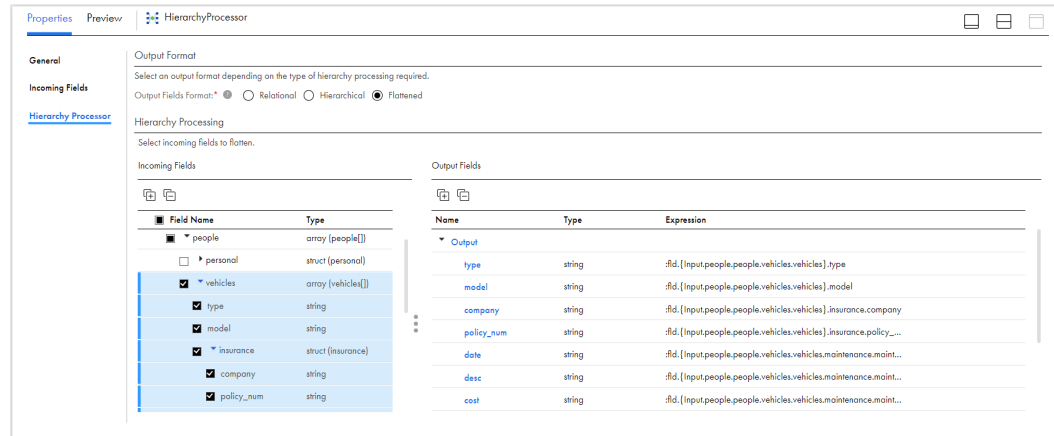
In this mapping, the data source is a JSON file that contains personal and vehicle data. The data flows into a flattened file that contains vehicle information. The Hierarchy Processor transformation is selected, and the **Hierarchy Processor** tab shows the structure of the incoming and output data.

For more information, see "Defining flattened output with the Hierarchy Processor transformation" on page 241.

Configuration for multibyte hierarchical data

If a mapping includes a transformation that processes hierarchical data and the data uses multibyte characters, configure the Secure Agent machine to use UTF-8.

On Windows, create the INFA_CODEPAGENAME=UTF-8 environment variable in Windows System Properties.

On Linux, set the LC_LOCALE variable to UTF-8.

Field restrictions

Elastic mappings support up to 7,000,000 input and output fields. All mappings are subject to this limit. However, mappings that include the Hierarchy Processor transformation are more likely to approach the limit because of the complex data involved.

The more joins, child fields, nested fields, and flattened arrays that the Hierarchy Processor transformation contains, the more likely the mapping will exceed the field limit.

If the mapping exceeds the limit, the following message appears in the mapping compilation log:

```
[LDTM_0502] The mapping [<mapping name>] failed because the number of fields in the
compiled mapping exceeds the threshold: [7,000,000]. Number of fields: [<actual
number>]. Create multiple mappings to process the data incrementally.
```

To resolve the error, create multiple mappings to process the complex data incrementally or reduce the size of the mapping.

Processing relational output

The Hierarchy Processor transformation can convert one hierarchical input group to multiple output groups. These output groups can contain delimited flat files or relational files.

For example, you have a customer order file with customer and order information. This file is in hierarchical JSON format. You wish to create a relational customers table to update information in the master database. You also want a separate delimited orders file to see which orders have been increasing.

Defining relational output with the Hierarchy Processor transformation

To define a Hierarchy Processor transformation, use the **Hierarchy Processor** tab to map incoming fields to output fields, configure the output data structure, and optionally generate keys for relational output.

The following image shows the **Hierarchy Processor** tab with hierarchical input and relational output:
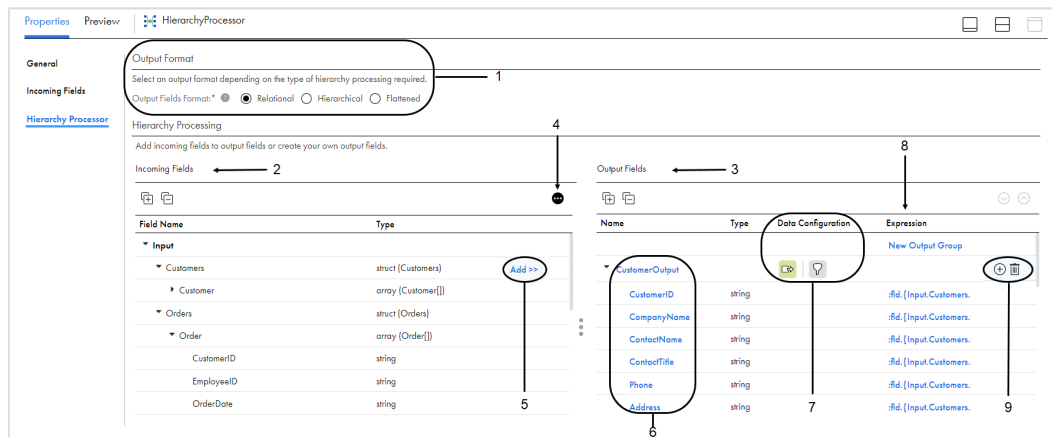
1. Output format. Select **Relational** to convert incoming hierarchical data into one or more relational output groups.
2. Input groups, incoming fields. Use these fields to map to the output fields.
3. Output group, output fields. Use these fields to create the complex output file.
4. Generate keys. Optionally generate keys for the input group to define relationships between output groups.
5. Add incoming field to output group. Use to add fields to the output group.
6. Output field names. Click on a field name to modify the field name or data type.
7. Data Configuration icons. Use to configure the output groups and fields.
8. Expression. Click on an expression to view or customize the output field expression.
9. Add or delete output field. Use to create or delete output fields.

Tip: You can resize the Incoming Fields or Output Fields panels to better see the information.

Configuring the Hierarchy Processor transformation

You can configure the Hierarchy Processor transformation to produce relational output.

1. Select the **Relational** output format.

2. Configure the transformation output by adding incoming fields to the output or by manually adding fields to the output.

3. Optionally, generate keys for the input group to define relationships between output groups.

4. Configure the output groups and fields:

   a. Configure data sources.

b. Optionally, customize the output field expressions.

c. Optionally, add filters to the transformation.

Generating unique keys

In a mapping that converts hierarchical data to relational output, you can optionally generate unique keys for the hierarchical input group.

The following image shows the option to generate unique keys in the **Hierarchy Processor** tab:
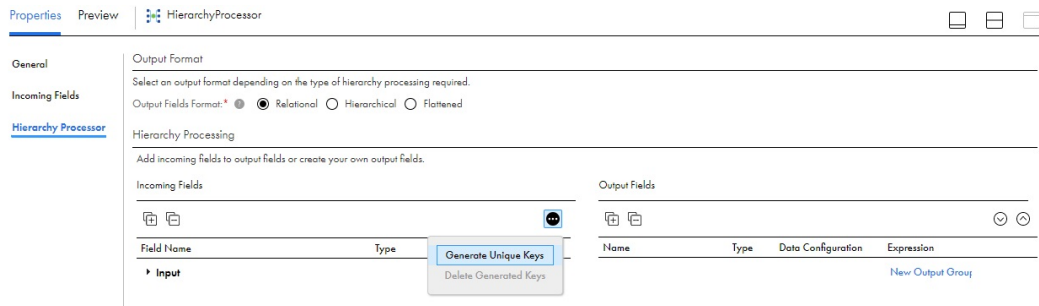
When you generate unique keys, you generate a primary key for the input group and a key for every array element within the input group. Each key is a combination of a global unique ID and a value that increases for each additional field. You can map the generated keys to the output groups just like any other incoming field.

When you map key fields from an input parent element to the output group of the child data set, the output data or output group has a primary key and foreign key relationship. This relationship is generated on the output side, based on how you mapped the generated keys.

Adding incoming fields to relational output groups

You can add incoming fields to the output individually or you can add an entire input group. You can add incoming fields to an output group, array, or struct field. You can rename or delete output fields as needed.

To add an output field, hover over the incoming field or input group that you want to add and then click the **Add** button.

You can add fields in the following ways based on the incoming field's data type and the output type:

Add incoming field

Adds the selected incoming field to the selected output group or field.

Select "Add incoming field" to add incoming fields with primitive data types to the output.

Add input group

Adds all incoming fields in the group to the selected output group or field.

Select "Add input group" to add incoming fields with primitive data types to the output.

Add single occurring children

Adds all single occurring children under the field to the output, including single occurring child fields that are nested under a struct. Does not add single occurring children that are nested under arrays.

You can select the "Add single occurring children" option only when you've select an incoming struct or array field and the output is relational. For more information, see "Add single occurring children" on page 202.

Add all descendants

Adds all children under the field to the output, including all arrays and structs. If the incoming field contains arrays, Data Integration creates a separate output group for each array.

You can select the "Add all descendants" option only when you've select an incoming struct or array field and the output is relational. For more information, see "Add all descendants" on page 203.

Add single occurring children

You can add single occurring children when adding incoming struct and array fields to the output and the output format is relational. The "Add single occurring children" option adds all single occurring children under the field to the output group, including the single occurring child fields that are nested under a struct.

When you select "Add single occurring children", the incoming field must have child objects.

If the field you select contains an array, the array and its children are not added because an array can contain multiple elements.

Example of adding single occurring children

You want to extract customer information from the Customer array, with each output record containing information about one customer.

Select **Add** on the Customer array, with the **Add Single Occurring Children** option selected. This action adds all single occurring children under Customer, including the children under the FullAddress struct to the output group.

Note that you cannot add single occurring children at the Customers struct level, because there are no single occurring children under that top-level struct.

The following image shows the incoming and output fields:

Add all descendants

You can add all descendants when adding incoming struct and array fields to the output and the output format is relational. The "Add all descendants" option adds all children under the field to the output group, including all arrays and structs.

You can't map hierarchical data with arrays to the same output group when you select the "Add all descendants" option. If you add an incoming field that contains hierarchical data with arrays to the output group, the Hierarchy Processor transformation creates a separate output group for each array.

Example of adding all descendants

You want to add all incoming fields to relational output groups. The incoming fields contain the arrays Customer and Order.

Click **Add** next to the parent incoming field and select the **Add All Descendants** option. This action maps the fields in the Customer array to the first output group, and maps all fields in the Order array to the second output group.

The following image shows the input and output fields:

Configuring relational output groups and fields

You can create and modify output groups and output fields. When the output format is relational, you can create multiple output groups.

After you add incoming fields to an output group, you can click the output field name on the **Hierarchy Processor** tab to modify the fields. You can also add and define output fields manually.

The following table describes the properties for output fields:

Property	Description
Child Of	The parent field or group that this field belongs to. The name structure describes the group, parent fields, and struct name. For example: `OUTGROUP.Grandparent.Parent.<struct name>`
Name	The name of the current output group or field.
Type	The data type of the current field. You can choose a primitive data type.
Precision	The total number of significant digits in the field.
Scale	The number of digits to the right of the decimal point.
Array Element Type	The data type of the current array element.
Array Element Precision	The precision for the current array element. Used when creating the target data.
Array Element Scale	The scale for the current array element. Used when creating the target data.
Struct Name	The struct name for the current struct field.
Element Struct Name	The element struct name for the current array of structs field.
Description	Optional description of the field and its usage.

Note: The output field properties that appear depend on the data type.

Configuring data sources

In a Hierarchy Processor transformation, a data source identifies the input group or incoming array that populates the primitive child fields of the output group or field.

When the output is relational, the data source for the output groups is always the input group or an incoming field. For example, you add an array to the output group. If you add single occurring children, the data source for the output group is the input group. If you add all descendants, the data source for each output group is an incoming field array.

Configuring filter conditions

You can define filter conditions to project a subset of the input data in the Hierarchy Processor transformation. You can filter based on incoming fields or output fields.

You can configure a filter condition to read data from primitive fields into an output array or struct field when the data in the array or struct field must correspond to the data in a sibling field in the output group.

Filter configuration example

You want to convert relational data to a JSON file. The incoming data is in a relational table that contains orders information. The orders table contains multiple rows for each order because each order can contain several products.

The incoming data looks like the following data:

```
OrderNumber,ProductName,ProductType,NumberOfItems,PricePerItem
12345,M&Ms Candies Chocolate Peanut Party Size - 38 Oz,Candy,2,14.49
12345,Stella Parm Shredded Cup - 20 Oz,Dairy,1,10.99
12345,AHA Sparkling Water Blueberry Pomegranate - 8-12 Fl. Oz.,Beverages,1,3.33
23456,Weetabix Biscuit Cereal Whole Grain 2 Count - 14 Oz,Breakfast & Cereal,2,4.99
23456,Producers Milk Lowfat 1% - Half Gallon,Dairy,1,2.79
23456,Egglands Best Eggs Cage Free Large Brown - 12 Count,Eggs,1,4.99
```

You want to read the product details into an array, where the product details are associated with a particular order number.

The following image shows the structure of the incoming and output fields:

In the Output Fields panel, set the data source for the Output group to `Input`, and configure the group by field as `Input.OrderNumber` to remove duplicate records from the output. Set the data source for the ProductDetails array to `Input`.

To ensure that the details in the ProductDetails array correspond to the order number in the output, configure the following filter condition for the array:

```
:fld.{Input.OrderNumber}= :fld.{Output.OrderNumber}
```

To further refine the records, use an AND condition in the filter. For example, to exclude records in which the product type is "Candy," configure the following filter condition:

```
:fld.{Input.OrderNumber}= :fld.{Output.OrderNumber} AND :fld.{Input.ProductType} != 'Candy'
```

The output contains one record for each order, and incoming records with the product type "Candy" are excluded.

The output data contains the following records:

```
{
    "OrderNumber":"12345",
    "ProductDetails":[
        {
            "ProductName":"AHA Sparkling Water Blueberry Pomegranate - 8-12 Fl. Oz.",
            "ProductType":"Beverages",
            "NumberOfItems":"1",
            "PricePerItem":"3.33"
        },
        {
            "ProductName":"Stella Parm Shredded Cup - 20 Oz",
            "ProductType":"Dairy",
            "NumberOfItems":"1",
            "PricePerItem":"10.99"
        }
    ]
```

```
        }
    {
        "OrderNumber":"23456",
        "ProductDetails":[
            {
                "ProductName":"Egglands Best Eggs Cage Free Large Brown - 12 Count",
                "ProductType":"Eggs",
                "NumberOfItems":"1",
                "PricePerItem":"4.99"
            },
            {
                "ProductName":"Producers Milk Lowfat 1% - Half Gallon",
                "ProductType":"Dairy",
                "NumberOfItems":"1",
                "PricePerItem":"2.79"
            },
            {
                "ProductName":"Weetabix Biscuit Cereal Whole Grain 2 Count - 14 Oz",
                "ProductType":"Breakfast & Cereal",
                "NumberOfItems":"2",
                "PricePerItem":"4.99"
            }
        ]
    }
```

Expression configuration

You can define expressions in the Hierarchy Processor transformation to create customized relational or hierarchical output, but not flattened output. You also use expressions to define filter conditions.

The Hierarchy Processor transformation can process information from different data sets. Some of the field names might not be unique among the different data sets. As a result, you can't simply reference the field by its name, because the same field name might be used in a different data set, or within the hierarchy of the same data set.

The syntax of the expressions in the Hierarchy Processor transformation differs from that used in the Expression transformation.

To reference a field in a Hierarchy Processor transformation, use the following syntax:

```
:fld.{input_group_name.field_name}.field_name
```

The following table describes the syntax in more detail:

Syntax part	Description
.fld.	Denotes the Hierarchy Processor transformation expression syntax.
input_group_name	Name of the input group or dataset.
field_name	Name of the field, including the full path name if it's not a top-level field. If any field is of the type array, include the array name. If an array is primitive and has no array name, use `elem` as the array name. For fields within a struct or an array, the actual field name is specified outside of the closing brace.
.field_name	Include the field_name portion only when referencing a field within a struct or an array. Follow these guidelines: - For fields within a struct, the field_name portion uses the format: `.structName.fieldname` - For fields within an array, the field_name portion uses the format: `.fieldName`

Configuring expressions

You can configure expressions when the output format is relational or hierarchical. For flattened output, the expression is fixed to the default format once you add the field and configuration is not possible.

To configure an expression in the Hierarchy Processor transformation, follow these steps:

1. Perform one of the following actions to create the output fields:

   a. To use an incoming field as an output, click the **Add** link that appears when you hover over the incoming field.

   b. To create a new field, click **New Field** in the Output Fields panel near the output group name or near an array name.

2. Click the expression in the Output Fields section.

   For newly created fields, the link appears as **Configure**. The **Field Expression** dialog box appears.

3. Enter the expression in the expression editor. You can add incoming fields, functions, and variables to the expression by clicking the **Add** link next to the object that you want to use. You can also type in the expression manually.

   The expression can contain constants, variables, built-in functions, and user-defined functions. You can nest functions to create a complex expression.

   Note: If you modify the references to any of the fields, your expression will fail.

4. Click **Validate**.

5. Correct any errors.

6. Click **OK**.

   You can close the editor and troubleshoot an invalid expression at another time.

Running a mapping with JSON data

To run a mapping that contains a Hierarchy Processor transformation with JSON-formatted data, you need to use a mapping task.

Reading JSON input

When you read JSON data, the input files can be based on a schema with multiple lines or on a schema with a single line.

The following sample shows a JSON schema on a single line:

```
{"Name":"Tom","Street":"2100 Seaport Blvd","City":"Redwood
City","State":"CA","Country":"USA","Zip":"94063"}
```

The following sample shows a JSON schema that spans across multiple lines:

```
{
    "Name": "Tom",
    "Surname": "Day",
    "City": "Redwood City",
    "State": "CA",
    "Country": "USA",
    "Zip": "94063"
}
```

By default, the Hierarchy Processor transformation reads each JSON schema as a single line. To read input that spans across multiple lines, you can configure the formatting options in the Source transformation to read multiple-line JSON files.

Writing JSON output

When you write JSON data, you can write each output record to a separate file, or you can write all output records to one file.

By default, each output record is written to a separate file. To write the output records to one JSON-formatted file, set the following Spark session property in the mapping task:

Session Property Name	Session Property Value
spark.sql.shuffle.partitions	1

Hierarchical to relational example

A customer order file contains the current customer contact information and the recent orders for those customers. The order file is in hierarchical JSON format and is generated by your company's cloud application. You want to process the hierarchical data and write the data to target files in relational and delimited formats.

Using the order file data, you want to create a relational customers table to use for an update on the customer information in the master database. Separately, you want to analyze the orders that have been increasing. You can use the order file to create a separate delimited orders file for the analysis.

You want to transform the data from the hierarchical input to relational and delimited output.

To create and run the mapping, perform the following tasks:

1. Ensure that you have access to an Amazon S3 V2 Connector for the S3 source and target objects.

2. Add a Source transformation that reads hierarchical data from the source JSON file.

3. Configure the following properties for the source object:

Property	Value
Connection	Amazon S3 V2
Source Format	JSON

4. Add a Hierarchy Processor transformation. The following image shows the data flow:
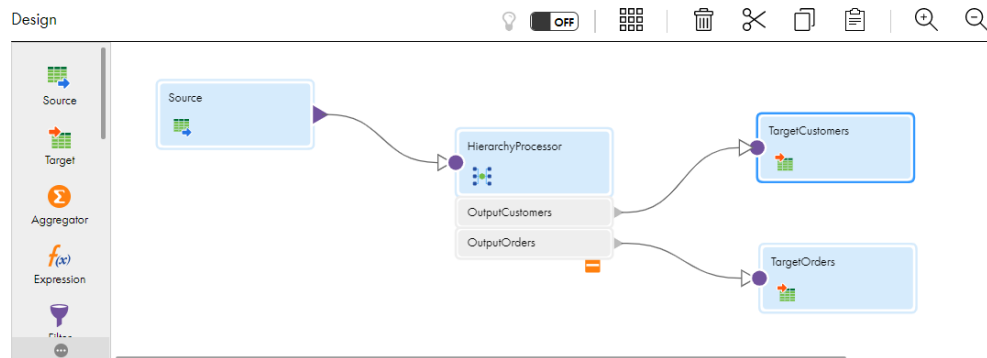
5. Create the OutputCustomers output group to create the relational customers data file.

The following image shows how to add incoming fields, which will create the output group:
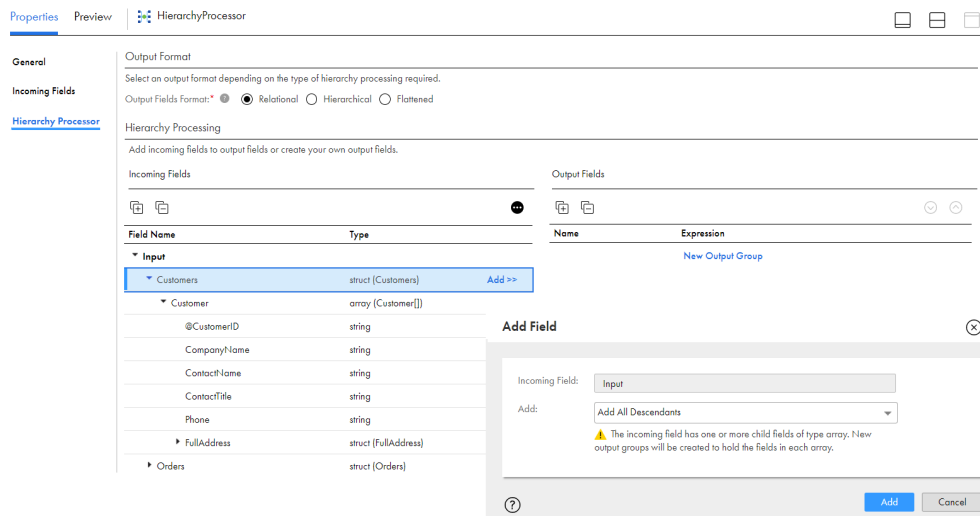
6. Create the `OutputOrders` output group to create the delimited orders data file.

7. In the **Hierarchy Processor** tab, map **Incoming Fields** to **Output Fields**.
   You can add fields individually or use the following options for struct and array fields:

   • Add All Descendants. Adds all children under the field, including all arrays and structs. If the incoming field contains arrays, Data Integration creates a separate output group for each array.

   • Add Single Occurring Children. Adds all single occurring children under the field, even if the single occurring child is nested under a struct.

8. Add a Target transformation to write the customers data output.

9. Configure the following properties for the target object:

Property	Value
Connection	Amazon S3 V2
Formatting Option	Relational

10. Add a Target transformation to write the orders data output.

11. Configure the following properties for the target object:

Property	Value
Connection	Amazon S3 V2
Formatting Option	Delimited

12. Link the `OutputCustomers` output group to the `TargetCustomers` Target transformation.

13. Link the `OutputOrders` output group to the `TargetOrders` Target transformation.

14. Run the mapping.

Processing hierarchical output

The Hierarchy Processor transformation can process the following types of hierarchical output:

- Relational to hierarchical. Convert up to five relational input groups to one hierarchical output group.

- Hierarchical to hierarchical. Convert one or more hierarchical input groups to one hierarchical output group with a different schema.

Defining hierarchical output with the Hierarchy Processor transformation

You can map the incoming fields to output fields to configure the output data structure for hierarchical output.

When the output data format is hierarchical, consider the following guidelines:

- There is a specific order that you should follow when performing operations.

- You can rename or delete output groups.

- You can aggregate both the input and output data.

The following image shows the **Hierarchy Processor** tab with relational input and hierarchical output:
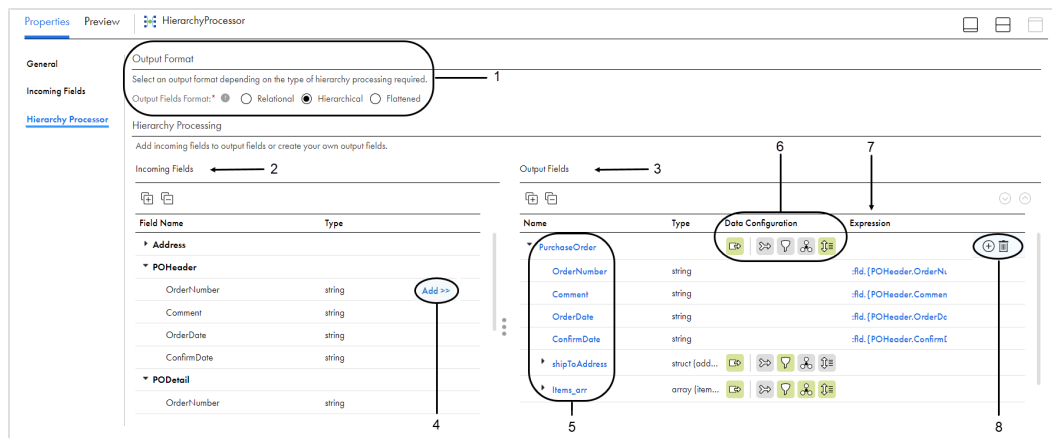
1. Output format. Select **Hierarchical** to construct a hierarchical schema for incoming fields.
2. Input groups, incoming fields. Use these fields to map to the output fields.
3. Output fields. Use these fields to create the complex output file.
4. Add incoming field to output group. Use to add fields to the output group.
5. Output field names. Click on a field name to modify the field name or data type.
6. Data Configuration icons. Use to configure the output groups and fields.
7. Expression. Click on an expression to view or customize the output field expression.
8. Add or delete output field. Use to create or delete output fields.

Tip: You can resize the Incoming Fields or Output Fields panels to better see the information.

Define hierarchical output

To define a Hierarchy Processor transformation with hierarchical output, perform the following tasks:

1. Select the **Hierarchical** output format.

2. Configure the transformation output by adding incoming fields to the output or by manually adding fields to the output.

3. Configure the output groups and fields.

    a. Configure data sources.

    b. Optionally, join data sources.

    c. Optionally, customize the output field expressions.

    d. Optionally, add filters to the transformation.

    e. Optionally, define output order using order by fields.

    f. Optional, aggregate input data using group by fields.

    g. Optionally, aggregate output data.

Order of operations

You should perform operations in a certain order when working with hierarchical output.

When the output data format is hierarchical, perform the operations in the following order:

1. Join. Specify join conditions in the data sources.
2. Filter. Optionally, specify filter conditions to include only a subset of the source data.
3. Group By. Optionally, specify the incoming fields for aggregate expressions.
4. Order By. Optionally, specify the incoming fields to create sorted output.

Renaming and deleting output groups

When the output data format is hierarchical, you can rename or delete input groups.

When you modify the input group, the following configurations are also updated if they refer to the input group:

- Input data source names
- Join data source names
- Order by fields
- Group by fields

However, you must modify the following configurations manually:

- Expressions
- Join conditions
- Filter conditions

Note: When the output data format is hierarchical, you cannot change the input group name.

Adding incoming fields to hierarchical output groups

You can add individual incoming fields or all fields in an input group to the output group, You can add incoming fields to an output group or to an array or struct field. You can rename or delete output fields as needed.

Click the **Add** link on the **Hierarchy Processor** tab next to the incoming field or input group that you want to add.

You can add fields in the following ways based on the incoming field's data type and the output type:

Add incoming field

Adds the selected incoming field to the selected output group or field.

The "Add incoming field" option is available when you add incoming fields with primitive data types to the output.

Add input group

Adds all incoming fields in the group to the selected output group or field.

The "Add input group" option is available when you add incoming fields with primitive data types to the output.

Add primitive single occurring children

Adds all primitive single occurring children under the field to the output, including the primitive, single occurring child fields that are nested under a struct. Adding primitive single occurring children doesn't add children that are nested under arrays.

You can add primitive single occurring children when you select an incoming struct or array field, and the output is hierarchical. For more information, see "Add primitive single occurring children" on page 212.

Preserve incoming field

Preserves the hierarchical structure of the selected field in the output. For example, if you add an array of structs to the output group, then the output group contains an array of structs with the same structure.

The "Preserve incoming field" option is available when you select an incoming struct or array field, and the output is hierarchical. For more information, see "Preserve incoming fields" on page 213.

Flatten selected array

Flattens an array of primitives into a primitive field. Creates one output record for each element in the array.

The "Flatten selected array" option is available when you select an incoming struct or array field, and the output is hierarchical. For more information, see "Flatten selected array" on page 214.

Add selected array as struct

Flattens an array of structs into a struct field. Creates one output record for each element the array.

The "Add selected array as struct" option is available when you select an incoming struct or array field, and the output is hierarchical. For more information, see "Add selected array as struct" on page 215.

Add primitive single occurring children

You can add primitive single occurring children when you are adding incoming struct and array fields to the output. The "Add primitive single occurring children" option is available when you add incoming struct or array fields to hierarchical output. It adds all primitive single occurring children under the field to the output group, including the primitive, single occurring child fields that are nested under a struct.

You can add primitive single occurring children only for incoming fields with child objects.

If the field you select contains an array, the array and its children are not added because an array can contain multiple elements.

Example

You want to extract the make, model, company, and policy number from an array of vehicle records. Each output record should contain information about one vehicle.

Add the vehicle array to the output group and choose **Add Primitive Single Occurring Children**.

The following image shows the incoming and output fields:

Note that the date field is not added to the output group because it is under a child array of the selected field and is not single occurring.

Preserve incoming fields

When you add incoming struct and array fields to the output, you can preserve the incoming field. When you preserve the incoming field, it's copied to hierarchical output without changing the structure.

When you add a nested array and select "Preserve incoming fields", the data source configuration for the output group determines how the records are created.

Example of preserving incoming fields

You want to extract the description information from a nested array of maintenance records. The description information is in an array of strings. You want the output data to also be in an array of strings.

Add the description array to the output group and choose **Preserve Incoming Field**.

The following image shows the incoming and output fields:

When Data Integration creates the output array, it sets the data source for the description array to `Input.vehicle.vehicle.maintenance.maintenance.desc.elem`, indicating that the information for the output array comes from the elements in the desc array in the Input group. The data source for the Output group determines the structure of the output records.

By default, Data Integration sets the data source for the Output group to `Input`. When you run the mapping, Data Integration collates all descriptions into one output record. To write the descriptions to separate records for each vehicle, set the data source to `Input.vehicle.vehicle`. To write the descriptions to separate records for each maintenance record, set the data source to `Input.vehicle.vehicle.maintenance.maintenance`. For more information about data source configuration, see .

Flatten selected array

When you add an incoming array of primitives to the output, you can flatten the selected array into a field of the same data type.

When you flatten a selected array, it creates one record for each element in the array.

Example of flattening an array

You want to extract the description information from a nested array of maintenance records. The description information is in an array of strings. You want to flatten the output into a string field.

Add the description array to the output group and choose **Flatten Selected Array**.

The following image shows the incoming and output fields:

The output contains one record for each occurrence of description in the incoming data.

For example, the incoming data contains the following record:

```
[
    {
        "vehicle": [
            {
                "make": "Toyota",
                "model": "Corolla",
                "insurance": {
                    "company": "Allstate",
                    "policy_num": "AS12876"
                },
                "maintenance": [
                    {
```

```
            "date": "01/01/2020",
            "description": ["oil filter1", "oil filter2"]
          },
          {
            "date": "01/08/2020",
            "description": ["tire rotation1", "tire rotation2"]
          }
        ]
      },
      {
        "make": "Toyota",
        "model": "RAV4",
        "insurance": {
          "company": "Allstate",
          "policy_num": "AS2033"
        },
        "maintenance": [
          {
            "date": "01/02/2020",
            "description": ["air filter replacement1", "air filter replacement2"]
          },
          {
            "date": "01/08/2020",
            "description": ["battery replacement1", "battery replacement2"]
          }
        ]
      }
    ]
  }
]
```

Data Integration creates the following output records, one for each occurrence of description in the incoming data:

```
{"description":"oil filter1"}
{"description":"oil filter2"}
{"description":"tire rotation1"}
{"description":"tire rotation2"}
{"description":"air filter replacement1"}
{"description":"air filter replacement2"}
{"description":"battery replacement1"}
{"description":"battery replacement2"}
```

Add selected array as struct

When you add an incoming array of primitives as a struct, the array is flattened into a struct field, without flattening any descendant arrays.

The "Add selected array as struct" option creates one record for each element in the selected array.

Example of adding an array as struct

You want to flatten an array of vehicle records into a struct without flattening the child array of maintenance records.

Add the vehicle array to the output group and choose **Add Selected Array as Struct**.

The following image shows the incoming and output fields:

The output contains one record for each element in the vehicle array.

For example, the incoming data contains the following record which contains data about two vehicles:

```
[
  {
    "vehicle": [
      {
        "make": "Toyota",
        "model": "Corolla",
        "insurance": {
          "company": "Allstate",
          "policy_num": "AS12876"
        },
        "maintenance": [
          {
            "date": "01/01/2020",
            "description": ["oil filter1", "oil filter2"]
          },
          {
            "date": "01/08/2020",
            "description": ["tire rotation1", "tire rotation2"]
          }
        ]
      },
      {
        "make": "Toyota",
        "model": "RAV4",
        "insurance": {
          "company": "Allstate",
          "policy_num": "AS2033"
        },
        "maintenance": [
          {
            "date": "01/02/2020",
            "description": ["air filter replacement1", "air filter replacement2"]
          },
          {
            "date": "01/08/2020",
            "description": ["battery replacement1", "battery replacement2"]
          }
        ]
      }
    ]
  }
]
```

```
        }
    ]
```
The Hierarchy Processor transformation creates the following output records, one for each vehicle:

```
{
    "vehicle":{
        "make":"Toyota",
        "model":"Corolla",
        "insurance":{
            "company":"Allstate",
            "policy_num":"AS12876"
        },
        "maintenance":[
            {
            "date":"01/08/2020",
            "description":["tire rotation2","tire rotation1"]
            },
            {
            "date":"01/01/2020",
            "description":["oil filter2","oil filter1"]
            }
        ]
    }
}
{
    "vehicle":{
        "make":"Toyota",
        "model":"RAV4",
        "insurance":{
            "company":"Allstate",
            "policy_num":"AS2033"
        },
        "maintenance":[
            {
                "date":"01/08/2020",
                "description":["battery replacement2","battery replacement1"]
            },
            {
                "date":"01/02/2020",
                "description":["air filter replacement2","air filter replacement1"]
            }
        ]
    }
}
```

Configuring hierarchical output group and fields

Uou can create and modify the output group and output fields for hierarchical output.

After you add incoming fields to an output group, you can select an output field name to modify the fields. You can also add and define output fields manually.

Name and Type

The following table describes the properties for hierarchical output fields:

Property	Description
Child Of	The parent field or group that this field belongs to. The name structure describes the group, parent fields, and struct name. For example: `OUTGROUP.Grandparent.Parent.struct_name`
Name	The name of the current output group or field.

Property	Description
Type	The data type of the current field. For hierarchical output, you can choose either a primitive or complex data type.
Precision	The total number of significant digits in the field.
Scale	The number of digits to the right of the decimal point.
Array Element Type	The data type of the current array element.
Array Element Precision	The precision for the current array element. Used when creating the target data.
Array Element Scale	The scale for the current array element. Used when creating the target data.
Struct Name	The struct name for the current struct field.
Element Struct Name	The element struct name for the current array of structs field.
Description	Optional description of the field and its usage.

Note: The output field properties that appear depend on the data type.

Aggregate Options

All output fields except for parent fields have aggregation options that you can set. The following table describes the aggregate options:

Property	Description
Use this field to aggregate values in an output field array.	Select this property to aggregate output data into the current field.
Output Field	If you are aggregating, specify the sibling array with fields to aggregate. The array must be a child of the current output field or group.

Aggregating values in an output field array

If the Hierarchy Processor transformation output contains an array, you might want to add an output field to aggregate values in the array. The output field must be a sibling of the array field.

For example, the Output group contains an array with orders information. The Orders array contains the OrderPrice field, which stores the price for each order. You want to find the total order price for each company.

The following image shows the output fields:

To find the total order price, add a field called TotalOrderPrice to the output group.

Edit the TotalOrderPrice field. Select **Use this field to aggregate values in an output field array**, and select the Orders array as the output field for which you want to aggregate values. Configure the following expression for the TotalOrderPrice field:

```
SUM(:fld.{Output.Orders.Orders.OrderPrice})
```

Configuring data sources

You can configure a data source for the output group and for all array and struct fields in the output. A data source identifies the input group or incoming array that populates the primitive child fields for the output group or field.

If you select a field as a data source, you have access to the following objects:

- All its primitive children
- All its ancestors' primitive children
- Individual elements of its array children
- Individual elements of its ancestors' array children

You can configure multiple data sources for an output group or field. If you do this, you must configure a join condition to join the data.

You can configure filters to exclude certain records. You can also specify group by fields for aggregating the data and order by fields for sorting records.

The data sources for hierarchical output group and fields can vary based on the output data structure.

Inheriting data sources from the parent

When you configure the data source for an array or struct field, you can populate the children of the field either with the incoming data or by inheriting the parent's data sources.

When you use the incoming data, the incoming data is used to populate the children of the array or struct.

When you inherit the parent's data sources, the data transformed into the parent output field populates the children of the array or struct. This preserves the data transformations (for example, joins and filters), in the

parent fields. You can apply filters to the field to further filter the data, but you can't configure data sources, joins, group by fields, or order by fields.

For example, you are reading data from a relational table of customer records in which the customer ID is unique. The incoming data contains the following records:

```
CustID,Name,Street,City,State,ZIP
00234,Ravindra Singh,123 6th St. Apt. 5A,Boston,MA,02134
14416,Melissa Clark,11 Winding Way,Watch Hill,RI,02891
```

You want to write the customer address fields to a struct.

The following image shows the incoming and output fields:
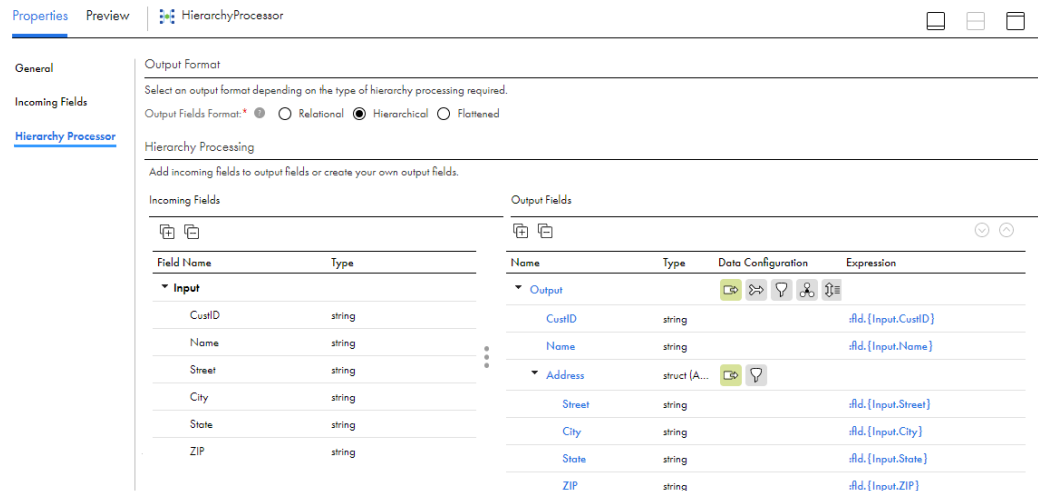
In the Output Fields panel, set the data source for the Output group to `Input` and the data source for the Address struct to `Inherit parent's data sources (Output)`. When you run the mapping, the Hierarchy Processor transformation creates one record for each occurrence of CustID in the input data and populates the struct with the address data that corresponds to the customer ID in the output:

```
{
    "CustID":"00234",
    "Name":"Ravindra Singh",
    "Address":{
        "Street":"123 6th St. Apt. 5A",
        "City":"Boston",
        "State":"MA",
        "ZIP":"02134"
    }
}
{
    "CustID":"14416",
    "Name":"Melissa Clark",
    "Address":{
        "Street":"11 Winding Way",
        "City":"Watch Hill",
        "State":"RI",
        "ZIP":"02891"
    }
}
```

If you set the data source for the Address struct to `Input`, then you must also configure the following filter condition on the struct to get the same output: `:fld.{Input.CustID} = :fld.{Output.CustID} AND :fld.{Input.Name} = :fld.{Output.Name}`. For more information about configuring filter conditions, see "Configure filter conditions" on page 224.

When the output field is an array that inherits its parent's data, the Hierarchy Processor transformation creates an array with one element.

Configure data source in hierarchical output example

When you add a nested array to the output and preserve the incoming field, the records that get created vary based on how you configure the data source for the output group.

For example, you want to extract the description information from a nested array of maintenance records in a JSON file. The description information is in an array of strings. You want the output data to also be in an array of strings.

The following sample data shows an incoming record:

```
[
  {
    "vehicle": [
      {
        "make": "Toyota",
        "model": "Corolla",
        "insurance": {
          "company": "Allstate",
          "policy_num": "AS12876"
        },
        "maintenance": [
          {
            "date": "01/01/2020",
            "description": ["oil filter1", "oil filter2"]
          },
          {
            "date": "01/08/2020",
            "description": ["tire rotation1", "tire rotation2"]
          }
        ]
      },
      {
        "make": "Toyota",
        "model": "RAV4",
        "insurance": {
          "company": "Allstate",
          "policy_num": "AS2033"
        },
        "maintenance": [
          {
            "date": "01/02/2020",
            "description": ["air filter replacement1", "air filter replacement2"]
          },
          {
            "date": "01/08/2020",
            "description": ["battery replacement1", "battery replacement2"]
          }
        ]
      }
    ]
  }
]
```

In the Hierarchy Processor transformation, you add the description array to the output group and choose **Preserve Incoming Field**.

The following image shows the incoming and output fields:
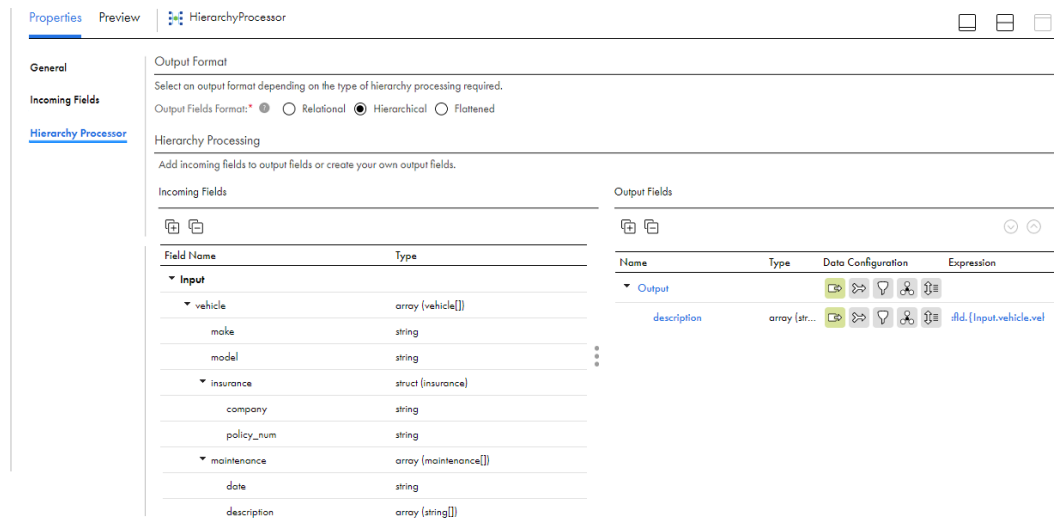
When the Hierarchy Processor transformation creates the output array, it sets the data source for the description array to `Input.vehicle.vehicle.maintenance.maintenance.desc.elem`, indicating that the information for the output array comes from the elements in the desc array in the Input group. By default, the Hierarchy Processor transformation sets the data source for the Output group to `Input`.

The data source configuration for the output group determines how the Hierarchy Processor transformation creates the output records.

To combine all descriptions into one output record, keep the data source for the output group as `Input`. This produces the following output:

```
{"description":["battery replacement2","battery replacement1","air filter
replacement2","air filter replacement1","tire rotation2","tire rotation1","oil
filter2","oil filter1"]}
```

To create one output record for each occurrence of vehicle in the incoming data, set the data source to `Input.vehicle.vehicle`. In this case, the output data contains one record for each vehicle:

```
{"description":["tire rotation2","tire rotation1","oil filter2","oil filter1"]}
{"description":["battery replacement2","battery replacement1","air filter
replacement2","air filter replacement1"]}
```

To create one output record for each occurrence of maintenance in the incoming data, set the data source to `Input.vehicle.vehicle.maintenance.maintenance`. In this case, the output contains one record for each maintenance record:

```
{"description":["oil filter2","oil filter1"]}
{"description":["tire rotation2","tire rotation1"]}
{"description":["air filter replacement2","air filter replacement1"]}
{"description":["battery replacement2","battery replacement1"]}
```

Data source conflicts

If you convert hierarchical data to hierarchical data and you add multi-level arrays to the output, you must ensure that the data sources for the output group or fields do not conflict. A conflict occurs when you select both an incoming array and one of its descendant arrays to be data sources for the same output group or field.

If you have a data source conflict, the transformation remains invalid until you resolve the conflict. Additionally, you cannot configure joins, filters, order by fields, or group by fields until you resolve the conflict.

Example of a data source conflict

A data source conflict occurs if you select both Array1 and Array2 as data sources for the output group in the following image:

The conflict occurs because there is no way to determine which data source provides the data for Field1 and Field2. In this case, the Hierarchy Processor transformation displays a conflicting data sources error.

To resolve the conflict, remove one of the data sources from the Output group.

Modify data sources for output group or fields

The output group and all array and struct fields in the output require a data source. When you add an input group or incoming field to the output, the Hierarchy Processor transformation usually sets a data source for you. You can manually add or edit a data source.

To configure a data source for the output group:

1. Click the **Data Sources** icon for the output group.
2. Examine if there is already a data source configured for the output group. If it requires any changes, continue with these steps, otherwise you can exit the dialog box.
3. Click the **Add Data Source** icon to add a new data source.
4. Click the ellipsis icon to select the input group or array to use as a data source.
5. Validate the configuration.
6. Click **Save**.

Tip: Click the trash icon to remove any incorrect or unneeded data source.

To configure a data source for struct or array fields:

1. Click the **Data Sources** icon for the output group.
2. Examine if there is already a data source configured for the struct or array. If it requires any changes, continue with these steps, otherwise you can exit the dialog box.
3. Select the type of data source to use:
    *   Inherit parent's data sources: Use the same data sources as the parent for the struct or array.
    *   Use input group or incoming fields: Select an input group or incoming field to use as a data source.
4. Validate the configuration.
5. Click **Save**.

Tip: Click the trash icon to remove any incorrect or unneeded data source.

Configuring output data

For hierarchical output, you can configure data source join and filter conditions as well as group by and order by fields. You can aggregate on both the input and output data.

Configure join conditions

When the output data format is hierarchical, you can define join conditions for the data sources. You must configure a join condition if an output group or field has multiple data sources. Configure a join condition to join the data from the input groups or incoming fields.

Configure the join conditions for the output groups on the **Hierarchy Processor** tab.

1. Click the **Join Conditions** icon for the output group.
2. Add a join condition.
3. Select the left data source.
4. Select the join type:
   - Inner. Includes rows with matching join conditions. Discards rows that do not match the join conditions.
   - Left Outer. Includes all rows from the right pipeline and the matching rows from the left pipeline. Discards the unmatched rows from the left pipeline.
   - Right Outer. Includes all rows from the left pipeline and the matching rows from the right pipeline. Discards the unmatched rows from the right pipeline.
   - Full Outer. Includes rows with matching join conditions and all incoming data from the left pipeline and right pipeline.

   Note: If you select an outer join on a large data set, you might need to increase the Spark driver memory in the mapping task. For more information about Spark session properties, see *Tasks*.
5. Select the right data source.
6. Click **Configure Join Condition**.
7. Select fields and built-in functions to create the expression.
8. Validate the expression.
9. Click **Save**.

Configure filter conditions

You can define filter conditions to project a subset of the input data in the Hierarchy Processor transformation. You can filter based on incoming fields or output fields.

You can configure a filter condition to read data from primitive fields into an output array or struct field when the data in the array or struct field must correspond to the data in a sibling field in the output group.

Filter configuration example

You want to convert relational data to a JSON file. The incoming data is in a relational table that contains orders information. The orders table contains multiple rows for each order because each order can contain several products.

The incoming data looks like the following data:

```
OrderNumber,ProductName,ProductType,NumberOfItems,PricePerItem
12345,M&Ms Candies Chocolate Peanut Party Size - 38 Oz,Candy,2,14.49
12345,Stella Parm Shredded Cup - 20 Oz,Dairy,1,10.99
12345,AHA Sparkling Water Blueberry Pomegranate - 8-12 Fl. Oz.,Beverages,1,3.33
```
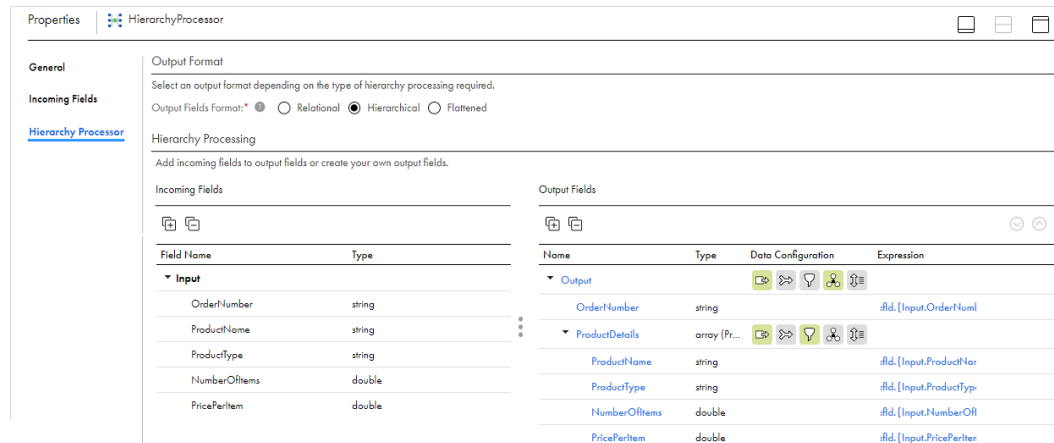
```
23456,Weetabix Biscuit Cereal Whole Grain 2 Count - 14 Oz,Breakfast & Cereal,2,4.99
23456,Producers Milk Lowfat 1% - Half Gallon,Dairy,1,2.79
23456,Egglands Best Eggs Cage Free Large Brown - 12 Count,Eggs,1,4.99
```

You want to read the product details into an array, where the product details are associated with a particular order number.

The following image shows the structure of the incoming and output fields:

In the Output Fields panel, set the data source for the Output group to `Input`, and configure the group by field as `Input.OrderNumber` to remove duplicate records from the output. Set the data source for the ProductDetails array to `Input`.

To ensure that the details in the ProductDetails array correspond to the order number in the output, configure the following filter condition for the array:

`:fld.{Input.OrderNumber}= :fld.{Output.OrderNumber}`

To further refine the records, use an AND condition in the filter. For example, to exclude records in which the product type is "Candy," configure the following filter condition:

`:fld.{Input.OrderNumber}= :fld.{Output.OrderNumber} AND :fld.{Input.ProductType} != 'Candy'`

The output contains one record for each order, and incoming records with the product type "Candy" are excluded.

The output data contains the following records:

```
{
    "OrderNumber":"12345",
    "ProductDetails":[
        {
            "ProductName":"AHA Sparkling Water Blueberry Pomegranate - 8-12 Fl. Oz.",
            "ProductType":"Beverages",
            "NumberOfItems":"1",
            "PricePerItem":"3.33"
        },
        {
            "ProductName":"Stella Parm Shredded Cup - 20 Oz",
            "ProductType":"Dairy",
            "NumberOfItems":"1",
            "PricePerItem":"10.99"
        }
    ]
}
{
    "OrderNumber":"23456",
    "ProductDetails":[
        {
```

```
                        "ProductName":"Egglands Best Eggs Cage Free Large Brown - 12 Count",
                        "ProductType":"Eggs",
                        "NumberOfItems":"1",
                        "PricePerItem":"4.99"
                    },
                    {
                        "ProductName":"Producers Milk Lowfat 1% - Half Gallon",
                        "ProductType":"Dairy",
                        "NumberOfItems":"1",
                        "PricePerItem":"2.79"
                    },
                    {
                        "ProductName":"Weetabix Biscuit Cereal Whole Grain 2 Count - 14 Oz",
                        "ProductType":"Breakfast & Cereal",
                        "NumberOfItems":"2",
                        "PricePerItem":"4.99"
                    }
                ]
        }
```

Configure group by fields

For hierarchical output, you can group primitive incoming fields for aggregation to produce one row for each group of input data.

To group fields for aggregation:

1. Click the **Group By fields** icon for the output group or array.
2. Examine if there is already a grouping present. If it requires any changes, continue with these steps, otherwise you can exit the dialog box.
3. Click the **Add New Group by Field** icon to add a new field for grouping.
4. Clip the ellipsis icon to select the primitive field to add to the grouping.
5. Validate the configuration.
6. Click **Save**.

Configure order by fields

When the output data format is hierarchical, you can define how the output is sorted.

Note: The following conditions must be true for the sort operation to take effect:

• The Hierarchy Processor transformation is connected directly to the Target transformation.
• All sort fields are connected to the Target transformation.
• The data types in the connected order by fields match the data types in the target fields.

Configure order by fields on the **Hierarchy Processor** tab.

1. Click the **Order By fields** icon for the output group or the array or struct field.
2. Add the incoming fields to order by and sort the data in ascending or descending order.
3. Rearrange the fields to adjust the sort order.
4. Validate the configuration.
5. Click **Save**.

Expression configuration

You can define expressions in the Hierarchy Processor transformation to create customized relational or hierarchical output, but not for flattened output. You also use expressions to define filter conditions.

The Hierarchy Processor transformation can process information from different data sets. Some of the field names might not be unique among the different data sets. As a result, you can't simply reference the field by its name, because the same field name might be used in a different data set, or within the hierarchy of the same data set.

The syntax of the expressions in the Hierarchy Processor transformation differs from that used in the Expression transformation.

To reference a field in a Hierarchy Processor transformation, use the following syntax:

```
:fld.{input_group_name.field_name}.field_name
```

The following table describes the syntax in more detail:

Syntax part	Description
.fld.	Denotes the Hierarchy Processor transformation expression syntax.
input_group_name	Name of the input group or dataset.
field_name	Name of the field, including the full path name if it's not a top-level field. If any field is of the type array, include the array name. If an array is primitive and has no array name, use `elem` as the array name. For fields within a struct or an array, the actual field name is specified outside of the closing brace.
.field_name	Include the field_name portion only when referencing a field within a struct or an array. Follow these guidelines: - For fields within a struct, the field_name portion uses the format: `.structName.fieldname` - For fields within an array, the field_name portion uses the format: `.fieldName`

Configure expressions

You can configure expressions when the output format is relational or hierarchical. For flattened output, can't configure the default expression once you add the field.

To configure an expression in the Hierarchy Processor transformation, follow these steps:

1. Perform one of the following actions to create the output fields:

    a. To use an incoming field as an output, click the **Add** link that appears when you hover over the incoming field.

    b. To create a new field, click **New Field** in the Output Fields panel near the output group name or near an array name.

2. Click the expression in the Output Fields section.

    For newly created fields, the link appears as **Configure**. The **Field Expression** dialog box appears.

3. Enter the expression in the expression editor. You can add incoming fields, functions, and variables to the expression by clicking the **Add** link next to the object that you want to use. You can also type in the expression manually.

The expression can contain constants, variables, built-in functions, and user-defined functions. You can nest functions to create a complex expression.

Note: If you modify the references to any of the fields, your expression will fail.

4. Click **Validate**.
5. Correct any errors.
6. Click **OK**.

   You can close the editor and troubleshoot an invalid expression at another time.

Running a mapping with JSON data

To run a mapping that contains a Hierarchy Processor transformation with JSON-formatted data, you need to use a mapping task.

Reading JSON input

When you read JSON data, the input files can be based on a schema with multiple lines or on a schema with a single line.

The following sample shows a JSON schema on a single line:

```
{"Name":"Tom","Street":"2100 Seaport Blvd","City":"Redwood
City","State":"CA","Country":"USA","Zip":"94063"}
```

The following sample shows a JSON schema that spans across multiple lines:

```
{
    "Name": "Tom",
    "Surname": "Day",
    "City": "Redwood City",
    "State": "CA",
    "Country": "USA",
    "Zip": "94063"
}
```

By default, the Hierarchy Processor transformation reads each JSON schema as a single line. To read input that spans across multiple lines, you can configure the formatting options in the Source transformation to read multiple-line JSON files.

Writing JSON output

When you write JSON data, you can write each output record to a separate file, or you can write all output records to one file.

By default, each output record is written to a separate file. To write the output records to one JSON-formatted file, set the following Spark session property in the mapping task:

Session Property Name	Session Property Value
spark.sql.shuffle.partitions	1

Relational to hierarchical example

You need to create a purchase order file in hierarchical format using customer sales data from two purchase order tables and the customer address table.

Use the Hierarchy Processor transformation to create purchase orders in hierarchical format.

The POHeader table contains basic information about the orders placed by customers:

OrderNumber	Comment	OrderDate	ConfirmDate
1	AppD for POD4	2020-10-01 00:00:00.0	2020-10-02 00:00:00.0
2	GoJS for IICS	2020-10-12 00:00:00.0	2020-10-12 00:00:00.0

The Address table contains customer address information for each order:

OrderNumber	AddressType	Name	Street	City	State	Country	Zip
1	ShipTo	Tom	2100 Seaport Blvd	Redwood City	CA	USA	94063
1	BillTo	Tom	2100 Seaport Blvd	Redwood City	CA	USA	94063
2	ShipTo	Bill	1630 S Delaware St	San Mateo	CA	USA	94402
2	BillTo	Bill	PO Box 313	San Mateo	CA	USA	94402

The PODetail table contains details about the customer purchase orders:

OrderNumber	ItemNum	ProductName	Quantity	Price	Comment	ShipDate	PartNum
1	1	AppD Agent for JVM	60	500	JVM agents	2020-10-15 00:00:00.0	1
1	3	ELB agents	10	200	ELB agents	2020-10-15 00:00:00.0	3
1	2	MySQL agents	2	120	MySQL agents	2020-10-16 00:00:00.0	2
1	4	MySQL agents	2	120	MySQL agents	2020-10-01 00:00:00.0	2
1	5	MySQL agents	2	120	MySQL agents	2020-10-01 00:00:00.0	2
2	1	GOJS OEM Edition	2	20000	GOJS Dev	2020-10-19 00:00:00.0	101
2	2	GOJS Professional Service	5	5000	GOJS Dev	2020-10-19 00:00:00.0	102

Perform the following steps to create purchase orders in hierarchical format:

1. Step 1. Design the mapping.
2. Step 2. Build the output group and create a struct.
3. Step 3. Create an array of structs.
4. Step 4. Aggregate the output data.
5. Step 5. Create an array of structs and join data sources.
6. Step 6. Run the mapping.

Step 1. Design the mapping

The first step is to configure the Hierarchy Processor in the Mapping Designer with the Source and Target transformations.

Perform the following steps in the Mapping Designer:

1. Add a Hierarchy Processor transformation and change the output data format to Hierarchical.
2. Add the POHeader, PODetail, and Address tables as source objects.
3. Connect the source objects to the Hierarchy Processor transformation in the data flow.
4. In the Hierarchy Processor transformation, add the PurchaseOrder output group and connect the target object in the data flow.

Your mapping should look like the following image:
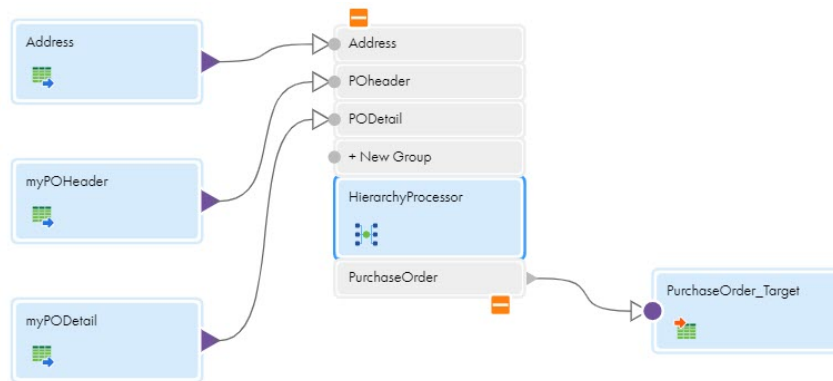
Step 2. Build the output group and create a struct

Once you configure the basic mapping, create the output group next.

Perform the following steps to create the output group with the basic purchase order data and add the ship-to address:

1. Add all the incoming fields from POHeader to the PurchaseOrder output group.
2. Add a new output field with the following properties:

Property	Value
Child Of	PurchaseOrder
Name	shipToAddress
Type	struct
Struct Name	address_struct

3. Add all the incoming fields from Address to the shipToAddress struct in the output group:

4. Delete the following fields that you do not need in the output group:

   - PurchaseOrder.shipToAddress.OrderNumber
   - PurchaseOrder.shipToAddress.AddressType

5. Add a filter condition for the PurchaseOrder.shipToAddress struct:

   ```
   :fld.{Address.OrderNumber}=:fld.{PurchaseOrder.OrderNumber} AND :fld.
   {Address.AddressType}='ShipTo'.
   ```

Step 3. Create an array of structs

Continue to configure the output by adding the purchase order details in the items array of structs. Configure the data processing strategies to sort by item number, group by part number, and aggregate the incoming quantity and price.

Perform the following steps

1. Add a new output field with the following properties:

Property	Value
Child Of	PurchaseOrder
Name	Items_arr
Type	array
Array Element Type	struct
Element Struct Name	item_str

2. Add all the incoming fields from PODetail to the Items_arr array in the output group.

3. Delete the following field that you do not need in the output group:
   PurchaseOrder.Items_arr.OrderNumber.

4. Add a filter condition for the PurchaseOrder.Items_arr array: `:fld.{PODetail.OrderNumber}=:fld.{PurchaseOrder.OrderNumber}`.

5. Configure a group by field for the PurchaseOrder.Items_arr array: PODetail.PartNum.

6. Configure an order by field in ascending order for the PurchaseOrder.Items_arr array: PODetail.ItemNum.

7. Update the field expression for PODetail.Quantity in the PurchaseOrder.Items_arr array: `SUM(:fld. {PODetail.Quantity})` to aggregate quantity.

8. Update the field expression for PODetail.Price in the PurchaseOrder.Items_arr array: `SUM(:fld. {PODetail.Price})` to aggregate price.

The following image shows the data configuration icons and expressions for the Items_arr array in the output group.
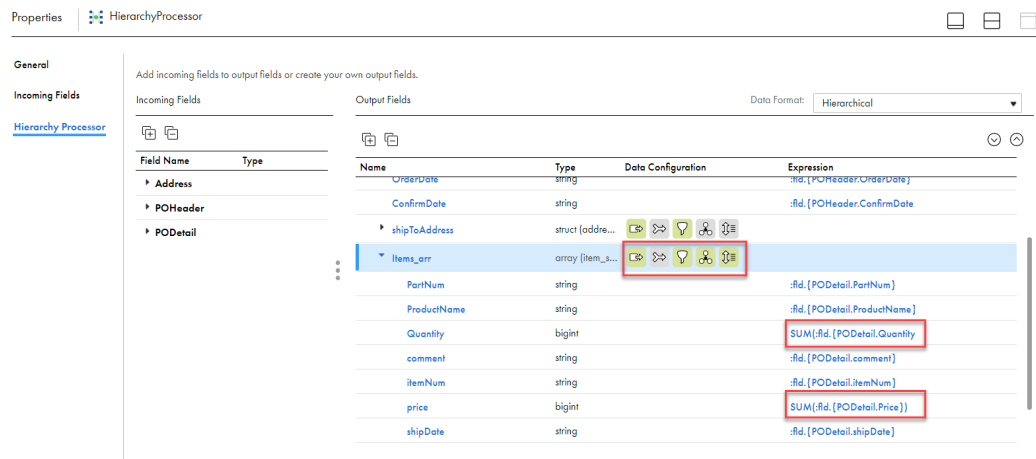
Step 4. Aggregate the output data

You aggregate the output data to calculate the total price.

Perform the following steps to aggregate all the items in a particular purchase order:

1. Add a new output field with the following properties:

Property	Value
Child Of	PurchaseOrder
Name	TotalPrice
Type	bigint
Aggregate Options: This field will aggregate values in an output field array	Enabled
Output Field	Items_arr

The following image shows the aggregate options for the TotalPrice output field:

2. Configure the following field expression for PurchaseOrder.TotalPrice to aggregate the total price:

```
SUM(:fld.{PurchaseOrder.Items_arr.item_str.Quantity}*:fld.
{PurchaseOrder.Items_arr.item_str.Price})
```

Step 5. Create an array of structs and join data sources

Add and configure the same-day items array of structs. Using a filter, a join, and a field expression, you output only the items that were ordered and shipped on the same date.

Perform the following steps:

1. Add a new output field with the following properties:

Property	Value
Child Of	PurchaseOrder
Name	SameDayItems
Type	array

Property	Value
Array Element Type	struct
Element Struct Name	sameday_str

2. Add all the incoming fields from PODetail to the SameDayItems array in the output group.

3. Delete the following field that you do not need in the output group:
   PurchaseOrder.SameDayItems.OrderNumber.

4. Add POHeader as a data source for PurchaseOrder.SameDayItems array.

5. Add a join condition for the PurchaseOrder.SameDayItems array with the following properties:

Property	Value
Left Group	POHeader
Join Type	Inner
Right Group	PODetail
Join Condition	:fld.{POHeader.OrderDate}=:fld.{PODetail.ShipDate} AND :fld.{POHeader.OrderNumber}=:fld.{PODetail.OrderNumber}

The following image shows the data sources and join condition for SameDayItems:
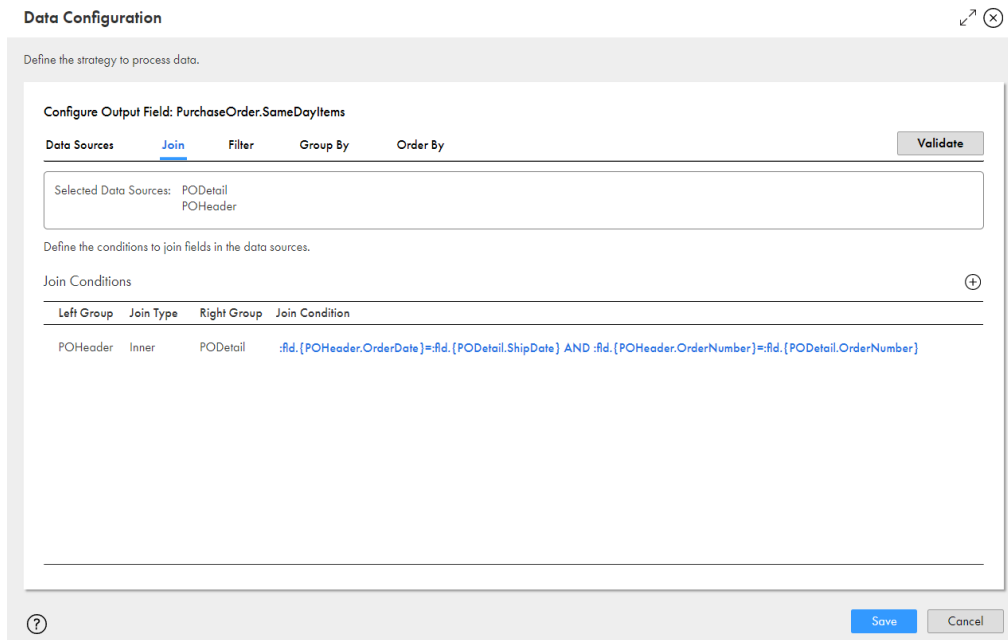
6. Add a filter condition for the PurchaseOrder.SameDayItems array:

```
:fld.{PODetail.OrderNumber}=:fld.{PurchaseOrder.OrderNumber}
```

.

Step 6. Run the mapping

The final step is to create and run the mapping task to produce the JSON output.

Perform the following steps:

1. Create a mapping task.

2. Run the mapping task.

3. Review your output.

Tip: For more information about mapping tasks, see the *Tasks* guide.

The following JSON shows the PurchaseOrder target output after you run the mapping:

```
{
    "OrderNumber": "1",
    "Comment": "AppD for POD4",
    "OrderDate": "2018-10-01 00:00:00.0",
    "ConfirmDate": "2018-10-02 00:00:00.0",
    "address_struct": {
        "Name": "Tom",
        "Street": "2100 Seaport blvd",
        "City": "Redwood City",
        "State": "CA",
        "Country": "USA",
        "Zip": "94063"
    },
    "Items_arr": [{
        "itemNum": "1",
        "ProductName": "AppD Agent for JVM",
        "Quantity": 60,
        "price": 500,
        "comment": "JVM agents",
        "shipDate": "2018-10-15 00:00:00.0",
        "PartNum": "1"
    }, {
        "itemNum": "2",
        "ProductName": "MySQL agents",
        "Quantity": 6,
        "price": 360,
        "comment": "MySQL agents",
        "shipDate": "2018-10-15 00:00:00.0",
        "PartNum": "2"
    }, {
        "itemNum": "3",
        "ProductName": "ELB agents",
        "Quantity": 10,
        "price": 200,
        "comment": "ELB agents",
        "shipDate": "2018-10-16 00:00:00.0",
        "PartNum": "3"
    }],
    "TotalPrice": 34160
} {
    "OrderNumber": "2",
    "Comment": "GoJS for IICS",
    "OrderDate": "2018-10-12 00:00:00.0",
    "ConfirmDate": "2018-10-12 00:00:00.0",
    "address_struct": {
        "Name": "Bill",
        "Street": "23rd Ave",
        "City": "San Mateo",
        "State": "CA",
        "Country": "USA",
        "Zip": "94401"
    },
    "Items_arr": [{
        "itemNum": "1",
        "ProductName": "GOJS OEM Edition",
        "Quantity": 2,
```

```
        "price": 20000,
        "comment": "GOJS Dev",
        "shipDate": "2018-10-19 00:00:00.0",
        "PartNum": "101"
    }, {
        "itemNum": "2",
        "ProductName": "GOJS Prefessional Service",
        "Quantity": 5,
        "price": 5000,
        "comment": "GOJS Dev",
        "shipDate": "2018-10-19 00:00:00.0",
        "PartNum": "102"
    }],
    "TotalPrice": 65000
}
```

Hierarchical to hierarchical example

You want to create a customer order file in hierarchical format, using an existing file of hierarchical data.

The existing customer order file CompanyOrders contains the names of companies that have placed orders and information about each order, including the price, date, shipping address, and ID numbers of ordered items.

The following image shows the structure of the CompanyOrders file:

You want to restructure the shipping address into a struct and add a field to calculate the total price of all orders for each company.

Perform the following steps to create and configure the target file:

1.   Step 1. Design the mapping.

2.   Step 2. Configure the output group.

3.   Step 3. Create an output field to aggregate the total price.

4.   Step 4. Create an output struct for the order address.

5. Step 5. Run the mapping.

Step 1. Design the mapping

The first step is to configure the Hierarchy Processor in the Mapping Designer with the Source and Target transformations.

Perform the following steps in the Mapping Designer:

1. Add the CompanyOrders file as a source object.
2. Add a Hierarchy Processor transformation to the mapping and connect CompanyOrders as an input source.
3. In the Hierarchy Processor transformation and change the output data format to Hierarchical. This creates an output group.
4. Add a Target transformation to the mapping, and connect the Hierarchy Processor transformation output to this target object.

The following image shows how the mapping should look like:
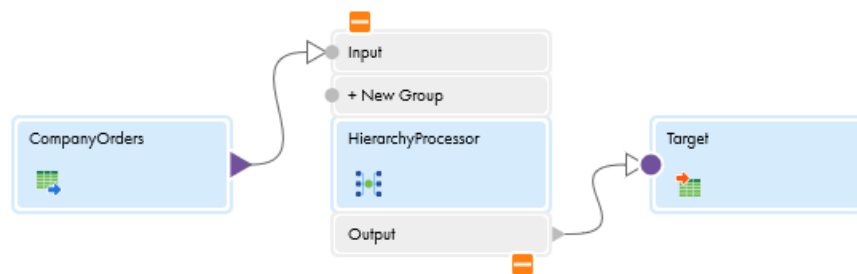
Step 2. Configure the output group

Once you configure the basic mapping, create the output group by adding all the incoming fields.

Perform the following steps in the Hierarchy Processor transformation:

1. Add all the incoming fields from the input to the output group. Set **Add** to *Preserve incoming field*. The following image shows the **Add Field** dialog:
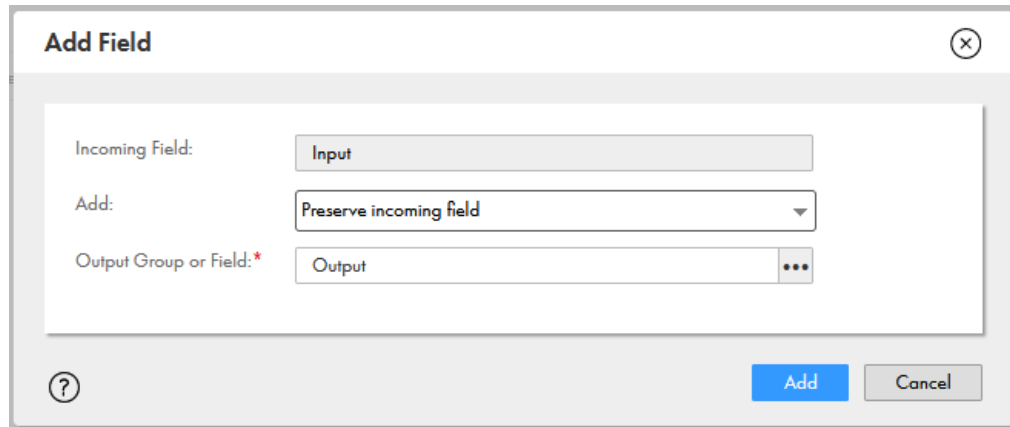
2. Verify that the data source for the output group is set to *Input*.

Step 3. Create an output field to aggregate the total price

Create an output field in the Hierarchy Processor transformation that calculates the total price of all orders for each company.

Perform the following steps:

1. Add a new output field with the following properties:

Property	Value
Child Of	Output
Name	TotalOrdersPrice
Type	double
Aggregate Options	Enabled
Output Field	Orders

The following image shows the aggregate options for the TotalOrdersPrice output field:

2. Configure the following field expression for Output.TotalOrdersPrice to aggregate the total price of all orders for a company:

    ```
    SUM(:fld.{Output.Orders.Orders.OrderPrice})
    ```

Step 4. Create an output struct for the order address

Create a structure for the order address in the output and remove fields from the output that you do not need.

Perform the following steps:

1. Add a new output field with the following properties:

Property	Value
Child Of	Output.Orders.Orders
Name	OrderAddress

Property	Value
Type	struct
Struct Name	Address

2. Add all the incoming fields from Orders to OrderAddress. Set **Add** to *Add primitive single occurring children*.

3. Set the data source for OrderAddress to *Use Output*.

4. Delete the following fields from the OrderAddress struct that you do not need:
   - Output.Orders.Orders.OrderAddress.OrderPrice
   - Output.Orders.Orders.OrderAddress.OrderDate

5. Delete the following fields from the Orders output group that you do not need:
   - Output.Orders.Orders.Street
   - Output.Orders.Orders.City
   - Output.Orders.Orders.State
   - Output.Orders.Orders.Country
   - Output.Orders.Orders.ZipCode

The following image shows the OrderAddress struct:

Step 5. Run the mapping

The final step is to create and run the mapping task to produce the JSON output.

Perform the following steps:

1. Create a mapping task.
2. Run the mapping task.
3. Review your output.

Tip: For more information about mapping tasks, see the *Tasks* guide.

Processing flattened output

The Hierarchy Processor transformation can convert a hierarchical input group to a flattened denormalized output group.

For example, you have a shop maintenance file that contains customer and vehicle information for all your customers. You want to denormalize the vehicle maintenance data and exclude personal information about the customers.

Defining flattened output with the Hierarchy Processor transformation

Use the **Hierarchy Processor** tab to map incoming fields to output fields. The flattened output option allows you to create denormalized output.

The following image shows the **Hierarchy Processor** tab with hierarchical input and flattened output:
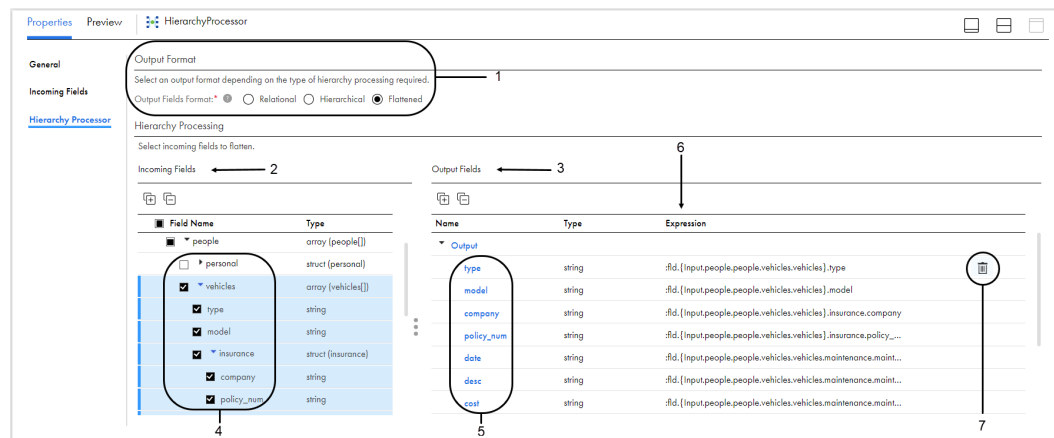
1. Output format. Select **Flattened** to convert hierarchical input into denormalized output.
2. Incoming fields. View the incoming data schema and field types.
3. Output fields. View the output fields as you create the output file.
4. Configure the output. Select incoming fields to build the flattened output file schema.
5. Output field names. Click on a field name to modify the field name.
6. Expression. View the field expressions to determine the input to output field mappings.
7. Delete. Use to delete output fields.

Tip: Use the maximize icon and resize the Incoming Fields panel or Output Fields panel to see the information you need.

To define a Hierarchy Processor transformation with flattened output, perform the following tasks:

1. Select the **Flattened** output format.
2. Configure the transformation output by selecting incoming fields to add to the output.
3. Optional. Rename output fields.
4. Optional. Delete output fields.

Adding incoming fields to flattened output groups

You can add incoming fields to the output individually or you can add an entire input group. After you add the incoming fields to the output group, you can rename or delete output fields as needed.

To add output fields, select the check box next to the input field or input group that you want to add. Selecting a parent automatically selects all child items, but you can deselect the entries that you do not want. You can also delete any unneeded entries from the output.

Flatten hierarchical data

When you add incoming hierarchical fields to the output, all fields in the output schema are automatically flattened.

Example of flattened hierarchical data

The incoming fields are contained within a hierarchy of structs and arrays. You just need four fields in the output: *OrderNumber, Name, ProductName,* and *Quantity*.

Select the appropriate check boxes to add the incoming fields to the output.

The following image shows the input and output:

Renaming flattened output group and fields

After you add incoming fields to the output group in the flattened format, you can change the names of these fields if necessary. You can also change the name of the output group.

To change the name of any output field, click on the field name. To change the name of the output group, click on the output group name.

The following table describes the properties for the output fields:

Property	Description
Name	The name of the output field.
Type	The data type of the current field. You cannot change the data type.
Precision	The total number of significant digits in the field. You cannot change the precision.
Scale	The number of digits to the right of the decimal point. You cannot change the scale.

Expression format

When you use the flattened output, the expression is fixed to the default and can't be changed. But if you understand the syntax of the default expression, you can easily identify the source location of each output field.

Tip: To add fields to the output group, use the check boxes next to the incoming field names. This differs from the Add link that you use for relational or hierarchical output.

The Hierarchy Processor transformation can process information from different data sets. Some of the field names might not be unique among the different data sets. As a result, you can't simply reference the field by its name, because the same field name might be used in a different data set, or within the hierarchy of the same data set.

The syntax of the expressions in the Hierarchy Processor transformation differs from that used in the Expression transformation.

To reference a field in a Hierarchy Processor transformation, use the following syntax:

```
:fld.{input_group_name.field_name}.field_name
```

The following table describes the syntax in more detail:

Syntax part	Description
.fld.	Denotes the Hierarchy Processor transformation expression syntax.
input_group_name	Name of the input group or dataset.
field_name	Name of the field, including the full path name if it's not a top-level field. If any field is of the type array, include the array name. If an array is primitive and has no array name, use `elem` as the array name. For fields within a struct or an array, the actual field name is specified outside of the closing brace.
.field_name	Include the field_name portion only when referencing a field within a struct or an array. Follow these guidelines: - For fields within a struct, the field_name portion uses the format: `.structName.fieldname` - For fields within an array, the field_name portion uses the format: `.fieldName`

Running a mapping with JSON data

To run a mapping that contains a Hierarchy Processor transformation with JSON-formatted data, you need to use a mapping task.

Reading JSON input

When you read JSON data, the input files can be based on a schema with multiple lines or on a schema with a single line.

The following sample shows a JSON schema on a single line:

```
{"Name":"Tom","Street":"2100 Seaport Blvd","City":"Redwood
City","State":"CA","Country":"USA","Zip":"94063"}
```

The following sample shows a JSON schema that spans across multiple lines:

```
{
    "Name": "Tom",
    "Surname": "Day",
    "City": "Redwood City",
    "State": "CA",
    "Country": "USA",
    "Zip": "94063"
}
```

By default, the Hierarchy Processor transformation reads each JSON schema as a single line. To read input that spans across multiple lines, you can configure the formatting options in the Source transformation to read multiple-line JSON files.

Writing JSON output

When you write JSON data, you can write each output record to a separate file, or you can write all output records to one file.

By default, each output record is written to a separate file. To write the output records to one JSON-formatted file, set the following Spark session property in the mapping task:

Session Property Name	Session Property Value
spark.sql.shuffle.partitions	1

Hierarchical to flattened example

You want to convert hierarchical data to relational data and write the data to a target file in denormalized format.

A shop maintenance file contains the customer and vehicle information for customers. The file is in hierarchical JSON format and is generated by your company's shop application.

The following JSON script shows the shop maintenance source input before you run the mapping:

```
{
    "people": [{
        "personal": {
            "age": 20,
            "gender": "M",
            "name": {
                "first": "John",
                "last": "Doe"
            }
        },
        "vehicles": [{
```

```
                        "type": "car",
                        "model": "Honda Civic",
                        "insurance": {
                            "policy_num": "HA12345"
                        },
                        "maintenance": [{
                            "desc": "oil change",
                            "cost": "111.50",
                            "summary": [{
                                "line1": "0w20",
                                "line2": "synthetic"
                            }, {
                                "line1": "2.0L 4-cyl",
                                "line2": "4.4 quarts"
                            }]
                        }, {
                            "desc": "new tires",
                            "cost": "425.00",
                            "summary": [{
                                "line1": "235/40R18",
                                "line2": "4 tires"
                            }, {
                                "line1": "All Season",
                                "line2": "No spare"
                            }]
                        }]
                    }, {
                        "type": "truck",
                        "model": "Dodge Ram",
                        "insurance": {
                            "policy_num": "DR12345"
                        },
                        "maintenance": [{
                            "desc": "new tires",
                            "cost": "299.99",
                            "summary": [{
                                "line1": "275/60R20",
                                "line2": "2 tires"
                            }, {
                                "line1": "All Season",
                                "line2": "No spare"
                            }]
                        }, {
                            "desc": "oil change",
                            "cost": "111.50",
                            "summary": [{
                                "line1": "5w30",
                                "line2": "conventional"
                            }, {
                                "line1": "5.7L V8",
                                "line2": "7.0 quarts"
                            }]
                        }]
                    }],
                    "source": "internet"
                }, {
                    "personal": {
                        "age": 24,
                        "gender": "F",
                        "name": {
                            "first": "Jane",
                            "last": "Roberts"
                        }
                    },
                    "vehicles": [{
                        "type": "car",
                        "model": "Toyota Camry",
                        "insurance": {
                            "policy_num": "TC98765"
                        },
```

```
            "maintenance": [{
                "desc": "tires rotated",
                "cost": "389.50",
                "summary": [{
                    "line1": "4 tires",
                    "line2": "leak repairs"
                }]
            }, {
                "desc": "oil change",
                "cost": "59.50",
                "summary": [{
                    "line1": "0w20",
                    "line2": "special"
                }]
            }]
        }, {
            "type": "car",
            "model": "Honda Accord",
            "insurance": {
                "policy_num": "HA98765"
            },
            "maintenance": [{
                "desc": "new air filter",
                "cost": "399.50",
                "summary": [{
                    "line1": "17220-6B2-A00",
                    "line2": "rebuild assembly"
                }]
            }, {
                "desc": "new brakes",
                "cost": "799.50",
                "summary": [{
                    "line1": "2-443344586",
                    "line2": "rear brake kit"
                }]
            }]
        }],
        "source": "phone"
    }]
}
```

You want to denormalize the vehicle maintenance data and exclude the customers' personal information.

Perform the following steps to create and configure the target file:

1. Step 1. Design the mapping.

2. Step 2. Configure the output group.

3. Step 3. Run the mapping.

Step 1. Design the mapping

The first step is to configure the Hierarchy Processor in the Mapping Designer with the Source and Target transformations.

Perform the following steps in the Mapping Designer:

1. Add a Source transformation to the mapping with the shop maintenance file as a source object.

2. Add a Hierarchy Processor transformation to the mapping and connect shop maintenance as an input source.

3. In the Hierarchy Processor transformation, select **Flattened** for the output format.

4. Add a Target transformation to the mapping, and connect the Hierarchy Processor transformation output to this target object.

Your mapping should look like the following image:
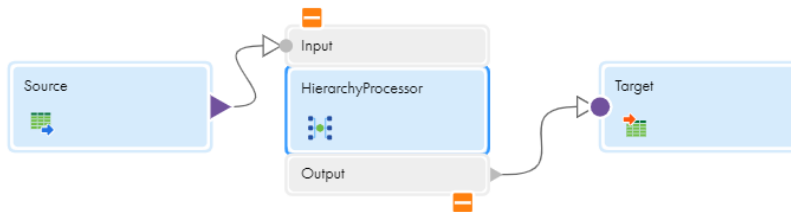
Step 2. Configure the output group

Once you configure the basic mapping, create an output group with the vehicle shop maintenance data.

Perform the following steps in the Hierarchy Processor transformation:

1. Select the top-level input group. This automatically selects all the input fields and adds them to the output.

2. Clear the personal struct. This automatically clears all the elements within the struct and removes them from the output.
   Your incoming and output fields should look like the following image:

3. Click on the output field for summary line1 and rename it to "Summary_line1."

4. Repeat the rename process for summary line2.

The following image shows how you edit the field name in the **Edit Output Field** dialog box:
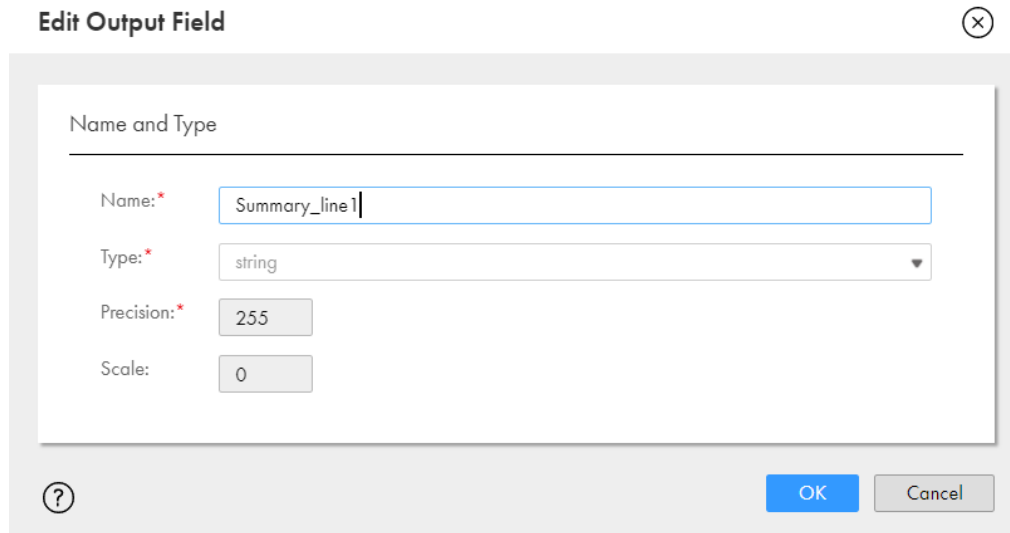
5. Verify the mapping.

Step 3. Run the mapping

The final step is to create and run the mapping task to produce the JSON output.

Perform the following steps:

1. Create a mapping task.
2. Run the mapping task.
3. Review your output.

The following table shows the partially denormalized target output after you run the mapping:

type	model	policy_num	desc	cost	Summary_line1	Summary_line2	source
car	Honda Civic	HA12345	oil change	111.5	0w20	synthetic	internet
car	Honda Civic	HA12345	oil change	111.5	2.0L 4-cyl	4.4 quarts	internet
car	Honda Civic	HA12345	new tires	425	235/40R18	4 tires	internet
car	Honda Civic	HA12345	new tires	425	All Season	No spare	internet
truck	Dodge Ram	DR12345	new tires	299.99	275/60R20	2 tires	internet
truck	Dodge Ram	DR12345	new tires	299.99	All Season	No spare	internet

type	model	policy_num	desc	cost	Summary_line1	Summary_line2	source
truck	Dodge Ram	DR12345	oil change	111.5	5w30	conventional	internet
truck	Dodge Ram	DR12345	oil change	111.5	5.7L V8	7.0 quarts	internet
car	Toyota Camry	TC98765	tires rotated	389.5	4 tires	leak repairs	phone
car	Toyota Camry	TC98765	oil change	59.5	0w20	special	phone
car	Honda Accord	HA98765	new air filter	399.5	17220-6B2-A00	rebuild assembly	phone
car	Honda Accord	HA98765	new brakes	799.5	2-443344586	rear brake kit	phone

CHAPTER 17

Input transformation

The Input transformation is a passive transformation that you use to configure the data that you want to pass into a mapplet. Use an Input transformation when you want a mapplet to receive data from an upstream transformation in a mapping or mapplet.

Add input fields to define the data fields that you want to pass into the mapplet from the upstream transformation.

You can add multiple Input transformations to a mapplet. Each Input transformation in a mapplet becomes an input group in when you use the mapplet in a Mapplet transformation. You must connect at least one input group to an upstream transformation.

You can use an Input transformation in a mapplet, but not in a mapping.

Input fields

Add input fields to an Input transformation to define the fields that you want to pass into the mapplet. You must add at least one input field to each Input transformation.

Add input fields on the **Input Fields** tab. To add a field, click **Add Fields**, and then enter the field name, data type, precision, scale, and optional description. The description can contain up to 4000 characters.

When you use the mapplet in a Mapplet transformation, map at least one input field to the upstream transformation.

CHAPTER 18

Java transformation

Extend Data Integration functionality with the Java transformation. The Java transformation provides a simple, native programming interface to define transformation functionality with the Java programming language.

You can use the Java transformation to quickly define simple or moderately complex transformation functionality without advanced knowledge of the Java programming language. The Java transformation can be an active or passive transformation.

The Secure Agent requires a Java Development Kit (JDK) to compile the Java code and generate byte code for the transformation. Azul OpenJDK is installed with the Secure Agent, so you do not need to install a separate JDK. Azul OpenJDK includes the Java Runtime Environment (JRE).

The Secure Agent uses the JRE to execute generated byte code at run time. When you run a mapping or mapping task that includes a Java transformation, the Secure Agent uses the JRE to execute the byte code, process input rows, and generate output rows.

To create a Java transformation, you write Java code snippets that define the transformation logic. Define transformation behavior for a Java transformation based on the following events:

- The transformation receives an input row.
- The transformation has processed all input rows.
- The transformation receives a transaction notification.

You can invoke expressions in a Java transformation only in advanced mode.

You cannot use the Java transformation with a Graviton-enabled cluster. For more information about Graviton-enabled clusters, see the Administrator help.

Note: When you create a Java transformation, ensure that you review the Java code to verify that it is free from potentially unsafe active content such as queries, remote scripts, or data connections before you run the code in a mapping task.

Defining a Java transformation

To define a Java transformation, configure the transformation fields and properties, enter Java code snippets on the **Java** tab, and compile the code.

Note: If you use third-party or custom Java packages, before you create a Java transformation, configure the classpaths to use when you compile the code and when you run the mapping that contains the transformation.

1. In the Mapping Designer, drag a Java transformation from the transformations palette onto the canvas and connect it to the upstream and downstream transformations.

2. Configure incoming field rules and output fields for the transformation.

3. Configure the transformation properties.

4. Write the Java code snippets that define the transformation functionality.

5. Compile the code.

6. Locate and fix compilation errors.

Classpath configuration

If you use third-party or custom Java packages in the Java transformation, configure the classpath and at least on class value. The Secure Agent includes the classes and resource files within the classpath when it compiles the Java code

For example, you import the Java package `converter` in the Java transformation and define the package in converter.jar. You must add converter.jar to the classpath before you compile the Java code. However, if you import the package java.io, you do not need to set the classpath because java.io is a built-in Java package. You do not need to configure the classpath if you use only built-in Java packages.

To make the JAR or class files accessible to the Secure Agent, set the classpath to each JAR file or class file directory associated with the Java package. Separate multiple classpath entries with a semicolon on Windows or a colon on UNIX.

JAR or class files on an advanced cluster

If you use third-party or custom Java packages in the Java transformation in advanced mode, store the JAR or class files in the Secure Agent installation directory.

Store the JAR or class files in the following directory on the Secure Agent machine:

```
<Secure Agent installation directory>/ext/ctjars
```

If you use native libraries in a Java transformation on in advanced mode, create the following native-lib and native-bin directories for the .so files and executables, respectively, at the same location as the ctjars directory:

```
<Secure Agent installation directory>/ext/native-lib
```

```
<Secure Agent installation directory>/ext/native-bin
```

If you update the JAR or class files on the Secure Agent machine, the files take effect the next time you run a job in advanced mode. If the Secure Agent machine stops unexpectedly and the agent restarts on a different machine, add the JAR or class files to the same directory on the new machine.

Note: In production environments, avoid frequent updates to the contents under the ext folder (ctjars, connectors, python, native-lib, native-bin) during workload execution. It can negatively impact long running jobs and cause failures. If necessary, you can set the k8s.infarpm.max.installers=x flag to a higher value than the default of 5, but doing so can cause disk pressure issues on the worker nodes.

If you use a serverless runtime environment, store the files in the supplementary file location. For more information about the supplementary file location, see the Administrator help.

Classpath values

If you use third-party or custom Java packages in the Java transformation, configure at least one classpath value.

You can set the following classpath values:

JVMClassPath property for the Secure Agent

The Secure Agent uses this classpath when you design and validate the Java transformation, run the mapping from the Mapping Designer, or run the mapping task. This classpath applies to all mappings and mapping tasks that run on the agent.

Set the JVMClassPath property for the Secure Agent in Administrator.

Set this property or set the CLASSPATH environment variable on the Secure Agent machine. You do not need to set both classpath values.

CLASSPATH environment variable

The Secure Agent uses this classpath when you design and validate the Java transformation, run the mapping from the Mapping Designer, or run the mapping task. This classpath applies to all mappings and mapping tasks that run on the agent.

Set the CLASSPATH environment variable on the Secure Agent machine.

Set the CLASSPATH environment variable or set the JVMClassPath property for the Secure Agent. You do not need to set both classpath values.

Design time classpath

The Secure Agent uses this classpath when you design and validate the Java transformation and when you run the mapping from the Mapping Designer. This classpath is not used when you run the mapping through a mapping task.

Set the design-time classpath when you want to test the transformation and neither the JVMClassPath property nor the CLASSPATH environment variable contain the required packages. If you configured the JVMClassPath property or the CLASSPATH environment variable to include the required packages, then you do not need to configure the design time classpath.

You configure the design-time classpath in the Java transformation advanced properties.

Java Classpath session property

The Secure Agent uses this classpath when you run the mapping task. This classpath applies only to the mapping task in which the property is set.

Set the Java Classpath session property when you want the classpath to apply to one mapping task but not others. If you configured the JVMClassPath property or the CLASSPATH environment variable to include the required packages, then you do not need to configure the Java Classpath session property.

Set the Java Classpath session property in the advanced session properties of the mapping task.

If you set multiple classpath values, the Secure Agent uses all of the classpaths that apply. For example, you set the JVMClassPath property for the Secure Agent, the CLASSPATH environment variable, and the design

time classpath in the Java transformation. When you compile the Java code in the Java transformation or run the mapping through the Mapping Designer, the Secure Agent uses all three classpaths. When you run the mapping through a mapping task, the Secure Agent uses the JVMClassPath and the CLASSPATH environment variable only.

Warning: If you set multiple classpaths, ensure that they do not create multiple copies of a class or resource which can cause runtime errors.

Configuring the JVM Classpath for the Secure Agent

Configure the DTM JVMClassPath property for the Secure Agent on the Secure Agent page in Administrator. The Secure Agent uses this classpath when you design and validate the Java transformation, run the mapping from the Mapping Designer, or run the mapping task. You must restart the Secure Agent after you configure this classpath.

1. In Administrator, select **Runtime Environments**.
2. Select the Secure Agent.
3. On the Secure Agent page, click **Edit**.
4. In the system configuration details, select the following values:

Option	Value
Service	Data Integration Server
Type	DTM

5. In the row for the JVMClassPath property, click **Edit**.
6. Append your classpath to the existing classpath value.

   On Windows, use a semicolon (`;`) to separate classpath entries. On UNIX, use a colon (`:`).

   Note: Do not delete the default value, `'pmserversdk.jar'`.
7. Click **Save**.

Restart the Secure Agent after you update the system configuration details.

Configuring the CLASSPATH environment variable

Configure the CLASSPATH environment variable on the Secure Agent machine. The Secure Agent uses this classpath when you design and validate the Java transformation, run the mapping from the Mapping Designer, or run the mapping task. You must restart the Secure Agent after you configure this classpath.

You configure the CLASSPATH environment variable differently on Windows and UNIX.

Configuring the CLASSPATH on Windows

On Windows, you configure the CLASSPATH environment variable from the Advanced System Properties in the Control Panel.

1. Open the Advanced System Properties from the Windows Control Panel.
2. Click **Environment Variables**.
3. Under System variables, click **New**.
4. Set the variable name to CLASSPATH and the variable value to the classpath.

5.   Click **OK**.

Restart the Secure Agent after you configure the environment variable.

Configuring the CLASSPATH on UNIX

On UNIX, you configure the CLASSPATH environment variable from a UNIX shell.

▶   Enter one of the following commands:

- In a UNIX C shell environment, type:

```
setenv CLASSPATH <classpath>
```

- In a UNIX Bourne shell environment, type:

```
CLASSPATH = <classpath>
export CLASSPATH
```

Restart the Secure Agent after you configure the environment variable.

Configuring the design time classpath

Configure the design time classpath in the Java transformation advanced properties. The Secure Agent uses the design time classpath when you design and validate the Java transformation and when you run the mapping from the Mapping Designer.

1.   Open the mapping that contains the Java transformation.

2.   Select the Java transformation and click the **Advanced Properties** tab.

3.   Enter the classpath in the **Design Time Classpath** field.

Configuring the Java Classpath session property

Configure the Java Classpath session property in the advanced session properties of the mapping task. The Secure Agent uses this classpath when you run the mapping task.

1.   Open the mapping task.

2.   On the **Schedule** page, scroll down to the advanced session properties, and click **Add**.

3.   Under **General Options Settings**, select **Java Classpath**.

4.   Enter the classpath in the **Set the Session Property Value** field.

5.   Click **Save**.

Java transformation fields

A Java transformation has incoming fields and output fields. Use incoming and output fields as variables in the Java code snippets that you define on the **Java** tab.

Incoming fields appear on the **Incoming Fields** tab. By default, the Java transformation inherits all incoming fields from the upstream transformation. If you do not need to use all of the incoming fields, you can define field rules to include or exclude certain fields. For more information about field rules, see "Field rules" on page 26.

Add output fields on the **Output Fields** tab. Add output fields for the output data that you want to pass to the downstream transformation. To add a field, click **Add Field**, and then enter the field name, data type, precision, scale, and optional description. The description can contain up to 4000 characters. You can also create output fields on the **Outputs** tab of the Java editor by clicking **Create New Field**.

The Java transformation initializes output field values based on the following field data types:

- Primitive data types. The transformation initializes the value of the field variable to 0.
- Complex data types. The transformation initializes the field variable to null.

Data type conversion

A Java transformation converts Data Integration transformation data types to Java data types, based on the Java transformation field type.

When a Java transformation reads input rows, it converts input field data types to Java data types. When a Java transformation writes output rows, it converts Java data types to output field data types.

For example, the following processing occurs for an input field with the integer data type in a Java transformation:

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

The following table shows how the Java transformation maps the Data Integration transformation data types to Java primitive and complex data types:

Transformation data type	Java data type
bigint	long
binary	byte[]
date/time	BigDecimal long (number of milliseconds since January 1, 1970 00:00:00.000 GMT)
decimal	double BigDecimal
double	double
integer	int
string	String
text	String

In Java, the String, byte[], and BigDecimal data types are complex data types, and the double, int, and long data types are primitive data types.

The Java transformation sets null values in primitive data types to zero. You can use the isNull and the setNull API methods in the On Input Row section of the Java editor to set null values in the input field to null values in the output field. For an example, see "setNull" on page 279.

When you configure the Java transformation, you can convert the date/time data type to BigDecimal or to long. BigDecimal has nanosecond precision while long has millisecond precision. For more information, see "Configuring Java transformation properties" on page 257.

The decimal data type maps to BigDecimal when high precision is enabled. BigDecimal cannot be used with some operators, such as the + operator. If the Java code contains an expression that uses a decimal field and the field is used with one of the operators, the Java code fails to compile.

Sort conditions

In advanced mode, you can configure a sort condition to specify the sort fields and the sort order. The mapping task uses the sort condition to sort the data before the Java code runs.

The sort fields are one or more fields that you want to use as the sort criteria. Configure the sort order to sort data in ascending or descending order.

If you configure a sort condition for data that is grouped into partitions, the mapping task sorts the data in each partition.

When you specify multiple sort conditions, the mapping task sorts each condition sequentially. The mapping task treats each successive sort condition as a secondary sort of the previous sort condition. You can configure the order of sort conditions.

If you use a parameter for the sort condition, define the sort fields and the sort order when you run the mapping or when you configure the mapping task.

Group by fields

In advanced mode, you can use group by fields to define how to group data into partitions before the Java code runs.

When you configure a group by field, the mapping task groups rows with the same data into a partition. Then, the Java code runs for each partition in the transformation. For example, the input row behavior is processed for each partition and each row in the partition, and the end of data behavior is processed for each partition after processing all rows in the partition.

When you select more than one group by field, the task creates a partition for each unique combination of data in the group by fields.

If you do not configure a group by field, the Java code runs based on the data's default partitioning scheme.

If you use a parameter for the group by fields, define the group by fields when you run the mapping or when you configure the mapping task.

Configuring Java transformation properties

The Java transformation includes properties for both the transformation code and the transformation. Configure Java transformation properties on the **Advanced** tab.

The properties that are available vary based on the mapping mode.

The following table describes the Java transformation properties:

Property	Description
Behavior	Transformation behavior, either active or passive. If active, the transformation can generate more than one output row for each input row. If passive, the transformation generates one output row for each input row. Default is Active.
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Transformation Scope	The method in which the Secure Agent applies the transformation logic to incoming data. Use the following options: - Row. Applies the transformation logic to one row of data at a time. Choose Row when the results of the transformation depend on a single row of data. - Transaction. Applies the transformation logic to all rows in a transaction. Choose Transaction when the results of the transformation depend on all rows in the same transaction, but not on rows in other transactions. For example, you might choose Transaction when the Java code performs aggregate calculations on the data in a single transaction. - All Input. Applies the transformation logic to all incoming data. When you choose All Input, the Secure Agent drops transaction boundaries. Choose All Input when the results of the transformation depend on all rows of data in the source. For example, you might choose All Input when the Java code for the transformation sorts all incoming data. For active transformations, default is All Input. For passive transformations, this property is always set to Row.
Defines Update Strategy	Specifies whether the transformation defines the update strategy for output rows. When enabled, the Java code determines the update strategy for output rows. When disabled, the update strategy is determined by the operation set in the Target transformation. You can configure this property for active Java transformations. Default is disabled.
Enable High Precision	Enables high precision to process decimal fields with the Java class BigDecimal. Enable this option to process decimal data types with a precision greater than 15 and less than 28. Default is disabled. In advanced mode, the Java transformation always uses high precision.
Use Nanoseconds in Date/Time	Specifies whether the generated Java code converts the transformation date/time data type to the Java BigDecimal data type, which has nanosecond precision. When enabled, the generated Java code converts the transformation date/time data type to the Java BigDecimal data type. When disabled, the code converts the date/time data type to the Java long data type, which has millisecond precision. Default is disabled.

Property	Description
Optional	Determines whether the transformation is optional. If a transformation is optional and there are no incoming fields, the mapping task can run and the data can go through another branch in the data flow. If a transformation is required and there are no incoming fields, the task fails. For example, you configure a parameter for the source connection. In one branch of the data flow, you add a transformation with a field rule so that only Date/Time data enters the transformation, and you specify that the transformation is optional. When you configure the mapping task, you select a source that does not have Date/Time data. The mapping task ignores the branch with the optional transformation, and the data flow continues through another branch of the mapping. Default is enabled.
Design Time Classpath	Classpath that the Secure Agent uses for custom or third-party packages when you design and validate the transformation and when you run the mapping from the Mapping Designer. This classpath is not used when you run the mapping through a mapping task. Set the design-time classpath when you want to test the transformation and neither the JVMClassPath property for the Secure Agent nor the CLASSPATH environment variable on the Secure Agent machine contain the required packages. If you configured the JVMClassPath property or the CLASSPATH environment variable to include the required packages, then you do not need to configure this property.

Active and passive Java transformations

When you create a Java transformation, you define the behavior as active or passive. Define the behavior on the **Advanced** tab.

A Java transformation runs the Java code that you define in the On Input Row section of the Java editor one time for each row of input data.

A Java transformation handles output rows based on the behavior as follows:

- An active Java transformation generates multiple output rows for each input row in the transformation.

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

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

You can change the transformation behavior. However, when you change the behavior, you must recompile the Java code.

Defining the update strategy

You can define the update strategy for active Java transformations. To define the update strategy, use the **Defines Update Strategy** option on the **Advanced** tab.

You can define the update strategy at the following levels:

Within the Java code

You can write the Java code to set the update strategy for output rows. The Java code can flag rows for insert, update, delete, or reject. To define the update strategy with in the Java code, enable the **Defines Update Strategy** option and use the setOutRowType method in the On Input Row section of the Java editor to flag rows. For more information about setting the update strategy, see "setOutRowType" on page 280.

Within the Target transformation

You can configure the Target transformation to set the update strategy. To configure the Target transformation to set the update strategy, disable the **Defines Update Strategy** option and configure the target operation in the target properties.

This option does not apply to passive transformations.

Using high precision

You can configure how the Java transformation handles high precision for decimal data types. To enable high precision, use the **Enable High Precision** option on the **Advanced** tab.

By default, the Java transformation converts fields of type decimal to double data types with a precision of 15. If you want to process a decimal data type with a precision greater than 15, enable high precision to process decimal fields with the Java class BigDecimal.

When you enable high precision, you can process decimal fields with precision less than 28 as BigDecimal. The Java transformation converts decimal data with a precision greater than 28 to the double data type.

For example, a Java transformation has an input field of type decimal that receives a value of 40012030304957666903. If you enable high precision, the value of the field is treated as it appears. If you do not enable high precision, the value of the field is 4.00120303049577 x 10^19.

High precision in advanced mode

To run a mapping task based on a mapping in advanced mode successfully, the Java transformation and the mapping must use the same precision mode. The Java transformation always uses high precision, but the mapping uses either high or low precision based on the types of hierarchical fields in the mapping.

If the mapping has hierarchical fields that contain child fields with decimal data types, the mapping runs using low precision. If the same mapping contains a Java transformation with a decimal field, the mapping task fails.

Processing subseconds

You can configure how the Java transformation processes subseconds for date/time data types. Define subsecond handling on the **Advanced** tab.

By default, the generated Java code converts the transformation date/time data type to the Java long data type, which has precision to the millisecond.

You can process subsecond data up to nanoseconds in the Java code. To process nanoseconds, enable the **Use Nanoseconds in Date/Time** option. When you enable this option, the generated Java code converts the transformation date/time data type to the Java BigDecimal data type, which has precision to the nanosecond.

Advanced mode supports precision to the microsecond. If a date/time value contains nanoseconds, the trailing digits are truncated.

Developing the Java code

To define the Java transformation functionality, you enter Java code snippets on the **Java** tab. Enter code snippets to import Java packages, write helper code, and write Java code that defines the behavior for specific transformation events. You can develop code snippets in the Java editor in any order.

Enter Java code snippets in the following sections of the Java editor:

Import Packages

Import third-party Java packages, built-in Java packages, or custom Java packages.

Helper Code

Define variables and methods available to all sections except Import Packages.

On Input Row

Define the transformation behavior when it receives an input row.

At End of Data

Define the transformation behavior when it has processed all input data.

On Receiving Transaction

Define the transformation behavior when it receives a transaction notification. Use with active Java transformations.

You cannot use the On Receiving Transaction section in advanced mode.

Access input data and set output data in the On Input Row section. For active transformations, you can also set output data in the At End of Data and On Receiving Transaction sections.

Creating Java code snippets

To create Java code snippets that define transformation behavior, use the Java editor on the **Java** tab.

The following image shows the **Java** tab with the Java editor expanded:
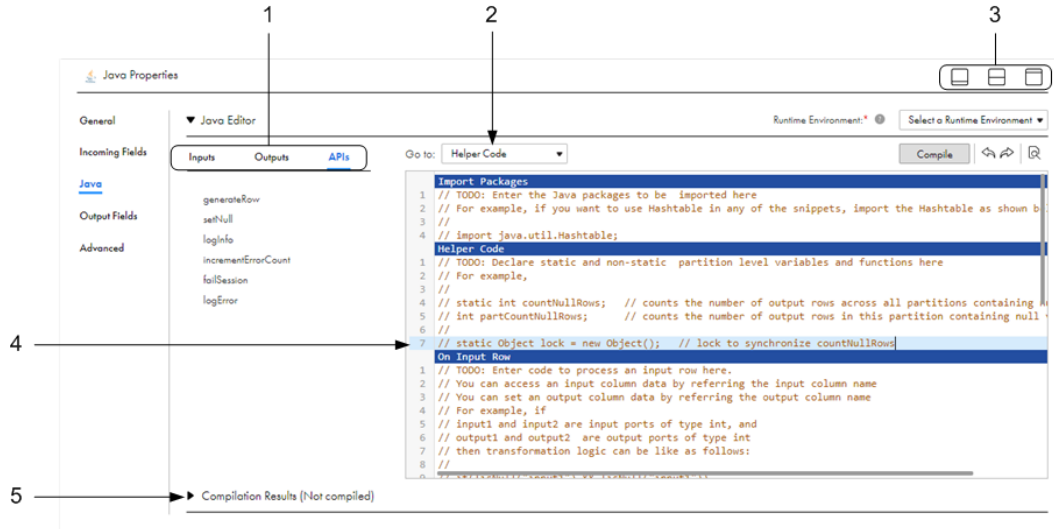
1. Inputs, Outputs, and APIs tabs. Use these tabs to add input and output fields as variables and to call API methods in the Java code snippets. The fields and methods displayed on these tabs vary based on which section of the code entry area is selected.
2. Go to list. Use to switch among the sections in the code entry area.
3. Minimize, Open Both, and Maximize icons. Use the Minimize and Maximize buttons to minimize and maximize the transformation properties. Use the Open Both icon to open the Mapping Designer canvas and the transformation properties at the same time.
4. Code entry area. Enter Java code snippets in the Import Packages, Helper Code, On Input Row, At End of Data, and On Receiving Transaction sections.
5. Compilation results. Expand the compilation results to see detailed compilation results, compilation errors, and view the full code.

Tip: To expand the transformation properties so that you can see the code entry area more fully, click **Maximize**.

1.   In the **Go to** list, select the section in which you want to enter a code snippet.

2.   To access an input or output field in the snippet, select the field on the **Inputs** or **Outputs** tab, and click **Add**.

     You can also create output fields on the **Outputs** tab by clicking **Create New Field**.

3.   To call a Java transformation API method in the snippet, select the method on the **APIs** tab, and click **Add**.

     The methods displayed on the **APIs** tab change based on which section is selected. For example, you can use the getInRowType method only in the On Input Row section but not in other sections. Therefore, this method is listed only when the On Input Row section is selected.

4.   If necessary, configure the method input values.

5.   Write appropriate Java code based on the section.

After you finish creating the Java code snippets, compile the code to validate the transformation.

Importing Java packages

You can import Java packages for active or passive Java transformations. Import packages in the Import Packages section of the Java editor.

For example, to import the Java I/O package, enter the following code in the Import Packages section:

```
import java.io.*;
```

You can import built-in, third-party, or custom Java packages. If you import third-party or custom Java packages, you must add the packages to the classpath. For more information about configuring the classpath, see "Classpath configuration" on page 252.

After you import Java packages, you can use the imported packages in the other sections.

You cannot declare or use static variables, instance variables, or user methods in the Import Packages section.

Note: When you export a mapping or mapping task that contains a Java transformation, the jar or class files that contain the third-party or custom packages required by the Java transformation are not included in the export XML file. Therefore, when you import the mapping or task, you must copy the jar or class files that contain the required third-party or custom packages to the Secure Agent machine.

Defining helper code

You can declare user-defined variables and methods for the Java transformation class in active or passive Java transformations. Define helper code in the Helper Code section of the Java editor.

After you declare variables and methods in the Helper Code area, you can use them in any code entry area except the Import Packages area.

You can declare the following types of code, variables, and methods:

Static code and static variables

Within a static block, you can declare static variables and static code. Static code runs before any other code in a Java transformation.

For example, the following code declares a static variable to store the error threshold for a Java transformation:

```
static int errorThreshold;
```

User-defined static or instance methods

These methods extend the functionality of the Java transformation. Java methods that you declare in the Helper Code section can use or modify output variables. You cannot access input variables from Java methods in the Helper Code section.

For example, use the following code in the Helper Code section to declare a function that adds two integers:

```
private int myTXAdd (int num1,int num2)
{
    return num1+num2;
}
```

Defining input row behavior

You can define the behavior of the Java transformation when it receives an input row. Define input row behavior in the On Input Row section of the Java editor. The Java code in this section executes one time for each input row.

Access and use the following input and output field data, variables, and methods in the On Input Row section:

Input field and output field variables

Access input and output field data as a variable by using the name of the field as the name of the variable. For example, if "in_int" is an integer input field, you can access the data for this field by referring as a variable "in_int" with the Java primitive data type int. You do not need to declare input and output fields as variables.

Do not assign a value to an input field variable. If you assign a value to an input variable in the On Input Row section, you cannot get the input data for the corresponding field in the current row.

Static variables and user-defined methods

Use any static variable or user-defined method that you declared in the Helper Code section.

For example, an active Java transformation has two input fields, BASE_SALARY and BONUSES, with an integer data type, and a single output field, TOTAL_COMP, with an integer data type. You create a user-defined method in the Helper Code section, myTXAdd, that adds two integers and returns the result.

Use the following Java code in the On Input Row section to assign the total values for the input fields to the output field and generate an output row:

```
TOTAL_COMP = myTXAdd (BASE_SALARY,BONUSES);
generateRow();
```

When the Java transformation receives an input row, it adds the values of the BASE_SALARY and BONUSES input fields, assigns the value to the TOTAL_COMP output field, and generates an output row.

Java transformation API methods

You can call API methods provided by the Java transformation.

Defining end of data behavior

You can define the behavior of an active or passive Java transformation when it has processed all input data. Define end of data behavior in the At End of Data section of the Java editor.

To generate output rows in the At End of Data section, set the transformation scope for the transformation to Transaction or to All Input on the **Advanced** tab. You cannot access or set the value of input field variables in this section.

Access and use the following variables and methods in the At End of Data section:

Output field variables

Use the names of output fields as variables to access or set output data for active Java transformations.

User-defined methods

Use any user-defined method that you declared in the Helper Code section.

Java transformation API methods

Call API methods provided by the Java transformation. For example, use the following Java code to write information to the session log when the end of data is reached:

```
logInfo("Number of null rows for partition is: " + partCountNullRows);
```

If you use the Java transformation in advanced mode, consider the following guidelines:

- You cannot write output to standard output, but you can write output to standard error which appears in the log files.

- You cannot pass binary null characters to an output field.

  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 field.

Defining transaction notification behavior

You can define the behavior of an active Java transformation when it receives a transaction notification. Define the transaction notification behavior in the On Receiving Transaction section of the Java editor.

The code snippet in the On Receiving Transaction section is only executed if the Transaction Scope for the transformation is set to Transaction. You cannot access or set the value of input field variables in this section.

Access and use the following output data, variables, and methods from the On Receiving Transaction section:

Output field variables

Use the names of output fields as variables to access or set output data.

User-defined methods

Use any iuser-defined method that you declare in the Helper Code section.

Java transformation API methods

Call API methods provided by the Java transformation.

Using Java code to parse a flat file

You can develop Java code to parse a flat file. Use Java code to extract specific columns of data from a flat file of varying schema or from a JMS message.

For example, you want to read the first two columns of data from a delimited flat file. Create a mapping that reads data from a delimited flat file and passes data to one or more output fields.

The mapping contains the following components:

Source transformation

The source is a delimited flat file. Configure the source to pass each row as a single string to the Java transformation. The source file contains the following data:

```
1a,2a,3a,4a,5a,6a,7a,8a,9a,10a
1b,2b,3b,4b,5b,6b,7b,8b,9b
1c,2c,3c,4c,5c,6c,7c
1d,2d,3d,4d,5d,6d,7d,8d,9d,10d
```

Java transformation

Define the Java transformation functionality in the Java editor.

Use the On Input Row section of the Java editor to read each input row and pass the first two fields to the output field, outputRow. Enter the following code in the On Input Row section:

```
// Collect the first two fields of the row and output them into outputRow.
String[] rowsSplit = row.split(",", 3);
if (rowsSplit.length >= 2) {
  outputRow = rowsSplit[0] + "," + rowsSplit[1];
}
generateRow();
```

Target transformation

Configure the target to receive the output field, outputRow, from the Java transformation. After you run the mapping, the target file has the following data:

```
1a,2a
1b,2b
1c,2c
1d,2d
```

Compiling the code

To validate a Java transformation, you must compile the Java code snippets that you enter in the Java editor. Data Integration uses the Secure Agent to compile the Java code snippets. Compile the code and view compilation results on the **Java** tab.

When you create a Java transformation, it contains a Java class that defines the base functionality for the transformation. When you compile the transformation, the Secure Agent adds the code that you enter in the Java editor to the template class for the transformation. This generates the full class code for the transformation.

The Secure Agent calls the JDK to compile the full class code. The JDK compiles the transformation and generates the byte code for the transformation.

Note: In a mapplet, the Java transformation compiles based on the data types and APIs that you can run on the Data Integration Server. If the code contains data types or APIs that you can run only on an advanced cluster, such as the invokeJExpression API method, the code fails to compile.

Before you can compile the code, you must complete the following tasks:

1. If you use non-standard Java packages, configure the classpath.

    For more information about configuring the classpath, see .
2. Select the runtime environment to use for compilation.

    Select the runtime environment on the **Java** tab. The Hosted Agent is not supported for compilation.

To compile the code, click **Compile** in the Java editor.

The compilation results shows the results of the compilation. Use the compilation results to identify and locate Java code errors.

Viewing the full class code

You can view and download the full class code that is generated when you compile a Java transformation. You can access the full class code from the Compilation Results on the **Java** tab.

To view the full class code, click **View Full Code** in the compilation results. Data Integration displays the full class code in the **Full Code** dialog box.

To download the full code, click **Download Full Code** in the **Full Code** dialog box.

Troubleshooting a Java transformation

You can identify Java code errors and locate the source of Java code errors for a Java transformation in the compilation results of the **Java** tab. Java transformation errors might occur because of an error in the Java editor or because of an error in the full code for the Java transformation class.

To troubleshoot a Java transformation:

- Find the source of the error in the Java code snippets or in the full class code for the transformation.
- Identify the type of error using the compilation results and the location of the error.

After you identify the source and type of error, fix the Java code on the **Java** tab and compile the transformation again.

Finding the source of compilation errors

When you compile a Java transformation, results are displayed in the compilation results. If the compilation succeeds, the compilation results displays the message, "Compilation Successful." If the compilation fails, the compilation results displays the compilation errors and error locations.

Compilation errors can occur in the following locations:

Java editor

If the error is located in a code snippet in Java editor, Data Integration lists the section and the line that contains the error. Data Integration also highlights the source of the error in the Java editor.

Full code

If the error is located in the full code, Data Integration lists the error location as "Full Code" and lists the line in the **Full Code** dialog box that contains the error.

You can locate errors in the **Full Code** dialog box, but you cannot edit the Java code. To fix errors that you find in the **Full Code** dialog box, edit the code in the appropriate section. You might need to use the **Full Code** dialog box to view errors caused by adding user code to the full class code for the transformation.

Identifying the error type

Compilation errors can appear as a result of errors in the user code. Errors in the user code might also generate an error in the non-user code for the class.

Compilation errors can be of the following types:

User code errors

Errors can occur in the user code in different sections of the Java editor. User code errors include standard Java syntax and language errors. User code errors might also occur when Data Integration adds the user code to the full class code.

For example, a Java transformation has an input field with a name of int1 and an integer data type. The full code for the class declares the input field variable with the following code:

```
int int1;
```

However, if you use the same variable name in the On Input Row section, the Java compiler issues an error for a redeclaration of a variable. To fix the error, rename the variable in the On Input Row section.

Non-user code errors

User code in sections of the Java editor can cause errors in non-user code.

For example, a Java transformation has an input field, int1, and an output field, out1, with integer data types. You enter the following code in the On Input Row section to calculate interest for input field int1 and assign it to output field out1:

```
int interest;
interest = CallInterest(int1); // calculate interest
out1 = int1 + interest;
}
```

When you compile the transformation, Data Integration adds the code from the On Input Row section to the full class code for the transformation. When the Java compiler compiles the Java code, the unmatched brace causes a method in the full class code to end prematurely, and the Java compiler issues an error.

Java transformation example

You can use the Java code in this example to create and compile an active Java transformation.

Use a Java transformation to process employee data for a fictional company. The Java transformation reads input rows from a flat file source and writes output rows to a flat file target. The source file contains employee data, including the employee identification number, name, job title, and the manager identification number.

The transformation finds the manager name for a given employee based on the manager identification number and generates output rows that contain employee data. The output data includes the employee identification number, name, job title, and the name of the employee's manager. If the employee has no manager in the source data, the transformation assumes the employee is at the top of the hierarchy in the company organizational chart.

Note: The transformation logic assumes that the employee job titles are arranged in descending order in the source file.

To create and run the mapping in this example, perform the following steps:

1. Create the source file.

    Create a comma-delimited flat file in a directory that the Secure Agent can access.
2. Configure the mapping.

    Add a source, target, and Java transformation to the mapping, and configure the fields.
3. Configure the Java code snippets in the Java transformation.

    Enter code snippets in the Import Packages, Helper Code, and On Input Row sections of the Java editor.
4. Compile the code and run the mapping.

Create the source file

Create the source file as a comma-delimited flat file. Save the file in a directory that the Secure Agent can access.

Use the following data:

```
EMP_ID,EMP_NAME,EMP_AGE,EMP_DESC,EMP_PARENT_EMPID
1,James Davis,50,CEO,
4,Elaine Masters,40,Vice President - Sales,1
5,Naresh Thiagarajan,40,Vice President - HR,1
6,Jeanne Williams,40,Vice President - Software,1
9,Geetha Manjunath,34,Senior HR Manager,5
```

```
10,Dan Thomas,32,Senior Software Manager,6
14,Shankar Rahul,34,Senior Software Manager,6
20,Juan Cardenas,32,Technical Lead,10
21,Pramodh Rahman,36,Lead Engineer,14
22,Sandra Patterson,24,Software Engineer,10
23,Tom Kelly,32,Lead Engineer,10
35,Betty Johnson,27,Lead Engineer,14
50,Dave Chu,26,Software Engineer,23
70,Srihari Giran,23,Software Engineer,35
71,Frank Smalls,24,Software Engineer,35
```

Configure the mapping

To configure the mapping in this example, create a new mapping, and add a Java transformation. Then configure the source fields, target fields, and Java transformation fields. In the Java transformation, verify that the transformation behavior is active.

The following image shows the mapping:
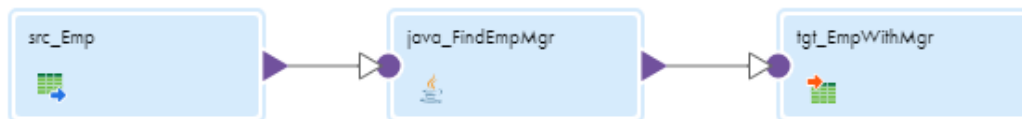
Configure the transformations as follows:

Source transformation

In the Source transformation, update the source field metadata as shown in the following table:

Name	Type	Precision	Scale	Origin
EMP_ID	integer	10	0	<source file name>
EMP_NAME	string	100	0	<source file name>
EMP_DESC	string	100	0	<source file name>
EMP_AGE	integer	10	0	<source file name>
EMP_PARENT_EMPID	integer	10	0	<source file name>

Java transformation

The Java transformation includes all incoming fields from the Source transformation.

Create the following output fields:

Name	Type	Precision	Scale
EMP_ID_OUT	integer	10	0
EMP_NAME_OUT	string	100	0

Name	Type	Precision	Scale
EMP_DESC_OUT	string	100	0
EMP_PARENT_EMPNAME	string	100	0

On the **Advanced** tab, verify that the behavior is set to active.

Target transformation

In the target transformation, create the following target fields:

Name	Type	Precision	Scale	Origin
EMP_ID	integer	10	0	<target file name>
EMP_NAME	string	100	0	<target file name>
EMP_DESC	string	100	0	<target file name>
EMP_PARENT_EMPNAME	string	100	0	<target file name>

Configure the field mapping as shown in the following table:

Target Field Name	Mapped Field
EMP_ID	EMP_ID_OUT
EMP_NAME	EMP_NAME_OUT
EMP_DESC	EMP_DESC_OUT
EMP_PARENT_EMPNAME	EMP_PARENT_EMPNAME

Configure the Java code snippets

Enter the Java code snippets that define the transformation functionality on the **Java** tab.

Enter the Java code snippets in the following sections of the Java editor:

Import Packages

Import the java.util.Map and java.util.HashMap packages.

Helper Code

Create a Map object, lock object, and boolean variables to track the state of data in the Java transformation.

On Input Row

Enter code that defines the behavior of the Java transformation when it receives an input row.

Import packages

Import third-party Java packages, built-in Java packages, or custom Java packages in the Import Packages section. This example uses the Map and HashMap packages.

Enter the following code in the Import Packages section:

```
import java.util.Map;
import java.util.HashMap;
```

Data Integration adds the import statements to the Java code for the transformation.

Helper code

Declare user-defined variables and methods for the Java transformation in the Helper Code section.

The Helper Code section defines the following variables that are used by the Java code in the On Input Row section:

Variable	Description
empMap	Map object that stores the identification number and employee name from the source.
lock	Lock object used to synchronize the access to empMap across partitions.
generateRow	Boolean variable used to determine whether an output row should be generated for the current input row.
isRoot	Boolean variable used to determine whether an employee is at the top of the company organizational chart (root).

Enter the following code in the Helper Code section:

```
// Static Map object to store the ID and name relationship of an employee.
// If a session uses multiple partitions, empMap is shared across all partitions.
private static Map <Integer, String> empMap = new HashMap <Integer, String> ();

// Static lock object to synchronize the access to empMap across partitions.
private static Object lock = new Object();

// Boolean to track whether to generate an output row based on validity of the
// input data.
private boolean generateRow;

// Boolean to track whether the employee is root.
private boolean isRoot;
```

Data Integration adds the import statements to the Java code for the transformation.

On Input Row

The Java transformation executes the Java code in the On Input Row section when the transformation receives an input row. In this example, the transformation might or might not generate an output row, based on the values of the input row.

Enter the following code in the On Input Row section:

```
// Initially set generateRow to true for each input row.
generateRow = true;

// Initially set isRoot to false for each input row.
isRoot = false;
```

```
        // Check if input employee id and name is null.
        if (isNull ("EMP_ID") || isNull ("EMP_NAME"))
        {
            incrementErrorCount(1);

            // If input employee id and/or name is null, don't generate a output row for this
            // input row.
            generateRow = false;

        } else {

            // Set the output field values.
            EMP_ID_OUT = EMP_ID;
            EMP_NAME_OUT = EMP_NAME;
        }

        if (isNull ("EMP_DESC"))
        {
            setNull("EMP_DESC_OUT");
        } else {
            EMP_DESC_OUT = EMP_DESC;
        }

        boolean isParentEmpIdNull = isNull("EMP_PARENT_EMPID");

        if(isParentEmpIdNull)
        {
            // This employee is the root for the hierarchy.
            isRoot = true;
            logInfo("This is the root for this hierarchy.");
            setNull("EMP_PARENT_EMPNAME");
        }

        synchronized(lock)

        {
            // If the employee is not the root for this hierarchy, get the corresponding
            // parent ID.
            if(!isParentEmpIdNull)
            EMP_PARENT_EMPNAME = (String) (empMap.get(new Integer (EMP_PARENT_EMPID)));

            // Add employee to the map for future reference.
            empMap.put (new Integer(EMP_ID), EMP_NAME);
        }

        // Generate row if generateRow is true.
        if(generateRow)
            generateRow();
```

Compile the code and run the mapping

To compile the Java code for the transformation, click **Compile** in the Java editor. After you compile the code successfully, you can run the mapping.

The compilation results displays the status of the compilation. If the Java code does not compile successfully, correct the errors in the Java editor and recompile the Java code. After you successfully compile the transformation, save and run the mapping.

To run the mapping, click **Run** in the Mapping Designer.

The target file contains the following results:

```
"EMP_ID","EMP_NAME","EMP_DESC","EMP_PARENT_EMPNAME"
1,"James Davis","CEO",
4,"Elaine Masters","Vice President - Sales","James Davis"
5,"Naresh Thiagarajan","Vice President - HR","James Davis"
6,"Jeanne Williams","Vice President - Software","James Davis"
9,"Geetha Manjunath","Senior HR Manager","Naresh Thiagarajan"
```

```
10,"Dan Thomas","Senior Software Manager","Jeanne Williams"
14,"Shankar Rahul","Senior Software Manager","Jeanne Williams"
20,"Juan Cardenas","Technical Lead","Dan Thomas"
21,"Pramodh Rahman","Lead Engineer","Shankar Rahul"
22,"Sandra Patterson","Software Engineer","Dan Thomas"
23,"Tom Kelly","Lead Engineer","Dan Thomas"
35,"Betty Johnson","Lead Engineer","Shankar Rahul"
50,"Dave Chu","Software Engineer","Tom Kelly"
70,"Srihari Giran","Software Engineer","Betty Johnson"
71,"Frank Smalls","Software Engineer","Betty Johnson"
```

CHAPTER 19

Java transformation API reference

When you create a Java transformation, you can add API methods to the Java code snippets to define the transformation behavior.

To add an API method to a code snippet, click the **APIs** tab in the Java editor, select the method that you want to add, and click **Add**. Alternatively, you can manually enter the API method in the Java code snippet.

You can add the following API methods to the Java code snippets in a Java transformation:

failSession

Throws an exception with an error message and fails the session.

generateRow

Generates an output row for active Java transformations.

getInRowType

Returns the input type of the current row in the transformation.

incrementErrorCount

Increments the error count for the session.

invokeJExpression

Invokes an expression and returns the value for the expression. Use only in advanced mode..

isNull

Checks for a null value in an input column.

logError

Writes an error message to the session log.

logInfo

Writes an informational message to the session log.

setNull

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

setOutRowType

Sets the update strategy for output rows. Can flag rows for insert, update, or delete.

failSession

Throws an exception with an error message and fails the session. Use failSession to terminate the session.

Use failSession in any section of the Java editor except Import Packages. Do not use failSession in a try/catch block in the Java editor.

Use the following syntax:

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

Argument	Data Type	Input/Output	Description
errorMessage	String	Input	Error message string.

Use the following Java code to test the input field input1 for a null value and fail the session if input1 is NULL:

```
if(isNull("input1")) {
      failSession("Cannot process a null value for field input1.");
}
```

generateRow

Generates an output row for active Java transformations.

When you call generateRow, the Java transformation generates an output row using the current value of the output field variables. If you want to generate multiple rows corresponding to an input row, you can call generateRow more than once for each input row. If you do not use generateRow in an active Java transformation, the transformation does not generate output rows.

Use generateRow in any section of the Java editor except Import Packages. You can use generateRow with active transformations only. If you use generateRow in a passive transformation, the session generates an error.

Use the following syntax:

```
generateRow();
```

Use the following Java code to generate one output row, modify the values of the output fields, and generate another output row:

```
// Generate multiple rows.
if(!isNull("input1") && !isNull("input2"))
{
     output1 = input1 + input2;
     output2 = input1 - input2;
}
generateRow();

// Generate another row with modified values.
output1 = output1 * 2;
output2 = output2 * 2;
generateRow();
```

getInRowType

Returns the input type of the current row in the transformation. The method returns a value of insert, update, delete, or reject.

You can only use getInRowType in the On Input Row section of the Java editor. You can only use the getInRowType method in active transformations that are configured to set the update strategy. If you use this method in an active transformation that is not configured to set the update strategy, the session generates an error.

Use the following syntax:

```
rowType getInRowType();
```

Argument	Data Type	Input/Output	Description
rowType	String	Output	Update strategy type. Value can be INSERT, UPDATE, DELETE, or REJECT.

Use the following Java code to propagate the input type of the current row if the row type is UPDATE or INSERT and the value of the input field input1 is less than 100 or set the output type as DELETE if the value of input1 is greater than 100:

```
// Set the value of the output field.
output1 = input1;

// Get and set the row type.
String rowType = getInRowType();
setOutRowType(rowType);

// Set row type to DELETE if the output field value is > 100.
if(input1 > 100
    setOutRowType(DELETE);
```

incrementErrorCount

Increases the error count for the session. If the error count reaches the error threshold for the session, the session fails.

Use incrementErrorCount in any section of the Java editor except Import Packages.

Use the following syntax:

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

Argument	Data Type	Input/Output	Description
nErrors	Integer	Input	Number of errors to increment the error count for the session.

Use the following Java code to increment the error count if an input field for a transformation has a null value:

```
// Check whether input employee ID or name is null.
if (isNull ("EMP_ID_INP") || isNull ("EMP_NAME_INP"))
{
    incrementErrorCount(1);

    // If input employee ID or name is null, don't generate an output row for this
```

```
input row.
    generateRow = false;
}
```

invokeJExpression

Invokes an expression and returns the value for the expression. Use only in advanced mode.

Use invokeJExpression in any section of the Java editor except Import Packages and Helper Code.

Use the following syntax:

```
(dataType)invokeJExpression(
                String expression,
                Object[] paramMetadataArray);
```

The following table describes the arguments:

Argument	Data Type	Input/Output	Description
dataType	-	Output	Data type that you want to cast the return value to. By default, the return data type is an object. You can cast the return value to an integer, double, string, or byte[] data type.
expression	String	Input	String that represents the expression to invoke. You must use the letter "x" and number the parameters consecutively. For example, if the invoked expression requires three parameters, name the parameters x1, x2, and x3.
paramMetadataArray	Object[]	Input	Array of objects that contains the input parameters for the invoked expression.

Use the following Java code to invoke the concat() method to concatenate the strings John and Smith:

```
(String)invokeJExpression("concat(x1,x2)", new Object [] { "John ", "Smith" });
```

The code returns the following string:

```
John Smith
```

Consider the following rules and guidelines for the invokeJExpression method:

- By default, the update strategy for return values is INSERT. To use a different update strategy, you must define the update strategy in the Java code.

- If an argument, parameter, or return value is NULL, the value is treated as a null indicator.

  For example, if the return value of the invoked expression is NULL and the return data type is a string, the invokeJExpression method returns a string with a value of NULL.

- If an input parameter to the invoked expression is a date/time data type, you must pass the parameter as a string and use the TO_DATE function to convert the string to a date/time data type.

  For example, use the following argument to pass a date/time value to the invoked expression:

```
new Object [] { "TO_DATE("01/22/98", "MM/DD/YY")" }
```

- If the invokeJExpression method returns a date/time data type, you must cast the return value to a string.

isNull

Checks the value of an input column for a null value. Use isNull to check if data of an input column is NULL before using the column as a value.

Use isNull in the On Input Row section of the Java editor.

Use the following syntax:

```
Boolean isNull(String satrColName);
```

Argument	Data Type	Input/Output	Description
strColName	String	Input	Name of an input column.

Use the following Java code to check the value of the SALARY input column before adding it to the output field TOTAL_SALARIES:

```
// If value of SALARY is not null
if (!isNull("SALARY")) {

    // Add to TOTAL_SALARIES.
    TOTAL_SALARIES += SALARY;
}
```

or

```
// If value of SALARY is not null
String strColName = "SALARY";
if (!isNull(strColName)) {

    // Add to TOTAL_SALARIES.
    TOTAL_SALARIES += SALARY;
}
```

logError

Writes an error message to the session log.

Use logError in any section of the Java editor except Import Packages.

Use the following syntax:

```
logError(String errorMessage);
```

Argument	Data Type	Input/Output	Description
errorMessage	String	Input	Error message string.

Use the following Java code to log an error if the input field is null:

```
// Log an error if BASE_SALARY is null.
if (isNull("BASE_SALARY")) {
```

```
        logError("Cannot process a null salary field.");
    }
```

The following message appears in the message log:

```
[JTX_1013] [ERROR] Cannot process a null salary field.
```

logInfo

Writes an informational message to the session log.

Use logInfo in any section of the Java editor except Import Packages.

Use the following syntax:

```
logInfo(String logMessage);
```

Argument	Data Type	Input/Output	Description
logMessage	String	Input	Information message string.

Use the following Java code to write a message to the session log after the Java transformation processes a message threshold of 1000 rows:

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

The following message appears in the session log:

```
[JTX_1012] [INFO] Processed 1000 rows.
```

setNull

Sets the value of an output column in an active or passive Java transformation to NULL. Once you set an output column to NULL, you cannot modify the value until you generate an output row.

Use setNull in any section of the Java editor except Import Packages.

Use the following syntax:

```
setNull(String strColName);
```

Argument	Data Type	Input/Output	Description
strColName	String	Input	Name of an output column.

Use the following Java code to check the value of an input column and set the corresponding value of an output column to null:

```
// Check the value of Q3RESULTS input column.
if(isNull("Q3RESULTS")) {

    // Set the value of output column to null.
```

```
        setNull("RESULTS");
    }
```

or

```
    // Check the value of Q3RESULTS input column.
    String strColName = "Q3RESULTS";
    if(isNull(strColName)) {

        // Set the value of output column to null.
        setNull(strColName);
    }
```

setOutRowType

Sets the update strategy for output rows. The setOutRowType method can flag rows for insert, update, or delete.

Use setOutRowType in the On Input Row section of the Java editor. You can use setOutRowType in active transformations that are configured to set the update strategy. If you use setOutRowType in an active transformation that is not configured to set the update strategy, the session generates an error and the session fails.

Use the following syntax:

```
    setOutRowType(String rowType);
```

Argument	Data Type	Input/ Output	Description
rowType	String	Input	Update strategy type. Value can be INSERT, UPDATE, or DELETE.

Use the following Java code to propagate the input type of the current row if the row type is UPDATE or INSERT and the value of the input field input1 is less than 100, or set the output type as DELETE if the value of input1 is greater than 100:

```
    // Set the value of the output field.
    output1 = input1;

    // Get and set the row type.
    String rowType = getInRowType();
    setOutRowType(rowType);

    // Set row type to DELETE if the output field value is > 100.
    if(input1 > 100)
        setOutRowType(DELETE);
```

CHAPTER 20

Joiner transformation

The Joiner transformation can join data from two related heterogeneous sources. For example, you can use the Joiner transformation to join account information from flat files with data from the Salesforce Account object.

The Joiner transformation joins data based on the join conditions and the join type. A join condition matches fields between the two sources. You can create multiple join conditions. A join type defines the set of data that is included in the results.

When you link a transformation to the Joiner transformation, you connect it to the Master or Detail group. To improve job performance, connect the transformation that represents the smaller data set to the Master group.

To join more than two sources in a mapping, you can use multiple Joiner transformations. You can join the output from the Joiner transformation with another source pipeline. You can add Joiner transformations to the mapping until you join all source pipelines.

Field name conflicts can occur when you join sources with matching field names. You can resolve the conflict in one of the following ways:

- Create a field name conflict resolution.
- Rename matching fields in an upstream transformation.
- Pass data through an Expression transformation to rename fields.

Join condition

The join condition defines when incoming rows are joined. It includes fields from both sources that must match to join source rows.

You define one or more conditions based on equality between the master and detail data. For example, if two sets of employee data contain employee ID numbers, the following condition matches rows with the same employee IDs in both sets of data:

    EMP_ID1 = EMP_ID2

Use one or more join conditions. Additional join conditions increase the time necessary to join the data. When you use multiple join conditions, the mapping task evaluates the conditions in the order that you specify.

Both fields in a condition must have the same data type. If you need to use two fields with non-matching data types, convert the data types so they match.

For example, when you try to join Char and Varchar data, any spaces that pad Char values are included as part of the string. Both fields might include the value "Shoes," but because the Char(40) field includes 35

trailing spaces, the values do not match. To ensure that the values match, change the data type of one field to match the other.

In advanced mode, the join condition can't contain a binary data type or evaluate to a binary data type.

Note: The Joiner transformation does not match null values. To join rows with null values, you can replace null values with default values, and then join on the default values.

Join type

The join type determines the result set that passes to the rest of the mapping.

You can use the following join types:

Normal Join

Includes rows with matching join conditions. Discards rows that do not match the join conditions.

Master Outer

Includes all rows from the detail pipeline and the matching rows from the master pipeline. It discards the unmatched rows from the master pipeline.

Detail Outer

Includes all rows from the master pipeline and the matching rows from the detail pipeline. It discards the unmatched rows from the detail pipeline.

Full Outer

Includes rows with matching join conditions and all incoming data from the master pipeline and detail pipeline.

Advanced properties

You can configure advanced properties for a Joiner transformation. The advanced properties control settings such as the tracing level for session log messages, cache settings, null ordering, and whether the transformation is optional or required.

The properties that are available vary based on the mapping mode.

You can configure the following properties:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Cache Directory	Specifies the directory used to cache master or detail rows and the index to these rows.
	By default, Data Integration uses the directory entered in the Secure Agent $PMCacheDir property for the Data Integration Server. If you enter a new directory, make sure that the directory exists and contains enough disk space for the cache files. The directory can be on a mapped or mounted drive.

Property	Description
Null Ordering in Master	Null ordering in the master pipeline. Select Null is Highest Value or Null is Lowest Value.
Null Ordering in Detail	Null ordering in the detail pipeline. Select Null is Highest Value or Null is Lowest Value.
Data Cache Size	Data cache size for the transformation. Select one of the following options: - Auto. Data Integration sets the cache size automatically. If you select Auto, you can also configure a maximum amount of memory for Data Integration to allocate to the cache. - Value. Enter the cache size in bytes. Default is Auto.
Index Cache Size	Index cache size for the transformation. Select one of the following options: - Auto. Data Integration sets the cache size automatically. If you select Auto, you can also configure a maximum amount of memory for Data Integration to allocate to the cache. - Value. Enter the cache size in bytes. Default is Auto.
Sorted Input	Specifies that data is sorted. Select this option to join sorted data, which can improve performance.
Master Sort Order	Specifies the sort order of the master source data. Select Ascending if the master source data is in ascending order. If you select Ascending, enable sorted input. Default is Auto.
Transformation Scope	Specifies how Data Integration applies the transformation logic to incoming data: - Transaction. Applies the transformation logic to all rows in a transaction. Choose Transaction when a row of data depends on all rows in the same transaction, but does not depend on rows in other transactions. - All Input. Applies the transformation logic on all incoming data. When you choose All Input, Data Integration drops incoming transaction boundaries. Choose All Input when a row of data depends on all rows in the source. - Row. Applies the transformation logic to one row of data at-a-time. Choose Row when a row of data does not depend on any other row.
Optional	Determines whether the transformation is optional. If a transformation is optional and there are no incoming fields, the mapping task can run and the data can go through another branch in the data flow. If a transformation is required and there are no incoming fields, the task fails. For example, you configure a parameter for the source connection. In one branch of the data flow, you add a transformation with a field rule so that only Date/Time data enters the transformation, and you specify that the transformation is optional. When you configure the mapping task, you select a source that does not have Date/Time data. The mapping task ignores the branch with the optional transformation, and the data flow continues through another branch of the mapping.

Hierarchical data in advanced mode

In advanced mode, you can use hierarchical fields that represent an array, map, or struct.

You can use hierarchical fields as pass-through fields.

Note: You cannot use a hierarchical field in a join condition.

Creating a Joiner transformation

Use a Joiner transformation to join data from two related heterogenous sources.

Before you create a Joiner transformation, add Source transformations to the mapping to represent source data. Include any other upstream transformations that you want to use.

If the data in the two pipelines include matching field names, rename one set of fields in a transformation upstream from the Joiner transformation.

1. In the **Transformation palette**, drag a Joiner transformation onto the mapping canvas.

2. Connect an upstream transformation that represents one data set to the Master group of the Joiner transformation.

    To improve job performance, use the transformation that represents the smaller data set.

3. Connect an upstream transformation that represents the other data set to the Detail group of the Joiner transformation.

4. On the **General** tab, enter a name and optional description for the transformation.

5. On the **Incoming Fields** tab, configure the field rules that define the data that enters the transformation.

6. On the **Join Condition** tab, select the join yype.

7. To configure a join condition, select Simple for the join condition. Click **Add New Join Condition**, and then select the master and detail fields to use and the operator. You can create multiple join conditions.

    Alternatively, to use a parameter for the join condition, select Completely Parameterized for the join condition.

You can add downstream transformations to the mapping and configure them. When the mapping is complete, you can validate and save the mapping.

Joiner transformation example

You're a marketing manager for an online retailer, and you want to merge order data with product and customer data from different Amazon S3 sources to understand what customers are purchasing. Use Joiner transformations to join the data from your sources.

You have three source data tables in an Amazon S3 bucket: *orders*, *order_items*, and *customer_data*. The following image shows a mapping that joins the data from these sources:
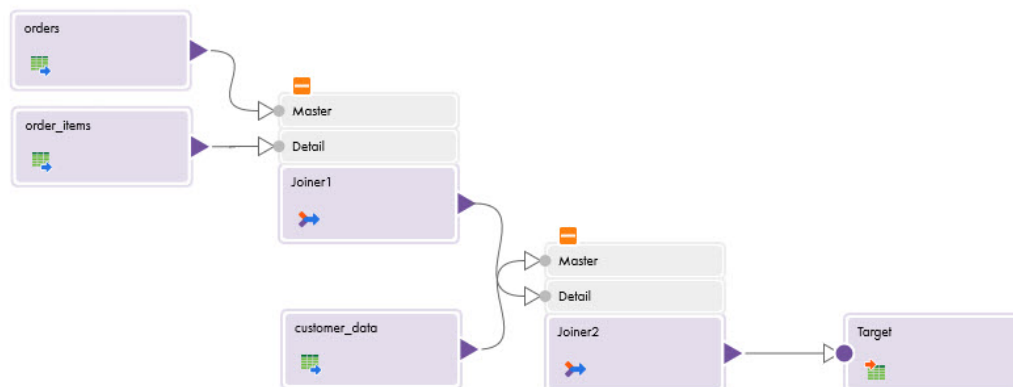
The mapping contains the following elements:

Source transformation for *orders*

The *orders* data table includes fields for the order number, date, price, and ID of the customer for each online order.

The following table shows a portion of *orders*:

order_id	order_date	customer_id	order_price
1005	2023-01-20	789	78.25
1006	2023-01-24	268	150.09
1007	2023-02-07	268	30.20

Source transformation for *order_items*

The *order_items* data table includes details about the items in each order, including the quantity and price.

The following table shows a portion of *order_items*:

order_id	item_id	qty	price
1005	5063	2	34.99
1006	2389	3	19.99
1006	5063	1	34.99
1007	9871	2	10.99

In the Source transformation, you rename the field `order_id` to `items_order_id` to avoid a field name conflict when you join *order_items* with *orders*.

Source transformation for *customer_data*

The *customer_data* table includes fields for information that the customers provide, including their name, date of birth, and phone number.

The following table shows a portion of *customer_data*:

c_id	c_name	c_dob
789	Kelcy Almeida	1969-07-20
268	Chidi Donalds	1972-12-07

Joiner transformation for *orders* and *order_items*

The first Joiner transformation performs a normal join between *orders* and *order_items*. The *orders* Source transformation is the master group and the *order_items* Source transformation is the detail group so that order information is added to each item ordered.

The Joiner transformation uses the following join condition to match the data by the order ID: `order_id = items_order_id`.

Joiner transformation for *customer_data*

The second Joiner transformation performs a detail outer join between *customer_data* and the output from the first Joiner transformation. The transformation uses the *customer_data* Source transformation as the master group since it is the smaller data set.

The second Joiner transformation uses the following join condition to match the data by the customer ID: `customer_id = c_id`.

Target transformation

The Target transformation writes the data to a new file in Amazon S3. You can configure the incoming fields to exclude the duplicate fields that result from the joins.

The following table shows a portion of the output data:

order_id	order_date	order_price	item_id	qty	price	c_id	c_name	c_dob
1005	2023-01-20	78.25	5063	2	34.99	789	Kelcy Almeida	1969-07-20
1006	2023-01-24	150.09	2389	3	19.99	268	Chidi Donalds	1972-12-07
1006	2023-01-24	150.09	5063	1	34.99	268	Chidi Donalds	1972-12-07
1007	2023-02-07	30.20	9871	2	10.99	268	Chidi Donalds	1972-12-07

CHAPTER 21

Labeler transformation

The Labeler transformation adds a labeler asset that you created in Data Quality to a mapping.

A labeler asset defines a set of operations that evaluates the types of information in an input field and assigns a label to each type of data that it finds.

When you configure the transformation, you map an input field to a target field that links to the labeler asset. When the mapping runs, the transformation searches the input field for values that match the labeling criteria that the asset defines. The transformation writes the results of the labeling operation to two output fields. For more information about the output fields, see "Labeler transformation output fields" on page 290.

A Labeler transformation is similar to a Mapplet transformation, as it allows you to add data transformation logic that you designed elsewhere to a mapping. Like mapplets, labeler assets are reusable assets.

A Labeler transformation shows incoming and outgoing fields. It does not display the logic that the labeler asset contains or allow you to edit the labeler asset. To edit the labeler asset, open it in Data Quality.

Labeler transformation configuration

When you configure a Labeler transformation in a mapping, you first select the labeler asset that contains the logic to include in the transformation. Next, you map an incoming field on the transformation to the target field that labeler asset specifies.

To configure the transformation, complete the following tasks:

1. Connect the Labeler transformation to a Source transformation or other upstream object.
2. On the **Labeler** tab, select the labeler asset that you want to include in the transformation.

The following image shows the options that you use to select the labeler asset:
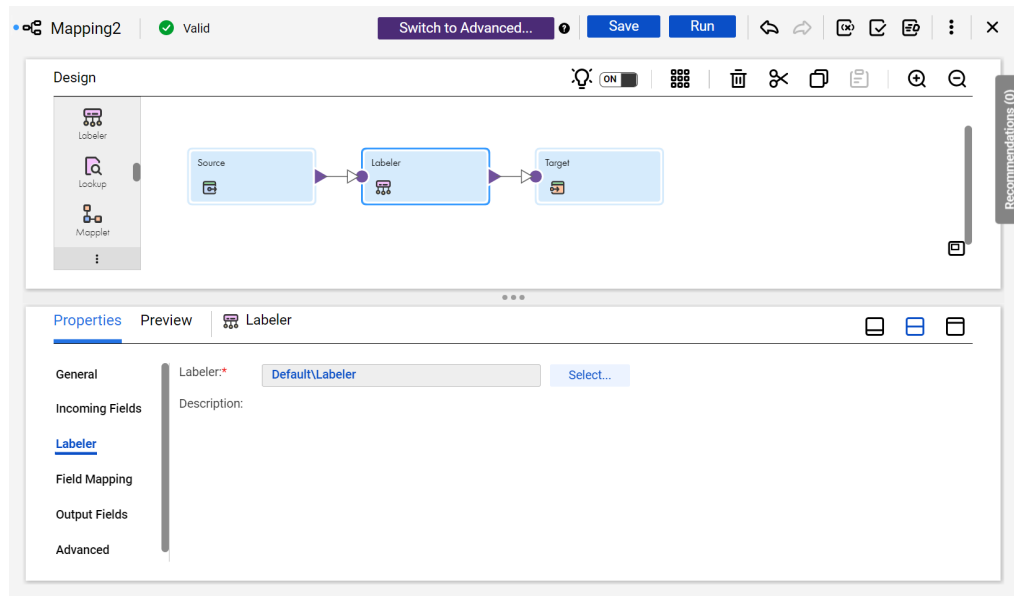
3. On the **Incoming Fields** tab, verify the incoming fields.

   By default, the transformation inherits all incoming fields from any connected upstream object in the mapping. You can define a field rule to limit or rename the incoming fields.

4. On the **Field Mapping** tab, connect a field from an upstream object to the target field.

   The labeler asset field name might reflect the name of an input field. If so, you can use the **Automap** options to connect the fields.

5. Verify the output field properties on the **Output Fields** tab.

   To learn more about the transformation output fields, see "Labeler transformation output fields" on page 290.

6. You can optionally rename the Labeler transformation and add a description on the **General** tab.

Synchronizing data quality assets

If you update an asset in Data Quality after you add it to a transformation, you may need to synchronize the asset version in the transformation with the latest version.

To synchronize the asset versions, open the transformation in the mapping and select the transformation name in the properties panel. For example, in a Cleanse transformation select **Cleanse** in the properties panel. If synchronization is necessary, Data Integration displays a message that prompts you to synchronize the assets.

Labeler transformation field mappings

Configure field mappings to define how data moves from the upstream transformation to the Labeler transformation. Configure field mappings on the **Field Mapping** tab of the **Properties** panel.

Note: You map an incoming field to a single field on a labeler asset.

You can configure the following field mapping options:

Field map options

Method of mapping fields to the transformation.

Select one of the following options:

- Manual. Manually link an incoming field to a transformation input field. Removes links for any automatically mapped field.

- Automatic. Automatically link fields with the same name. Use when all of the fields that you want to link share the same name. You cannot manually link fields with this option.

- Completely Parameterized. Use a parameter to represent the field mapping.
Choose the Completely Parameterized option when the labeler asset in the transformation is parameterized or any upstream transformation in the mapping is parameterized.

- Partially Parameterized. Configure links in the mapping that you want to enforce and use a parameter to allow other fields to be mapped in the mapping task. Or, use a parameter to configure links in the mapping, and allow all fields and links to display in the task for configuration.

Parameter

Select the parameter to use for the field mapping, or create a new parameter. This option appears when you select Completely Parameterized or Partially Parameterized as the field map option. The parameter must be of type *field mapping*.

Do not use the same field mapping parameter in more than one Labeler transformation in a single mapping.

Options

Controls how fields are displayed in the **Incoming Fields** and **Target Fields** lists.

Configure the following options:

- The fields that appear. You can show all fields, unmapped fields, or mapped fields.

- Field names. You can use technical field names or labels.

Automap

Links fields with matching names. Allows you to link matching fields and to manually configure other field mappings. The Automap options appear when you select the Manual or Partially Parameterized field map option.

You can map fields in the following ways:

- Exact Field Name. Data Integration matches fields of the same name.

- Smart Map. Data Integration matches fields with similar names. For example, if you have an incoming field `Cust_Name` and a target field `Customer_Name`, Data Integration automatically links the `Cust_Name` field with the `Customer_Name` field.

You can use both Exact Field Name and Smart Map in the same field mapping. For example, use Exact Field Name to match fields with the same name and then use Smart Map to map fields with similar names.

You can undo all automapped field mappings by clicking **Automap** > **Undo Automap**.

To unmap a single field, select the field to unmap and click **Actions** > **Unmap** on the context menu for the field. To unmap one or more fields that you selected, click **Unmap Selected** on the Target Fields context menu.

To clear all field mappings from the transformation, click **Clear Mapping** on the Target Fields context menu.

Labeler transformation output fields

A Labeler transformation creates one or more output fields to contain the outcome of the labeling operation. The number of fields depends on whether the labeler asset in the transformation specifies a token labeling or character labeling operation. View the output fields on the **Output Fields** tab of the **Properties** panel. The tab displays the name, type, precision, and scale of the output fields.

Token labeling output

The Labeler transformation creates the following output fields in token labeling mode:

LabeledOutput

> A copy of the input field data in which any value that matches the labeling criteria is replaced with the label that the asset specifies.

TokenizedData

> A copy of the input field data. If the labeler asset reads a dictionary, the transformation may replace any input value that matches a dictionary value with an alternative value from the dictionary. A Data Quality user can configure the labeler asset to replace the input values with the alternative values.

The transformation copies any value in the input field that the labeling operation does not identify to each output field.

Character labeling output

The Labeler transformation creates the following output field in character labeling mode:

LabeledOutput

> A copy of the input field data in which any character that matches the labeling criteria is replaced with the label that the asset specifies.
>
> The field can contain character labels and characters from the input data to which a label does not apply.

CHAPTER 22

Lookup transformation

Use a Lookup transformation to retrieve data based on a specified lookup condition. For example, you can use a Lookup transformation to retrieve values from a database table for codes used in source data.

When a mapping task includes a Lookup transformation, the task queries the lookup source based on the lookup fields and a lookup condition. The Lookup transformation returns the result of the lookup to the target or another transformation. You can configure the Lookup transformation to return a single row or multiple rows. When you configure the Lookup transformation to return a single row, the Lookup transformation is a passive transformation. When you configure the Lookup transformation to return multiple rows, the Lookup transformation is an active transformation. You can use multiple Lookup transformations in a mapping.

Perform the following tasks with a Lookup transformation:

- Get a related value. Retrieve a value from the lookup table based on a value in the source. For example, the source has an employee ID. Retrieve the employee name from the lookup table.
- Get multiple values. Retrieve multiple rows from a lookup table. For example, return all employees in a department.
- Update slowly changing dimension tables. Determine whether rows exist in a target.

You can perform the following types of lookups:

Connected or unconnected lookup

A connected Lookup transformation receives source data, performs a lookup, and returns data.

An unconnected Lookup transformation is not connected to a source or target. A transformation calls the Lookup transformation with a lookup expression. The unconnected Lookup transformation returns one column to the calling transformation.

Cached or uncached lookup

Cache the lookup source to optimize performance. If you cache the lookup source, you can use a static or dynamic cache. You can also use a persistent or non-persistent cache.

By default, the lookup cache remains static and does not change as the mapping task runs. With a dynamic cache, the task inserts or updates rows in the cache as the target table changes. When you cache the target table as the lookup source, you can look up values in the cache to determine if the values exist in the target.

By default, the lookup cache is also non-persistent. Therefore, Data Integration deletes the cache files after the mapping task completes. If the lookup table does not change between mapping runs, you can use a persistent cache to increase performance.

Lookup object

The Lookup object is the source object that Data Integration queries when it performs the lookup. The lookup object is also called the lookup source.

Select the lookup source on the **Lookup Object** tab of the Properties panel. The properties that you configure for the lookup source vary based on the connection type

The following image shows the **Lookup Object** tab for a relational lookup:
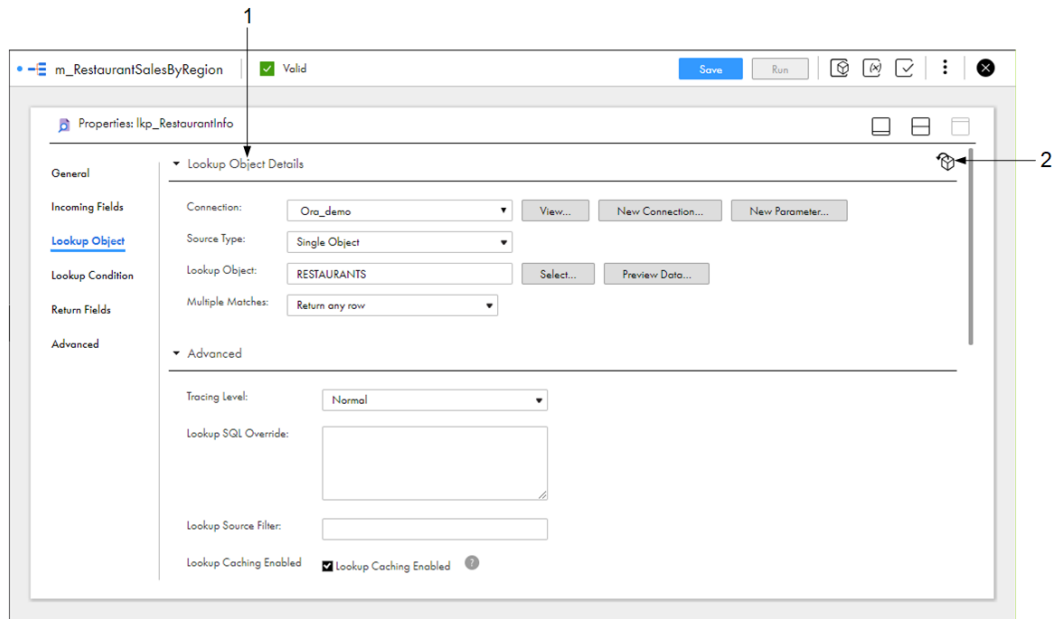
1. Lookup object details where you configure the connection, source type, lookup object, and multiple match behavior.
2. Select the lookup source from the mapping inventory.

You can select the lookup source in the following ways:

Select the connection and lookup object.

In the **Lookup Object Details** area, select the connection, source type, and lookup object. You can also create a new connection.

Select the lookup source from the mapping inventory.

If your organization administrator has configured Enterprise Data Catalog integration properties, and you have added objects to the mapping from the **Data Catalog** page, you can select the lookup source from the **Inventory** panel. If your organization administrator has not configured Enterprise Data Catalog integration properties or you have not performed data catalog discovery, the **Inventory** panel is empty. For more information about data catalog discovery, see *Mappings*.

Use a parameter.

You can use an input parameter to define the connection or lookup object when you run the mapping task. For more information about parameters, see *Mappings*.

Use a custom query.

You can use a custom query to reduce the number of columns to query. You might want to use a custom query when the source object is large.

You must also specify the transformation behavior when the lookup returns multiple matches.

Lookup object properties

When you configure a lookup, you select the lookup connection and lookup object. You also define the behavior when a lookup condition returns more than one match.

The following table describes the lookup object properties:

Property	Description
Connection	Name of the lookup connection.
Source Type	Source type. For database lookups, the source type can be single object, parameter, or query. For flat file lookups, the source type can be single object, file list, command, or parameter.
Lookup Object	If the source type is a single object, this property specifies the lookup file, table, or object. If the source property is a file list, this property specifies the text file that contains the file list. If the source type is a command, this property specifies the sample file from which Data Integration imports the return fields.
Parameter	If the source type is a parameter, this property specifies the parameter.
Define Query	If the source type is a query, displays the **Edit Custom Query** dialog box. Enter a valid custom query and click **OK**.
Multiple Matches	Behavior when the lookup condition returns multiple matches. You can return all rows, any row, the first row, the last row, or an error. If you choose all rows and there are multiple matches, the Lookup transformation is an active transformation. If you choose any row, the first row, or the last row and there are multiple matches, the Lookup transformation is a passive transformation.
Formatting Options	File formatting options which are applicable if the lookup object is a flat file. Opens the **Formatting Options** dialog box to define the format of the file. Configure the following file format options: - Delimiter. Delimiter character. - Treat multiple characters as a single delimiter. Treats the specified set of delimiters as one delimiter. For example, a source file contains the following record: abc~def|ghi~|~|jkl|~mno. If you specify the delimiter as (~ |), Data Integration reads the record as three columns separated by two delimiters: abc~def|ghi, NULL, jkl|~mno. If you disable this option, Data Integration reads the record as nine columns separated by eight delimiters: abc, def, ghi, NULL, NULL, NULL, jkl, NULL, mno. - Treat consecutive delimiters as one. Treats one or more consecutive column delimiters as one. The default is to treat consecutive delimiters as a null value. - Row Delimiter. Line break character. Select a line break character from the list. The default is line-feed, \012 LF (\n). - Text Qualifier. Character to qualify text. - Escape character. Escape character. - Field labels. Determines if the task generates field labels or imports labels from the source file. - First data row. The first row of data. The task starts the read at the row number that you enter.
Command	If the source type is a command, this property specifies the command that generates the file list.

For more information about file lists and commands, see "File lists" on page 45. For more information about parameters and file formats, see *Mappings*.

Multiple match policy restrictions

When you configure a lookup, you define the behavior when a lookup condition returns more than one match. Some types of lookups have restrictions on the multiple match policy.

The following types of lookups have multiple match policy restrictions:

Uncached lookups

Some connector types do not support the multiple match policies **Return first row** and **Return last row** in uncached lookups. If you select either of these policies, and the connector does not support the policy in uncached lookups, Data Integration enables the **Lookup Caching Enabled** advanced property, and you cannot edit it.

Lookups that use a dynamic cache

If the Lookup transformation uses a dynamic cache, you must configure the multiple match policy to return an error. Other multiple match policies are not supported.

Salesforce lookups

When you perform a lookup against a Salesforce object, you can return any row or return an error.

Lookups in advanced mode

When you use the Lookup transformation in advanced mode, you can return all rows, return any row, or return an error. The multiple match policies **Return first row** and **Return last row** are not supported.

When you define the behavior for multiple matches in advanced mode to return an error, the Lookup transformation drops duplicate rows and does not include the duplicate rows in the log files.

For more information about the multiple match policies supported by different connectors, see the help for the appropriate connector.

Custom queries

You can create a custom query for database lookups. You might create a custom query to reduce the number of columns to query.

To use a custom query as a lookup source, select **Query** as the source type, and then define the query. When you define the query, enter an SQL SELECT statement to select the source columns that you want to use and specify a FROM clause. Data Integration uses the SQL statement to retrieve source column information.

When you use a custom query in a lookup transformation, use the following format for the SQL statement:

- For a relational database connection, use an alias for each column in the SQL statement, for example:

      SELECT COL1 AS COL1, COL2 AS COL2, COL3 AS COL3 from TABLE_NAME

- For other types of database connections, use SQL that is valid for the source database. You can use database-specific functions in the query.

To use a custom query as a lookup source, you must enable lookup caching.

When you change a custom query in a saved mapping, at design time Data Integration replaces the field metadata with metadata using the revised query. Typically, this is the desired behavior. However, if the mapping uses a relational source and you want to retain the original metadata, use the **Retain existing field metadata** option, which is available in the following connectors:

- JDBC V2 Connector
- PostgreSQL Connector
- MongoDB V2 Connector

- Couchbase Connector

When you use the **Retain existing field metadata** option, Data Integration doesn't refresh the field metadata during design time. Data Integration maps the existing fields with the fields from the revised query at run time. Fields that can't be mapped will cause run time failure.

Tip: Test the SQL statement you want to use on the source database before you create a custom query. Data Integration does not display specific error messages for invalid SQL statements.

Lookup condition

The lookup condition defines when the lookup returns values from the lookup object. When you configure the lookup condition, you compare the value of one or more fields from the data flow with values in the lookup object.

A lookup condition includes an incoming field from the data flow, a field from the lookup object, and an operator. For flat file and database connections, you can use the following operators in a lookup condition:

= (Equal to)

< (Less than)

> (Greater than)

<= (Less than or equal to)

>= (Greater than or equal to)

!= (Not equal to)

For other connections and for Lookup transformations that use a dynamic cache, you can use the = (Equal to) operator in a lookup condition.

Note the following information about lookup conditions:

- When you enter multiple conditions, the mapping task evaluates the lookup conditions using the AND logical operator to join the conditions. It returns rows that match all of the lookup conditions.

- When you include multiple conditions, to optimize performance enter the conditions in the following order:

  1. = (Equal to)

  2. < (Less than), <= (Less than or equal to), > (Greater than), >= (Greater than or equal to)

  3. != (Not equal to)

- The lookup condition matches null values. When an input field is NULL, the mapping task evaluates the NULL equal to null values in the lookup.

- In advanced mode, the mapping becomes invalid if the lookup condition contains a binary data type.

- If the lookup condition is completely parameterized, then you need to enter the lookup condition in the format:

  ```
  <Lookup field><Operator><Incoming field>
  ```

  If multiple conditions are required, join them with AND.

Lookup return fields

Select the fields to return from the lookup object on the **Return Fields** tab of the Properties panel.

The **Return Fields** tab displays all fields from the selected lookup object. By default, the mapping includes all fields in the list in the data flow. Remove fields that you do not want to use.

For Lookup transformations that use a dynamic cache, the task returns the NewLookupRow return field. You cannot remove this field. For more information about the NewLookupRow return field, see "Dynamic cache updates" on page 304.

You can edit the name of a field and edit the metadata for a field. When you edit field metadata, you can change the name, native data type, native precision, and native scale.

For some relational connection types, you can specify the default column value for lookup return fields. You can use a string or an expression for the default value. If you use a string, enclose the string in single quotes, for example, 'ABC'. To see if you can set default values for lookup return fields, see the help for the appropriate connector.

Note: When you change field metadata, you cannot automatically revert your changes. Avoid making changes that can cause errors when you run the task.

You can add a field to the field list if the field exists in the lookup object. To add a field, you need exact field details, including the field name, data type, precision, and scale.

To restore the original fields from the lookup object, use the Synchronize icon. Synchronization restores deleted fields, adds new fields, and retains added fields with corresponding fields in the lookup object. Synchronization removes any added fields that do not have corresponding fields in the lookup object. Synchronization does not revert any local changes to the field metadata.

The following table describes the options that you can use on the **Return Fields** tab:

Field option	Description
Add Field icon	Adds a field from the selected lookup object. Use to retrieve a field from the object that does not display in the list. Opens the **New Field** dialog box. Enter the exact field name, data type, precision, and scale, and click **OK**.
Delete icon	Deletes the field from the list, removing the field from the data flow.
Sort icon	Sorts fields in native order, ascending order, or descending order.
Find field	Enter a search string to find the fields with names that contain the string.
Options menu	Contains the following options: - Use Technical Field Names. Displays field names by technical field name. - Use Labels. Displays field names by label. - Edit Metadata. Change the name, native type, native precision, or native scale, if applicable for the data type. For some relational connection types, you can set the default value for lookup return fields. When you edit metadata, you can display native names by technical field name or label.
Synchronize icon	Synchronizes the list of fields with the lookup object. **Note:** If you select this option, you lose any changes you make to the metadata for return fields.

Field option	Description
Ignore in Comparison	When the transformation uses a dynamic cache, by default, Data Integration compares the values in all lookup fields with the values in the associated incoming fields to determine whether to update the row in the lookup cache. Enable this property if you want Data Integration to ignore the field when it compares the values before updating a row. You must configure the transformation to compare at least one field. This property is displayed for each field when the Lookup transformation uses a dynamic cache.
Ignore Null Inputs for Updates	When enabled, if the value in the associated incoming field contains a null value, Data Integration doesn't update the column in the cache. This property is displayed for each field except the NewLookupRow field when the Lookup transformation uses a dynamic cache. Default is disabled.
Retain existing fields at runtime	If field metadata changes after a mapping is saved, Data Integration uses the updated field metadata when you run the mapping. Typically, this is the desired behavior. However, if the mapping uses a native flat file connection and you want to retain the metadata used at design time, enable the **Retain existing fields at runtime** option. When you enable this option, Data Integration mapping tasks will use the field metadata that was used when you created the mapping.

Advanced properties

You can configure advanced properties for a Lookup transformation. The connection type and the mapping type determine which advanced properties are available for the Lookup transformation.

The properties that are available vary based on the mapping mode.

You can set the following properties:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Lookup Source File Directory	Name of the directory for a flat file lookup source. By default, Data Integration reads files from the lookup source connection directory. You can also use an input parameter to specify the source file directory. If you use the service process variable directory $PMLookupFileDir, the task writes target files to the configured path for the system variable. To find the configured path of a system variable, see the pmrdtm.cfg file located at the following directory: `<Secure Agent installation directory>\apps\Data_Integration_Server\<Data Integration Server version>\ICS\main\bin\rdtm` You can also find the configured path for the $PMLookupFileDir variable in the Data Integration Server system configuration details in Administrator.
Lookup Source File Name	File name, or file name and path of the lookup source file.

Property	Description
Lookup SQL Override	Overrides the default SQL statement to query the lookup table. Specifies the SQL statement you want to use for querying lookup values.
Lookup Source Filter	Restricts the lookups based on the value of data in any field in the Lookup transformation. Use with lookup cache enabled.
Lookup Caching Enabled	Determines whether to cache lookup data during the runtime session. When you enable caching, Data Integration queries the lookup source once and caches the values for use during the session, which can improve performance. When you disable caching, a SELECT statement gets the lookup values each time a row passes into the transformation. Caching is enabled and is not editable in the following circumstances: - When the lookup source type does not support uncached lookups. - When you select a multiple match policy, but the lookup source type does not support the policy in uncached lookups. For example, you cannot disable caching when you select **Return first row** or **Return last row** as the multiple match policy for a lookup against an Amazon Redshift V2 source. Default is enabled. This property is not displayed for flat file lookups because flat file lookups are always cached.
Use Lookup Field Default Value	Determines whether to return the default value when there's no match in the lookup source. By default, the transformation returns NULL when there's no match in the lookup source. Doesn't apply when you parameterize the lookup object, perform an unconnected lookup, or disable lookup caching.
Lookup Cache Directory Name	Specifies the directory to store cached lookup data when you select Lookup Caching Enabled. The directory name can be an environment variable.
Lookup Cache Persistent	Determines whether to save the lookup cache file to reuse it the next time Data Integration processes a Lookup transformation configured to use the cache.
Cache File Name Prefix	Use with persistent lookup cache. Specifies the file name prefix to use with persistent lookup cache files. Data Integration uses the file name prefix as the file name for the persistent cache files that it saves to disk. If the named persistent cache files exist, Data Integration builds the memory cache from the files. If the named persistent cache files do not exist, Data Integration rebuilds the persistent cache files. Enter the prefix. Do not include a file extension such as .idx or .dat.
Re-cache from Lookup Source	Use with persistent lookup cache. When selected, Data Integration rebuilds the persistent lookup cache from the lookup source when it first calls the Lookup transformation instance.
Data Cache Size	Data cache size for the transformation. Select one of the following options: - Auto. Data Integration sets the cache size automatically. If you select Auto, you can also configure a maximum amount of memory for Data Integration to allocate to the cache. - Value. Enter the cache size in bytes. Default is Auto.
Index Cache Size	Index cache size for the transformation. Select one of the following options: - Auto. Data Integration sets the cache size automatically. If you select Auto, you can also configure a maximum amount of memory for Data Integration to allocate to the cache. - Value. Enter the cache size in bytes. Default is Auto.

Property	Description
Dynamic Lookup Cache	Determines whether to use a dynamic cache instead of a static cache. When you enable dynamic caching, the task updates the cache as it inserts or updates rows in the target so that the cache and target remain in sync. Use when lookup cache is enabled.
Update Dynamic Cache Condition	When the lookup uses a dynamic lookup cache, Data Integration uses the expression condition to determine whether to update the dynamic cache. Data Integration updates the cache when the condition is true and the data exists in the cache. Enter a boolean expression. You can include field names and values, and in-out parameters in the expression. Doesn't apply to mappings in advanced mode. Default is TRUE.
Output Old Value On Update	When you enable this property, when the task updates a row in the cache, it outputs the value that existed in the lookup cache before it updated the row. When it inserts a row, it returns a null value. Use when dynamic lookup cache is enabled.
Synchronize dynamic cache	When you enable this property, the task retrieves the latest values from the lookup source and updates the dynamic cache. This is helpful when multiple tasks that use the same lookup source are running simultaneously. Use when dynamic lookup cache is enabled. Cache synchronization is not available for some connection types. For more information, see the help for the appropriate connector.
Insert Else Update	Applies to rows entering the Lookup transformation with the row type of insert. When enabled, the mapping task inserts rows in the cache and updates existing rows. When disabled, the mapping task does not update existing rows. Use when dynamic lookup cache is enabled.
Lookup Source is Static	When you enable this property, the lookup source does not change when the task runs.
Datetime Format	Sets the datetime format and field width. Milliseconds, microseconds, or nanoseconds formats have a field width of 29. If you do not specify a datetime format here, you can enter any datetime format for fields. Default is YYYY-MM-DD HH24:MI:SS. The format does not change the size of the field. Default is YYYY-MM-DD HH24:MI:SS. The Datetime format does not change the size of the field.
Thousand Separator	Specifies the thousand separator. Enter a comma (,) a period (.) or None. Default is None.
Decimal Separator	Specifies the decimal separator. Enter a comma (,) or a period (.). Default is period.
Case Sensitive String Comparison	Determines whether to enable case-sensitive string comparisons when you perform lookups on string columns in flat files. For relational uncached lookups, the column types that support case-sensitive comparison depend on the database. Case-sensitivity is automatically enabled for lookups in advanced mode.

Property	Description
Null Ordering	Determines how the null values are ordered. You can choose to sort null values high or low. By default, null values are sorted high. This overrides configuration to treat nulls in comparison operators as high, low, or null. For relational lookups, null ordering depends on the database default value.
Sorted Input	Indicates whether or not the lookup file data is in sorted order. This increases lookup performance for file lookups. If you enable sorted input and the condition columns are not grouped, the session fails. If the condition columns are grouped but not sorted, the lookup is processed as if you did not configure sorted input.
Pre-build Lookup Cache	Specifies to build the lookup cache before the Lookup transformation receives data. Multiple lookup cache files can be built at the same time to improve performance.
Subsecond Precision	Sets the subsecond precision for datetime fields. For relational lookups, you can change the precision for databases that have an editable scale for datetime data. You can change the subsecond precision for Oracle Timestamp, Informix Datetime, and Teradata Timestamp data types. Enter a positive integer value from 0 to 9. Default is 6 microseconds. If you enable SQL ELT optimization in a task, the database returns the complete datetime value, regardless of the subsecond precision setting.
Optional	Determines whether the transformation is optional. If a transformation is optional and there are no incoming fields, the mapping task can run and the data can go through another branch in the data flow. If a transformation is required and there are no incoming fields, the task fails. For example, you configure a parameter for the source connection. In one branch of the data flow, you add a transformation with a field rule so that only Date/Time data enters the transformation, and you specify that the transformation is optional. When you configure the mapping task, you select a source that does not have Date/Time data. The mapping task ignores the branch with the optional transformation, and the data flow continues through another branch of the mapping.

Lookup SQL overrides

When a mapping includes a Lookup transformation, the mapping task queries the lookup object based on the fields and properties that you configure in the Lookup transformation. The mapping task runs a default lookup query when the first row of data enters the Lookup transformation. If the Lookup transformation performs a relational lookup, you can override the default query.

You can override the default query for both cached and uncached lookups. To view the default query, run the mapping task. The default query appears in the log file.

Lookup SQL overrides for cached lookups

In a cached lookup, you can override the default SQL query to change the ORDER BY clause, add a WHERE clause, or transform the data. The mapping task uses the override statement to build the lookup cache.

The default query contains a SELECT statement that includes all lookup fields in the mapping. The SELECT statement also contains an ORDER BY clause that orders all columns in the same order in which they appear in the Lookup transformation.

If you want to change the ORDER BY clause, add a WHERE clause, or transform the lookup data before it is cached, override the default query. For example, you might use database functions to adjust the data types or

formats in the lookup table to match the data types and formats of fields that are used in the mapping. Or, you might override the default query to query multiple tables.

Override the default query on the **Advanced** tab of the Lookup transformation. Enter the entire SELECT statement in the **Lookup SQL Override** field. Use an alias for each column in the query. If you want to change the ORDER BY clause, append "--" to the end of the query to suppress the ORDER BY clause that the mapping task generates.

Example

A Lookup transformation returns the following fields from Microsoft SQL Server table ALC_ORDER_DETAILS:

The transformation uses the following lookup condition:

```
ORDERID=in_ORDERID
```

When you run the mapping task, the following query appears in the log file:

```
LKPDP_1> DBG_21097 [2018-11-07 14:11:33.509] Lookup Transformation [lkp_ALC_ORDER_DETAILS]:
Default sql to create lookup cache: SELECT PRODUCTID,UNITPRICE,QUANTITY,DISCOUNT,ORDERID FROM
"icsauto"."ALC_ORDER_DETAILS" ORDER BY ORDERID,PRODUCTID,UNITPRICE,QUANTITY,DISCOUNT
```

To override the ORDER BY clause and sort by PRODUCTID, enter the following query in the **Lookup SQL Override** field on the **Advanced** tab:

```
SELECT PRODUCTID AS PRODUCTID, UNITPRICE AS UNITPRICE, QUANTITY AS QUANTITY, DISCOUNT AS
DISCOUNT, ORDERID AS ORDERID FROM "icsauto"."ALC_ORDER_DETAILS" ORDER BY PRODUCTID --
```

When you run the mapping task again, the following query appears in the log file:

```
LKPDP_1> DBG_21312 [2018-11-07 14:14:36.734] Lookup Transformation [lkp_ALC_ORDER_DETAILS]:
Lookup override sql to create cache: SELECT PRODUCTID AS PRODUCTID, UNITPRICE AS UNITPRICE,
QUANTITY AS QUANTITY, DISCOUNT AS DISCOUNT, ORDERID AS ORDERID FROM
"icsauto"."ALC_ORDER_DETAILS" ORDER BY PRODUCTID -- ORDER BY
ORDERID,PRODUCTID,UNITPRICE,QUANTITY,DISCOUNT
```

Lookup SQL override for uncached lookups

You can override the default SQL query for an uncached lookup including the WHERE and ORDER BY clauses. The mapping task doesn't build a cache from the override statement for an uncached lookup.

When you run the mapping task, the task generates a default SELECT statement that includes the lookup fields, output fields, and WHERE clause based on the lookup condition. If the Lookup transformation is an unconnected lookup, the SELECT statement includes the lookup fields and the return field, but the mapping task doesn't generate the WHERE clause from the lookup condition.

Override the default query on the **Advanced** tab of the Lookup transformation. Enter the entire SELECT statement in the **Lookup SQL Override** field. Use an alias for each column in the query. If you want to change the ORDER BY clause, append " --" to the end of the query to suppress the ORDER BY clause that the mapping task generates.

To reference incoming fields in the WHERE clause, configure parameter binding by adding a "?" before and after the field name. The query data changes based on the value of the data in the field. The following example includes a WHERE statement that references the Name field:

```
SELECT EMPLOYEE.NAME as NAME, max(EMPLOYEE.ID) as ID from EMPLOYEE WHERE EMPLOYEE.NAME=?
NAME1?
```

If you add a function to the SQL statement, the return data type must match the data type of the alias column. For example, the data type of ID matches the return type of the MAX function:

```
SELECT EMPLOYEE.NAME as NAME, MAX(EMPLOYEE.ID) as ID FROM EMPLOYEE
```

Note: You can't use subqueries in the SQL override for uncached lookups.

Guidelines for overriding the lookup query

Certain rules and guidelines apply when you override a lookup query.

Use the following guidelines:

- You can override the lookup SQL query for relational lookups.
- Enter the entire SELECT statement using the syntax that is required by the database.
- Enclose all database reserved words in quotes.
- Include all lookup and return fields in the SELECT statement.

  If you add or subtract fields in the SELECT statement, the mapping task fails.

- Use an alias for each column in the query.

  If you do not use column aliases, the mapping task fails with the following error:

  ```
  Failed to initialize transformation [<Lookup Transformation Name>]
  ```

- The query must contain a FROM clause.
- To override the ORDER BY clause, append "--" at the end of the query.

  The mapping task generates an ORDER BY clause, even when you enter one in the override. Therefore, you must enter two dashes (--) at the end of the query to suppress the generated ORDER BY clause.

- If the ORDER BY clause contains multiple columns, enter the columns in the same order as the fields in the lookup condition.
- If the mapping task uses SQL ELT optimization or is based on a mapping in SQL ELT mode, you can't override the ORDER BY clause or suppress the generated ORDER BY clause with comment notation.
- If multiple Lookup transformations share a lookup cache, use the same lookup SQL override for each Lookup transformation.
- When you configure a Lookup transformation that returns all rows, the mapping task builds the lookup cache with sorted keys. When the transformation retrieves all rows in a lookup, the mapping task builds the data cache with the keys in sorted order. The mapping task cannot retrieve all the rows from the cache if the rows are not sorted. If the data is not sorted on the keys, you might get unexpected results.
- You can't include parameters in the lookup SQL override.
- If you configure a lookup SQL override and a lookup source filter in the same transformation, the mapping task ignores the filter.

Lookup source filter

You can configure a lookup source filter for a relational Lookup transformation that has caching enabled. Add the lookup source filter to limit the number of lookups that the mapping task performs on a lookup source table. When you configure a lookup source filter, the mapping task performs lookups based on the results of the filter statement.

To configure a lookup source filter, open the **Advanced** tab of the Lookup transformation, and enter the filter in the **Lookup Source Filter** field. Do not include the WHERE keyword in the filter condition.

For example, you might need to retrieve the last name of every employee whose ID is greater than 510.

You configure the following lookup source filter on the EmployeeID field in the Lookup transformation:

```
EmployeeID >= 510
```

When the mapping task reads the source row, it performs a lookup on the cache when the value of EmployeeID is greater than 510. When EmployeeID is less than or equal to 510, the Lookup transformation does not retrieve the last name.

When you add a lookup source filter to the Lookup query for a mapping task that uses SQL ELT optimization or is based on a mapping in SQL ELT mode, the task creates a view to represent the SQL override. The mapping task runs an SQL query against this view to push the transformation logic to the database.

Note: If you configure a lookup source filter and a lookup SQL override in the same transformation, the mapping task ignores the filter.

Dynamic lookup cache

Use a dynamic lookup cache to keep the lookup cache synchronized with the target.

When you enable lookup caching, a mapping task builds the lookup cache when it processes the first lookup request. The cache can be static or dynamic. If the cache is static, the data in the lookup cache doesn't change as the mapping task runs. If the task uses the cache multiple times, the task uses the same data. If the cache is dynamic, the task updates the cache based on the actions in the task, so if the task uses the lookup multiple times, downstream transformations can use updated data.

You can use a dynamic cache with most types of lookup sources. You cannot use a dynamic cache with flat file or Salesforce lookups. For more information about using a dynamic cache with a specific type of lookup source, see the help for the appropriate connector.

Based on the results of the lookup query, the row type, and the Lookup transformation properties, the mapping task performs one of the following actions on the dynamic lookup cache when it reads a row from the source:

Inserts the row into the cache

> The mapping task inserts the row when the row is not in the cache. The mapping task flags the row as insert.

Updates the row in the cache

> The mapping task updates the row when the row exists in the cache. The mapping task updates the row in the cache based on the input fields. The mapping task flags the row as an update row.

Makes no change to the cache

>   The mapping task makes no change when the row is in the cache and nothing changes. The mapping task flags the row as unchanged.

The dynamic Lookup transformation includes the return field, NewLookupRow, which describes the changes the task makes to each row in the cache. Based on the value of the NewLookupRow, you can also configure a Router or Filter transformation with the dynamic Lookup transformation to route insert or update rows to the target table. You can route unchanged rows to another target table or flat file, or you can drop them.

You cannot use a parameterized source, target, or lookup with a Lookup transformation that uses a dynamic cache.

Static and dynamic lookup comparison

You might want to use dynamic cache instead of a static cache if the source might contain duplicate private keys. Or, you might want to use a dynamic cache when the source contains a large table of data to optimize performance.

Data Integration processes lookup conditions differently based on whether you configure the Lookup transformation to use a static or dynamic cache.

The following table compares a Lookup transformation that uses a static cache to a Lookup transformation that uses a dynamic cache:

Static Lookup Cache	Dynamic Lookup Cache
The cache does not change during the task run.	The task inserts or updates rows in the cache as it passes rows to the target.
You can use a flat file, relational database, and other connection types such as Salesforce for lookup.	You cannot use a flat file or Salesforce connection type.
When the lookup condition is true, the task returns a value from the lookup table or cache. When the condition is not true, the task returns the default value.	When the lookup condition is true, the task either updates the row in the cache and target or leaves the cache unchanged. This indicates that the row is in the cache and target table. When the lookup condition is not true, the task either inserts the row in the cache and target or leaves the cache unchanged based on the row type. This indicates that the row is not in the cache or target table.

Dynamic cache updates

When the mapping task reads a row, it changes the lookup cache depending on the results of the lookup query and the Lookup transformation properties that you define. The mapping task assigns a value to the NewLookupRow return field that indicates the action it takes.

The following table lists the possible NewLookupRow values:

NewLookupRow Value	Description
0	Mapping task does not update or insert the row in the cache.
1	Mapping task inserts the row into the cache.
2	Mapping task updates the row in the cache.

You can use the NewLookupRow values in downstream transformations.

Inserts and updates for insert rows

You can configure how the mapping task handles inserts and updates to the cache for insert rows. To update existing rows in the dynamic lookup cache when the row type is insert, enable the **Insert Else Update** advanced property for the transformation.

Note: This property only applies to rows entering the Lookup transformation where the row type is insert. When a row of any other row type, such as update, enters the Lookup transformation, the **Insert Else Update** property has no effect on how the mapping task handles the row.

When you enable **Insert Else Update** and the row type entering the Lookup transformation is insert, the mapping task inserts the row into the cache if it is new. If the row exists in the index cache but the data cache is different than the current row, the mapping task updates the row in the data cache.

If you do not enable **Insert Else Update** and the row type entering the Lookup transformation is insert, the mapping task inserts the row into the cache if it is new, and makes no change to the cache if the row exists.

The following table describes how the mapping task changes the lookup cache when the row type of the rows entering the Lookup transformation is insert:

Insert Else Update Option	Row found in cache?	Data cache is different?	Lookup Cache Result	NewLookupRow Value
Disabled - insert only	Yes	-	No change	0
Disabled - insert only	No	-	Insert	1
Enabled	Yes	Yes	Update	2
Enabled	Yes	No	No change	0
Enabled	No	-	Insert	1

Dynamic cache and lookup source synchronization

The Lookup transformation maintains a dynamic lookup cache to track the rows that it passes to the target. When multiple tasks update the same target, you can configure the Lookup transformation in each task to synchronize the dynamic lookup cache to the same lookup source instead of a target.

To synchronize the cache with the lookup source, enable the **Synchronize Dynamic Cache** property for the Lookup transformation.

When you configure a Lookup transformation to synchronize the cache with the Lookup source, the Lookup transformation performs a lookup on the lookup source. If the data does not exist in the Lookup source, the Lookup transformation inserts the row into the lookup source before it updates the dynamic lookup cache.

The data might exist in the lookup source if another task inserted the row. To synchronize the lookup cache to the lookup source, the task retrieves the latest values from the lookup source. The Lookup transformation inserts the values from the Lookup source in the dynamic lookup cache.

For example, you have multiple tasks running simultaneously. Each task generates product numbers for new product names. When a task generates a product number, the other tasks must use the same product number to identify the product. The product number is generated once and inserted in the lookup source. If another task processes a row containing the product, it must use the product number that is in the lookup

source. Each task performs a lookup on the lookup source to determine which product numbers have already been generated.

When you configure the Lookup transformation to synchronize the cache with the lookup source, the task performs a lookup on the dynamic lookup cache for insert rows. If data does not exist in the dynamic lookup cache, the task performs a lookup on the lookup source. It then completes one of the following tasks:

- If data exists in the lookup source, the task inserts a row in the dynamic lookup cache with the columns from the lookup source. It does not update the cache with the source row.
- If data does not exist in the lookup source, the task inserts the data into the lookup source and inserts the row into the cache.

The lookup source contains the same fields as the lookup cache. The task does not insert a row in the lookup cache unless the column is projected from the Lookup transformation or the field is part of a lookup condition.

Dynamic cache and target synchronization

Configure downstream transformations to ensure that the dynamic lookup cache and target are synchronized.

When you use a dynamic lookup cache, the mapping task writes to the lookup cache before it writes to the target table. The lookup cache and target table can become unsynchronized if the task does not write the data to the target. For example, the target database might reject the data.

Consider the following guidelines to keep the lookup cache synchronized with the lookup table:

- Use the Router transformation to pass rows to the cached target when the NewLookupRow value equals one or two.
- Use the Router transformation to drop rows when the NewLookupRow value equals zero. Or, output the rows to a different target.

Field mapping

When you use a dynamic lookup cache, map incoming fields with lookup cache fields on the **Field Mapping** tab. The **Field Mapping** tab is only available when you configure the Lookup transformation to use a dynamic cache.

You must map all of the incoming fields when you use a dynamic cache so that the cache can update as the task runs. Optionally, you can map the Sequence-ID field instead of an incoming field if you want to create a generated key for a field in the target object.

Generated key fields

When you configure a dynamic lookup cache, you can create a generated key for a field in the target object.

To create a generated key for a field in the target object, map the Sequence-ID field to a lookup cache field on the **Field Mapping** tab. You can map the Sequence-ID field instead of an incoming field to lookup cache fields with the integer or Bigint data type. For integer lookup fields, the generated key maximum value is 2,147,483,647. For Bigint lookup fields, the generated key maximum value is 9,223,372,036,854,775,807.

When you map the Sequence-ID field, Data Integration generates a key when it inserts a row into the lookup cache.

Data Integration uses the following process to generate sequence IDs:

1. When Data Integration creates the dynamic lookup cache, it tracks the range of values for each field that has a sequence ID in the dynamic lookup cache.

2. When Data Integration inserts a row of data into the cache, it generates a key for a field by incrementing the greatest sequence ID value by one.

3. When Data Integration reaches the maximum number for a generated sequence ID, it starts over at one. Data Integration increments each sequence ID by one until it reaches the smallest existing value minus one. If Data Integration runs out of unique sequence ID numbers, the mapping task fails.

Data Integration generates a sequence ID for each row it inserts into the cache.

Ignore fields in comparison

When you use a dynamic lookup cache, you can configure fields to be ignored when Data Integration compares the values in the lookup fields with the values in the associated incoming fields. Ignoring some fields in the comparison can improve mapping performance.

When you run a mapping that uses a dynamic lookup cache, by default, Data Integration compares the values in all lookup fields with the values in the associated incoming fields. Data Integration compares the values to determine whether to update the row in the lookup cache. When a value in an incoming field differs from the value in the lookup field, Data Integration updates the row in the cache.

If you do not want to compare all fields, you can choose the fields that you want Data Integration to ignore when it compares fields. For example, the source data includes a column that indicates whether the row contains data that you need to update. Enable the **Ignore in Comparison** property for all lookup fields except the field that indicates whether to update the row in the cache and target table.

Configure the fields to be ignored on the **Return Fields** tab of the Lookup transformation. To ignore a field, enable the **Ignore in Comparison** property for the field.

You must configure the transformation to compare at least one field unless the lookup condition contains all the lookup fields.

Dynamic lookup query overrides

When you add a WHERE clause in a Lookup transformation that uses a dynamic cache, connect a Filter transformation before the Lookup transformation to filter rows that you do not want to insert into the cache or target table. If you do not include the Filter transformation, you might get inconsistent results between the cache and the target table.

For example, you configure a Lookup transformation to perform a dynamic lookup on the employee table, EMP, matching rows by EMP_ID. You define the following lookup SQL override:

```
SELECT EMP_ID, EMP_STATUS FROM EMP ORDER BY EMP_ID, EMP_STATUS WHERE EMP_STATUS = 4
```

When you first run the mapping, the mapping task builds the lookup cache from the target table based on the lookup SQL override. All rows in the cache match the condition in the WHERE clause, `EMP_STATUS = 4`.

The mapping task reads a source row that meets the lookup condition you specify, but the value of EMP_STATUS is 2. Although the target might have the row where EMP_STATUS is 2, the mapping task does not find the row in the cache because of the SQL override. The mapping task inserts the row into the cache and passes the row to the target table. When the mapping task inserts this row in the target table, you might get inconsistent results when the row already exists. In addition, not all rows in the cache match the condition in the WHERE clause in the SQL override.

To verify that you only insert rows into the cache that match the WHERE clause, you add a Filter transformation before the Lookup transformation and define the filter condition as the condition in the WHERE clause in the lookup SQL override.

You enter the following filter condition in the Filter transformation and the WHERE clause in the SQL override:

```
EMP_STATUS = 4
```

Persistent lookup cache

You can configure a Lookup transformation to use a persistent cache. When you use a persistent cache, Data Integration saves and reuses the cache files from mapping run to mapping run.

By default, Data Integration uses a non-persistent cache when you enable caching in a Lookup transformation. When you use a non-persistent cache, Data Integration deletes the cache files at the end of the mapping run. The next time you run the mapping, Data Integration builds the memory cache from the database.

If the lookup table does not change between mapping runs, you can use a persistent cache. A persistent cache can improve mapping performance because it eliminates the time required to read the lookup table. The first time that Data Integration runs a mapping using a persistent lookup cache, it saves the cache files to disk. The next time that Data Integration runs the mapping, it builds the memory cache from the cache files.

Configure the Lookup transformation to use a persistent lookup cache in the transformation advanced properties. To use a persistent cache, enable the **Lookup Cache Persistent** property.

You can configure the following options when you use a persistent cache:

Specify a name for the cache files.

When you use a persistent lookup cache, you can specify a name for the cache files.

To specify a name, enter the file name prefix in the **Cache File Name Prefix** field on the **Advanced** tab of the Lookup transformation. Do not enter a suffix such as .idx or .dat.

Rebuild the lookup cache.

If the lookup table changes occasionally, you can configure the Lookup transformation to rebuild the lookup cache. When you do this, Data Integration rebuilds the lookup cache from the lookup source when it first calls the Lookup transformation instance.

To configure the transformation to rebuild the cache, enable the **Re-cache from Lookup Source** property on the **Advanced** tab of the Lookup transformation.

Rebuilding the lookup cache

You can rebuild the lookup cache if you think the lookup source changed since the last time Data Integration built the persistent cache.

When you rebuild a cache, Data Integration creates new cache files, overwriting existing persistent cache files. Data Integration writes a message to the session log when it rebuilds the cache.

If Data Integration cannot reuse the cache, it rebuilds the cache or fails the mapping task. The behavior can differ based on whether the cache is named or unnamed.

The following table summarizes how Data Integration handles named and unnamed persistent caches when the mapping changes between runs:

Mapping changes between runs	Named cache	Unnamed cache
Data Integration cannot locate cache files. For example, the file no longer exists.	Rebuilds cache	Rebuilds cache
Enable or disable the Enable High Precision option in the mapping task advanced session properties.	Fails mapping task	Rebuilds cache

Mapping changes between runs	Named cache	Unnamed cache
Edit the transformation in the Mapping Designer or Mapplet Designer, excluding editing the transformation description.	Fails mapping task	Rebuilds cache
Edit the mapping, excluding the Lookup transformation.	Reuses cache	Rebuilds cache
Change the number of partitions in the pipeline that contains the Lookup transformation.	Fails mapping task	Rebuilds cache
Change database connection or the file location used to access the lookup table.	Fails mapping task	Rebuilds cache

Unconnected lookups

An unconnected Lookup transformation is a Lookup transformation that is not connected to other transformations in a mapping. A transformation in the mapping pipeline calls the Lookup transformation with a :LKP expression. The unconnected Lookup transformation returns one column to the calling transformation.

You can use an unconnected Lookup transformation to perform a lookup against the following types of data objects:

- Flat file
- Relational database
- Amazon Redshift V2
- Amazon S3 V2
- Google BigQuery V2
- Microsoft Azure Synapse SQL
- Snowflake Data Cloud

The following table lists the differences between connected and unconnected Lookup transformations:

Functionality	Connected lookup	Unconnected lookup
Input values	Receives input values directly from the mapping pipeline.	Receives input values from the result of a :LKP expression in another transformation.
Cache	Cache includes all lookup columns used in the mapping. This includes columns in the lookup condition and columns linked as output fields to other transformations. Can use static or dynamic cache.	Cache includes all lookup/output fields in the lookup condition and the lookup/return field. Cannot use dynamic cache.
Return values	Returns multiple values from the same row.	Returns the specified field for each row.

Functionality	Connected lookup	Unconnected lookup
Lookup conditions	If there is no match for a lookup condition, Data Integration returns the default value for all output fields. If there is a match, Data Integration returns the results of the lookup condition for all lookup/output fields.	If there is no match for the lookup condition, Data Integration returns NULL. If there is a match, Data Integration returns the result of the lookup condition to the return field.
Output values	Passes multiple output values to another transformation. Links lookup/output fields to another transformation.	Passes one output value to another transformation. The lookup/output/return field passes the value to the transformation that contains the :LKP expression.

Configuring an unconnected Lookup transformation

To configure an unconnected Lookup transformation, select the **Unconnected Lookup** option, add incoming fields, configure the lookup condition, and designate a return value. Then configure a lookup expression in a different transformation.

1. On the **General** tab of the Lookup transformation, enable the **Unconnected Lookup** option.
2. Create the incoming fields.

   On the **Incoming Fields** tab of the Lookup transformation, create an incoming field for each argument in the :LKP expression. For each lookup condition you plan to create, you need to add an incoming field to the Lookup transformation. You can create a different field for each condition, or use the same incoming field in more than one condition.
3. Designate a return value.

   You can pass multiple input values into a Lookup transformation and return one column of data. Data Integration can return one value from the lookup query. Use the return field to specify the return value.
4. Configure a lookup expression in another transformation.

   Supply input values for an unconnected Lookup transformation from a :LKP expression in a transformation that uses expressions such as an Expression, Aggregator, Filter, or Router transformation. The arguments are local input fields that match the Lookup transformation input fields used in the lookup condition.

Calling an unconnected lookup from another transformation

Supply input values for an unconnected Lookup transformation from a :LKP expression in another transformation such as an Expression transformation or Aggregator transformation. You can call the same lookup multiple times in one mapping. You cannot call an unconnected lookup from a Joiner or Java transformation.

Use the following syntax for a :LKP expression:

```
:LKP.<Lookup transformation name> (<argument>, <argument>, ...)
```

The arguments are local input fields that match the Lookup transformation input fields used in the lookup condition.

For example, the following expression passes the ITEM_ID and PRICE fields to an unconnected Lookup transformation named lkp_ItemPrices:

```
:LKP.lkp_ItemPrices (ITEM_ID, PRICE)
```

Use the following guidelines to write an expression that calls an unconnected Lookup transformation:

- The order in which you list each argument must match the order of the lookup conditions in the Lookup transformation.
- The datatypes for the fields in the expression must match the datatypes for the input fields in the Lookup transformation.
- The argument fields in the expression must be in the same order as the input fields in the lookup condition.
- If you call a connected Lookup transformation in a :LKP expression, Data Integration marks the mapping invalid.

Connected Lookup example

In the following example, the Lookup transformation performs a lookup on an Orders table and returns all the orders for a specific customer. You could also define the Lookup to return only the first order or last order for the customer.

First, you configure a Lookup Condition, which is an expression that identifies what rows to return from the lookup table. For example, create a Simple Lookup Condition to find all the records where the CUSTOMER_ID Lookup Field is equal to the Incoming Field, CUSTOMER_IN.

Based on this condition, the Lookup finds all the rows where the customer ID is equal to the customer number that is passed to the Lookup transformation.

You can also add multiple conditions. For example, if you add this condition, the Lookup returns only the orders that are greater than $100.00.

O_ORDER_AMT > 100.00

The Lookup returns data only when all conditions are true.

Dynamic Lookup example

To configure a Lookup transformation to be dynamic, use a dynamic lookup cache.

A dynamic cache is helpful when the source table contains a large amount of data or it contains duplicate primary keys.

The following example illustrates the advantage of using a dynamic cache rather than a static cache when the source table includes duplicate primary keys.

You want to update your payroll table with data from your Human Resources department. The payroll table includes the following data, where ID is the primary key:

ID	Name	Location
1	Abhi	USA
2	Alice	UK

You create a mapping with a Lookup transformation and use the payroll table for the target and the lookup cache. You configure the cache to be dynamic because you expect the Human Resources department's table to contain duplicate keys.

In the mapping, you specify the Human Resources department's table to be the source. The source table includes the following data:

ID	Name	Location
1	Abhi	India
2	Alice	UK
3	James	Japan
3	James	USA

You create a mapping task to update the payroll table. When the mapping task begins, it creates the cache file that contains the rows in the target table. As the task processes the rows, it flags the first row as an update and it updates the cache. It flags the third row as an insert and inserts the row in the cache. It flags the fourth row as an update because the row exists in the cache.

If you follow the same scenario using a static cache, the task flags the fourth row as an insert. The cache does not contain the row for James because it does not update as the task processes the rows. The target database produces an error because it cannot handle two rows with the same primary key.

Unconnected Lookup example

You can use an unconnected Lookup transformation to replace cryptic or numeric ID values in a table with meaningful names from a lookup table.

For example, you need to load some sales order data from SAP transactional tables to a relational table in your data warehouse. The SAP tables contain numeric IDs for values such as the sales group and sales office. In the data warehouse table, you want to replace the numeric IDs with the corresponding names in your local language. The name that is associated with each ID is stored in a reference table. Use an unconnected Lookup transformation to retrieve the names from the reference table.

The following image shows the mapping:
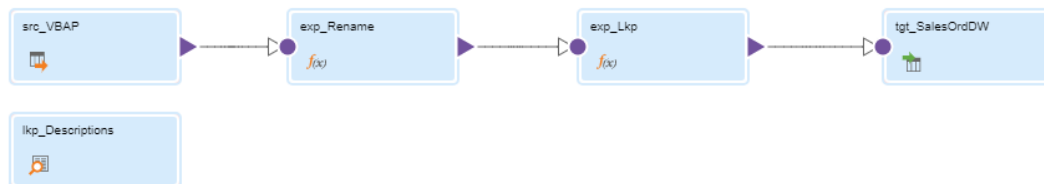
Configure the transformations in the following ways:

Source transformation

Use a Source transformation to specify the tables from which to extract data.

On the **Source** tab, configure the source connection and select the tables from which you want to extract data.

First Expression transformation (optional)

Optionally, use an Expression transformation to rename fields and replace null values.

On the **Incoming Fields** tab, use the **Named Fields** field selection criteria to select the fields that you want to load to the target table. If required, rename the selected fields to give them more meaningful names.

On the **Expression** tab, create output fields to replace the null values. For example, to replace null values for the sales group code and sales office code with spaces, you might create the following output fields:

Output Field	Expression
in_sales_group	IIF(ISNULL(sales_group_code),' ',sales_group_code)
in_sales_office	IIF(ISNULL(sales_office_code),' ',sales_office_code)

Unconnected Lookup transformation

Use an unconnected Lookup transformation to retrieve the descriptions from the reference table.

On the **General** tab, enable the **Unconnected Lookup** option.

On the **Incoming Fields** tab, create an incoming field for each value that you need to pass to the Lookup transformation to retrieve the data that you want. For example, to pass the domain name, language, and code value to the Lookup transformation, create the in_domain_name, in_language, and in_lookup_code fields.

On the **Lookup Object** tab, configure the lookup connection and select the reference table that you want to use as the lookup table.

On the **Lookup Condition** tab, specify the lookup condition for each incoming field. For example:

Lookup Field	Operator	Incoming Field
domain_name	=	in_domain_name
language_code	=	in_language
lookup_code	=	in_lookup_code

On the **Return Fields** tab, select the field in the reference table that you want to return. For example, to return a description, you might select lookup_description as the return field.

Second Expression transformation

Use an Expression transformation to call the unconnected Lookup transformation and retrieve the name that is associated with each ID value.

On the **Incoming Fields** tab, include all fields from the upstream transformation.

On the **Expression** tab, create an output field to retrieve each description from the Lookup transformation. Call the Lookup transformation with a :LKP expression. For example, to retrieve the

sales group and sales office names from the appropriate domain in English, you might create the following output fields:

Output Field	Expression
sales_group	`:LKP.lkp_Descriptions('sales_group','en',in_sales_group)`
sales_office	`:LKP.lkp_Descriptions('sales_office','en',in_sales_office)`

Target transformation

On the **Target** tab, configure the target connection and select the relational table to which you want to load data.

On the **Field Mapping** tab, map the output fields from the upstream transformation to the appropriate target fields. For example, to map the sales_group and sales_office output fields from the second Expression transformation to the SALES_GROUP and SALES_OFFICE target fields, configure the following field mapping:

Target Field	Mapped Field
SALES_GROUP	sales_group
SALES_OFFICE	sales_office

CHAPTER 23

Machine Learning transformation

In advanced mode, the Machine Learning transformation runs a machine learning model and returns predictions. It uses a REST API to pass incoming data to the machine learning model and pass the predictions to downstream transformations.

The following video shows a use case for how you can use the Machine Learning transformation to incorporate a machine learning model into data integration jobs in your organization:

Consuming a Machine Learning Model in Informatica Cloud Data Integration Elastic

Before you use the Machine Learning transformation, verify that the following requirements are met:

- The machine learning model is deployed on a machine learning platform, such as Amazon SageMaker or Azure Machine Learning, and a REST endpoint is available to get predictions from the model.

- An API collection has a POST request to access the REST endpoint using a REST V3 connection.

For information about API collections, see *Components*.

Deploying the model as a REST endpoint

The machine learning model must be deployed as a REST endpoint. The Machine Learning transformation uses the endpoint to communicate with the model.

Deploy the model as a REST endpoint according to your machine learning platform:

Amazon SageMaker

In Amazon SageMaker, use Amazon API Gateway and AWS Lambda to deploy the model as an endpoint.

For more information, refer to the instructions in the following AWS Machine Learning blog post:

https://aws.amazon.com/blogs/machine-learning/call-an-amazon-sagemaker-model-endpoint-using-amazon-api-gateway-and-aws-lambda/

Azure Machine Learning

In Azure Machine Learning, deploy the model as a real-time endpoint.

For more information about real-time endpoints, refer to the Microsoft Azure documentation.

After you deploy the model as a REST endpoint, create an API collection and configure a REST API request to access the endpoint.

Accessing the machine learning model

Define how the Machine Learning transformation accesses the machine learning model on the **Model** tab.

Select a REST API request from an API collection. You can use the same REST V3 connection as the API collection, or you can select a different REST V3 connection.

You cannot use the following authorization methods in the REST V3 connection because they are not available in the Machine Learning transformation:

- OAuth 2.0 authorization code
- OAuth 2.0 client credentials

If you run the Machine Learning transformation in a serverless runtime environment and you want to configure TLS for server authentication between the transformation and the machine learning model, see the Administrator help for details. To configure TLS for a non-serverless runtime environment, contact Informatica Global Customer Support.

The API collection must have a POST request to access the machine learning model. If you change the request type and synchronize the API collection, the Machine Learning transformation clears the **Model**, **Request Mapping**, and **Response Fields** tabs. You must select a different POST request and reconfigure the transformation.

Mapping fields to the request schema

Map incoming fields to request schema fields on the **Request Mapping** tab. The Machine Learning transformation uses the request schema fields to send data to the machine learning model at the REST endpoint.

The request schema fields come from the request schema in the REST API request from the API collection. The working field name is the key that the Machine Learning transformation passes in the REST API request. If the REST API can't process a special character in the field name, the working field name replaces the special character. For example, underscores are replaced with a period character `(.)`.

The names of incoming fields and request schema fields can differ, but the data types and hierarchies must match. All request schema fields must have a mapped field, even if the fields are not mandatory in the REST API.

Note that only parent fields appear in the request mapping. To review child fields, refer to the incoming fields in the Machine Learning transformation and the request schema fields in the API collection.

Mapping hierarchical fields

To map hierarchical fields in the request mapping, the hierarchies of the incoming fields and request schema fields must match.

For example, the following image shows a request schema in an API collection:
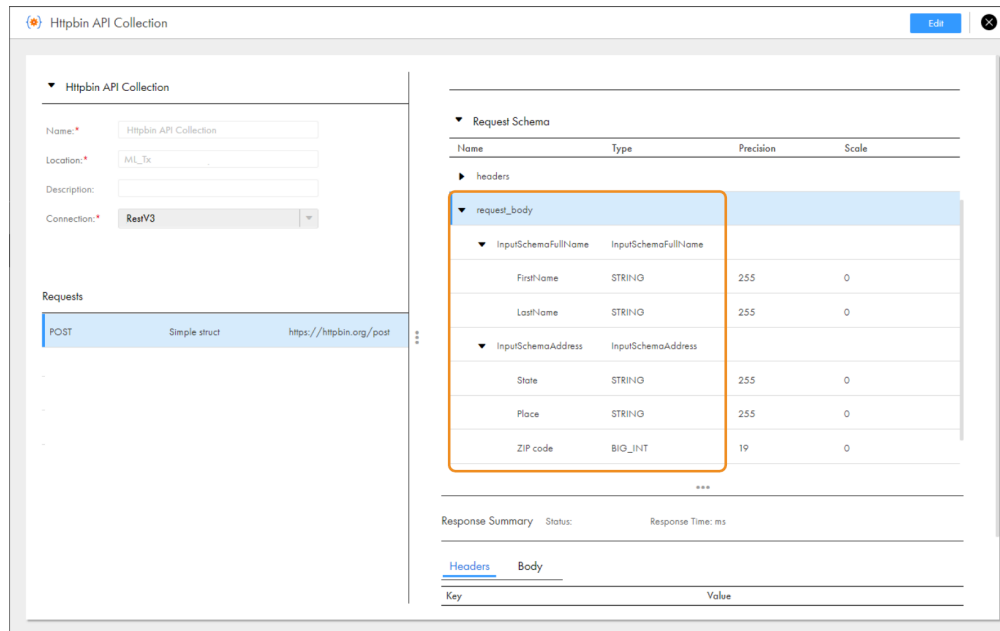
The following image shows incoming fields in a Machine Learning transformation with a hierarchy that matches the request schema:
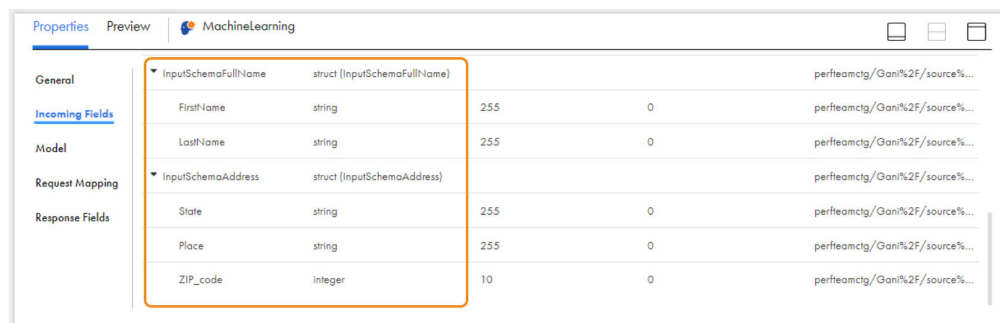
The following image shows the request mapping between the incoming fields and the request schema fields:
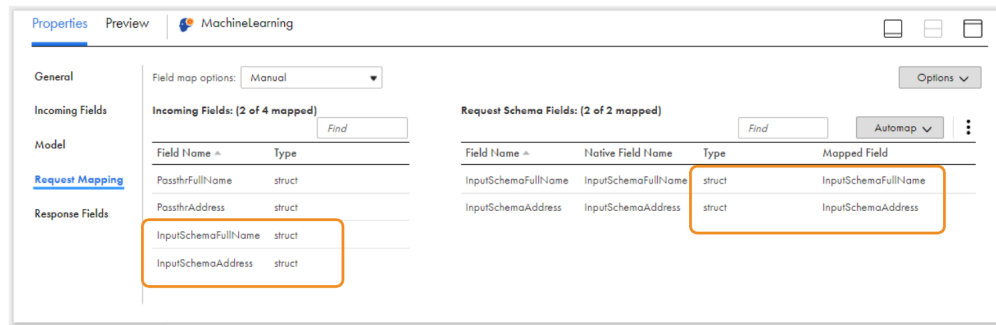
If the hierarchies don't match, use a Hierarchy Processor transformation before the Machine Learning transformation to restructure the incoming data according to the request schema fields. For more information, see "Hierarchy Processor transformation overview" on page 196.

Request mapping options

Configure request mapping options to define how incoming fields are mapped to request schema fields.

You can configure the following request mapping options:

Field Map Options

Method to map fields to the target. Use one of the following options:

- Manual. Manually link incoming fields to target fields. Selecting this option removes links for automatically mapped fields. To map fields manually, drag a field from the incoming fields list and position it next to the appropriate field in the target fields list. Or, you can map selected fields, unmap selected fields, or clear all of the mappings using the **Actions** menu.

- Automatic. Automatically link fields with the same name. Use when all of the fields that you want to link share the same name. You cannot manually link fields with this option.

- Completely Parameterized. Use a parameter to represent the field mapping. In the task, you can configure all field mappings.

- Partially Parameterized. Configure links in the mapping that you want to enforce and use a parameter to allow other fields to be mapped in the mapping task. Or, use a parameter to configure links in the mapping, and allow all fields and links to display in the task for configuration.
  To see more information on field mapping parameters, see *Mappings*.

Parameter

Parameter to use for the field mapping.

New Parameter

Creates a field mapping parameter.

Show Fields

Controls the fields that appear in the **Incoming Fields** list. Show all fields, unmapped fields, or mapped fields.

Automap

Links fields with matching names. Allows you to link matching fields, and then manually configure other field mappings.

You can map fields in the following ways:

- Exact Field Name. Data Integration matches fields of the same name.
- Smart Map. Data Integration matches fields with similar names. For example, if you have an incoming field `Cust_Name` and a target field `Customer_Name`, Data Integration automatically links the `Cust_Name` field with the `Customer_Name` field.

You can use both Exact Field Name and Smart Map in the same field mapping. For example, use Exact Field Name to match fields with the same name and then use Smart Map to map fields with similar names.

You can undo all automapped field mappings by clicking **Automap** > **Undo Automap**. To unmap a single field, select the field to unmap and click **Actions** > **Unmap**.

Data Integration highlights newly mapped fields. For example, when you use Exact Field Name, Data Integration highlights the mapped fields. If you then use Smart Map, Data Integration only highlights the fields mapped using Smart Map.

Action menu

Additional field link options. Provides the following options:

- Map Selected. Links the selected incoming field with the selected target field.
- Unmap Selected. Clears the link for the selected field.
- Clear Mapping. Clears all field mappings.

Show

Determines how field names appear in the target fields list. Use technical field names or labels.

Viewing response fields

View the response schema fields that the REST API returns on the **Response Fields** tab. The response schema fields capture the predictions that you can pass to downstream transformations.

The response schema fields come from the response schema in the REST API request from the API collection. The working field name is the key that the Machine Learning transformation reads from the REST API response. The working field name might replace special characters that the REST API can't process. The field name is the corresponding output field that the Machine Learning transformation uses to represent the response data in the mapping.

Additionally, the ouput fields that are passed to downstream transformations can't contain special characters except for the following characters: `@ # ( )`. The Machine Learning transformation replaces the special characters with an underscore (_).

Configuring bulk requests

Configure bulk requests to optimize performance by combining multiple requests into one request that the Machine Learning transformation sends to the machine learning model. Bulk requests can improve

performance by reducing processing overhead and the amount of time that it takes to communicate with the model.

To create a bulk request, the Machine Learning transformation selects the highest-level array field in the request schema. In the JSON request body for the bulk request, the transformation combines request rows as elements of the selected array field so that one JSON request body contains data for multiple requests. You can configure bulk request options to determine how much data each bulk request contains.

For example, the request schema might have the following structure:

Field Name	Working Field Name	Type
▼ instances	▼ instances	struct (instances)
▼ data	▼ data	array (data [])
▼ features	▼ features	array (features [])
keys	keys	struct (data_keys)
values	values	struct (data_values)
index	index	integer
pages	pages	integer

The Machine Learning transformation selects the `data` array as the highest-level array field to combine requests. If you configure each bulk request to send 2 MB of data to the machine learning model, the Machine Learning transformation configures the `data` array in the JSON request body to include data for 2 MB of request rows.

The highest-level array cannot have sibling array fields. If the highest-level array has a sibling field of a primitive data type, the data in the sibling field will not be combined. Instead, one random record in the sibling field will be sent to the machine learning model.

To create a bulk response, the machine learning endpoint must combine response rows as elements of the highest-level array in the response schema. The Machine Learning transformation parses the array into output rows. Review the bulk request options to verify which field the Machine Learning transformation will parse.

To use bulk requests in the Machine Learning transformation, the machine learning endpoint must be configured to accept bulk requests and send bulk responses.

Bulk request options

Bulk request options determine the fields that the Machine Learning transformation uses to combine requests and parse responses. They also determine how much data each bulk request contains.

The following table describes the bulk request options:

Option	Description
Combine Requests on Field	The transformation combines request rows as elements of this array field and sends one request to the machine learning model. The request schema determines this field automatically.
Request Size	Maximum size of each request. To choose a size, consult the best practices on your machine learning platform.
Parse Requests on Field	The machine learning endpoint combines response rows as elements of this array field. Then, the transformation parses the array into output rows. The response schema determines this field automatically.

Configuring an API proxy

You can configure an API proxy to enforce security for APIs at run time.

The following table describes the types of proxies that are available:

Proxy type	Description
None	Bypasses the proxy server configured at the agent, Spark, or connection level.
Spark	Considers the proxy configured for the Spark engine.
Custom	Considers the proxy configured at the connection level.

Note: A platform proxy is not available and you must configure a Spark proxy instead. A platform proxy considers the proxy configured at the agent level, but mappings that use the Machine Learning transformation refer to the Spark engine for proxy details.

API proxies do not apply if the Machine Learning transformation runs in a serverless runtime environment.

Bypassing the proxy server

To bypass the proxy server, select **None** as the proxy type in the REST V3 connection. Do not configure a proxy server at the agent or Spark level. For more information, see the help for the appropriate connector.

Configuring a Spark proxy

A Spark proxy replaces a platform proxy. You must configure the Spark proxy in the mapping task. For more information, see the help for the appropriate connector.

1. In the REST V3 connection, select **Platform** as the proxy type.

2. In the mapping task, set the Spark session property `spark.driver.extraJavaOptions` to the following value: `-Dhttp.proxyHost=<host> -Dhttp.proxyPort=<port> -Dhttp.proxyUser=<user> -Dhttp.proxyPassword=<password>`

Configuring a custom proxy

Configure a custom proxy to use the proxy details in the REST V3 connection. Do not configure a proxy server at the agent or Spark level. For more information, see the help for the appropriate connector.

1. In the REST V3 connection, select **Custom** as the proxy type.

2. Configure the host, post, user, and password.

3. Save the connection.

Troubleshooting

Use Monitor to access log files to troubleshoot REST API requests and responses in the Machine Learning transformation. Set the tracing level to verbose data to view the details.

The following table lists the details that are available in each log file:

Log file	Details
Spark driver log	- Input schema - Output schema - URL - HTTP method
Spark executor log	If the mapping fails, the following details become available for each failed row: - Response code* - Request details, such as the URL, headers, and request body - Response details, such as the headers and response body The following details are available for each successful row: - Response code - Time to receive the response
Session log	Stack trace, if the mapping fails*
Also available when the log level is set to INFO.	

For more information about log files in advanced mode, see the Monitor help.

Error handling

The Machine Learning transformation handles errors based on the type of issue that occurs and the response code that it receives from the REST API. You can review the response details in the Spark executor log.

The mapping fails when the following types of issues occur:

- Connection issues when there is a connection timeout, the route cannot be found, or there is an unknown host.
- Syntax issues in the URL.
- SSL handshake issues such as certificate issues during SSL handshakes with the server.

The following table describes the action that occurs for each response code:

HTTP Status Code	Action
100 Continue	Row is skipped. Partial requests are not permitted.
101,102,103 Informational, interim	Mapping fails.
3XX Redirection	Row is skipped. Mapping fails if the redirected URL cannot be reached or the redirected URL returns a code that fails the mapping.
400 Bad request	Row is skipped.
401 Unauthorized	Mapping fails.
403 Forbidden	Mapping fails.
404 Resource not found	Mapping fails.
405 Method not allowed	Mapping fails.
407 Proxy authentication required	Request is tried again.
408 Request timeout	Mapping fails.
409 Conflict	Row is skipped.
415 Unsupported media type	Mapping fails.
418 I'm a teapot	Mapping fails.
421 Misdirected request	Mapping fails.
423 Resource is locked	Row is skipped.
429 Too many requests	Mapping fails.
500 Internal server error	Mapping fails.
501 Not implemented	Mapping fails.

HTTP Status Code	Action
502 Bad gateway	Mapping fails.
503 Service unavailable	Mapping fails.
504 Gateway timeout	Mapping fails.
505 HTTP version not supported	Mapping fails.
506 Server internal configuration error	Mapping fails.
511 Network authentication required	Mapping fails.

Machine Learning transformation example

You're a data scientist at a popular video streaming site and you need to generate a report on which users are likely to cancel their subscription so the marketing team can target their upcoming ad campaign.

You built a machine learning model to predict how likely users are to cancel their subscription based on their watch history, content preferences, and other user data. Your company stores this user data in a cloud data lake. To generate the report, you'll create a mapping to read from the data lake, pass the data through the machine learning model, and filter out results that aren't important to the marketing campaign.

Before you create the mapping, you set up a REST endpoint for the machine learning model. You also create an API collection with a POST request in Data Integration.

Then, you create a mapping that includes a Machine Learning transformation and a Filter transformation. The following image shows an overview of the mapping:

You configure the transformations in the following ways:

Machine Learning transformation

On the **Model** tab, select a POST request from your API collection as the operation and select the connection to the machine learning model. The following image shows the configured **Model** tab:
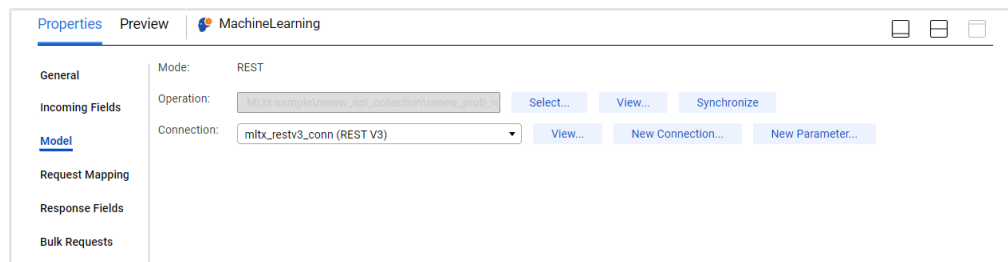
On the **Request Mapping** tab, verify that the mapping of the incoming field from the Source transformation to the request schema field is correct. If needed, manually map the fields. The following image shows the configured **Request Mapping** tab:
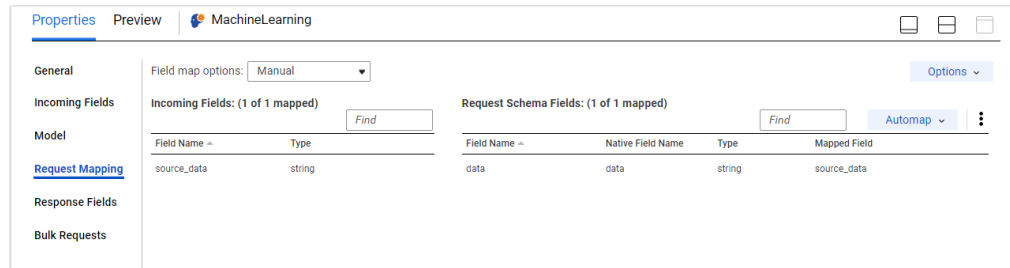
Filter transformation

Since the marketing team is only interested in users who are unlikely to renew their subscription, set the filter condition to filter for users who have a probability of renewing less than 50%. The following image shows the configured filter condition:
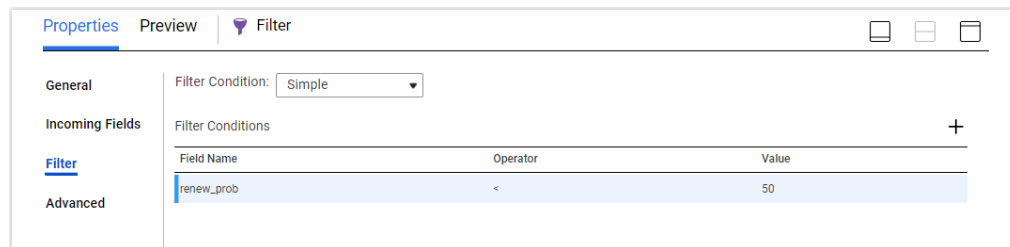
When you run the mapping, it reads user data from the cloud data lake. The Machine Learning transformation makes a REST API call to pass the data through the request schema fields to the machine learning model. The model generates predictions about each user's likelihood of renewing, and the Machine Learning transformation captures the predictions in the response schema fields. The Machine Learning transformation passes the results to the Filter transformation, which filters out users who are likely to renew and passes users who are unlikely to renew to the Target transformation. The Target transformation writes the results to the target file, which you can send over to the marketing team.

CHAPTER 24

Mapplet transformation

The Mapplet transformation inserts a mapplet that you created in Data Integration, imported from PowerCenter, or generated from an SAP asset into a mapping. Each Mapplet transformation can contain one mapplet. You can add multiple Mapplet transformations to a mapping or mapplet.

The Mapplet transformation can be active or passive based on the transformation logic within the mapplet. An active mapplet includes at least one active transformation. An active mapplet can return a number of rows that is different from the number of source rows passed to the mapplet. A passive mapplet includes only passive transformations. A passive mapplet returns the same number of rows that are passed from the source.

Use a Mapplet transformation to accomplish the following goals:

Extend the data transformation capabilities of Data Integration.

For example, you want to create a mapping that passes customer records to a target if the customers pass a credit check. You create a Web Services transformation to run a credit check on each customer. You include the Web Services transformation in a mapplet and use the mapplet in a mapping to perform the credit check.

Reuse transformation logic in different mappings.

For example, you have different fact tables that require a series of dimension keys. You create a mapplet that contains a series of Lookup transformations to find each dimension key. You include the mapplet in different fact table mappings instead of re-creating the lookup logic in each mapping.

Hide complex transformation logic.

The Mapplet transformation shows the mapplet incoming and outgoing fields. It does not display the transformations that the mapplet contains.

For more information about mapplets, see *Components*.

Mapplet transformation configuration

When you add a Mapplet transformation to a mapping, you must first select the mapplet that contains the transformation logic that you want to include in the mapping. If the mapplet includes one or more input groups, configure the incoming fields and field mappings. If the mapplet includes one or more output groups, verify the output fields.

To configure a Mapplet transformation, complete the following tasks:

1. Select the mapplet that you want to include in the transformation.
2. If the mapplet includes one or more input groups, configure the incoming fields.

By default, the transformation inherits all incoming fields from the upstream transformation. You can define a field rule to limit or rename the incoming fields. If the mapplet contains multiple input groups, configure incoming fields for each input group.

For information about configuring incoming fields for a transformation, see "Incoming fields" on page 25.

3. If the mapplet includes one or more input groups, configure field mappings to define how data moves from the upstream transformation to the Mapplet transformation.

   If the mapplet contains multiple input groups, configure the field mappings for each input group.

4. If the mapplet contains one or more output groups, verify the mapplet output fields on the **Output Fields** tab. Connect at least one output group to a downstream transformation.

Selecting a mapplet

Select the mapplet that you want to use in the Mapplet transformation on the **Mapplet** tab of the **Properties** panel. You can select a mapplet that you created or imported into Data Integration, or you can select a mapplet that is included in an installed bundle.

1. In the **Properties** panel for the Mapplet transformation, click the **Mapplet** tab.
2. Click **Select**.
3. Open the project and folder that contains the mapplet and click **Select**.

   Mapplets in installed bundles are in the Add-On Bundles project.

   The selected mapplet appears in the **Properties** panel.

If the mapplet that you select does not include a source, configure the incoming fields and field mappings after you select the mapplet.

If the mapplet that you select does not contain a target, configure the output fields and field mappings after you select the mapplet.

Mapplet transformation field mappings

Configure field mappings in a Mapplet transformation to define how data moves from the upstream transformation to the Mapplet transformation. Configure field mappings on the **Field Mapping** tab of the **Properties** panel.

Note: Mapped field names can have a maximum of 72 characters.

You can configure the following field mapping options:

Mapplet Input Group

The input group for which you want to configure field mappings. This option appears when the Mapplet transformation has multiple input groups.

Field Map Options

Method of mapping fields to the Mapplet transformation. Select one of the following options:

- Manual. Manually link incoming fields to Mapplet transformation input fields. Removes links for automatically mapped fields.

- Automatic. Automatically link fields with the same name. Use when all of the fields that you want to link share the same name. You cannot manually link fields with this option.

Options

Controls how fields are displayed in the **Incoming Fields** and **Mapplet Input Fields** lists. Configure the following options:

- The fields that appear. You can show all fields, unmapped fields, or mapped fields.

- Field names. You can use technical field names or labels.

Automap

Links fields with matching names. Allows you to link matching fields, and then manually configure other field mappings.

You can map fields in the following ways:

- Exact Field Name. Data Integration matches fields of the same name.

- Smart Map. Data Integration matches fields with similar names. For example, if you have an incoming field `Cust_Name` and a target field `Customer_Name`, Data Integration automatically links the `Cust_Name` field with the `Customer_Name` field.

You can use both Exact Field Name and Smart Map in the same field mapping. For example, use Exact Field Name to match fields with the same name and then use Smart Map to map fields with similar names.

You can undo all automapped field mappings by clicking **Automap** > **Undo Automap**. To unmap a single field, select the field to unmap and click **Actions** > **Unmap**.

Data Integration highlights newly mapped fields. For example, when you use Exact Field Name, Data Integration highlights the mapped fields. If you then use Smart Map, Data Integration only highlights the fields mapped using Smart Map.

Action menu

Additional field mapping options. Provides the following options:

- Map Selected. Links the selected incoming field with the selected mapplet input field.

- Unmap Selected. Clears the link for the selected field.

- Clear Mapping. Clears all field mappings.

Mapplet parameters

When you select a mapplet that contains parameters, the parameters are renamed in the Mapplet transformation. You can view the corresponding parameter names on the **Parameter** tab of the **Properties** panel.

In the Mapplet transformation, mapplet parameter names are prefixed with the Mapplet transformation name.

The following image shows the Parameters tab:

You can edit the properties of the mapplet parameters, but you cannot change the parameter type or delete the parameters. To delete a parameter, open the mapplet and remove the parameter.

To edit the parameter properties, click the new parameter name on the **Parameters** tab or on the **Parameters** panel. When you change the parameter properties in a Mapplet transformation, the changes do not affect the mapplet.

You can reuse the parameters in other transformations in the mapping.

Mapplet transformation output fields

When you select a mapplet that contains an Output transformation to use in a Mapplet transformation, the mapplet output fields appear on the **Output Fields** tab of the **Properties** panel.

The Mapping Designer displays the name, type, precision, scale, and origin for each output field in each output group. You cannot edit the transformation output fields. If you want to exclude output fields from the data flow or rename output fields before you pass them to a downstream transformation, configure the field rules in the downstream transformation.

Mapplet transformation names

Data Integration renames the transformations in a mapplet when you use the mapplet in a Mapplet transformation. If you reference a transformation in the mapplet in a downstream transformation, be sure to use the updated name.

The mapplet that you use in the Mapplet transformation might contain transformations with names that conflict with the names of transformations in the mapping. To avoid name conflicts with transformations in the mapping, Data Integration prefixes the names of transformations in the mapplet with the Mapplet transformation name at run time.

For example, a mapplet contains an Expression transformation named Expression_1. You create a mapping and use the mapplet in the Mapplet transformation Mapplet_Tx_1. When you run the mapping, the Expression transformation is renamed to Mapplet_Tx_1_Expression_1.

If the mapplet contains another Mapplet transformation, Data Integration also prefixes the transformation names with the second Mapplet transformation name. For example, the mapplet in the previous example also contains a Mapplet transformation named MappletTx, which contains FilterTx_1. In the mapping, FilterTx_1 is renamed to Mapplet_Tx_1_MappletTx_FilterTx_1.

In most cases, Data Integration truncates transformation names that contain more than 80 characters. Data Integration doesn't truncate the names of Mapplet transformations that contain Hierarchy Builder, Hierarchy Parser, or Structure Parser transformations. When you use a Hierarchy Builder, Hierarchy Parser, or Structure Parser transformation in a mapplet, to prevent runtime errors, be sure that the combined Mapplet transformation name doesn't exceed 80 characters.

Note: Data Integration only renames transformations in mapplets when the mapplet is used in a mapping created after the April 2022 release.

Synchronizing a mapplet

If the interface of a mapplet changes after it has been added to a Mapplet transformation, you must synchronize the mapplet to get the changes. Synchronize a mapplet on the **Mapplet** tab.

Mappings and mapplets that use the mapplet are invalid until the mapplet is synchronized. If you run a mapping task that includes a changed mapplet, the task fails.

When you synchronize a mapplet, the updates might cause validation errors in other transformations in the mapping or mapplet.

You cannot synchronize a mapplet that you imported into Data Integration from PowerCenter or SAP.

To synchronize a mapplet, perform the following steps:

1. Open the mapping or mapplet that uses the mapplet.
2. Select the mapplet transformation.
3. On the **Mapplet** tab, click **Synchronize**.
4. Correct any resulting errors in the transformation logic.

CHAPTER 25

Normalizer transformation

The Normalizer transformation is an active transformation that transforms one incoming row into multiple output rows. When the Normalizer transformation receives a row that contains multiple-occurring data, it returns a row for each instance of the multiple-occurring data.

For example, a relational source includes four fields with quarterly sales data. You can configure a Normalizer transformation to generate a separate output row for each quarter.

When the Normalizer transformation returns multiple rows from an incoming row, it returns duplicate data for single-occurring incoming columns.

When you configure a Normalizer transformation, you define Normalizer properties on the following tabs of the **Properties** panel:

- **Normalized Fields** tab. Define the multiple-occurring fields and specify additional fields that you want to use in the mapping.
- **Field Mapping** tab. Connect the incoming fields to the normalized fields.

Normalized fields

Define the fields to be normalized on the **Normalized Fields** tab. You can also include other incoming fields that you want to use in the mapping.

When you define normalized fields, you can create fields manually or select fields from a list of incoming fields. When you create a normalized field, you can set the data type to String or Number, and then define the precision and scale. In advanced mode, you can use any primitive data type.

In advanced mode, the Normalizer transformation produces multiple output groups. Otherwise, the Normalizer transformation produces only one output group.

When incoming fields include multiple-occurring fields without a corresponding category field, you can create the category field to define the occurs for the data. For example, to represent three fields with different types of income, you can create an Income category field and set the occurs value to 3.

Occurs configuration

Configure the occurs value for a normalized field to define the number of instances the field occurs in incoming data.

To define a multiple occurring field, set the occurs value for the field to an integer greater than one. When you set an occurs value to greater than one, the Normalizer transformation creates a generated column ID field for the field. The Normalizer transformation also creates a generated key field for all normalized data.

The Normalizer transformation also uses the occurs value to create a corresponding set of output fields. The output fields display on the **Field Mapping** tab of the Normalizer transformation. The naming convention for the output fields is <occurs field name>_<occurs number>.

To define a single-occurring field, set the occurs value for the field to one. Define a single-occurring field to include incoming fields that do not need to be normalized in the normalized fields list.

Unmatched groups of multiple-occurring fields

You can normalize more than one group of multiple-occurring fields in a Normalizer transformation. When you include more than one group and the occurs values do not match, configure the mapping to avoid validation errors.

Use one of the following methods to process groups of multiple-occurring fields with different occurs values.

Write the normalized data to different targets

> You can use multiple-occurring fields with different occurs values when you write the normalized data to different targets.
>
> For example, the source data includes an Expenses field with four occurs and an Income field with three occurs. You can configure the mapping to write the normalized expense data to one target and to write the normalized income data to a different target.

Use the same occurs value for multiple occurring fields

> You can configure the multiple-occurring fields to use the same number of occurs, and then use the generated fields that you need. When you use the same number of occurs for multiple-occurring fields, you can write the normalized data to the same target.
>
> For example, when the source data includes an Expenses field with four occurs and an Income field with three occurs, you can configure both fields to have four occurs.
>
> When you configure the Normalizer field mappings, you can connect the four expense fields and the three income fields, leaving the unnecessary income output field unused. Then, you can configure the mapping to write all normalized data to the same target.

Generated keys

The Normalizer transformation generates key values for normalized data.

Generated keys fields appear on the **Normalized Fields** tab when you configure the field to have more than one occurrence.

The mapping task generates the following fields for normalized data.

Generated Key

> A key value that the task generates each time it processes an incoming row. When a task runs, it starts the generated key with one and increments by one for each processed row.
>
> The Normalizer transformation uses one generated key field for all data to be normalized.
>
> The naming convention for the Normalizer generated key is GK_<redefined_field_name>.

Note: The generated key is not applicable in advanced mode.

Generated Column ID

> A column ID value that represents the instance of the multiple-occurring data. For example, if an Expenses field that includes four occurs, the task uses values 1 through 4 to represent each type of occurring data.

The Normalizer transformation uses a generated column ID for each field configured to occur more than one time.

The naming convention for the Normalizer generated key is GCID_<redefined_field_name>.

An advanced cluster processes the generated column ID field as a bigint. The Data Integration Server processes the ID as an integer.

Normalizer field mapping

Map incoming fields to normalized fields on the **Field Mapping** tab of the Normalizer transformation.

When you configure the Normalizer field mappings, complete the following steps:

1. Map the multiple-occurring incoming fields that you want to normalize to the corresponding output fields that the Normalizer transformation created.

   Note: Map at least one output field for each set of normalized fields.

2. Map incoming fields to all normalized fields with a single occurrence.

Normalizer field mapping options

You can use the following field mapping options on the Normalizer **Field Mapping** tab.

Show Fields

Controls the fields that appear in the **Incoming Fields** list. Show all fields, unmapped fields, or mapped fields.

Automap

Links fields with matching names. Allows you to link matching fields, and then manually configure other field mappings.

You can map fields in the following ways:

- Exact Field Name. Data Integration matches fields of the same name.

- Smart Map. Data Integration matches fields with similar names. For example, if you have an incoming field `Cust_Name` and a target field `Customer_Name`, Data Integration automatically links the `Cust_Name` field with the `Customer_Name` field.

You can use both Exact Field Name and Smart Map in the same field mapping. For example, use Exact Field Name to match fields with the same name and then use Smart Map to map fields with similar names.

You can undo all automapped field mappings by clicking **Automap** > **Undo Automap**. To unmap a single field, select the field to unmap and click **Actions** > **Unmap**.

Data Integration highlights newly mapped fields. For example, when you use Exact Field Name, Data Integration highlights the mapped fields. If you then use Smart Map, Data Integration only highlights the fields mapped using Smart Map.

Action menu

Additional field link options. Provides the following options:

- Map Selected. Links the selected incoming field with the selected target field.

- Unmap Selected. Clears the link for the selected field.

- Clear Mapping. Clears all field mappings.

Advanced properties

You can configure advanced properties for a Normalizer transformation. The advanced properties control settings such as the tracing level for session log messages and whether the transformation is optional or required.

The properties that are available vary based on the mapping mode.

You can configure the following properties:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Optional	Determines whether the transformation is optional. If a transformation is optional and there are no incoming fields, the mapping task can run and the data can go through another branch in the data flow. If a transformation is required and there are no incoming fields, the task fails.
	For example, you configure a parameter for the source connection. In one branch of the data flow, you add a transformation with a field rule so that only Date/Time data enters the transformation, and you specify that the transformation is optional. When you configure the mapping task, you select a source that does not have Date/Time data. The mapping task ignores the branch with the optional transformation, and the data flow continues through another branch of the mapping.

Target configuration for Normalizer transformations

When you use a Normalizer transformation in a mapping, consider the following configuration guidelines for targets and Target transformations:

- To write normalized data to the target, map the multiple occurring field to a target field in the Target transformation field mapping.

- To include generated keys or generated column IDs in target data, create additional target fields as required, and then map the fields in the Target transformation field mapping.

Note: Generated keys are not applicable in advanced mode.

Normalizer field rule for parameterized sources

When you use a parameter as the source object to provide data for a Normalizer transformation, use a **Named Fields** field rule in the Normalizer to define incoming fields.

When you configure a Normalizer transformation, you define the field mappings between the incoming fields and the normalized fields. When you use a parameter for a source object, field names do not appear in the list of incoming fields until you use a Named Field rule to add fields.

To add fields, on the **Incoming Fields** tab of the Normalizer transformation, create a **Named Fields** rule. In the **Include Named Fields** dialog box, click **Add** and enter the name of an incoming field.

Create fields to represent all of the source fields that you want to use in the Normalizer transformation.

Mapping example with a Normalizer and Aggregator

A relational table includes the quarterly unit sales for each store. To calculate the annual unit sales for each store, use a Normalizer transformation to create a row for each quarter. Then, use an Aggregator transformation to aggregate the quarterly sales for each store.

The following image shows the mapping to create:
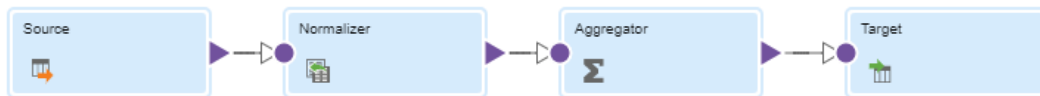
The source data includes the following rows:

StoreNo	Q1	Q2	Q3	Q4	Year
1001	30	50	48	80	2014
1022	100	120	125	140	2014
1190	80	108	123	134	2014

Use the Normalizer transformation to pivot source data before the data passes to the Aggregator transformation, as follows:

StoreNo	QuarterlySales
1001	30
1001	50
1001	48
1001	80
1022	100
1022	120
1022	125
1022	140
1190	80
1190	108
1190	123
1190	134

Define the normalized fields

On the **Normalized Fields** tab, create a normalized field called "QuarterlySales." To indicate that this field represents four fields, set the occurs value to four.

To include the store number data in the mapping, from the menu, select **Generate From Incoming Fields** and select **StoreNo**. Use the default occurs value of one because this field does not include multiple-occurring data.

The following image shows the **Normalized Field** tab after adding both fields:
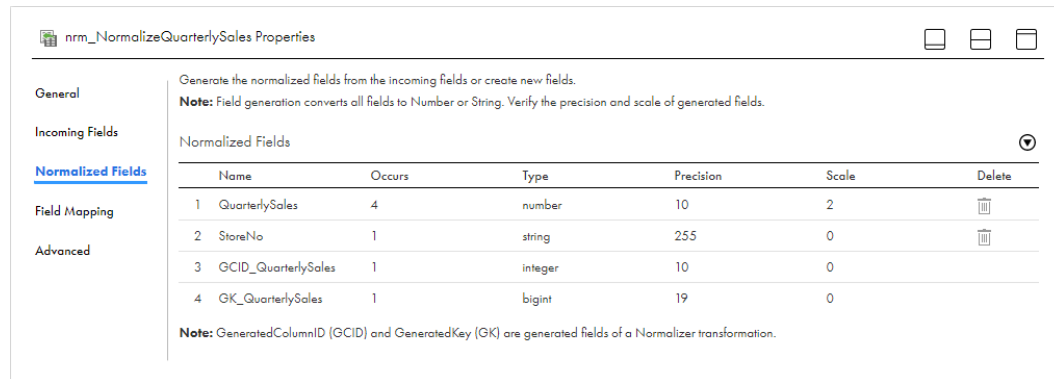
Notice that when you set the QuarterlySales occurs value to four, the Normalizer creates the generated column ID field and the generated key field.

Configure the Normalizer field mappings

On the **Field Mapping** tab of the Normalizer transformation, connect the incoming fields to the normalized fields.

In the **Normalized Fields** list, the Normalizer replaces the multiple-occurring QuarterlySales field with corresponding fields to hold the normalized data: QuarterlySales_1, QuarterlySales_2, QuarterlySales_3, and QuarterlySales_4. The list also includes the StoreNo field.

Connect the incoming fields to the StoreNo and QuarterlySales normalized fields as follows:
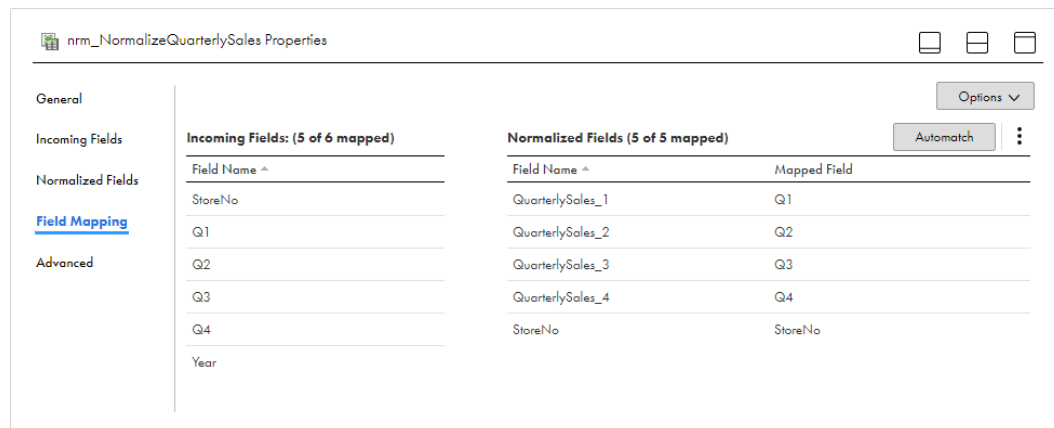
Configure the Aggregator transformation

To calculate the annual sales by store, add an Aggregator transformation to the mapping and connect the Normalizer to the Aggregator.

In the Aggregator transformation, use the default All Fields rule to pass all fields from the Normalizer to the Aggregator.

To group data by store number, add a group by field on the **Group By** tab, and then select the **StoreNo** field.

The following image shows the **Group By** tab with the StoreNo group by field:
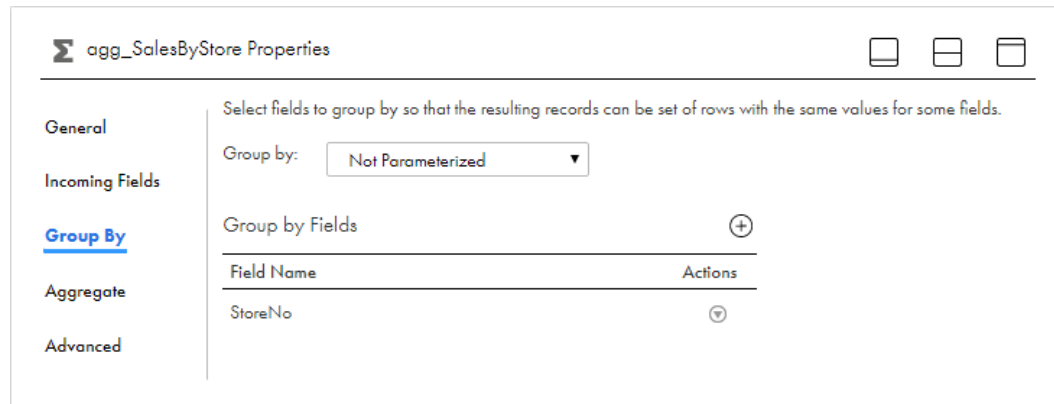
On the **Aggregate** tab, create a Decimal output field named AnnualSales_byStore. To configure the output field, use the QuarterlySales field in the following aggregate expression: `SUM(QuarterlySales)`. The QuarterlySales field represents all of the normalized quarterly data.

The following image shows the **Aggregate** tab with the AnnualSales_byStore output field:
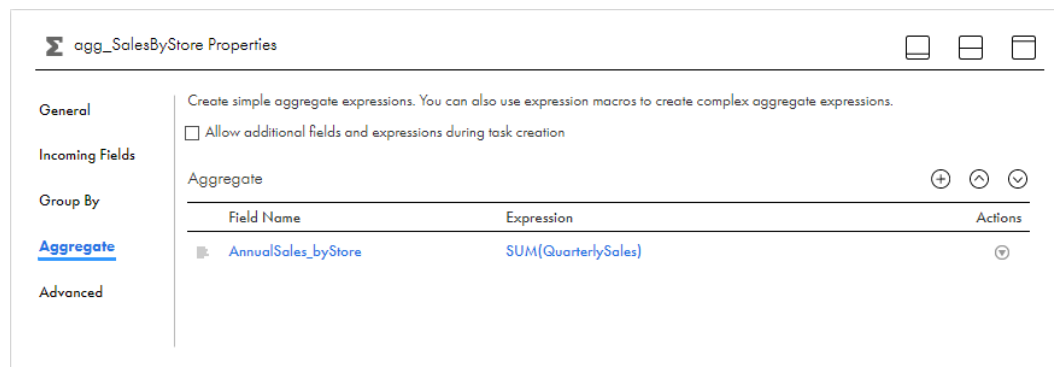
Configure the Target

Add a Target transformation and connect the Aggregator transformation to the Target transformation.

Use the default All Fields rule to pass all fields from the Aggregator to the Target transformation.

On the **Target** tab, select the target connection and the target object.

On the **Field Mapping** tab, the incoming fields list includes the AnnualSales_byStore field created in the Aggregator transformation, and the StoreNo field that passed through the mapping from the source.

The incoming fields list also includes the QuarterlySales and generated key columns created by the Normalizer. These fields do not need to be written to the target.

Connect the StoreNo and AnnualSales_byStore fields to corresponding target fields.

The following image shows the configured **Field Mapping** tab:
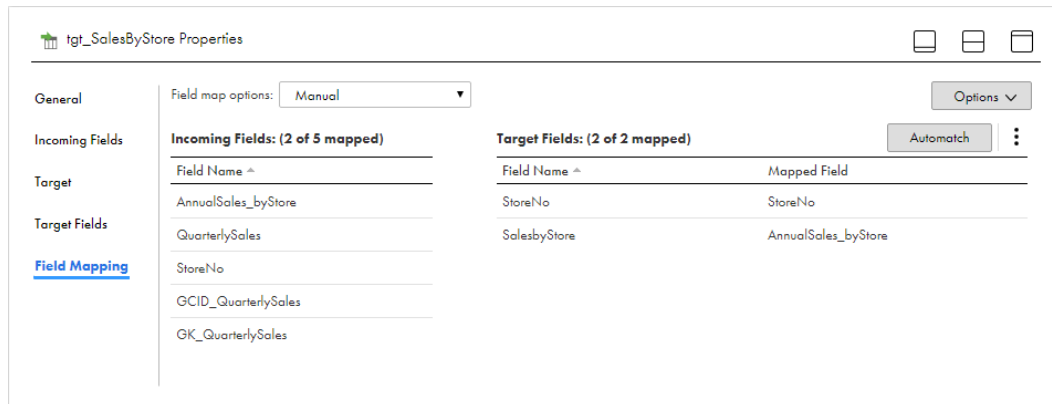
Task results

When you run the task, the mapping task normalizes the source data, creating one row for each quarter. The task groups the normalized data by store, and then aggregates the quarterly unit sales for each store.

The task writes the following data to the target:

```
StoreNo    SalesbyStore
1001        208
1022        485
1190        445
```

CHAPTER 26

Output transformation

The Output transformation is a passive transformation that you use to pass data from a mapplet to a downstream transformation.

Add output fields to the Output transformation to define the data fields you want to pass from the mapplet. You must add at least one output field to each Output transformation. You can add multiple Output transformations to a mapplet. Each Output transformation becomes an output group when you use the mapplet in a Mapplet transformation. You must connect at least one output group to a downstream transformation. You can connect an output group to multiple downstream transformations.

You can use an Output transformation in a mapplet but not in a mapping.

Output fields

Add output fields to an Output transformation to define the data fields you want to pass from the mapplet to the downstream transformation. You must add at least one output field to each Output transformation.

Add output fields on the **Output Fields** tab of the properties panel. You can add fields manually or automatically generate fields based on incoming fields.

When you use the mapplet in a Mapplet transformation, map at least one output field to the downstream transformation.

To add an output field, click **Add** and then choose to add fields manually or to automatically create fields based on incoming fields. You can also generate some fields automatically and create other fields manually. If you choose to create output fields based on incoming fields, you can edit the generated metadata. When you add fields manually, enter the field name, data type, precision, and scale for each field.

After you add output fields, map them to incoming fields on the **Field Mapping** tab.

Generating output fields based on incoming fields

When you generate output fields based on incoming fields, Data Integrationuses incoming field metadata to automatically create corresponding output fields. You can choose to copy all incoming fields or only certain fields.

1. On the **Output Fields** tab, click **Add** and then select **Generate output fields based on incoming fields**.
2. Select the incoming fields that you want to use as the basis for output fields.

    Tip: Search for specific fields by entering the field name in the find field.

3. Click **Select Fields**.

   Data Integration creates output fields with the same name and metadata as the incoming fields that you selected.

Field mapping

Map fields to configure how data moves from the upstream transformation to the Output transformation.

Configure field mappings on the **Field Mapping** tab.

The **Field Mapping** tab includes a list of incoming fields and a list of target fields.

Note: Mapped field names can have a maximum of 72 characters.

You can configure the following field mapping options:

Field Map Options

Method of mapping fields to the Mapplet transformation. Select one of the following options:

- Manual. Manually link incoming fields to Mapplet transformation input fields. Removes links for automatically mapped fields.
- Automatic. Automatically link fields with the same name. Use when all of the fields that you want to link share the same name. You cannot manually link fields with this option.

Options

Controls how fields are displayed in the **Incoming Fields** and **Output Fields** lists. Configure the following options:

- The fields that appear. You can show all fields, unmapped fields, or mapped fields.
- Field names. You can use technical field names or labels.

Automap

Links fields with matching names. Allows you to link matching fields, and then manually configure other field mappings.

You can map fields in the following ways:

- Exact Field Name. Data Integration matches fields of the same name.
- Smart Map. Data Integration matches fields with similar names. For example, if you have an incoming field `Cust_Name` and a target field `Customer_Name`, Data Integration automatically links the `Cust_Name` field with the `Customer_Name` field.

You can use both Exact Field Name and Smart Map in the same field mapping. For example, use Exact Field Name to match fields with the same name and then use Smart Map to map fields with similar names.

You can undo all automapped field mappings by clicking **Automap** > **Undo Automap**. To unmap a single field, select the field to unmap and click **Actions** > **Unmap**.

Data Integration highlights newly mapped fields. For example, when you use Exact Field Name, Data Integration highlights the mapped fields. If you then use Smart Map, Data Integration only highlights the fields mapped using Smart Map.

Action menu

Additional field mapping options. Provides the following options:

- Map Selected. Links the selected incoming field with the selected mapplet input field.
- Unmap Selected. Clears the link for the selected field.
- Clear Mapping. Clears all field mappings.

CHAPTER 27

Parse transformation

The Parse transformation adds a parse asset that you created in Data Quality to a mapping.

A parse asset defines a set of operations that identify tokens in an input field based on the content or structure of the token. In a parsing operation, a token is a discrete word or string.

The Parse transformation parses the tokens to output fields that the asset specifies. When you configure the transformation, you map an input field to the appropriate target field in the transformation. When the mapping runs, the transformation searches the input field for tokens that meet the parsing criteria and writes the tokens to the associated output fields. If the transformation can identify an input data value but a defined output field is not available, the transformation may write the value to a predefined field for overflow data. If the transformation cannot identify a value in the input data, it may write the value to a predefined field for unparsed data. The asset that you add to the transformation determines the number of overflow and unparsed data fields that the transformation creates.

A Parse transformation is similar to a Mapplet transformation, as it allows you to add data transformation logic that you designed elsewhere to a mapping. Like mapplets, parse assets are reusable assets.

A Parse transformation shows incoming and outgoing fields. It does not display the logic that the parse asset contains or allow to you edit the parse asset. To edit the parse asset, open it in Data Quality.

Parse transformation configuration

When you configure a Parse transformation in a mapping, you first select the parse asset that contains the logic to include in the transformation. Next, you map an incoming field on the transformation to the target field that parse asset specifies.

To configure the transformation, complete the following tasks:

1. Connect the Parse transformation to a Source transformation or other upstream object.
2. On the **Parse** tab, select the parse asset that you want to include in the transformation.

The following image shows the options that you use to select the parse asset:
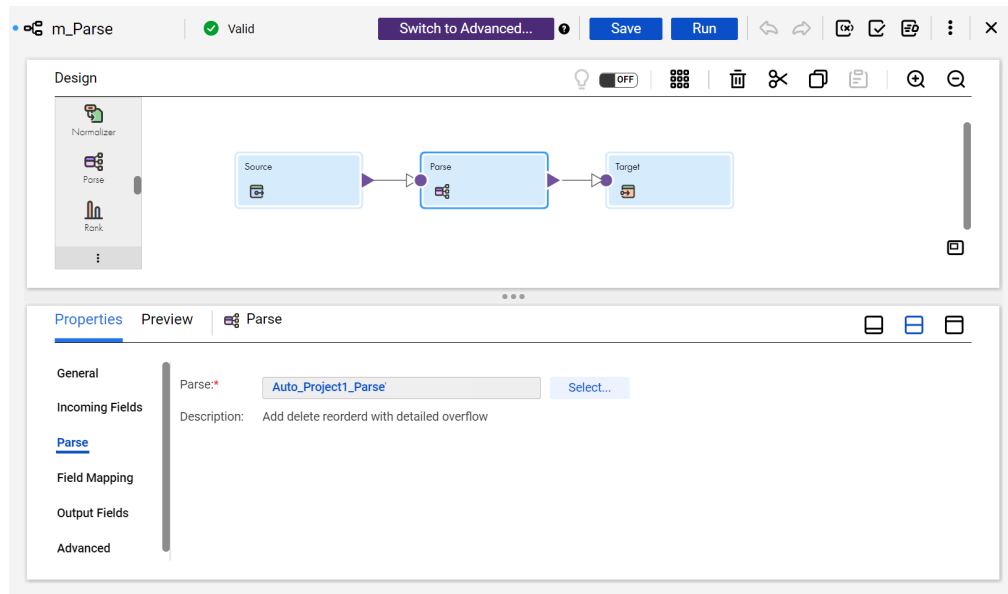
3. On the **Incoming Fields** tab, verify the incoming fields.

   By default, the transformation inherits all incoming fields from any connected upstream object in the mapping. You can define a field rule to limit or rename the incoming fields.

4. On the **Field Mapping** tab, connect an incoming field to the target field.

   The parse asset input names might reflect the name of an input field. If so, you can use the **Automap** options to connect the fields.

5. Verify the output field properties on the **Output Fields** tab.

   To learn more about the transformation output fields, see "Parse transformation output fields" on page 345.

6. You can optionally rename the Parse transformation and add a description on the **General** tab. You can also update the tracing level for the transformation on the **Advanced** tab. The default tracing level is Normal.

Note: If you update an asset in Data Quality after you add it to a transformation, you may need to synchronize the asset version in the transformation with the latest version. For more information about data quality asset synchronization, see "Synchronizing data quality assets" on page 116.

Parse transformation field mappings

Configure field mappings to define how data moves from the upstream transformation to the Parse transformation. Configure field mappings on the **Field Mapping** tab of the **Properties** panel.

Note: You map an incoming field to a single field on a parse asset.

You can configure the following field mapping options:

Field map options

   Method of mapping fields to the transformation.

Select one of the following options:

- Manual. Manually link an incoming field to a transformation input field. Removes links for any automatically mapped field.

- Automatic. Automatically link fields with the same name. Use when all of the fields that you want to link share the same name. You cannot manually link fields with this option.

- Completely Parameterized. Use a parameter to represent the field mapping.
  Choose the Completely Parameterized option when the parse asset in the transformation is parameterized or any upstream transformation in the mapping is parameterized.

- Partially Parameterized. Configure links in the mapping that you want to enforce and use a parameter to allow other fields to be mapped in the mapping task. Or, use a parameter to configure links in the mapping, and allow all fields and links to display in the task for configuration.

Parameter

Select the parameter to use for the field mapping, or create a new parameter. This option appears when you select Completely Parameterized or Partially Parameterized as the field map option. The parameter must be of type *field mapping*.

Do not use the same field mapping parameter in more than one Parse transformation in a single mapping.

Options

Controls how fields are displayed in the **Incoming Fields** and **Target Fields** lists.

Configure the following options:

- The fields that appear. You can show all fields, unmapped fields, or mapped fields.

- Field names. You can use technical field names or labels.

Automap

Links fields with matching names. Allows you to link matching fields and to manually configure other field mappings. The Automap options appear when you select the Manual or Partially Parameterized field map option.

You can map fields in the following ways:

- Exact Field Name. Data Integration matches fields of the same name.

- Smart Map. Data Integration matches fields with similar names. For example, if you have an incoming field `Cust_Name` and a target field `Customer_Name`, Data Integration automatically links the `Cust_Name` field with the `Customer_Name` field.

You can undo all automapped field mappings by clicking **Automap** > **Undo Automap**.

To unmap a single field, select the field to unmap and click **Actions** > **Unmap** on the context menu for the field. To unmap one or more fields that you selected, click **Unmap Selected** on the Target Fields context menu.

To clear all field mappings from the transformation, click **Clear Mapping** on the Target Fields context menu.

Parse transformation output fields

A Parse transformation creates the output fields that the parse asset specifies. The type and number of output fields depends on the parsing operations that the user configures in the parse asset. View the output fields on the **Output Fields** tab of the **Properties** panel.

The Parse transformation can create some or all of the following types of output field:

Parsed data fields

Contain data values that meet the parsing criteria that the asset defines.

Overflow fields

Contain data values that meet the parsing criteria but for which a corresponding output field is not available. The Parse transformation writes a value to an overflow field when all appropriate output fields for the value are already populated.

Unparsed field

A field that contains any value that does not meet the parsing criteria that the asset defines.

Rule and guidelines for Parse transformation output fields

Consider the following rules and guidelines when you review the output fields on the transformation:

- The type and number of output fields depends on the parsing operations that the user configures in the parse asset.

- When the asset specifies a regular expression or a dictionary, the transformation creates one or more output fields for the data that each regular expression or dictionary parses successfully.

  The user who configures the asset determines the number of output fields for each regular expression or dictionary operation. Each regular expression or dictionary operation is called a *step* in the asset configuration.

- When the asset specifies pattern-based parsing, the transformation creates a range of output fields that represent the types of information that the pattern logic might find.

  A pattern-based parsing operation can generate output fields for the following types of information:

  - Person names, such as first names, family names, name prefixes, and name suffixes.

  - Derived information, such as gender, formal greetings, and informal greetings.

  - Label values that represent the pattern that the parsing operation identified in the input data row. The Data Quality user can use the labels to enhance the pattern logic in the asset.

  The asset logic determines the number of output fields for parsed data.

- When the asset specifies a regular expression or a dictionary, the transformation may create a single overflow field for all overflow data. Or, the transformation may create an overflow field for each regular expression or dictionary that the asset defines. The user can update the asset properties to determine the policy for overflow fields.

  When the asset specifies a pattern parsing operation, the transformation may or may not create a single overflow field. The presence or absence of the overflow field depends on the locale that the asset specifies for the input data. For example, the Parse transformation creates an overflow field for pattern parsing operations when the asset specifies the locale as Portugal or Brazil. The Data Quality user sets the locale.

- The transformation creates a single field for all unparsed data when the asset specifies a regular expression or a dictionary.

  The transformation may or may not create an unparsed data field when the asset specifies a pattern parsing operation. The presence or absence of the unparsed data field in pattern-based parsing depends on the locale that the asset specifies for the input data.

Advanced properties

You can configure advanced properties for a Parse transformation. The advanced properties control the tracing level for session log messages.

You can configure the following property:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

CHAPTER 28

Python transformation

In advanced mode, you can use the Python transformation to define transformation functionality using the Python programming language. The Python transformation can be an active or passive transformation.

You can use the Python transformation to define simple or complex transformation functionality. You can also use the Python transformation to implement machine learning. For example, you can load a pre-trained model through a resource file and use the model to classify input data or to create predictions.

To create a Python transformation, you write the following types of Python code snippets:

- Pre-partition Python code that runs one time before it processes any input rows.
- Main Python code that runs when the transformation receives an input row.
- Post-partition Python code that runs after the transformation processes all input rows.

You can't use the Python transformation with a Graviton-enabled cluster. For more information about Graviton-enabled clusters, see the Administrator help.

Note: When you create a Python transformation, ensure that you review the Python code to verify that it is free from potentially unsafe active content such as queries, remote scripts, or data connections before you run the code in a mapping task.

Install and configure Python

Install and configure your Python code to run on a Secure Agent machine. Configure tasks such as setting the advanced session property and installing the Python packages.

Note: You can't create a custom Python installation when you use a serverless runtime environment, so these steps don't apply.

To install and configure Python, complete the following tasks:

1. To allow the Python transformation to use third-party libraries, make sure that the runtime environment has access to an installation of Python and the referenced resource files.
2. Install the prerequisite packages on the Secure Agent machine.
3. Download the version of Python that you want to install on the Secure Agent machine and extract the files.
4. Prepare the installation directory.
5. Compile and build the installation directory.
6. Set environment variables to allow the Secure Agent machine access to the installation.
7. Verify the installation contains the correct folders.

8. For custom installations, install the str2bool library.

9. If your installation uses third-party libraries, install them.

10. Copy the installation to the Secure Agent machine.

11. If your installation uses third-party libraries, verify their installation.

Step 1. Enable access to third-party libraries

To use the Python transformation with third-party libraries, the runtime environment must have access to an installation of Python and the resource files that you reference in the Python code.

When you install Python, you can include any third-party libraries you want to access in the Python code.

In the mapping task, use an advanced session property to specify the locations of the Python installation directory and the Python executable:

1. In the mapping task advanced session properties, click **Add**.

2. Select **advanced.custom.property** as the session property name.

3. For the session property value, enter the following text:

```
infaspark.pythontx.executorEnv.PYTHONHOME=$INFA_HOME/python&:infaspark.pythontx.exec=
$INFA_HOME/python/bin/python3
```

4. Click **Finish**.

Step 2. Install prerequisite packages

Before you install Python, make sure that the Secure Agent machine has the required packages.

The following table lists the commands to install a package on the Secure Agent machine:

Package	Command
OpenSSL	`sudo yum install openssl-devel`
zlib	`sudo yum install zlib-devel`
libffi	`sudo yum install libffi-devel`

Step 3. Download the Python distribution

Find the version of Python that you want to install on the Secure Agent machine and extract the files.

1. Navigate to the home directory.

2. Download the Python distribution onto the Secure Agent machine. For example, you can run the following command to get an installation of Python 3.6.5 from the internet:

```
wget https://www.python.org/ftp/python/3.6.5/Python-3.6.5.tgz
```

This command downloads the Python distribution in the home directory.

3. To untar the file, run the following command :

```
tar -xvf Python-3.6.5.tgz
```

The extracted files appear in the directory `~/Python-3.6.5`.

Step 4. Prepare a directory to install Python

Specify a location to contain the Python installation. To prepare an installation directory, run commands to create a directory and set it as the working directory.

1. To create a Python3 directory under the home directory, run the following command:

   ```
   mkdir Python3
   ```

2. To create an environment variable called $workingdir and set the value to the present working directory, run the following command :

   ```
   export workingdir=`pwd`
   ```

3. To verify the $workingdir environment variable, run the following command:

   ```
   echo $workingdir
   ```

Step 5. Install Python

Run commands to compile and build the Python installation.

1. Navigate to the `~/Python-3.6.5` directory where you extracted the Python distribution:

   ```
   cd Python-3.6.5
   ```

2. To compile Python under `$workingdir/Python3`, run the following command:

   ```
   ./configure --enable-shared --prefix=$workingdir/Python3
   ```

3. To build the Python installation, run the following command:

   ```
   make
   make install
   ```

Step 6. Set environment variables

Configure the Secure Agent machine to access the Python installation.

1. To set the PYTHONHOME environment variable, run the following command:

   ```
   export PYTHONHOME=$workingdir/Python3
   ```

   Notice that `$PYTHONHOME` is the directory where we installed Python.

2. To set the LD_LIBRARY_PATH environment variable, run the following command:

   ```
   export LD_LIBRARY_PATH=$PYTHONHOME/lib
   ```

3. To set the PATH environment variable, run the following command:

   ```
   export PATH=$PYTHONHOME/bin:$PATH
   ```

Step 7. Verify the Python installation

Confirm the location of the installation folders.

Verify that the Python installation directory contains the correct folders:

1. To change the directory to `$PYTHONHOME`, run the following command:

   ```
   cd $PYTHONHOME
   ```

2. To verify that `$PYTHONHOME` contains the following folders, run the `ls` command :

   ```
   bin
   include
   lib
   share
   ```

Step 8. Install str2bool library

Custom Python installations require the str2bool library.

Use the following pip command to install the str2bool library in `$PYTHONHOME`:

```
bin/pip3 install str2bool
```

Step 9. Optionally, install third-party libraries

Identify and install any additional libraries that you need in your Python installation.

If your Python installation uses third-party libraries, use the pip command to install them in `$PYTHONHOME`.

For example, you can run the following commands to install numpy and scikit-learn:

```
bin/pip3 install numpy
bin/pip3 install scikit-learn
```

Step 10. Copy Python to the Secure Agent machine

Copy the Python installation to the Secure Agent machine.

To complete the Python installation, run the following commands:

1. Create the following directory on the Secure Agent machine if it doesn't exist:
    ```
    <Secure Agent installation directory>/ext/python/
    ```
2. To copy the Python installation to the Secure Agent machine, run the following command:
    ```
    cp -r $PYTHONHOME/* <Secure Agent installation directory>/ext/python/
    ```
3. To check the contents of the Informatica domain, run the following command:
    ```
    ll <Secure Agent installation directory>/ext/python/
    ```
4. Verify that the contents include the following folders:
    ```
    bin
    include
    lib
    share
    ```

Step 11. Optionally, verify third-party library installations

If your Python installation uses third-party libraries, make sure they are installed on the Secure Agent machine.

1. To change directories to the Secure Agent machine, run the following command:
    ```
    cd <Secure Agent installation directory>/ext/python/
    ```
2. To invoke the Python interpreter, run the following command:
    ```
    bin/python3
    ```
3. Run the following Python code:
    ```
    import numpy
    import sklearn
    ```
4. Verify that the import statement is successful for each Python third-party library that you installed.

    Press `Ctrl+D` to exit the Python interpreter.

Python transformation fields

A Python transformation has incoming fields and output fields. Use incoming fields and output fields as variables in the Python code snippets that you define on the **Python** tab.

Add output fields on the **Output Fields** tab.

Add output fields for the output data that you want to pass to the downstream transformation. To add a field, click **Add Field**, and then enter the field name, data type, precision, and scale. You can also create output fields on the **Outputs** tab of the Python editor by clicking **Create New Field**.

After you add fields to the transformation, you can use the field names as variables in the Python code.

Use the following guidelines when you create output fields:

- A field name can contain only ASCII characters.
- Don't use a Python keyword for a field name. For example, don't use fields names such as `import`, `global`, or `class`.
- Don't use these reserved terms for a field name: `resourceJepFile` or `resourceFilesArray`.
- Set user-defined default values in the Python code.
  For example, you can enter `output_field = 'value'` to set the default value `'value'` for the output field `output_field`.

You can configure one or more input fields as partition keys. Data Integration uses the partition keys to repartition the data before the code runs. If you don't add an incoming fields as a partition key, the data is processed using its default partitioning scheme.

Data type conversion

A Python transformation converts transformation data types to Python data types based on the Python transformation field type.

When a Python transformation reads input rows, it converts incoming field data types to Python data types. When a Python transformation writes output rows, it converts Python data types to output field data types.

For example, the following processing occurs for an input field with the double data type:

1. The Python transformation converts the double data type in the incoming field to the Python float data type.
2. The Python transformation uses the value in the incoming field as the value for the Python float data type.
3. When the transformation generates the output row, it converts the Python float data type to the double data type.

The following table shows how the Python transformation maps transformation data types to Python data types:

Transformation data type	Python data type
Integer	Int
Decimal	Float
Double	Float

Transformation data type	Python data type
Timestamp	Datetime
String	Str
text	Text

Data types in input and output fields

In the Python transformation, the data types in corresponding incoming fields and output fields must be the same. If the data types are not the same, convert them in the Python code.

For example, you create an incoming field with the integer data type and an output field with the string data type. You define the Python code to process the data in the incoming field and write the data to the output field. In the Python code, you can use the `str()` function to convert the integer data type in the incoming field and write the output as a string data type in the output field.

Partition keys

You can use partition keys to define how to group data into partitions before the Python code runs.

You can configure one or more input fields as partition keys. The Python transformation uses the partition keys to repartition the data before the code runs. If you do not add an incoming field as a partition key, the data is processed using its default partitioning scheme.

Active and passive Python transformations

When you create a Python transformation, you specify how it generates output rows by defining the behavior as active or passive. You define the behavior on the **Advanced** tab. By default, the Python transformation is active.

A Python transformation handles output rows based on the behavior as follows:

- 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 incoming fields 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.

- 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.

Resource files

The Python transformation uses resource files and the Python code to define the transformation functionality. If you use a pre-trained model, you specify the pre-trained model as a resource file in the Python transformation.

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 Data Integration. 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.

Runtime environment

Add resource files based on the type of runtime environment. If you reference resource files in the Python code, add the resource files to the same directory. To maintain consistency, you can store the resource files in a dedicated folder named python_resources.

Consider the following guidelines:

- If the Secure Agent machine stops unexpectedly and the agent restarts on a different machine, you must add the Python installation and resource files to the same directory on the new machine.
- If you update the Python installation or resource files on the Secure Agent machine, the files take effect the next time that you run a job in advanced mode.
- To prevent long-running jobs from failing, do not update the files on the Secure Agent machine more than four times while you have jobs running.

Serverless runtime environment

Add resource files in the supplementary file location.

If you update the Python installation or resource files, you must redeploy the serverless runtime environment for the changes to take effect.

For more information about the supplementary file location, see *Administrator* in the Administrator help.

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, or define output variables.

Developing the Python code

To define the Python transformation functionality, you enter Python code snippets on the **Python** tab. Enter code snippets to reconstruct input variables, load a pre-trained model, define output variables, and define additional transformation functionality.

Enter Python code snippets in the following sections of the Python editor:

Pre-Partition Python Code

Defines code that can be interpreted one time and shared among all rows of data.

Use the **Pre-Partition Python Code** section to perform the following tasks:

- Declare import statements.
- Declare variables.
- Initialize variables.
- Define helper methods.

Main Python Code

Defines how the Python transformation behaves when it receives an input row while processing a partition. The Python transformation processes the code on the **Main Python Code** section for each partition and each row.

Post-Partition Python Code

Defines how the Python transformation behaves after it processes all input data in a partition. You can call the `generateRow()` method to generate output rows.

Use the following guidelines when you write Python code:

- Define variables before you use them. For example, you cannot reference a variable in the **Pre-Partition Python Code** section if the variable is defined in the **Main Python Code** section.
- Call the incoming field name to access incoming fields.
- The Python code must assign a value to each output field.
- To define how the transformation writes data from the incoming fields to output fields, set the output field to the value of the incoming field.
  For example, write `output_field = incoming_field` to write the data from the incoming field `incoming_field` to the output field `output_field`.
- To access resource files, use the variable `resourceFilesArray`. Specify the resource file using an index such as `resourceFilesArray[0]`.
- The Mapping Designer does not validate Python code.

Creating Python code snippets

To create Python code snippets that define transformation functionality, use the Python editor on the **Python** tab.

The following image shows the **Python** tab with the Python editor expanded:

1. Inputs and Outputs tabs. Use these tabs to add incoming fields and output fields as variables in the Python code snippets. The fields and methods displayed on these tabs vary based on which section of the code entry area is selected.

2. Go to list. Use to switch among the sections in the code entry area.

3. Minimize, Open Both, and Maximize icons. Use the Minimize and Maximize buttons to minimize and maximize the transformation properties. Use the Open Both icon to open the Mapping Designer canvas and the transformation properties at the same time.

4. Code entry area. Enter Python code snippets in the **Pre-Partition Python Code**, **Main Python Code**, and **Post-Partition Python Code** sections.

Tip: To expand the transformation properties so that you can see the code entry area more fully, click **Maximize**.

To create Python code snippets, perform the following tasks:

1. Select the section in which you want to enter a code snippet in the **Go to** list.

2. To access an incoming field or output field in the snippet, select the field on the **Inputs** or **Outputs** tab, and click **Add**.

   You can also create output fields on the **Outputs** tab by clicking **Create New Field**.

3. Write appropriate Python code based on the section.

Referencing a resource file

A resource file contains the resources that you use in the Python code.

In the Python transformation, enter the relative path of the resource file. For example, if the resource file is stored in `<Secure Agent installation directory>/ext/python/folder1/myfile`, then the relative path would be `/folder1/myfile`.

When you access a resource file in the Python code, you reference the array `resourceFilesArray[index]`. 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 when you specify several resource files, you reference the first resource file in the Python code using `resourceFilesArray[0]`. You reference the second resource using `resourceFilesArray[1]`.

Example: Add an ID column to nonpartitioned data

Your organization runs a solar thermal power system that uses sensors to monitor the health of the system. 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, perform the following tasks:

Step 1. Create a Python transformation.

Create a Python transformation. On the **Advanced** tab, set the behavior to Passive.

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, it contains the following incoming fields:

- SensorLocation
- LastReadingTime

Step 3. Create output fields.

Use the **Output Fields** tab in the Python transformation to create the output field `SensorID_out` to represent the ID column.

Additionally, create the following output fields to pass incoming data to downstream transformations:

- SensorLocation_out
- LastReadingTime_out

Step 4. Set the ID value for each row.

In the **Main Python Code** section, set the ID value for each row that is processed and write the data to the output fields 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 fields 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. The project is part of the wellness program at your organization. You want to use the information to better personalize the wellness program.

You can use the Python transformation 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 to determine which employee earns the highest salary in their department, perform the following tasks:

Step 1. Add a Python transformation to the mapping.

Create a Python transformation. On the **Advanced** tab, set the behavior to Active.

Step 2. Pass data to the Python transformation.

Pass the following fields 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, add the incoming field `DepartmentID` as a partition key on the **Partition Keys** tab.

Step 4. Create output fields.

Create the following output fields on the **Output Fields** tab 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 in the **Pre-Partition Python Code** section:

```
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 salary of the employee is higher than the maximum salary of the previous rows that have been processed. If the salary of the employee is higher, update the employee who has the maximum salary in the department.

Add the following code in the **Main Python Code** section:

```
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 files.

In the **Post-Partition Python Code** section of the **Python** 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 in the **Post-Partition Python Code** section:

```
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 fields in the Python transformation are linked directly to a Target 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

Example: Operationalize a pre-trained model

You work for a pharmaceutical company and you are studying data on flower formation in foxgloves 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.

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 Data Integration.

To operationalize the pre-trained model, complete the following tasks:

1. Create a mapping that contains a passive Python transformation and list the pre-trained model as a resource file.
2. Write a Python script that accesses the pre-trained model.
3. Pass the data on flower sepals and petals to the Python transformation to classify the data by foxglove species.

The following table shows sample sepals and petals data that you can pass to the Python transformation:

Name	Type	Precision
sepal_length	decimal	10
sepal_width	decimal	10
petal_length	decimal	10

Name	Type	Precision
petal_width	decimal	10
true_class	string	50

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:

- Path that is relative to the location on the Secure Agent machine.

    For example, if the resource file is under `<Secure Agent installation directory>/ext/python/folder1/foxgloveDataMLmodel.pkl`, then the relative path would be `/folder1/foxgloveDataMLmodel.pkl`.

- The supplementary file location for a serverless runtime environment.

    `/data/home/dtmqa/data/foxgloveDataMLmodel.pkl`

Python Code

Specify the Python code in the **Pre-Partition Python Code** and **Main Python Code** sections.

Use the **Pre-Partition Python Code** section to import libraries, load the resource file, and initialize variables.

For example, you might enter the following code in the **Pre-Partition Python Code** section:

```
from sklearn import svm
from sklearn.externals import joblib
import numpy as np
clf = joblib.load(resourceFileArrays[0])
classes = ['common', 'woolly']
```

Use the **Main Python Code** section to define how the Python transformation uses the pre-trained model to evaluate each row of data.

For example, you might enter the following code in the **Main Python Code** section:

```
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
```

CHAPTER 29

Rank transformation

The Rank transformation selects the top or bottom range of data. Use the Rank transformation to return the largest or smallest numeric values in a group. You can also use the Rank transformation to return strings at the top or bottom of the mapping sort order.

For example, you can use a Rank transformation to select the top 10 customers by region. Or, you might identify the three departments with the lowest expenses in salaries and overhead.

The Rank transformation differs from the transformation functions MAX and MIN because the Rank transformation returns a group of values, not just one value. While the SQL language provides many functions designed to handle groups of data, identifying top or bottom strata within a set of rows is not possible using standard SQL functions.

The Rank transformation is an active transformation because it can change the number of rows that pass through it. For example, you configure the transformation to select the top 10 rows from a source that contains 100 rows. In this case, 100 rows pass into the transformation but only 10 rows pass from the Rank transformation to the downstream transformation or target.

When you run a mapping that contains a Rank transformation, Data Integration caches input data until it can perform the rank calculations.

Ranking string values

You can configure the Rank transformation to return the top or bottom values of a string field. To sort string fields, Data Integration uses the session sort order configured for the mapping task.

You set the **Session Sort Order** property in the advanced session properties for the mapping task. You can select binary or a specific language such as Danish or Spanish. If you select binary, Data Integration calculates the binary value of each string and sorts the strings using the binary values. If you select a language, Data Integration sorts the strings alphabetically using the sort order for the language.

The following image shows the **Session Sort Order** property in the advanced session properties for a mapping task:
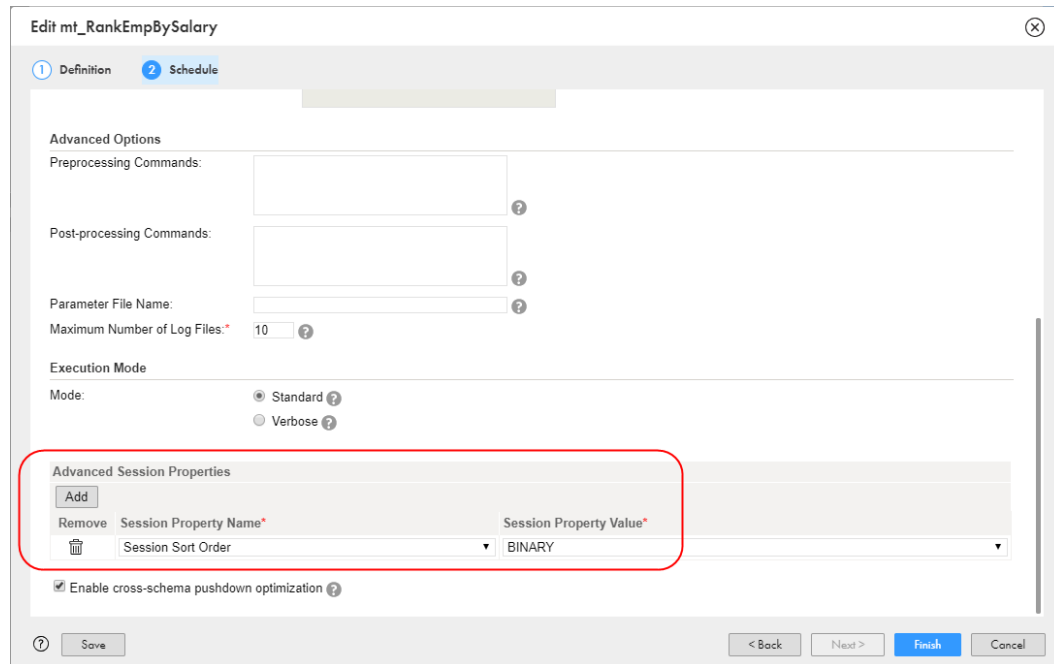
Rank caches

When you run a mapping that contains a Rank transformation, Data Integration creates data and index cache files to run the transformation. By default, Data Integration stores the cache files in the directory entered in the Secure Agent $PMCacheDir property for the Data Integration Server.

You can change the cache directory and cache sizes on the **Advanced** tab of the Rank transformation.

Data Integration creates the following caches for the Rank transformation:

- Data cache that stores row data based on the group by fields.
- Index cache that stores group values as configured in the group by fields.

When you run a mapping that contains a Rank transformation, Data Integration compares an input row with rows in the data cache. If the input row out-ranks a cached row, Data Integration replaces the cached row with the input row. If you configure the Rank transformation to rank across multiple groups, Data Integration ranks incrementally for each group that it finds.

If you create multiple partitions in the Source transformation, Data Integration creates separate caches for each partition.

Defining a Rank transformation

To define a Rank transformation, you drag the transformation into the Mapping Designer, configure fields, and configure the transformation properties.

1. In the Mapping Designer, drag a Rank transformation from the transformation palette onto the canvas and connect it to the upstream and downstream transformations.

2. Configure the transformation fields.

   By default, the transformation inherits all incoming fields from the upstream transformation. If you do not need to use all of the incoming fields, you can configure field rules to include or exclude certain fields.

3. Configure the rank properties.

   Select the field that you want to rank by, the rank order, and the number of rows to rank.

4. Optionally, configure rank groups.

   You can configure the Rank transformation to create groups for ranked rows.

5. Optionally, configure the transformation advanced properties.

   You can update the cache properties, tracing level for log messages, transformation scope, case-sensitivity for string comparisons, and whether the transformation is optional.

Rank transformation fields

A Rank transformation inherits incoming fields from the upstream transformation. When you create a Rank transformation, Data Integration also creates a RANKINDEX output field.

The Rank transformation uses the following fields:

Incoming fields

Incoming fields appear on the **Incoming Fields** tab. By default, the Rank transformation inherits all incoming fields from the upstream transformation. If you do not need to use all of the incoming fields, you can define field rules to include or exclude certain fields. For more information about field rules, see "Field rules" on page 26.

RANKINDEX

After the Rank transformation identifies all rows that belong to a top or bottom rank, it assigns rank index values. Data Integration creates the RANKINDEX field to store the rank index value for each row in a group.

For example, you create a Rank transformation to identify the five retail stores in the company with the highest monthly gross sales. The store with the highest sales receives a rank index of 1. The store with the next highest sales receives a rank index of 2, and so on. If two stores have the same gross sales, they receive the same rank index, and the transformation skips the next rank index.

For example, in the following data set, the Long Beach and Anaheim stores have the same gross sales, so they are assigned the same rank index:

RANKINDEX	STORE	GROSS_SALES
1	Long Beach	100000

RANKINDEX	STORE	GROSS_SALES
1	Anaheim	100000
3	Riverside	90000
4	Chula Vista	80050

When measuring a bottom rank, such as the 10 lowest priced products in the inventory, the Rank transformation assigns a rank index from lowest to highest. Therefore, the least expensive item receives a rank index of 1.

The RANKINDEX is an output field. It appears on the **Incoming Fields** tab of the downstream transformation.

Defining rank properties

When you define the rank properties for a Rank transformation, you select the field to rank by, specify the rank order, and specify number of rows to rank by. Define rank properties on the **Rank** tab.

The following image shows the **Rank** tab:

Configure the following properties:

Rank By

Specify the field that you want to use for ranking in the **Rank By** field.

For example, you create a Rank transformation to rank the top 10 employees in each department based on salary. The EMP_SALARY field contains the salary for each employee. Select EMP_SALARY as the **Rank By** field.

In advanced mode, the **Rank By** field can't be a binary data type.

Rank Order

Specify the rank order in the **Rank Order** field. Select **Top** or **Bottom**.

Number of Rows

Specify the number of rows to include in each rank group in the **Number of Rows** field. For example, to rank the top 10 employees in each department based on salary, enter 10 in the **Number of Rows** field.

You can use a parameter to specify the number of rows. To use a parameter, select **Parameterized** in the **Parameterize Number of Rows** field, and enter the parameter in the **Parameter** field.

Defining rank groups

You can configure the Rank transformation to define groups for ranked rows. For example, to select the 10 most expensive items by manufacturer, define a rank group for the manufacturer. Define rank groups on the **Group By** tab.

To define rank groups, select one or more incoming fields as **Group By Fields**. For each unique value in a rank group, the transformation creates a group of rows that fall within the rank definition (top or bottom, and number in each rank).

Note: Define rank groups to improve performance in a mapping in advanced mode that processes a large volume of data. When you define rank groups, processing is distributed across multiple worker nodes. If you do not define rank groups, the data is processed on one worker node. Depending on the volume of data, performance is impacted and the mapping might fail due to a lack of storage space on the EBS volume that is attached to the worker node.

For example, you create a Rank transformation that ranks the top five salespersons grouped by quarter. The rank index numbers the salespeople from 1 to 5 for each quarter as follows:

RANKINDEX	SALES_PERSON	SALES	QUARTER
1	Alexandra B.	10000	1
2	Boris M.	9000	1
3	Chanchal R.	8000	1
4	Dong T.	7000	1
5	Elias M.	6000	1
1	Elias M.	11000	2
2	Boris M.	10000	2
3	Alexandra B.	9050	2
4	Dong T.	7500	2
5	Frances Z.	6900	2

If you define multiple rank groups, the Rank transformation groups the ranked rows in the order in which the fields are selected in the **Group By Fields** list.

Advanced properties

Configure advanced properties to define how the Rank transformation processes data. Configure advanced properties on the **Advanced** tab.

The properties that are available vary based on the mapping mode.

Configure the following properties:

Property	Description
Cache Directory	Directory where Data Integration creates the data cache and index cache files. By default, Data Integration stores the cache files in the directory entered in the Secure Agent $PMCacheDir property for the Data Integration Server. If you change the cache directory, verify that the directory exists and contains enough disk space for the cache files. To increase performance during cache partitioning, enter multiple directories separated by semicolons. Cache partitioning creates a separate cache for each partition that processes the transformation. Default is $PMCacheDir.
Rank Data Cache Size	Data cache size for the transformation. Select one of the following options: - Auto. Data Integration sets the cache size automatically. If you select Auto, you can also configure a maximum amount of memory for Data Integration to allocate to the cache. - Value. Enter the cache size in bytes. Default is Auto.
Rank Index Cache Size	Index cache size for the transformation. Select one of the following options: - Auto. Data Integration sets the cache size automatically. If you select Auto, you can also configure a maximum amount of memory for Data Integration to allocate to the cache. - Value. Enter the cache size in bytes. Default is Auto.
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Transformation Scope	The method in which Data Integration applies the transformation logic to incoming data. Select one of the following values: - Transaction. Applies the transformation logic to all rows in a transaction. Choose Transaction when the results of the transformation depend on all rows in the same transaction, but not on rows in other transactions. - All Input. Applies the transformation logic to all incoming data. When you choose All Input, Data Integration drops transaction boundaries. Select All Input when the results of the transformation depend on all rows of data in the source. Default is All Input.

Property	Description
Case Sensitive String Comparison	Specifies whether Data Integration uses case-sensitive string comparisons when it ranks strings. To ignore case in strings, disable this option. Default is enabled. On an advanced cluster, comparisons are always case-sensitive.
Optional	Determines whether the transformation is optional. If a transformation is optional and there are no incoming fields, the mapping task can run and the data can go through another branch in the data flow. If a transformation is required and there are no incoming fields, the task fails. For example, you configure a parameter for the source connection. In one branch of the data flow, you add a transformation with a field rule so that only Date/Time data enters the transformation, and you specify that the transformation is optional. When you configure the mapping task, you select a source that does not have Date/Time data. The mapping task ignores the branch with the optional transformation, and the data flow continues through another branch of the mapping. Default is enabled.

Hierarchical data in advanced mode

In advanced mode, you can use hierarchical fields that represent an array, map, or struct.

You can use the hierarchical fields as pass-through fields.

Consider the following guidelines when you pass hierarchical fields to the Rank transformation:

- You cannot rank data using a hierarchical field.
- You cannot define a rank group using a hierarchical field.

Rank transformation example

You store customer data in a relational database table. You want send a promotion to the top three customers in each tier. Use a Rank transformation to rank the customers in each tier by order amount.

The source, mapping, and target are configured as follows:

Source Data

The following table shows the source data:

```
CUST_ID        CUST_TIER      CUST_NAME                    ORDER_AMT

10110102       Gold           Brosseau, Derrick            63508.12

10110109       Platinum       Acheson, Jeff                139824.15

10110143       Silver         Cudell, Bob                  49614.00

10110211       Silver         Amico, Paul                  47677.30

10110215       Platinum       Bergeron, Kim                148871.25
```

CUST_ID	CUST_TIER	CUST_NAME	ORDER_AMT
10110224	Silver	Madison, Shelley	40497.10
10110235	Gold	Anderson, Rick	50429.27
10110236	Silver	Tucker, Paul	42585.00
10110237	Silver	Smith, Robert	38563.98
10110393	Gold	Washington, Rochelle	73767.96
10110425	Gold	Nguyen, Trang	65522.25
10110434	Silver	Keane, Thomas	38055.40
10110436	Platinum	Catherwood, Jennifer	117107.44
10110442	Platinum	Charest, Walter	126618.60
10110458	Gold	Coutts, Sylvain	70646.32
10110497	Platinum	Zheng, Wei	191422.00
10110506	Gold	Gonzales, Roberto	79342.90
10110526	Gold	Vanelo, Susan	81978.06
10110528	Platinum	Abedini, John	136506.32
10110530	Silver	Sousa, Maria	10155.42

Mapping Configuration

Configure the mapping as shown in the following image:
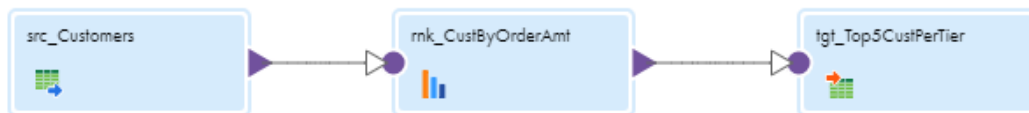
Configure the Rank transformation as follows:

Rank tab

Configure the following properties:

Field	Value
Rank By	ORDER_AMT
Rank Order	Top
Parameterize Number of Rows	Not Parameterized
Number of Rows	3

Group By tab

Select CUST_TIER as the group by field.

Target Data

The following table shows the data that is written to the target when you run the mapping:

RANKINDEX	CUST_ID	CUST_TIER	CUST_NAME	ORDER_AMT
1	10110526	Gold	Vanelo, Susan	81978.06
2	10110506	Gold	Gonzales, Roberto	79342.90
3	10110393	Gold	Washington, Rochelle	73767.96
1	10110497	Platinum	Zheng, Wei	191422.00
2	10110215	Platinum	Bergeron, Kim	148871.25
3	10110109	Platinum	Acheson, Jeff	139824.15
1	10110143	Silver	Cudell, Bob	49614.00
2	10110211	Silver	Amico, Paul	47677.30
3	10110236	Silver	Tucker, Paul	42585.00

CHAPTER 30

Router transformation

The Router transformation is an active transformation that you can use to apply a condition to incoming data.

In a Router transformation, Data Integration uses a filter condition to evaluate each row of incoming data. It tests the conditions of each user-defined group before processing the default group. If a row meets more than one group filter condition, Data Integration passes the row multiple times. You can either drop rows that do not meet any of the conditions or route those rows to a default output group.

If you need to test the same input data based on multiple conditions, use a Router transformation in a mapping instead of creating multiple Filter transformations to perform the same task.

The following table compares the Router transformation to the Filter transformation:

Options	Router	Filter
Conditions	Test for multiple conditions in a single Router transformation	Test for one condition per Filter transformation
Handle rows that do not meet the condition	Route rows to the default output group or drop rows that do not meet the condition	Drop rows that do not meet the condition
Incoming data	Process once with a single Router transformation	Process in each Filter transformation

The following figure shows a mapping with a Router transformation that filters data based on region and routes it to a different target, either NA, EMEA, or APAC. The transformation routes data for other regions to the default target:
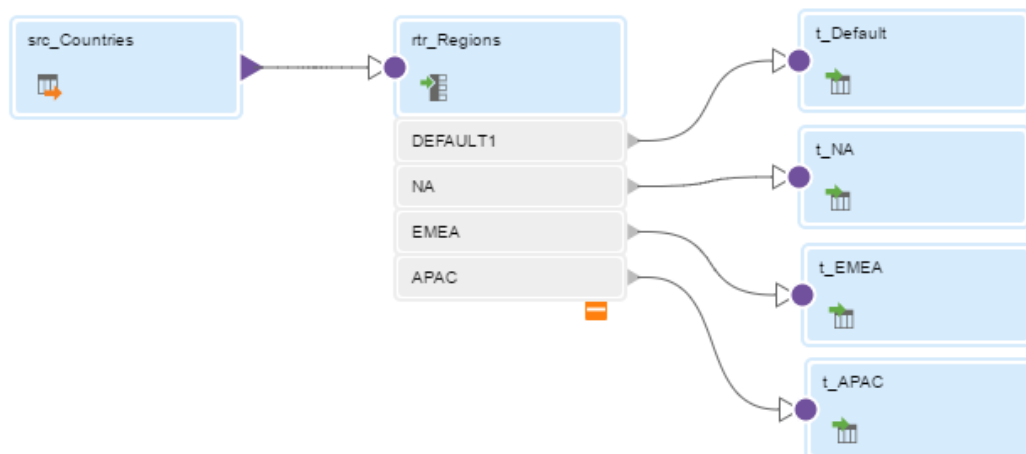
Working with groups

You use groups in a Router transformation to filter the incoming data.

Data Integration uses the filter conditions to evaluate each row of incoming data. It tests the conditions of each user-defined group before processing the default group. If a row meets more than one group filter condition, Data Integration passes the row to multiple groups.

A Router transformation has the following types of groups:

Input

Data Integration copies properties from the input group fields to create the fields for each output group.

Output

There are two types of output groups:

- User-defined groups. Create a user-defined group to test a condition based on incoming data. A user-defined group consists of output ports and a group filter condition. Create one user-defined group for each condition that you want to specify. Data Integration processes user-defined groups that are connected to a transformation or a target.

- Default group. The default group captures rows that do not satisfy any group condition. You cannot edit, delete, or define a group filter condition for the default group. If all of the conditions evaluate to FALSE, Data Integration passes the row to the default group. If you want to drop rows that do not satisfy any group condition, do not connect the default group to a transformation or a target.

You can modify a user-defined output group name. Click the row to open the **Edit Output Group** dialog box.

Guidelines for connecting output groups

Note the following guidelines when you connect Router transformation output groups to downstream transformations:

- You can connect each output group to one or more transformations or targets.

- You cannot connect more than one group to one target or a single input group transformation.

- You can connect more than one output group to a downstream transformation if you connect each output group to a different input group.

- If you want Data Integration to drop all rows in the default group, do not connect it to a transformation or a target in a mapping.

Group filter conditions

You can test data based on one or more group filter conditions. You can enter any expression that returns a single value. You can also specify a constant for the condition. In advanced mode, the filter condition must evaluate to a numeric result.

A group filter condition returns TRUE or FALSE for each row that passes through the transformation, depending on whether a row satisfies the specified condition. Zero (0) is the equivalent of FALSE, and any non-zero value is the equivalent of TRUE.

When the task runs, Data Integration handles the data in the following ways:

- Passes the rows of data that evaluate to TRUE to each transformation or target that is associated with each user-defined group.

   The Router transformation can pass data through multiple output groups. For example, if the data meets three output group conditions, the Router transformation passes the data through three output groups.

- Passes the row to the default group if all of the conditions evaluate to FALSE.

You cannot configure a group filter condition for the default group. However, you can add an Expression transformation to perform a calculation and handle the rows in the default group.

Configuring a group filter condition

Configure a group filter condition for each user-defined output group.

1. Connect the source to the Router transformation.
2. Click the **Output Groups** tab.
3. Click the + sign and add the group you want to configure.
4. Click **Configure**.
5. In the **Edit Filter Condition** dialog box, select one of the following filter condition types:
   - **Simple**. Enter a simple condition based on available field names and operators.
   - **Advanced**. Select this option to open the expression editor and validate the syntax for a complex condition.
   - **Completely Parameterized**. Select this option to base the filter condition on an input parameter.

   The following image shows the **Edit Filter Condition** dialog box:

Edit Filter Condition for NA			⊗
Filter Condition:	Simple ▾		
Filter Conditions			⊕
Field Name	Operator	Value	
Region	=	NA	🗑
			OK Cancel

6. Click the + sign to add a row for each condition that you want to apply to this group.
7. Choose a Field Name, Operator, and Value for each condition.
8. Click **OK** to save the conditions.

Advanced properties

You can configure advanced properties for a Router transformation. The advanced properties control settings such as the tracing level for session log messages and whether the transformation is optional or required.

The properties that are available vary based on the mapping mode.

You can configure the following properties:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Optional	Determines whether the transformation is optional. If a transformation is optional and there are no incoming fields, the mapping task can run and the data can go through another branch in the data flow. If a transformation is required and there are no incoming fields, the task fails. For example, you configure a parameter for the source connection. In one branch of the data flow, you add a transformation with a field rule so that only Date/Time data enters the transformation, and you specify that the transformation is optional. When you configure the mapping task, you select a source that does not have Date/Time data. The mapping task ignores the branch with the optional transformation, and the data flow continues through another branch of the mapping.

Hierarchical data in advanced mode

In advanced mode, you can use hierarchical fields that represent an array, map, or struct.

You can use the hierarchical fields as pass-through fields. You can also use hierarchical fields in an advanced filter condition or use a hierarchical field with a complex operator to access primitive child fields in the filter condition. For more information about complex operators, see *Function Reference*.

Consider the following guidelines when you use hierarchical fields in a filter condition:

- You cannot use a hierarchical field in a simple filter condition.
- Do not use a parameter for the filter condition.

Router transformation examples

You can use a Router transformation to complete the following tasks:

- Group data by different country attributes and route each group to different target tables based on conditions that test for the region.
- Group inventory items by different price categories and route each group to different target tables based on conditions that test for low, medium, and high prices.

Example 1: Route data to different targets by region.

Your source includes data for customers in different regions. You want to configure a marketing campaign with variants for customers in the North America region, the Europe, Middle East, and Africa region, and the

Asia Pacific region. All other customers see the default ad campaign. Use a Router transformation to route the data to four different Target transformations.

The following figure shows a mapping with a Router transformation that filters data based on these conditions:
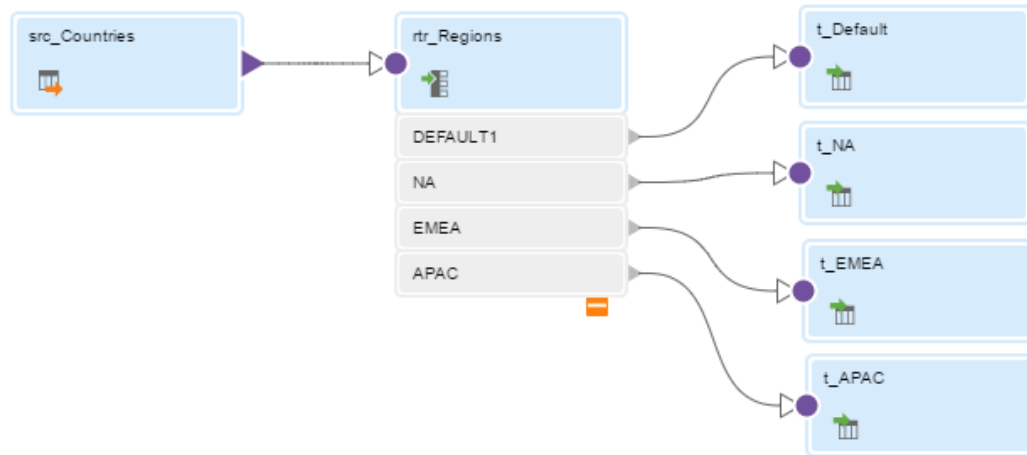
Create three output groups and specify the group filter conditions on the **Output Groups** tab as shown in the following table:

Group Name	Condition
NA	region = 'NA'
EMEA	region = 'EMEA'
APAC	region = 'APAC'

The default group includes data for all customers that are not in the NA, EMEA, or APAC region.

Example 2: Route some rows to multiple output groups.

The Router transformation passes data through all output groups that meet the filter condition. In the following example, the conditions test for a price threshold, but the filter conditions for the two output groups overlap:

Group Name	Condition
PriceGroup1	item_price > 100
PriceGroup2	item_price > 500

When the Router transformation processes an input row with item_price=510, it routes the row to both output groups.

If you want to pass the data through a single output group, define the filter conditions so that they do not overlap. For example, you might change the filter condition for PriceGroup1 to item_price <= 500.

CHAPTER 31

Rule Specification transformation

The Rule Specification transformation adds a rule specification asset that you created in Data Quality to a mapping.

A rule specification is a set of one or more logical operations that analyze data according to business criteria that you define. The rule specification generates an output that indicates whether the data satisfies the business criteria. The rule specification can also update the data that it analyzes. You define the logical operations as IF/THEN/ELSE statements in Data Quality.

Each Rule Specification transformation can contain a single rule specification. You can add multiple Rule Specification transformations to a mapping.

Use a Rule Specification transformation to accomplish the following goals:

- Define the types of data that a business data set contains.
- Define a set of conditions that the business data must satisfy.
- Define the actions to take when the data satisfies the conditions of the business rule.
- Define the actions to take when the data fails to satisfy the conditions of the business rule.

A Rule Specification transformation is similar to a Mapplet transformation, as it allows you to add data analysis and data transformation logic that you designed elsewhere to a mapping. Like mapplets, rule specifications are reusable assets. A Rule Specification transformation shows incoming and outgoing fields. It does not display the logic that the rule specification contains or allow to you edit the rule specification. To edit the rule specification, open it in Data Quality.

To use the Rule Specification transformation, you need the appropriate license.

Rule Specification transformation configuration

When you configure a Rule Specification transformation in a mapping, you first select the rule specification asset that contains the logic to include in the mapping. Next, you configure the incoming fields and the field mappings. Then, you verify the output fields.

To configure the transformation, complete the following tasks:

1. Connect the Rule Specification transformation to a Source transformation or other upstream object.
2. On the **Rule Specification** tab, select the rule specification that you want to include in the transformation.

The following image shows the options that you use to select the rule specification:
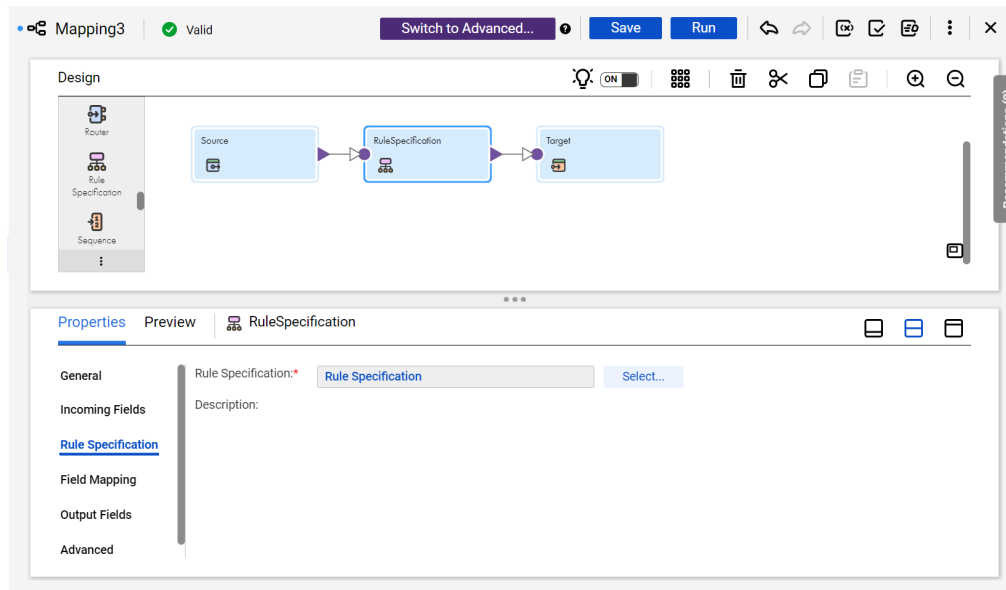
3. On the **Incoming Fields** tab, configure the incoming fields.

   By default, the transformation inherits all incoming fields from any connected upstream object in the mapping. You can define a field rule to limit or rename the incoming fields.

4. On the **Field Mapping** tab, configure the field mapping options to connect the data from the incoming fields to the target fields on the rule specification asset.

   The rule specification inputs might already reflect the names of the input fields in the data. If so, you can use the **Automap** options to connect the fields. For more information about connecting to upstream object fields, see "Rule Specification transformation field mappings" on page 377.

5. Verify the rule specification output fields on the **Output Fields** tab.

6. You can optionally rename the Rule Specification transformation and add a description on the **General** tab. You can also update the tracing level for the transformation on the **Advanced** tab. The default tracing level is Normal.

Note: Consider the following rules and guidelines when you use a parameter to identify the rule specification that the transformation uses:

- If you use a parameter to identify the rule specification, you must use a parameter to define the field mappings on the **Field Mapping** tab. The parameter must be of type *string*.

- If you use a parameter to identify the rule specification, any downstream object in the mapping that connects to the Rule Specification transformation must be completely parameterized.

Note: If you update an asset in Data Quality after you add it to a transformation, you may need to synchronize the asset version in the transformation with the latest version. For more information about data quality asset synchronization, see "Synchronizing data quality assets" on page 116.

Rule Specification transformation field mappings

Configure field mappings to define how data moves from the upstream transformation to the Rule Specification transformation. Configure field mappings on the **Field Mapping** tab of the **Properties** panel.

You can configure the following field mapping options:

Field map options

Method of mapping fields to the transformation.

Select one of the following options:

- Manual. Manually link incoming fields to transformation input fields. Removes links for automatically mapped fields.

- Automatic. Automatically link fields with the same name. Use when all of the fields that you want to link share the same name. You cannot manually link fields with this option.

- Completely Parameterized. Use a parameter to represent the field mapping. In the task, you can configure all field mappings.
  Choose the Completely Parameterized option when the rule specification in the transformation is parameterized or any upstream transformation in the mapping is parameterized

- Partially Parameterized. Configure links in the mapping that you want to enforce and use a parameter to allow other fields to be mapped in the mapping task. Or, use a parameter to configure links in the mapping, and allow all fields and links to display in the task for configuration.

Parameter

Select the parameter to use for the field mapping, or create a new parameter. This option appears when you select Completely Parameterized or Partially Parameterized as the field map option. The parameter must be of type *field mapping*.

Do not use the same field mapping parameter in more than one Rule Specification transformation in a single mapping.

Options

Controls how fields are displayed in the **Incoming Fields** and **Rule Specification Input Fields** lists.

Configure the following options:

- The fields that appear. You can show all fields, unmapped fields, or mapped fields.

- Field names. You can use technical field names or labels.

Automap

Links fields with matching names. Allows you to link matching fields and to manually configure other field mappings. The Automap options appear when you select the Manual or Partially Parameterized field map option.

You can map fields in the following ways:

- Exact Field Name. Data Integration matches fields of the same name.

- Smart Map. Data Integration matches fields with similar names. For example, if you have an incoming field `Cust_Name` and a target field `Customer_Name`, Data Integration automatically links the `Cust_Name` field with the `Customer_Name` field.

You can use both Exact Field Name and Smart Map in the same field mapping. For example, use Exact Field Name to match fields with the same name and then use Smart Map to map fields with similar names.

You can undo all automapped field mappings by clicking **Automap** > **Undo Automap**.

To unmap a single field, select the field to unmap and click **Actions** > **Unmap** on the context menu for the field. To unmap one or more fields that you selected, click **Unmap Selected** on the Rule Specification Input Fields context menu.

To clear all field mappings from the transformation, click **Clear Mapping** on the Rule Specification Input Fields context menu.

Rule Specification transformation output fields

A Rule Specification transformation displays an output field for each rule set in the rule specification. View the output fields on the **Output Fields** tab of the **Properties** panel. The transformation outputs include the output from each rule set in the rule specification.

The tab displays the name, type, precision, and scale for each output field. The output field names are the names of the rule sets in the rule specification.

You cannot edit the output field properties in the Rule Specification transformation. To edit the properties, open the rule specification in Data Quality.

Advanced properties

You can configure advanced properties for a Rule Specification transformation. The advanced properties control the tracing level for session log messages.

You can configure the following property:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

CHAPTER 32

Sequence transformation

The Sequence transformation is a passive and connected transformation that generates numeric values. Use the Sequence transformation to create unique primary key values, replace missing primary keys, or cycle through a sequential range of numbers.

The Sequence transformation contains pass-through fields and two output fields, NEXTVAL and CURRVAL. In advanced mode, the transformation contains one output field, NEXTVAL. You can connect the output fields to one or more downstream transformations.

The mapping task generates a numeric sequence of values each time the mapped fields enter a connected transformation. You set the range of numbers in the Mapping Designer. You can change the initial number in the sequence when you run the task.

After the task completes, you can see the current value and the initial value for a Sequence transformation in the mapping task details.

Sequence transformation uses

Use a Sequence transformation to generate a sequence of numbers in the following ways:

Generate a sequence of unique numbers.

The sequence begins with the Initial Value that you specify.

Generate a cyclic sequence of numbers.

You can establish a range of values for the Sequence transformation. If you use the cycle option, the Sequence transformation repeats the range when it reaches the end value. For example, if you set the sequence range to start at 10 and end at 50, and you set an increment value of 10, the Sequence transformation generates the following values: 10, 20, 30, 40, 50. The sequence starts over again at 10.

Doesn't apply in advanced mode or SQL ELT mode.

Continue an existing sequence of numbers.

Each time you run the mapping task, the task updates the value to reflect the last-generated value plus the Increment By value. If you want the numbering to start over each time you run the task, you can enable the Reset configuration property.

Generate a sequence of unique numbers for multiple partitions using the same Sequence transformation.

You can specify the number of sequential values the mapping task caches at a time so that the task doesn't generate the same numbers for each partition. You can't generate a cyclic sequence of numbers when you use the same Sequence transformation for multiple partitions.

Doesn't apply in advanced mode or SQL ELT mode.

Sequence output fields

The Sequence transformation has two output fields, NEXTVAL and CURRVAL. In advanced mode and SQL ELT mode, the transformation has one output field, NEXTVAL. You can't edit or delete these fields.

You can connect a Sequence transformation to any transformation. If the mapping contains both fields, you don't need to map both of the output fields. If you don't map one of the output fields, the mapping task ignores the unmapped field.

In mappings in SQL ELT mode, you can only connect a Sequence transformation directly to a downstream Expression transformation or target.

NEXTVAL field

Use the NEXTVAL field to generate a sequence of numbers based on the Initial Value and Increment By properties.

Map the NEXTVAL field to an input field in a Target transformation or other downstream transformation to generate a sequence of numbers. If you don't configure the Sequence transformation to cycle through the sequence, the NEXTVAL field generates sequence numbers up to the configured End Value.

If you map the NEXTVAL field to multiple transformations, the mapping task generates the same sequence or a unique sequence of numbers for each downstream transformation based on the mapping type and whether incoming fields are disabled.

The following table lists the situations where the Sequence transformation generates the same sequence or a unique sequence of numbers:

Mapping type	Incoming fields are…	Same sequence or unique sequence?
Mapping	Not disabled	Same sequence
Mapping	Disabled	Unique sequence*
Mapping (Advanced mode)	-	Same sequence
Mapping (SQL ELT mode)	-	Same sequence
* To generate the same sequence of numbers when incoming fields are disabled, you can place an Expression transformation between the Sequence transformation and its downstream transformations to stage the sequence of numbers.		

CURRVAL field

The CURRVAL field value is the NEXTVAL value plus the Increment By value. For example, if the Initial Value is 1 and Increment By is 1, the mapping task generates the following values for NEXTVAL and CURRVAL:

NEXTVAL	CURRVAL
1	2
2	3
3	4

NEXTVAL	CURRVAL
4	5
5	6

Typically, you map the CURRVAL field when the NEXTVAL field is already mapped to a downstream transformation in the map. If you map the CURRVAL field without mapping the NEXTVAL field, the mapping task generates the same number for each row.

Sequence properties

Configure Sequence properties to define how the Sequence transformation generates numeric values.

The properties that are available vary based on the mapping mode.

Configure the following Sequence properties on the **Sequence** tab:

Property	Description
Use Shared Sequence	Enable to generate sequence values using a shared sequence. When enabled, the sequence starts with the Current Value of the shared sequence. For information about shared sequences, see *Components*. Default is disabled.
Increment By	The difference between two consecutive values in a generated sequence. For example, if Increment By is 2 and the existing value is 4, then the next value generated in the sequence will be 6. Default is 1. Maximum value is 2,147,483,647.
End Value	Maximum value that the mapping task generates. If the sequence reaches this value during the task run and the sequence is not configured to cycle, the run fails. Maximum value is 9,223,372,036,854,775,807. If you connect the NEXTVAL field to a downstream integer field, set the End Value to a value no larger than the integer maximum value. If the NEXTVAL exceeds the data type maximum value for the downstream field, the mapping run fails. In advanced mode, set the end value to at least the maximum number of rows that you process.
Initial Value	The value you want the mapping task to use as the first value in the sequence. If you want to cycle through a series of values, the value must be greater than or equal to the Start Value and less than the End Value. Default is 1.
Cycle	If enabled, the mapping task cycles through the sequence range. If disabled, the task stops the sequence at the configured End Value. The session fails if the task reaches the End Value and still has rows to process. Default is disabled.

Property	Description
Cycle Start Value	Start value of the generated sequence that you want the mapping task to use if you use the Cycle option. When the sequence values reach the End Value, they cycle back to this value. Default is 0. Maximum value is 9,223,372,036,854,775,806.
Number of Cached Values	Number of sequential values the mapping task caches for each run. Each subsequent run uses a new batch of values. The task discards unused sequences for the batch. The mapping task updates the repository as it caches each value. When set to 0, the task does not cache values. Use this option when multiple partitions use the same Sequence transformation at the same time to ensure each partition receives unique values. Default is 0. This option is not available when the Cycle property is enabled. In advanced mode, you can't set the number of cached values. However, your organization administrator can optimize how values are cached. For more information, contact Informatica Global Customer Support. (Ref 619019)
Reset	If enabled, the mapping task generates values based on the original Initial Value for each run. Default is disabled.

Configure advanced Sequence properties on the **Advanced** tab:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Optional	Determines whether the transformation is optional. If a transformation is optional and there are no incoming fields, the mapping task can run and the data can go through another branch in the data flow. If a transformation is required and there are no incoming fields, the task fails. For example, you configure a parameter for the source connection. In one branch of the data flow, you add a transformation with a field rule so that only Date/Time data enters the transformation, and you specify that the transformation is optional. When you configure the mapping task, you select a source that does not have Date/Time data. The mapping task ignores the branch with the optional transformation, and the data flow continues through another branch of the mapping.
Disable incoming fields	Disable incoming fields to connect only the generated sequence to a downstream transformation. If you disable incoming fields, you must connect at least one field from another transformation to the downstream transformation.

Sequence advanced properties aren't available in mappings in SQL ELT mode.

Disabling incoming fields

You can disable incoming fields to connect only the generated sequence in the output fields, NEXTVAL and CURRVAL, to one or more downstream transformations. When you disable incoming fields, you can't connect the Sequence transformation to upstream transformations.

When you disable incoming fields, at least one field from another transformation must be connected to the downstream transformation along with the Sequence transformation fields. For example, if the mapping contains a Sequence transformation and a Source transformation and the Sequence transformation is connected to a Target transformation, you must connect at least one field from the Source transformation to the Target transformation.

The following image shows a mapping where incoming fields are disabled in the Sequence transformation:
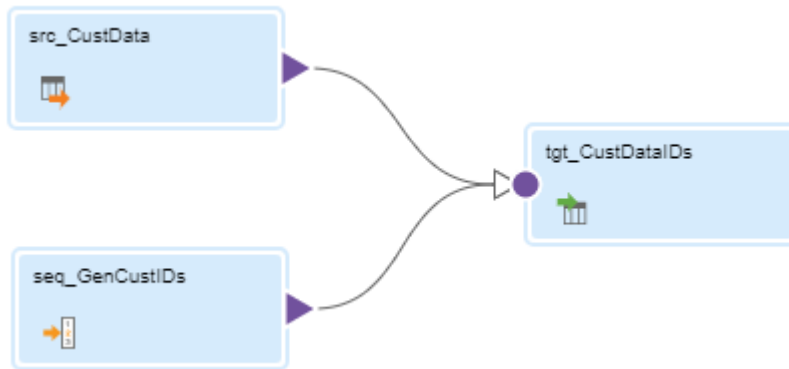
Note: You can't disable incoming fields in advanced mode.

Hierarchical data in advanced mode

In advanced mode, you can use hierarchical fields that represent an array, map, or struct.

You can use the hierarchical fields as pass-through fields.

Sequence transformation rules and guidelines

Consider the following guidelines when you create a Sequence transformation:

- When you map the NEXTVAL or CURRVAL output fields, ensure that the data type of the mapped field is appropriate.
- When you run the mapping in the Mapping Designer, the current value is not saved so each time you run the mapping, it begins with the initial value.
- When you run the task in the mapping task wizard, you can edit the current value to start the sequence with a specified value.
- Before the Sequence transformation runs, Data Integration checks for sufficient space between the starting value and the end value. If you set an end value that's too close to the starting value, the mapping fails with a sequence overflow error. This error can occur even if the starting value plus the number of rows processed would be less than the end value. If you encounter a sequence overflow error, increase the end value.
- You can't use a Sequence transformation in a mapplet.
- To use the Sequence transformation in advanced mode, the Secure Agent must run on a virtual machine.
- A self-service cluster might intermittently fail to connect to the agent, which can cause the mapping to fail. Error messages related to this appear in the session logs.
- In advanced mode, generated values might not increase monotonically. Values are generated based on the order that the Spark engine processes the data.
- In mappings in SQL ELT mode, you can't use shared sequences.

- In mappings in SQL ELT mode, a Sequence transformation needs to be connected directly to a downstream Expression transformation or target.

Sequence transformation example

The following example shows how you can use the Sequence transformation to generate primary keys.

You are gathering customer data and need to assign customer IDs to each customer. The CustomerData.csv flat file contains your source customer data. You create a mapping that includes the Sequence transformation to create customer IDs, using the following process:

1. Create a copy of the CustomerData.csv file to use as the target and then add the cust_id field to the file to hold the generated customer ID values. You name the file CustomerData_IDs.csv. The following image shows a sample of the file:

last_name	city	state	country_c	cust_id
Smith	Los Angel	California	USA	
Jones	Chicago	Illinois	USA	
Winston	Houston	Texas	USA	
Leesom	Philadelpl	Pennsylva	USA	
Marks	Phoenix	Arizona	USA	
Weston	San Diego	California	USA	
West	Dallas	Texas	USA	
Arlington	San Anton	Texas	USA	
Book	Detroit	Michigan	USA	
Ferris	San Jose	California	USA	
Richards	Indianapo	Indiana	USA	
Benton	San Franci	California	USA	
Martinez	Jacksonvil	Florida	USA	
Alton	Columbus	Ohio	USA	

2. Create a connection that has access to the CustomerData.csv and CustomerData_IDs.csv files.

3. Create a mapping in the Mapping Designer and add a Source transformation to the mapping. You configure the transformation to use the CustomerData.csv file.

4. Add a Sequence transformation to the mapping.

5. To generate a simple sequence starting with 1, set the **Initial Value** to 1 and the **Increment By** value to 1. This configuration starts the sequence at 1 and increments the value by 1, for example, 1, 2, 3.

You leave the default values for the other properties. The following image shows this configuration:
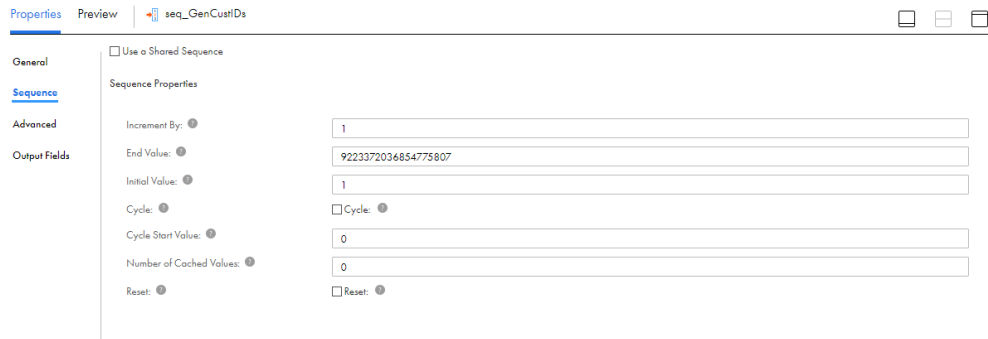
6. Add a Target transformation to the mapping and configure the transformation to use the CustomerData_IDs.csv file that you created.

7. Connect the Source transformation to the Sequence transformation and the Sequence transformation to the Target transformation:
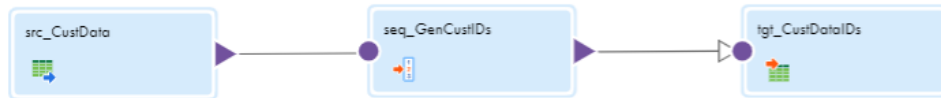
8. In the Target transformation, map the NEXTVAL output field to the cust_id field.
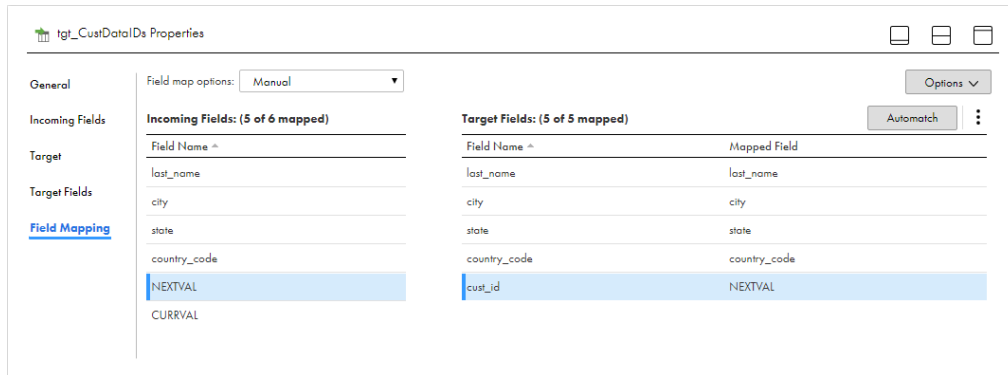
9. Save the mapping and create a mapping task in the mapping task wizard. The **Current Value** is 1 because the mapping hasn't been run yet and the **Initial Value** is 1.
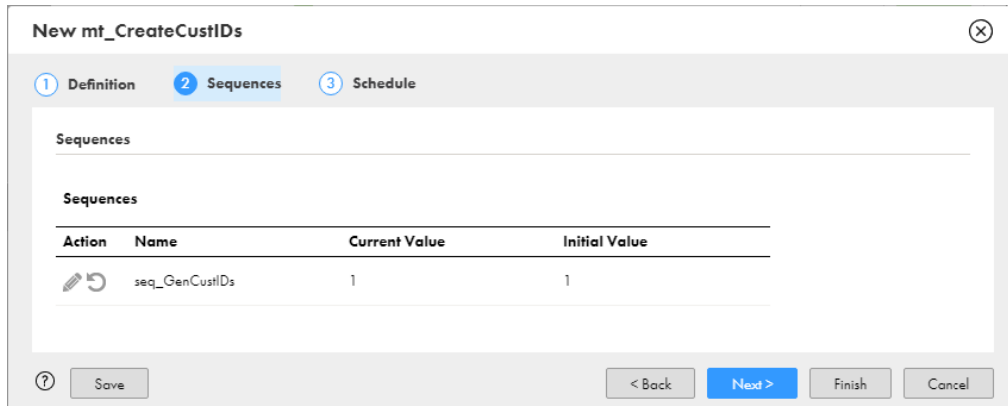
10. After running the mapping task, view the mapping task details. The task details show the current value and the initial value of the sequence.

11. Open the CustomerData_IDs.csv file and note that the cust_id field is populated with a numeric sequence:

last_name	city	state	country_c	cust_id
Smith	Los Angel	California	USA	1
Jones	Chicago	Illinois	USA	2
Winston	Houston	Texas	USA	3
Leesom	Philadelpl	Pennsylva	USA	4
Marks	Phoenix	Arizona	USA	5
Weston	San Diego	California	USA	6
West	Dallas	Texas	USA	7
Arlington	San Anton	Texas	USA	8
Book	Detroit	Michigan	USA	9
Ferris	San Jose	California	USA	10
Richards	Indianapo	Indiana	USA	11
Benton	San Franci	California	USA	12
Martinez	Jacksonvil	Florida	USA	13
Alton	Columbus	Ohio	USA	14

CHAPTER 33

Sorter transformation

Use a Sorter transformation to sort data in ascending or descending order, according to a specified sort condition. You can configure the Sorter transformation for case-sensitive sorting and for distinct output. The Sorter transformation is a passive transformation.

You can use the Sorter transformation to increase performance with other transformations. For example, you can sort data that passes through a Lookup or an Aggregator transformation configured to use sorted incoming fields.

When you create a Sorter transformation, specify fields as sort conditions and configure each sort field to sort in ascending or descending order. You can use a parameter for the sort condition and define the value of the parameter when you configure the mapping task.

Note: In advanced mode, make sure that the following conditions are true for the Sorter transformation to take effect:

- The Sorter transformation is connected directly to the Target transformation.
- All sort fields used in the sort condition are connected to the Target transformation.
- The data types in the connected sort fields match the data types in the target fields.

Sort conditions

Configure the sort condition to specify the sort fields and the sort order. The mapping task uses the sort condition to sort the data.

The sort fields are one or more fields that you want to use as the sort criteria. Configure the sort order to sort data in ascending or descending order. In advanced mode, you can override the sort order using the advanced session properties when you schedule the mapping task.

When you specify multiple sort conditions, the mapping task sorts each condition sequentially. The mapping task treats each successive sort condition as a secondary sort of the previous sort condition. You can configure the order of sort conditions.

If you use a parameter for the sort condition, define the sort fields and the sort order when you run the mapping or when you configure the mapping task.

Sorter caches

The mapping task passes all incoming data into the Sorter transformation before it performs the sort operation. The mapping task uses cache memory to process Sorter transformations. If the mapping task cannot allocate enough memory, the mapping fails.

By default, the mapping task determines the cache size at run time. Before starting the sort operation, the mapping task allocates the amount of memory configured for the Sorter cache size.

Configure the Sorter cache size with a value less than the amount of available physical RAM on the machine that hosts the Secure Agent. Allocate at least 16 MB (16,777,216 bytes) of physical memory to sort data with a Sorter transformation. When you allocate memory to the Sorter cache, consider other transformations in the mapping and the volume of data in the mapping task.

If the amount of incoming data is greater than the Sorter cache size, the mapping task temporarily stores data in the work directory. When storing data in the Sorter transformation work directory, the mapping task requires disk space of at least twice the amount of incoming data.

When you configure the tracing level to Normal, the mapping task writes the memory amount that the Sorter transformation uses to the session log.

Advanced properties

You can specify additional sort criteria in the Sorter transformation advanced properties. The mapping task applies the properties to all sort fields. The Sorter transformation properties also determine the system resources that the mapping task allocates when it sorts data.

The properties that are available vary based on the mapping mode.

You can configure the following advanced properties for a Sorter transformation:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Sorter Cache Size	The maximum amount of memory required to perform the sort operation. The mapping task passes all incoming data into the Sorter transformation before it performs the sort operation. If the mapping task cannot allocate enough memory, the mapping fails. You can configure a numeric value for the sorter cache. Allocate at least 16 MB of physical memory. Default is Auto.
Case Sensitive	Determines whether the mapping task considers case when sorting data. When you enable a case-sensitive sort, the mapping task sorts uppercase characters higher than lowercase characters. Default is Case Sensitive. On an advanced cluster, sorting is always case-sensitive.
Work Directory	The mapping task uses the work directory to create temporary files while it sorts data. After the mapping task sorts data, it deletes the temporary files. You can specify any directory on the Secure Agent machine to use as a work directory. Allocate at least 16 MB (16,777,216 bytes) of physical memory for the work directory. You can configure a system parameter or a user-defined parameter in this field. Default is the TempDir system parameter.

Property	Description
Distinct	Treats output rows as distinct. If you configure the Sorter transformation for distinct output rows, the mapping task configures all fields as part of the sort condition. The mapping task discards duplicate rows compared during the sort operation.
Null Treated Low	Treats a null value as lower than any other value. For example, if you configure a descending sort condition, rows with a null value in the sort field appear after all other rows. On an advanced cluster, null values are treated as high.
Transformation Scope	The transaction is determined by the commit or rollback point. The transformation scope specifies how the mapping task applies the transformation logic to incoming data: - Transaction. Applies the transformation logic to all rows in a transaction. Choose Transaction when the results of the transformation depend on all rows in the same transaction, but not on rows in other transactions. - All Input. Applies the transformation logic to all incoming data. When you choose All Input, the mapping task drops incoming transaction boundaries. Choose All Input when the results of the transformation depend on all rows of data in the source.
Optional	Determines whether the transformation is optional. If a transformation is optional and there are no incoming fields, the mapping task can run and the data can go through another branch in the data flow. If a transformation is required and there are no incoming fields, the task fails. For example, you configure a parameter for the source connection. In one branch of the data flow, you add a transformation with a field rule so that only Date/Time data enters the transformation, and you specify that the transformation is optional. When you configure the mapping task, you select a source that does not have Date/Time data. The mapping task ignores the branch with the optional transformation, and the data flow continues through another branch of the mapping.

Hierarchical data in advanced mode

In advanced mode, you can use hierarchical fields that represent an array, map, or struct.

You can use the hierarchical fields as pass-through fields.

Note: You cannot use a hierarchical field in a sort condition.

Sorter transformation example

You need to create an invoice for customer sales from a customer database. Use a Sorter transformation on the customer sales object to sort the data in ascending order according to the order number. Use the result of the Sorter transformation as an input to the Aggregator transformation. You can increase Aggregator transformation performance with the sorted incoming fields option.

The Product_Orders table contains information about all the orders placed by customers.

OrderID	ItemID	Item	Quantity	Price
43	123456	ItemA	3	3.04
41	456789	ItemB	2	12.02

OrderID	ItemID	Item	Quantity	Price
43	000246	ItemC	6	34.55
45	000468	ItemD	5	0.56
43	NULL	ItemE	1	0.75
41	123456	ItemA	4	3.04
45	123456	ItemA	5	3.04
45	456789	ItemB	3	12.02

In the Mapping Designer, add Product_Orders as a source object.

Add a Sorter transformation to the mapping canvas, and connect it to the data flow. Sort the product orders by order ID and item ID.

The following image shows a sort condition with the order ID and item ID fields configured to sort in descending order:
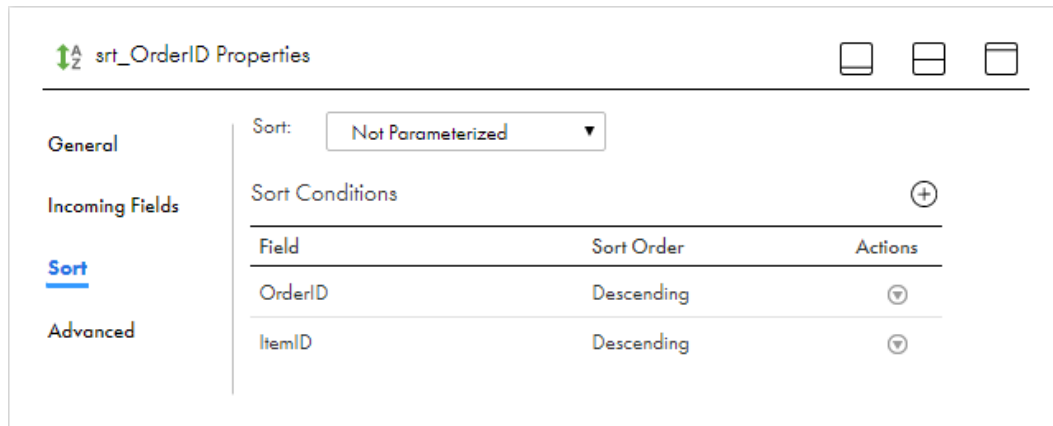
Enable a null treated low sort so that the mapping task considers null values to be lower than other values.

The following image shows the advanced properties for the Sorter transformation, with the Null Treated Low option selected:
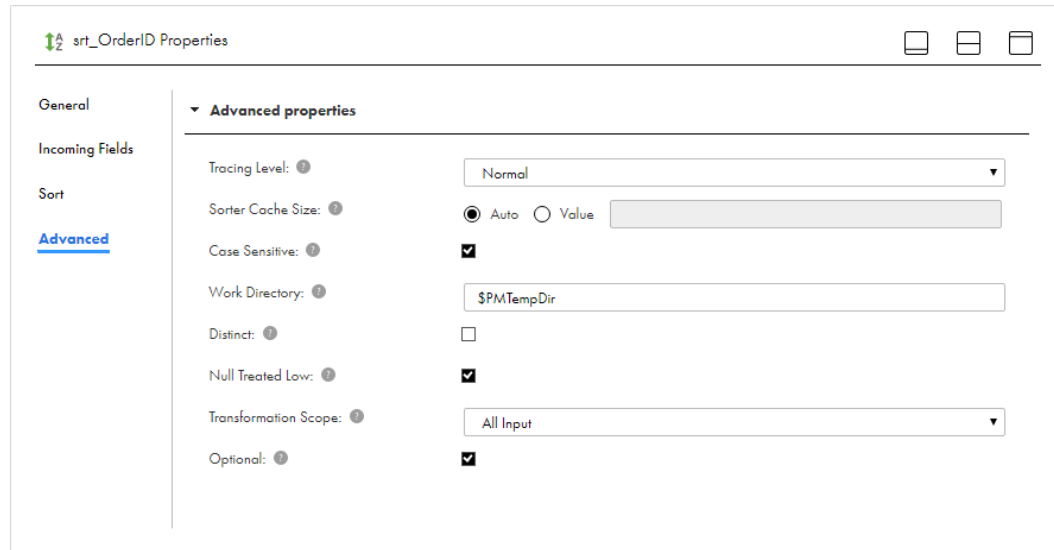
After the mapping task sorts the data, it passes the following rows out of the Sorter transformation:

OrderID	ItemID	Item	Quantity	Price
45	456789	ItemB	3	12.02
45	123456	ItemA	5	3.04
45	000468	ItemD	5	0.56
43	123456	ItemA	3	3.04
43	000246	ItemC	6	34.55
43	NULL	ItemE	1	0.75
41	456789	ItemB	2	12.02
41	123456	ItemA	4	3.04

You need to find out the total amount and the item quantity for each order. You can use the result of the Sorter transformation as an input to an Aggregator transformation to increase performance. Add the Aggregator transformation to the mapping, and connect the transformation to the data flow. Group the fields in the Aggregator transformation by the Order ID, and add an expression to sum the orders by price.

When you pass the data from the Sorter transformation, the Aggregator transformation groups the order ID to calculate the total amount for each order.

OrderID	Sum
45	54.06
43	217.17
41	36.2

Write the order summaries to an object in the data warehouse that stores all order totals.

The mapping task reads the orders data from the source, sorts the orders, totals the prices, and then writes the data to the target.

The following image shows the mapping of the data flow:
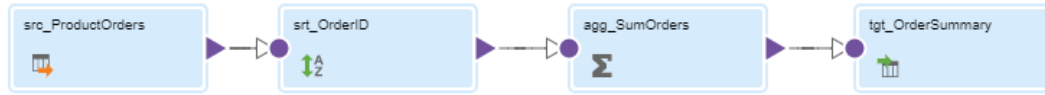
CHAPTER 34

SQL transformation

Use the SQL transformation to call a stored procedure or function in a relational database or to processes SQL queries midstream in a pipeline. The transformation can call a stored procedure or function, process a saved query, or process a query that you create in the transformation SQL editor.

The SQL transformation can process the following types of SQL statements:

Stored procedure or stored function

A stored procedure is a precompiled collection of database procedural statements and optional flow control statements, similar to an executable script. Stored procedures reside in the database and run within the database. A stored function is similar to a stored procedure, except that a function returns a single value.

When the SQL transformation processes a stored procedure or function, it passes input parameters to the stored procedure or function. The stored procedure or function passes the return value or values to the output fields of the transformation.

Saved or user-entered query

You can configure the SQL transformation to process a saved query that you create in Data Integration or you can enter a query in the SQL editor. The SQL transformation processes the query and returns rows and database errors.

You can pass strings or parameters to the query to define dynamic queries or change the selection parameters. You can output multiple rows when the query has a SELECT statement.

Stored procedure or function processing

You can use the SQL transformation to call a stored procedure or function in a Microsoft SQL Server, MySQL, ODBC, Oracle, PostgreSQL, Redshift V2, or Snowflake database. The stored procedure or function must exist in the database before you create the SQL transformation.

You can call a stored procedure or function with the following types of SQL transformations:

Connected SQL transformation

The transformation is connected to the mapping pipeline. The stored procedure or function runs on a row by row basis and can return a single output parameter or multiple output parameters.

You can map the incoming fields of the SQL transformation to the input fields of a stored procedure. The output fields in the SQL transformation consist of the stored procedure output parameters or return values.

A return value is a code or text string that you define in the stored procedure. For example, a stored procedure can return a value that indicates the date the stored procedure was run. When a stored procedure has a return value, the SQL transformation has a return value field.

Unconnected SQL transformation

The SQL transformation is not connected to the mapping pipeline. An Expression transformation calls the SQL transformation with a stored procedure expression, or the stored procedure runs before or after the mapping.

You can configure the expression to return the stored procedure output to expression output fields and variables. You can call the stored procedure from multiple expressions and nest stored procedures.

You cannot process a stored function with an unconnected SQL transformation.

You might use a stored procedure to perform the following tasks:

- Check the status of a target database before loading data into it.
- Determine if enough space exists in a database.
- Perform a specialized calculation.
- Retrieve data by a value.
- Drop and re-create indexes.
- Remove temporary tables.
- Verify that a table exists in a database

You can use a stored procedure to perform a calculation that you would otherwise make part of a mapping. For example, if you have a stored procedure to calculate sales tax, perform that calculation in an SQL transformation instead of re-creating the calculation in an Expression transformation.

When you run a mapping, the SQL transformation passes input parameters to the stored procedure. The stored procedure passes the return value or values to the output fields of the transformation.

Connected SQL transformation example

Your mapping includes user IDs in the data flow. You want to include user names in addition to user IDs.

You have a stored procedure that matches user IDs with user names in the database. You add an SQL transformation to your mapping, select the stored procedure, and map the userId incoming field with the userId input field in the stored procedure. You check the **Output Fields** tab for the SQL transformation to confirm that it includes the username field. When you run the mapping, the username value is returned with the user ID.

Unconnected SQL transformation example

Your mapping includes employee salary data and you want to update each employee's salary with a raise.

You have a stored procedure that calculates employee salary increases. The stored procedure returns the new salary and the percentage of increase. You add an unconnected SQL transformation and select the stored procedure.

You then add an Expression transformation to the mapping pipeline. In the Expression transformation, you add a variable field to capture the new salary. You add an output field and use the stored procedure function to configure the expression. You configure the arguments so that the output field returns the increase percentage and you create a second output field to return the new salary. You then map the new output fields to the downstream transformation.

Connected or unconnected SQL transformation for stored procedure processing

When you call a stored procedure in an SQL transformation, you can use a connected or an unconnected SQL transformation.

You process a stored procedure with a connected SQL transformation when you need data from an input field sent as an input parameter to the stored procedure, or you need the results of a stored procedure sent as an output parameter to another transformation.

You process a stored procedure with an unconnected SQL transformation when you need the stored procedure to run before or after a mapping, run nested stored procedures, or call the stored procedure multiple times.

The following table describes when you would use a connected or unconnected SQL transformation to process a stored procedure:

Scenario	SQL transformation type
Run a stored procedure before or after a mapping.	Unconnected
Run a stored procedure once during a mapping.	Unconnected
Run a stored procedure every time a row passes through the SQL transformation.	Connected or unconnected
Run a stored procedure based on data that passes through the mapping such as when a specific field does not contain a null value.	Unconnected
Pass parameters to the stored procedure and receive a single output parameter.	Connected or unconnected
Pass parameters to the stored procedure and receive multiple output parameters. **Note:** To get multiple output parameters from an unconnected SQL transformation, you must create variables for each output parameter.	Connected or unconnected
Run nested stored procedures.	Unconnected
Call a stored procedure multiple times within a mapping.	Unconnected

Unconnected SQL transformations

An unconnected SQL transformation is an SQL transformation that is not connected to the mapping pipeline. Use an unconnected SQL transformation to call a stored procedure.

You use an Expression transformation to call the unconnected SQL transformation with an :SP expression. Or, you configure the SQL transformation to invoke a stored procedure before or after a mapping run. For example, you might use an unconnected SQL transformation to remove temporary source tables after the mapping receives data from the source.

You might also use an unconnected SQL transformation when you want to call a stored procedure multiple times in a mapping.

Calling an unconnected SQL transformation from an expression

Call an unconnected SQL transformation from an Expression transformation with an :SP expression.

When you call a stored procedure from an expression, you configure the expression to return the stored procedure output values to fields in the expression. Use one of the following methods to return the output values:

- Assign the output value to a local variable field.
- Assign the output value to the system variable PROC_RESULT.

When you use the PROC_RESULT variable, Data Integration assigns the value of the return parameter directly to the output field, which you can write to a target. You can also assign one output parameter to PROC_RESULT and the other parameter to a variable.

Use expression variables to access OUT or INOUT parameters in the stored procedure. If the stored procedure returns multiple output parameters, you must create variables for each output parameter.

Use the following syntax to call a stored procedure in an expression:

```
:SP.<SQL transformation name> (arg1, arg2, PROC_RESULT)
```

If the stored procedure returns a single output parameter or return value, use the reserved variable PROC_RESULT as the output variable.

For example, the following expression calls a stored procedure called GET_NAME_FROM_ID:

```
:SP.GET_NAME_FROM_ID(inID, PROC_RESULT)
```

inID can be either an input field in the stored procedure or a variable in the Expression transformation. When you run the mapping, Data Integration applies the value of PROC_RESULT to the output field for the expression.

If the stored procedure returns multiple output parameters, you must create expression variables for each output parameter. For example, if the stored procedure also returns a title, create a variable field called varTitle1 in the Expression transformation and use the field as the expression for an output field called Title. You write the following expression:

```
:SP.GET_NAME_FROM_ID(inID, varTitle1, PROC_RESULT)
```

The following image shows how you configure the Expression transformation:

Data Integration returns output parameters in the order they are declared in the stored procedure. In this example, Data Integration applies the value of the first output field in the stored procedure to varTitle1 and passes it to the Title field in the Expression transformation. It applies the value of the second stored procedure output field to the output field for the expression.

The data types for the expression fields and variables must match the data types for the stored procedure input/output variables and return value.

Invoking a stored procedure before or after a mapping run

You can configure an unconnected SQL transformation to process a stored procedure before or after a mapping run. Data Integration invokes the stored procedure at the specified time. You do not need to call the stored procedure with an :SP expression. You can configure the stored procedure to run before or after the mapping receives data from the source, or before or after the mapping loads data to the target.

You can configure the following stored procedure types:

- Source Pre-load. The stored procedure runs before the mapping retrieves data from the source.
- Source Post-load. The stored procedure runs after the mapping retrieves data from the source.
- Target Pre-load. The stored procedure runs before the mapping sends data to the target.
- Target Post-load. The stored procedure runs after the mapping sends data to the target.

On the **Advanced** tab, configure the stored procedure type and enter the call text for the stored procedure. The call text is the name of the stored procedure followed by any applicable input parameters in parentheses. If there are no input parameters, you must include an empty pair of parentheses. Do not include the SQL statement EXEC or use the :SP keyword.

For example, to call the stored procedure Drop_Table, enter the following call text:

```
Drop_Table()
```

To pass a string input parameter, enter it without quotes. If the string has spaces in it, enclose the parameter in double quotes. For example, if the stored procedure Drop_Table requires a table name as an input parameter, enter the following call text:

```
Drop_Table(Customer_list)
```

Stored procedure run order

Specify the stored procedure run order when you have multiple source or target pre- or post-load stored procedures. Data Integration calls the stored procedures in the order that you specify. If you don't specify a run order, Data Integration calls the stored procedures in the order they are received at run time.

You can specify the stored procedure run order when the stored procedure type is set to anything except Normal and more than one stored procedure of that type exists in the mapping. For example, if you have multiple stored procedures that you want to call before Data Integration loads the source, you can specify the run order of the source pre-load stored procedures.

To set the stored procedure run order, open the **Actions** menu and select **Stored Procedure Order**.

The **Stored Procedure Order** dialog displays all source and target pre- and post-load stored procedures in the mapping even if there is only one stored procedure of a particular type. The tabs that you see in the **Stored Procedure Order** dialog depend on the type of stored procedures in the mapping. For example, if the mapping doesn't contain any target pre-load stored procedures, the Target pre-load tab doesn't display.

The following image shows the **Stored Procedure Order** dialog:
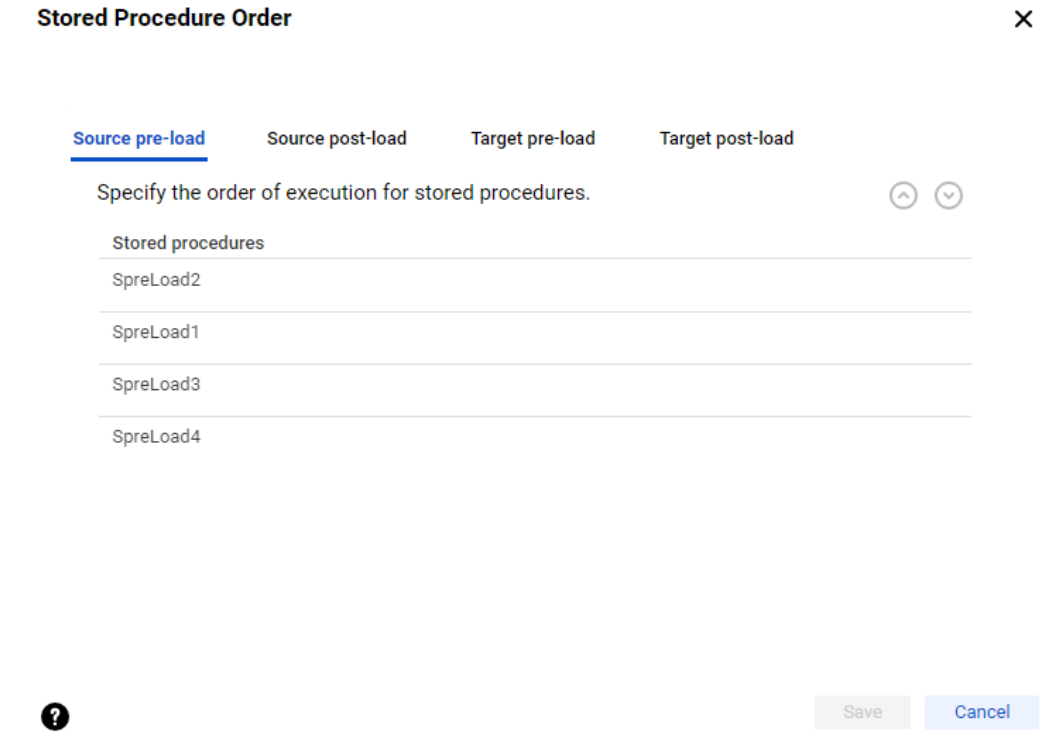
To set the stored procedure run order, select a stored procedure and use the up and down arrows to change the order. Data Integration calls the stored procedures from the top down.

You can't specify the run order for stored procedures with a stored procedure type of Normal or for connected stored procedures.

Unconnected SQL transformation example

You are updating employee salaries with a cost of living increase. You have a CSV file that contains employee names and IDs. You need to add each employee's salary, calculate their increase, and write the data to a new CSV file.

You have the following source data:

```
EMP_ID, EMP_NAME
1001, John
1002, Alice
1003, Mary
1004, Mark
1005, Stephan
```

You have a stored procedure called ADD_SALARY in an Oracle database that adds the employee's salary to the file and a second stored procedure called SALARY_INCREASE which calculates the salary increase. In the mapping, use two unconnected SQL transformations to call the stored procedures and two Expression transformations to call the SQL transformations.

The following image shows the mapping:
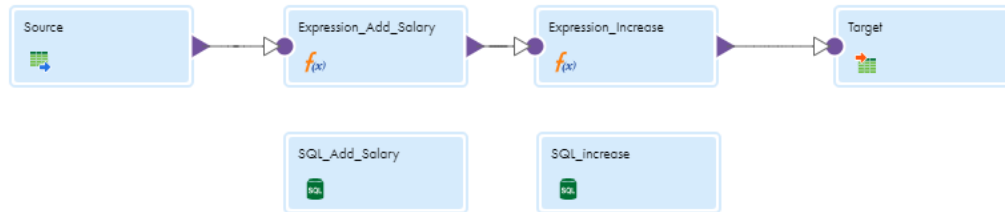
Configure the transformations in the following ways:
Source transformation

Configure the Source transformation to load the source data that you want to use.

SQL_Add_Salary transformation

Configure the first SQL transformation to call the ADD_SALARY stored procedure.

On the **SQL** tab, select the connection that contains the ADD_SALARY stored procedure and then select the stored procedure.

Select **Unconnected stored procedure**.

The stored procedure has one input field for the employee ID and returns the current salary in the output field.

SQL_Increase transformation

Configure the second SQL transformation to call the SALARY_INCREASE stored procedure.

On the **SQL** tab, select the connection that contains the SALARY_INCREASE stored procedure and then select the stored procedure.

Select **Unconnected stored procedure**.

The stored procedure has one input field for the employee name and returns the new salary in the output field.

Expression_Add_Salary transformation

Configure the first Expression transformation to call the SQL_Add_Salary transformation. Create a variable field for the input parameter and an output field to capture the output of the stored procedure.

On the **Expression** tab, add a variable field named ID and configure its value as the EMP_ID field in the source. Create an output field called salary to capture the return value of the ADD_SALARY stored procedure in the first SQL transformation. Configure the salary field to call the ADD_SALARY stored procedure with the following expression:

```
:SP.SQL_Add_Salary(ID, PROC_RESULT)
```

The expression takes the variable field ID as the input parameter of the stored procedure and returns the salary value to the SALARY output field.

The following image shows how you configure the Expression transformation:
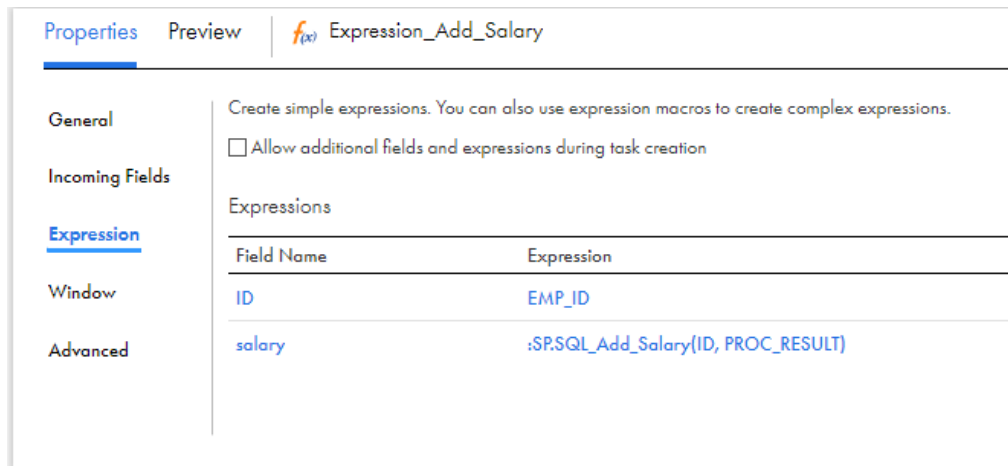
Expression_Increase transformation

Configure the second Expression transformation to call the SQL_increase transformation. Add a variable field called CurrentSalary and configure its value as the incoming salary field. Add a output field called newSalary to capture the return value of the SALARY_INCREASE stored procedure. Configure the newSalary field to call the SALARY_INCREASE stored procedure with the following expression:

```
:SP.SQL_increase(CurrentSalary, PROC_RESULT)
```

The expression takes the variable field CurrentSalary as the input parameter of the stored procedure and returns the new salary to the newSalary output field.

The following image shows how you configure the Expression transformation:
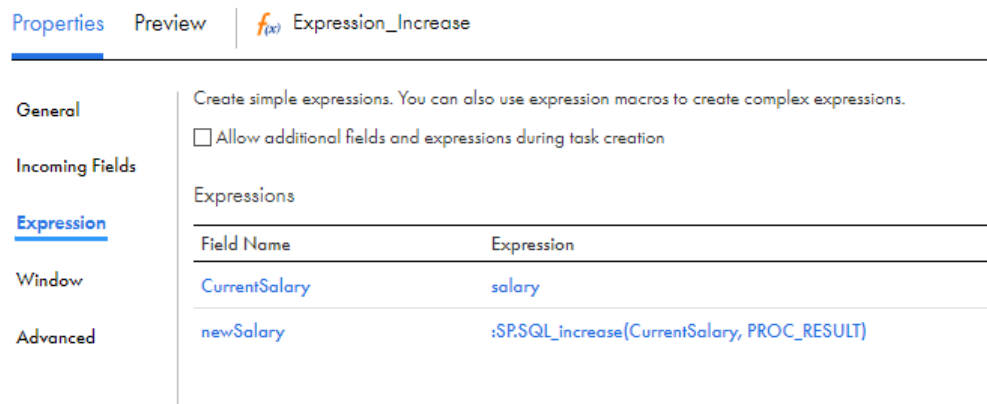
Target transformation

Configure the Target transformation to create a target file at run time.

When you run the mapping, you get the following results:

```
EMP_ID, EMP_NAME, salary, newSalary
1001, John, 400, 480
1002, Alice, 500, 600
1003, Mary, 400, 480
1004, Mark, 700, 840
1005, Stephan, 600, 720
```

Query processing

You can configure the SQL transformation to process a saved query or a user-entered query that runs against a Microsoft SQL Server or Oracle database. When you configure the SQL transformation to process a query, you create an active transformation. The transformation can return multiple rows for each input row.

When you enter a query, you can format the SQL and validate the syntax. Alternatively, you can create a string parameter to define the query in the mapping task.

You can create the following types of SQL queries:

Static SQL query

> The query statement does not change, but you can use query parameters to change the data. Data Integration prepares the SQL query once and runs the query for all input rows.

Dynamic SQL query

> You can change the query statements and the data. Data Integration prepares an SQL query for each input row.

You can optimize performance by creating static queries.

Static SQL queries

Create a static SQL query when you need to run the same query statements for each input row, but you want to change the data in the query for each input row. When you create a static SQL query, you use parameter binding in the SQL editor to define parameters for query data.

To change the data in the query, you configure query parameters and bind them to input fields in the transformation. When you bind a parameter to an input field, you identify the field by name in the query. Enclose the field name in question marks (?). The query data changes based on the value of the data in the input field.

For example, the following static queries use parameter binding:

```
DELETE FROM Employee WHERE Dept = ?Dept?
INSERT INTO Employee(Employee_ID, Dept) VALUES (?Employee_ID?, ?Dept?)
UPDATE Employee SET Dept = ?Dept? WHERE Employee_ID > 100
```

Example

The following static SQL query uses query parameters that bind to the Employee_ID and Dept input fields of an SQL transformation:

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

The source has the following rows:

Employee_ID	Dept
100	Products
123	HR
130	Accounting

Data Integration generates the following query statements from the rows:

```
SELECT Name, Address FROM Employees WHERE Employee_ID = '100' and DEPT = 'Products'
SELECT Name, Address FROM Employees WHERE Employee_ID = '123' and DEPT = 'HR'
SELECT Name, Address FROM Employees WHERE Employee_ID = '130' and DEPT = 'Accounting'
```

Selecting multiple database rows

When the SQL query contains a SELECT statement, the transformation returns one row for each database row it retrieves. You must configure an output field for each column in the SELECT statement. The output fields must be in the same order as the columns in the SELECT statement.

When you configure output fields for database columns, you must configure the data type of each database column that you select. Select a native data type from the list. When you select the native data type, Data Integration configures the transformation data type for you.

The native data type in the transformation must match the database column data type. Data Integration matches the column data type in the database with the native database type in the transformation at run time. If the data types do not match, Data Integration generates a row error.

Dynamic SQL queries

A dynamic SQL query can execute different query statements for each input row. When you create a dynamic SQL query, you use string substitution to define string variables in the query and link them to input fields in the transformation.

To change a query statement, configure a string variable in the query for the portion of the query that you want to change. To configure the string variable, identify an input field by name in the query and enclose the name in tilde characters (~). The query changes based on the value of the data in the field.

The transformation input field that contains the query variable must be a string data type. You can use string substitution to change the query statement and the query data.

When you create a dynamic SQL query, Data Integration prepares a query for each input row. You can pass the following types of dynamic queries in an input field:

Full query

> You can substitute the entire SQL query with query statements from source data.

Partial query

> You can substitute a portion of the query statement, such as the table name.

Passing the full query

You can pass the full SQL query through an input field in the transformation. To pass the full query, create a query in the SQL editor that consists of one string variable to represent the full query: `~Query_Field~`

To pass the full query, configure the source to pass the full query in an output field. Then, configure the SQL transformation to receive the query in the Query_Field input field.

The following image shows the transformation configuration:
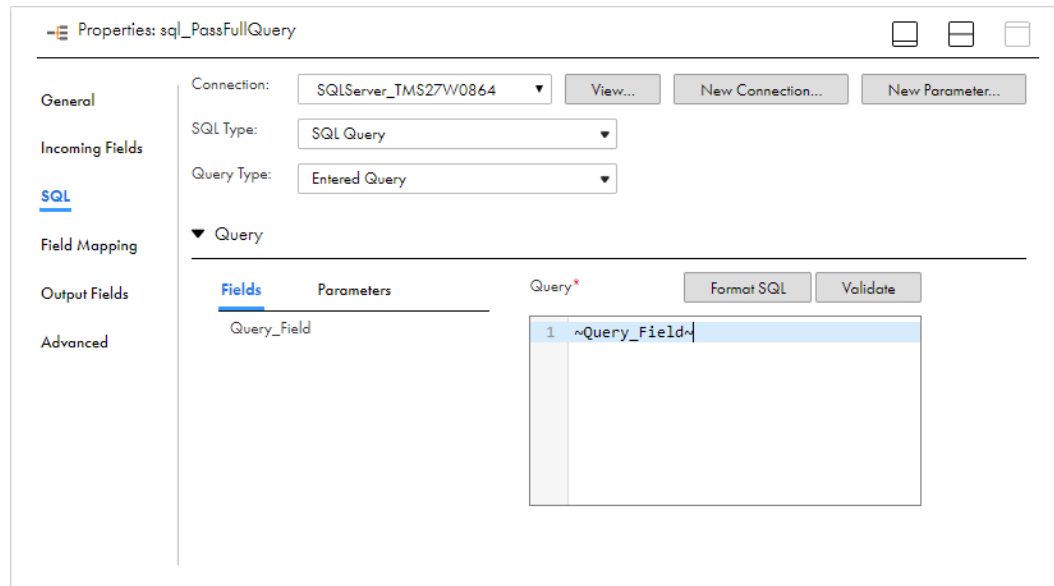
Data Integration replaces the `~Query_Field~` variable in the dynamic query with the SQL statements from the source. It prepares the query and sends it to the database to process. The database executes the query. The SQL transformation returns database errors to the SQLError output field.

When you pass the full query, you can pass more than one query statement for each input row. For example, the source might contain the following rows:

```
DELETE FROM Person WHERE LastName = 'Jones'; INSERT INTO Person (LastName, Address)
VALUES ('Smith', '38 Summit Drive')
DELETE FROM Person WHERE LastName = 'Jones'; INSERT INTO Person (LastName, Address)
VALUES ('Smith', '38 Summit Drive')
DELETE FROM Person WHERE LastName = 'Russell';
```

You can pass any type of query in the source data. When you configure SELECT statements in the query, you must configure output fields for the database columns that you retrieve from the database. When you mix SELECT statements and other types of queries, the output fields that represent database columns contain null values when no database columns are retrieved.

Substituting the table name in a string

You can use a partial query to substitute the table name. To substitute the table name, configure an input field to receive the table name from each input row. Identify the input field by name in the query and enclose the name with tilde characters (~).

For example, the following dynamic query contains a string variable, `~Table_Field~`:

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

The source might pass the following values to the Table_Field column:

Table_Field

```
Employees_USA
```

```
Employees_England
```

```
Employees_Australia
```

Data Integration replaces the ~Table_Field~ variable with the table name in the input field:

```
SELECT Emp_ID, Address from Employees_USA where Dept = 'HR'
SELECT Emp_ID, Address from Employees_England where Dept = 'HR'
SELECT Emp_ID, Address from Employees_Australia where Dept = 'HR'
```

Passive mode configuration

When you create an SQL transformation, you can configure it to run in passive mode instead of active mode. A passive transformation does not change the number of rows that pass through it. It maintains transaction boundaries and row types.

If you configure the SQL transformation to process a query, you can configure passive mode when you create the transformation. Configure passive mode in the transformation advanced properties.

When you configure the transformation as a passive transformation and a SELECT query returns more than one row, Data Integration returns the first row and an error to the SQLError field. The error states that the SQL transformation generated multiple rows.

If the SQL query has multiple SQL statements, Data Integration executes all statements but returns data for the first SQL statement only. The SQL transformation returns one row. The SQLError field contains the errors from all SQL statements. When multiple errors occur, they are separated by semicolons (;) in the SQLError field.

SQL statements that you can use in queries

You can use certain data definition, data manipulation, data language control, and transaction control statements with the SQL transformation.

The following table lists the statements that you can use in an SQL query in the SQL transformation:

Statement Type	Statement	Description
Data Definition	ALTER	Modifies the structure of the database.
Data Definition	COMMENT	Adds comments to the data dictionary.
Data Definition	CREATE	Creates a database, table, or index.
Data Definition	DROP	Deletes an index, table, or database.
Data Definition	RENAME	Renames a database object.
Data Definition	TRUNCATE	Removes all rows from a table.
Data Manipulation	CALL	Calls a PL/SQL or Java subprogram.
Data Manipulation	DELETE	Deletes rows from a table.
Data Manipulation	EXPLAIN PLAN	Writes the access plan for a statement into the database Explain tables.
Data Manipulation	INSERT	Inserts rows into a table.
Data Manipulation	LOCK TABLE	Prevents concurrent application processes from using or changing a table.
Data Manipulation	MERGE	Updates a table with source data.

Statement Type	Statement	Description
Data Manipulation	SELECT	Retrieves data from the database.
Data Manipulation	UPDATE	Updates the values of rows of a table.
Data Control Language	GRANT	Grants privileges to a database user.
Data Control Language	REVOKE	Removes access privileges for a database user.
Transaction Control	COMMIT	Saves a unit of work and performs the database changes for that unit of work.
Transaction Control	ROLLBACK	Reverses changes to the database since the last COMMIT.

Rules and guidelines for query processing

Use the following rules and guidelines when you configure the SQL transformation to process a query:

- The number and the order of the output fields must match the number and order of the fields in the query SELECT clause.
- The native data type of an output field in the transformation must match the data type of the corresponding column in the database. Data Integration generates a row error when the data types do not match.
- When the SQL query contains an INSERT, UPDATE, or DELETE clause, the transformation returns data to the SQLError field, the pass-through fields, and the NumRowsAffected field when it is enabled. If you add output fields, the fields receive NULL data values.
- When the SQL query contains a SELECT statement and the transformation has a pass-through field, the transformation returns data to the pass-through field whether or not the query returns database data. The SQL transformation returns a row with NULL data in the output fields.
- When the number of output fields is more than the number of columns in the SELECT clause, the extra fields receive a NULL value.
- When the number of output fields is less than the number of columns in the SELECT clause, Data Integration generates a row error.
- You can use string substitution instead of parameter binding in a query. However, the input fields must be string data types.

SQL transformation configuration

Use the **Properties** panel to configure the SQL transformation. When you configure the transformation, you specify the transformation name and description, configure fields and field mapping, and specify the type of SQL that the transformation processes. You also set the advanced properties for the transformation.

The following image shows the **Properties** panel of an SQL transformation that is configured to process a stored procedure:
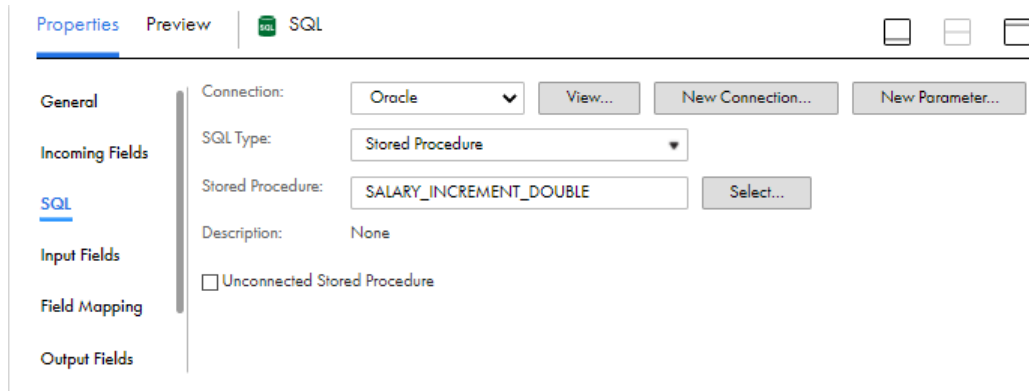
Configure the transformation using the following tabs on the **Properties** panel:

General

Configure the SQL transformation name and optional description.

Incoming Fields

Define field rules that determine the data to include in the transformation.

Not applicable to unconnected SQL transformations.

SQL

Define the database connection and the type of SQL that the transformation processes: either a stored procedure, stored function, or query.

If you configure the transformation to process a stored procedure, stored function, or saved query, you select the stored procedure, stored function, or saved query on this tab.

If you configure the transformation to process a stored procedure, you can choose to run the transformation as an unconnected transformation.

If you configure the transformation to process a user-entered query, the SQL editor appears on this tab. Enter the query in the SQL editor.

Input Fields

For transformations that process stored procedures, displays the stored procedure input fields.

Field Mapping

For stored procedure and stored functions, specify how to map incoming fields to the input fields of the selected stored procedure or function.

You do not configure the field mapping for queries or unconnected SQL transformations.

Output Fields

For stored procedures, stored functions, and saved queries, displays a preview of the SQL transformation output fields. For user-entered queries, configure output fields for the columns retrieved from the database.

For queries, the output fields also include the SQLError field, the optional NumRowsAffected field, and optional pass-through fields.

Advanced

Define advanced properties for the transformation. Advanced properties differ based on the type of SQL that the transformation processes.

Note: Field name conflicts must be resolved in an upstream transformation. You cannot use field name conflict resolution rules in an SQL transformation.

Configuring the SQL type

You can configure the SQL transformation to process a stored procedure, stored function, saved query, or user-entered query on the **SQL** tab of the SQL transformation.

The steps for configuring the transformation vary based on the type of SQL that the transformation processes.

Selecting a stored procedure or function

You can configure the SQL transformation to process a stored procedure or stored function on the **SQL** tab of the SQL transformation.

1. In the **Properties** panel of the SQL transformation, click the **SQL** tab.

   The following image shows the **SQL** tab when the SQL type is set to **Stored Procedure**:
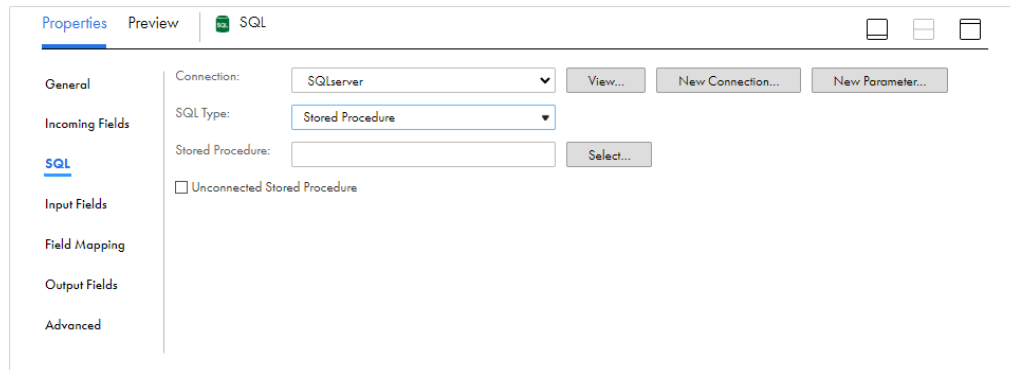
2. Select the connection to the database.

   You can select the connection or use a parameter.

   Note: If you want to parameterize the connection, create the parameter after you select the stored procedure or function.

3. Set the **SQL Type** to **Stored Procedure** or **Stored Function**.

4. Click **Select** to select the stored procedure or function from the database, or enter the exact name of the stored procedure or function to call.

   The stored procedure or function name is case-sensitive.

Note: If you add a new stored procedure to the database while you have the mapping open, the new stored procedure does not appear in the list of available stored procedures. To refresh the list, close and reopen the mapping.

5. If the transformation processes a stored procedure, and you want to run the transformation in unconnected mode, select **Unconnected Stored Procedure**.

   Note: This option is not available for all connector types. For more information, see the help for the appropriate connector.

6. If you want to parameterize the connection, click **New Parameter** and enter the details for the connection parameter.

Selecting a saved query

You can configure the SQL transformation to process a saved query on the **SQL** tab of the SQL transformation.

1. In the **Properties** panel of the SQL transformation, click the **SQL** tab.
2. Select the connection to the database.

   You can select the connection or use a parameter.
3. Set the SQL type to **SQL Query**.
4. Set the query type to **Saved Query**.
5. Click **Select** to select the saved query from a list of Data Integration assets.

Entering a query

You can configure the SQL transformation to process a user-entered query on the **SQL** tab of the SQL transformation. Optionally, you can parameterize the query. When you parameterize the query, you enter the full query in the mapping task.

1. In the **Properties** panel of the SQL transformation, click the **SQL** tab.
2. Select the connection to the database or use a parameter.
3. Set the SQL type to **SQL Query**.
4. Set the query type to **Entered Query**.
5. If you do not want to parameterize the query, enter the query in the query editor.

   Incoming fields are listed on the **Fields** tab. To add a field to the query, select the field and click **Add**.

   You can format the SQL and validate the syntax.

   Note: The syntax validation performs a general SQL syntax check but does not verify the SQL against the database. The validation can return a syntax error even though the SQL is valid for the database. In this case, you can still save and run the mapping.

   If you update the incoming fields after you configure the query, open the **SQL** tab to refresh the changes.

6. If you want to parameterize the query, perform the following steps:

   a. Open the **Parameters** tab and create a new string parameter.

   b. Select the parameter, and then click **Add** to add the parameter to the query editor.

      When you add the parameter to the query editor, Data Integration encloses it in dollar sign characters ($).

      Do not format the SQL or validate the query.

SQL transformation field mapping

Configure field mapping in an SQL transformation to define how to use data from the upstream transformation in a stored procedure or function. You do not configure field mapping when the transformation processes a query.

Configure field mapping on the **Field Mapping** tab of the **Properties** panel.

You can configure the following field mapping options:

Field Map Options

Method of mapping fields to the SQL transformation. Select one of the following options:

- Manual. Manually link incoming fields to the store procedure or function's input fields. Removes links for automatically mapped fields.

- Automatic. Automatically link fields with the same name. Use when all of the fields that you want to link share the same name. You cannot manually link fields with this option.

- Completely Parameterized. Use a parameter to represent the field mapping. In the task, you can configure all field mappings.

- Partially Parameterized. Configure links in the mapping that you want to enforce and use a parameter to allow other fields to be mapped in the mapping task. Or, use a parameter to configure links in the mapping, and allow all fields and links to display in the task for configuration.
  To see more information on field mapping parameters, see *Mappings*.

Show Fields

Controls the fields that appear in the **Incoming Fields** list. Show all fields, unmapped fields, or mapped fields.

Automap

Links fields with matching names. Allows you to link matching fields, and then manually configure other field mappings.

You can map fields in the following ways:

- Exact Field Name. Data Integration matches fields of the same name.

- Smart Map. Data Integration matches fields with similar names. For example, if you have an incoming field `Cust_Name` and a target field `Customer_Name`, Data Integration automatically links the `Cust_Name` field with the `Customer_Name` field.

You can use both Exact Field Name and Smart Map in the same field mapping. For example, use Exact Field Name to match fields with the same name and then use Smart Map to map fields with similar names.

You can undo all automapped field mappings by clicking **Automap** > **Undo Automap**. To unmap a single field, select the field to unmap and click **Actions** > **Unmap**.

Data Integration highlights newly mapped fields. For example, when you use Exact Field Name, Data Integration highlights the mapped fields. If you then use Smart Map, Data Integration only highlights the fields mapped using Smart Map.

Action menu

Additional field link options. Provides the following options:

- Map selected. Links the selected incoming field with the selected stored procedure or function input field.

- Unmap selected. Clears the link for the selected field.

- Clear all. Clears all field mappings.

Show

Determines how field names appear in the **Stored Procedure Input Fields** list. Use technical field names or labels.

SQL transformation output fields

You can view output fields for the SQL transformation on the **Output Fields** tab of the **Properties** panel. The Mapping Designer displays the name, type, precision, scale, and origin for each output field.

Information on the **Output Fields** tab varies based on the SQL type.

Output fields for stored procedures and functions

When the SQL transformation processes a stored procedure or function, the output fields include output parameters from the database. You cannot edit the transformation output fields. If you want to exclude output fields from the data flow or rename output fields before you pass them to a downstream transformation, configure the field rules for the downstream transformation.

Output fields for queries

When the SQL transformation processes a query, the output fields include the following fields:

Retrieved column fields

When the SQL query contains a SELECT statement, the transformation returns one row for each database row that it retrieves.

For user-entered queries, you must configure an output field for each column in the SELECT statement. The output fields must be in the same order as the columns in the SELECT statement.

For saved queries, Data Integration creates the output fields.

SQLError field

Data Integration returns row errors to the SQLError field when it encounters a connection or syntax error. It returns NULL to the SQLError field when no SQL errors occur.

For example, the following SQL query generates a row error from an Oracle database when the Employees table does not contain Product_ID:

```
SELECT Product_ID FROM Employees
```

Data Integration generates one row. The SQLError field contains the following error text in one line:

```
ORA-0094: "Product_ID": invalid identifier Database driver error... Function Name:
Execute SQL Stmt: SELECT Product_ID from Employees Oracle Fatal Error
```

When a query contains multiple statements, and you configure the SQL transformation to continue on SQL error, the SQL transformation might return rows from the database for one query statement, but return database errors for another query statement. The SQL transformation returns any database error in a separate row.

NumRowsAffected field

You can enable the NumRowsAffected output field to return the number of rows affected by the INSERT, UPDATE, or DELETE query statements in each input row. Data Integration returns the NumRowsAffected for each statement in the query. NumRowsAffected is disabled by default.

When you enable NumRowsAffected and the SQL query does not contain an INSERT, UPDATE, or DELETE statement, NumRowsAffected is zero in each output row.

The following table lists the output rows that the SQL transformation generates when you enable NumRowsAffected:

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

When a query contains multiple statements, Data Integration returns the NumRowsAffected for each statement. NumRowsAffected contains the sum of the rows affected by each INSERT, UPDATE, and DELETE statement in an input row.

For example, a query contains the following statements:

```
DELETE from Employees WHERE Employee_ID = '101';
SELECT Employee_ID, LastName from Employees WHERE Employee_ID = '103';
INSERT into Employees (Employee_ID, LastName, Address)VALUES ('102', 'Gein', '38
Beach Rd')
```

The DELETE statement affects one row. The SELECT statement does not affect any row. The INSERT statement affects one row.

Data Integration returns one row from the DELETE statement. NumRowsAffected is equal to one. It returns one row from the SELECT statement, NumRowsAffected is zero. It returns one row from the INSERT statement with NumRowsAffected equal to one.

Pass-through fields

Define incoming fields as pass-through fields to pass data through the SQL transformation. The SQL transformation returns data from pass-through fields whether or not the SQL query returns rows.

When the source row contains a SELECT statement, the SQL transformation returns the data in the pass-through field in each row it returns from the database. If the query result contains multiple rows, the SQL transformation repeats the pass through field data in each row.

When a query returns no rows, the SQL transformation returns the pass-through column data and null values in the output fields. For example, queries that contain INSERT, UPDATE, and DELETE statements return no rows. If the query has errors, the SQL transformation returns the pass-through column data, the SQLError message, and null values in the output fields.

To define a pass-through field, click **Add** in the Pass-Through Fields area, and then select the field you want to pass through the SQL transformation. When you configure an incoming field as a pass-through field, Data Integration adds the field with the suffix "_output" in the Pass-Through Fields area.

If you configure a field as a pass-through field and then change the field name in the source, Data Integration does not update the pass-through field name and no data is passed through the field. In the SQL transformation, delete the old pass-through field and configure the updated incoming field as a pass-through field.

Advanced properties

Configure advanced properties for the SQL transformation on the **Advanced** tab. The advanced properties vary based on whether the transformation processes a stored procedure or function or a query.

Advanced properties for stored procedures or functions

The following table describes the advanced properties when the transformation processes a stored procedure or function:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Subsecond Precision	Subsecond precision for datetime fields. You can change the precision for databases that have an editable scale for datetime data. If you enable SQL ELT optimization, the database returns the complete datetime value, regardless of the subsecond precision setting. Enter a positive integer value from 0 to 9. Default is 6 microseconds.
Stored Procedure Type	For unconnected transformations, determines when the stored procedure runs. Select one of the following options: - Target Pre Load. Runs before the target is loaded. - Target Post Load. Runs after the target is loaded. - Normal. Runs on a row-by-row basis. - Source Pre Load. Runs before the mapping receives data from the source. - Source Post Load. Runs after the mapping receives data from the source. **Note:** When the SQL transformation is part of a mapplet, the stored procedure type is always Normal.
Call Text	For unconnected transformations with stored procedure type Target Pre/Post Load or Source Pre/Post Load, enter the call text for the stored procedure. The call text is the stored procedure name followed by the input parameters in parentheses. If there are no input parameters, you must include an empty pair of parentheses. Do not include the SQL statement EXEC or use the :SP keyword. The call text does not apply to Normal stored procedure types.

Advanced properties for queries

The following table describes the advanced properties when the transformation processes a saved or user-entered query:

Property	Description
Behavior	Transformation behavior, either active or passive. If active, the transformation can generate more than one output row for each input row. If passive, the transformation generates one output row for each input row. Default is Active.
Continue on SQL Error within Row	Continues processing the remaining SQL statements in a query after an SQL error occurs. Enable this option to ignore SQL errors in a statement. Data Integration continues to run the rest of the SQL statements for the row. The SQL transformation does not generate a row error, but the SQLError field contains the failed SQL statement and error messages. **Tip:** Disable this option to debug database errors. Otherwise, you might not be able to associate errors with the query statements that caused them. Default is disabled.
Auto Commit	Enables auto-commit for each database connection. Each SQL statement in a query defines a transaction. A commit occurs when the SQL statement completes or the next statement is executed, whichever comes first. Default is disabled.
Max Output Row Count	The maximum number of rows that the SQL transformation can output from a SELECT query. To configure unlimited rows, set this property to zero. Default is 600.
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Transformation Scope	The method in which Data Integration applies the transformation logic to incoming data. Select one of the following options: - Row. Applies the transformation logic to one row of data at a time. Choose Row when the results of the transformation depend on a single row of data. - Transaction. Applies the transformation logic to all rows in a transaction. Choose Transaction when a row of data depends on all rows in the same transaction, but does not depend on rows in other transactions. - All Input. Applies the transformation logic on all incoming data. When you choose All Input, Data Integration drops incoming transaction boundaries. Choose All Input when a row of data depends on all rows in the source. Default is Row.

CHAPTER 35

Structure Parser transformation

The Structure Parser transformation transforms your input data into a user-defined structured format based on an intelligent structure model. You can use the Structure Parser transformation to analyze data such as log files, clickstreams, XML or JSON files, Word tables, and other unstructured or semi-structured formats.

You can connect a Structure Parser transformation to the following types of sources:

- A Source transformation based on a flat file to process local input files
- A Source transformation based on a Hadoop Files V2 connection to stream input files in HDFS or to process local input files

When you configure a Structure Parser transformation, you associate it with an intelligent structure model. An intelligent structure model is an asset that Intelligent Structure Discovery generates to represent the data that you expect the model to parse at run time. You can create a model before you configure the Structure Parser transformation or as you configure it.

Intelligent Structure Discovery generates the intelligent structure model based on a sample of your input data or a schema that you provide. You can create a model from the following input types:

- Avro files
- Cobol copybooks
- Data within PDF form fields
- Data within Microsoft Word tables
- JSON files
- Machine generated files such as weblogs and clickstreams
- Microsoft Excel files
- ORC files
- PDF files
- Parquet files
- Text files, including delimited files such as CSV files and complex files that contain textual hierarchies
- XML files
- XSD files

After Intelligent Structure Discovery generates the intelligent structure model, you can refine the model and customize the structure of the output data. You can edit the nodes in the model to combine, exclude, flatten, or collapse them.

Processing input from a Hadoop Files source

When you connect a Hadoop Files source to the Structure Parser transformation, you can process local file input or HDFS input. For example, the Structure Parser transformation can process big data input from a Hadoop Files source in streaming mode in a Hadoop environment.

You can create a Hadoop Files connection to a Hortonworks Data Platform or Cloudera distribution to provide input to a Structure Parser transformation.

Hadoop Files V2 Connector can process a single file or an entire directory. When the connector processes a directory, it processes the files recursively.

To ensure that the Structure Parser transformation can process output from the Hadoop Files source, define the field mapping settings in the Structure Parser transformation. Map the **Data** and **File Path** fields in the **Incoming Field** panel to the **Data** and **File Path** fields in the **Structure Parser Input Fields** panel. You can map the fields automatically or manually.

Processing input from a flat file source

The Structure Parser transformation can process input from one or more files when you add a flat file source.

The Source transformation uses a reference file that contains a file path or a list of file paths to one or more files that you want the Structure Parser transformation to process. Ensure that you use a reference file when you configure the Source transformation. On the **Source** tab, select the reference file in the **Object** field.

Configuring the flat file source

To use a flat file source with a Structure Parser transformation, ensure that the Source transformation has one source field.

1. On the Source transformation **Source** tab, select **Formatting Options** near the **Object** field.
2. Select **None** for the **Text Qualifier**.
3. Select **Auto-generate** for the **Field Labels**.
4. Click **OK**.

Configuring the Structure Parser transformation to access flat files

To ensure that the Structure Parser transformation can access the files in the reference file, configure the Field Mapping settings.

▶ On the Structure Parser transformation **Field Mapping** tab, map the field in the **Incoming Field** panel to the **File Path** field in the **Structure Parser Input Fields** panel.

Structure Parser field mapping

Configure the field mapping in a Structure Parser transformation to define which fields in the source transformation to map to fields in the Structure Parser transformation. Configure field mappings on the **Field Mapping** tab of the **Properties** panel.

The Structure Parser contains the following Field Mapping fields in the **Structure Parser Input Fields** panel:

Data

> Used to map the data field in the source transformation to the data field in the Structure Parser transformation. Map this field for a Hadoop Files source and for a Structure Parser transformation that is used midstream.

File Path

> Used to map the file path or reference file in the source transformation to the file path field in the Structure Parser transformation.

You can configure the following field mapping options:

Field Map Options

> Determines how field names appear in the Structure Parser transformation input fields list. Use technical field names or labels.
>
> Select one of the following options:
>
> - Manual. Manually link incoming fields to target fields. Removes links for automatically mapped fields.
> - Automatic. Automatically link fields with the same name. Use when all of the fields that you want to link share the same name. You can't manually link fields with this option.
>
> **Note:** You can't use parameterized field names with the Structure Parser transformation. Do not select the options **Completely Parameterized** or **Partially Parameterized**.

Automap

> Links fields with matching or similar names. Allows you to link matching fields, and then manually configure other field mappings.

Show Fields

> Controls the fields that appear in the **Incoming Fields** list. Show all fields, unmapped fields, or mapped fields.

For a Hadoop Files source

For a Hadoop Files source, map the **Data** and the **File Path** fields in the **Incoming Field** panel to the same fields in the **Structure Parser Input Fields** panel. You can use the **Automap** option to automatically map these fields.

For a flat file source

For a flat file source, map the field in the **Incoming Field** panel to the **File Path** field in the **Structure Parser Input Fields** panel.

For a Structure Parser transformation that is used midstream

When using a Structure Parser transformation midstream, that is not after the source transformation, map the incoming data ports to fields in the **Structure Parser Input Fields** panel.

Output fields

When you select an intelligent structure to use in a Structure Parser transformation, the intelligent structure model output fields appear on the **Output Fields** tab of the Properties panel.

In a mapping, the output fields are grouped into the following groups:

- Unidentified group. This group contains the data that was not identified by the intelligent structure. You might want to pass this data to a target file for further analysis.

- Output groups. One or more groups that contain the data that the intelligent structure model identifies.

You can select a group or groups to transfer pass-through fields to. Pass-through fields are fields that you don't map in the transformation field mapping and that the transformation transfers to the selected group or groups as is. Use this option if you want to use the pass-through fields later in the mapping, for example, to pass a timestamp field to the next downstream transformation. The Structure Parser transformation passes the pass-through fields to each selected group.

In a mapping in advanced mode, the output fields are put in one group. This group includes pass-through fields if you configure the transformation to enable pass-through fields.

The **Output Fields** tab displays the first five fields in each output group, including the name, type, precision, scale, and origin of each field. You can click the link at the bottom of the group to display all the fields in the group. The origin of a field shows the path of the respective node in the intelligent structure model. If a name of a node in the intelligent structure model contains special characters, the Secure Agent replaces them with an underscore (_) character, and the **Output Fields** tab displays the revised name as the output field name. The **Output Fields** tab doesn't display the origin of fields with revised names.

To edit the precision of an output field, click the precision value and enter the precision you require.

Note: Intelligent Structure Discovery doesn't enforce precision and scale on decimal fields.

You can't edit the transformation output fields. If you want to exclude output fields from the data flow or rename output fields before you pass them to a downstream transformation or target, configure the field rules for the downstream transformation.

Output groups for relational output

If you configure the transformation to generate relational output, the transformation can contain multiple output groups. Each output group contains one or more fields that you can write to a relational target or pass to a downstream transformation for further processing.

In advanced mode, the Structure Parser transformation can't generate relational output.

Output groups for JSON, JSON lines, and XML output

If you configure the transformation to generate JSON, JSON lines, or XML output, the transformation contains one output group. The output group contains one string field which contains the output data. The default precision is 1,000,000 characters. You can pass the data in the string field to a downstream transformation or to a target of the appropriate type. For example, if you configure the transformation to generate XML output, you can connect the output field to a flat file target to save the output to an XML file.

Output groups for Avro, Parquet, and ORC output

If you configure the transformation to generate Avro, Parquet, or ORC output, the transformation contains one output group. The output group contains one binary field which contains the output data. The default precision is 1,000,000 characters. You can pass the data in the binary field to a downstream transformation or to a target of the appropriate type. For example, if you configure the transformation to generate Parquet output, you can connect the output field to a target such as Amazon S3 or Hadoop Files.

In advanced mode, the Structure Parser transformation can't generate Avro, Parquet, or ORC output.

Advanced properties

You can configure advanced properties for a Structure Parser transformation. The advanced properties control the transformation scope.

The properties that are available vary based on the mapping mode.

You can configure the following property:

Property	Description
Transformation Scope	Specifies how Data Integration applies the transformation logic to incoming data. Select one of the following options: - Transaction. Applies the transformation logic to all rows in a transaction. Choose Transaction when a row of data depends on all rows in the same transaction, but does not depend on rows in other transactions. - All Input. Applies the transformation logic on all incoming data. When you choose All Input, Data Integration drops incoming transaction boundaries. Choose All Input when a row of data depends on all rows in the source. - Row. Applies the transformation logic to one row of data at a time. Choose Row when the results of the transformation depend on a single row of data.

Structure Parser transformation configuration

When you configure a Structure Parser transformation, you associate an intelligent structure model with the transformation and select the output type.

To associate an intelligent structure model with the transformation, you can select an existing model or create a new model. If you create a new model, you can select one of the following actions:

- Design new. You create the model in the Intelligent Structure Model page. For more information about creating an intelligent structure model, see *Components*.

- Auto-generate from a sample. You select a sample file and Intelligent Structure Discovery creates the model and saves it in the location that you select.

You can also select and view a model on the Intelligent Structure Model page before you use it.

When you configure a Structure Parser transformation, you select the data output type. The transformation can generate the following output types:

- Relational

- JSON

- JSON lines

- XML

- Avro

- Parquet

- ORC

Configuring a Structure Parser transformation

Add a Structure Parser transformation to a mapping and configure transformation settings.

1. Add a Structure Parser transformation to the mapping and configure general settings.
2. On the **Properties** panel, click **Structure Parser** and choose one of the following options to associate an intelligent structure model with the transformation:

Option	Description
Select	Select an existing intelligent structure model.
New > Design New	Create a new model in the Intelligent Structure Model page.
New > Auto-generate from sample file	Select a sample file and a location for the model and click **Create**. Intelligent Structure Discovery creates the model and saves it in the selected location.

3. Select the output type for the Structure Parser transformation from the **Output As** list.
4. If you select the JSON output type, you can select **Include empty tags** to add all the model tags to the output at run time, including tags that don't exist in the input. The transformation adds tags that don't exist in the input as empty tags with a NULL value.
5. On the **Properties** panel, click **Incoming Fields** and configure the incoming fields.
6. On the **Properties** panel, click **Field Mappings** and configure field mapping.
7. Optionally, on the **Properties** panel, click **Advanced** and configure the transformation scope.
8. Link the previous transformation on the map to the Structure Parser transformation
9. Link the Structure Parser transformation to the downstream transformation and select an output group.

Selecting an intelligent structure model

Select an intelligent structure model to include in the Structure Parser transformation on the **Structure Parser** tab of the **Properties** panel.

1. In the **Properties** panel of the Structure Parser transformation, click the **Structure Parser** tab.
2. Click **Select**.

   The **Select Intelligent Structure** dialog box appears.
3. In the **Explore** list, select an intelligent structure model.
4. To search for an intelligent structure model, select the search criteria, enter the characters to search for in the **Find** field.

   You can search for an intelligent structure model by name or description. You can sort the intelligent structure models by name, type, description, tags, status, or date last modified.
5. Choose the intelligent structure model to include in the Structure Parser transformation and click **Select**.

   The intelligent structure model appears in the **Properties** panel.

Selecting an output group

When you link the Structure Parser transformation to a downstream transformation, you must select one output group to map to the downstream transformation. If you want to map more than one output group, create a different downstream transformation for each output group.

1. Link the Structure Parser transformation to a downstream transformation.

   The **Select Output Group** dialog box appears.

2. To map the data that was identified by the intelligent structure, select one of the output groups. Alternatively, to map the data that was not identified by the intelligent structure, select the **Unidentified** output group.

3. Click **OK**.

Rules and guidelines for the Structure Parser transformation

Consider the following rules and guidelines when you use a Structure Parser transformation:

- When you open a mapping that contains a Structure Parser transformation in the Mapping Designer, and the associated intelligent structure model changed after it was associated with the transformation, a message appears in the Mapping Designer. Click the link that is provided in the message to update the model so that it complies with the mapping.

- If you update an intelligent structure model that was created with an older version of Intelligent Structure Discovery, where all output fields were assigned a string data type, the Structure Parser transformation might change field data types during the update. If, for any of the affected fields, the downstream transformation or the target expects a string, edit the model to change the data type back to string. Then, in the Mapping Designer, click the provided link to update the model in the transformation again.
  For more information about editing an intelligent structure model, see *Components*.

- The Structure Parser transformation can parse recursive elements in an XSD input file if all of the following conditions are true:

  - The recursive element isn't nested inside a repeating element.

  - The recursive element always uses the same name element.

  - The Structure Parser transformation isn't used in a mapping in advanced mode.

  If the transformation can't parse a recursive element, it outputs the data in a single output group. The transformation can parse recursive elements in mappings created after the October 2023 release or when you reupload the schema file after upgrading to the October 2023 release.

- When you use a Structure Parser transformation in a mapplet, to prevent runtime errors, be sure that the combined Mapplet transformation name doesn't exceed 80 characters. For more information, see .

Rules and guidelines for mappings in advanced mode

Consider the following rules and guidelines when you use the Structure Parser transformation in a mapping in advanced mode:

- In advanced mode, a Structure Parser transformation can process a JSON object as input, but not a JSON array.

- By default in advanced mode, the schema of the source data must match exactly the schema of the intelligent structure model associated with the transformation, and the transformation can't process pass-through fields. To process pass-through fields, set the following Spark custom property in the Spark session properties for the mapping task: `Spark.MSPEnableUnassigedData=true`

  When you set this property, the output group contains an array called *UnassignedData* that contains the data that was not identified by the intelligent structure.

Rules and guidelines for output types

Consider the following rules and guidelines when you select the output type for a Structure Parser transformation:

- Before you use the Parquet output type, set the *HADOOP_HOME* and *hadoop.home.dir* environment variables.

- When you use an AVRO output type, the transformation doesn't generate output for input fields with identical names.

- The transformation creates ORC output based on the local date and time. Processing the same input in different locations or environments might result in output with different times and time formats.

- We recommend that you use a binary output type to create large XML and JSON files.

Structure Parser transformation example

You need to parse the unstructured data in a log file and write the data to a target file in relational format.

You need to configure an intelligent structure model that analyzes unstructured data and discovers its structure.

The following image shows the log file that you want to parse:

```
[2015-06-19 08:32:55] [info]   [ 9936] Commons Daemon procrun (1.0.13.0 64-bit) started
[2015-06-19 08:32:55] [info]   [ 9936] Running 'VDS_NODE_65_inw00004001' Service...
[2015-06-19 08:32:55] [info]   [ 4156] Starting service...
[2015-06-19 08:32:56] [info]   [ 4156] Service started in 1159 ms.
[2015-06-19 16:03:42] [info]   [ 9704] Console SHUTDOWN event signaled
[2015-06-19 16:03:42] [info]   [ 9704] Stopping service...
[2015-06-19 16:03:48] [info]   [ 9704] Service stopped.
[2015-06-19 16:03:48] [info]   [ 9936] Run service finished.
[2015-06-19 16:03:48] [info]   [ 9936] Commons Daemon procrun finished
[2015-06-19 16:03:48] [error] [ 4156] Failed to set service status
```

Create the intelligent structure model in Informatica Cloud. The following image shows the intelligent structure model details:

Intelligent Structure Details

Name	node-log
Location	log
Description	
Created On	Nov 29, 2016 4:00:31 AM
Updated On	Jul 18, 2017 6:03:39 AM
Created By	log
Updated By	log

Output Groups

Group	Info	Name
Unidentified	▤	unidentified
element	▤	index
	▤	date
	▤	time
	▤	firstName
	▤	number
	▤	money

To parse the log file, use a Structure Parser transformation in a mapping to transform the data in the log file.

In the Mapping Designer, you add a source object that is flat file that contains the path to the log file you want to parse.

The following image shows the selected source file properties:

You add a Structure Parser transformation. Configure it to use the intelligent structure model that you configured. Select the relational output type.

You connect the source object to the Structure Parser transformation. To map the incoming data to the fields of the transformation, select the Structure Parser transformation. In the **Field Mapping** tab, map **Path** to the Structure Parser Input Fields **Field Path**.

The following image shows the field mapping:

Add a text file target transformation for the parsed output group that you want to process named TargetFile. Add a separate text file target transformation for data that was not identified named Unidentified.

The following image shows the mapping:
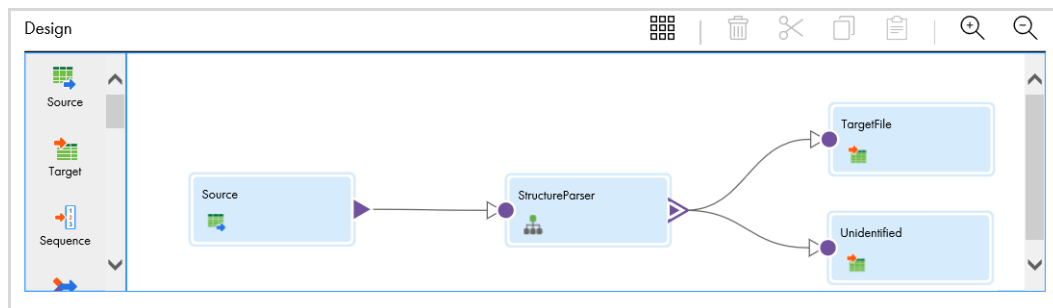
Run the mapping to write the data in a structured format to the TargetFile transformation. The mapping sends any data that was not identified by the intelligent structure to the Unidentified transformation.

The following image shows the parsed data output file from the TargetFile transformation:

```
"date","time","token","number","value"
"2015-06-19","08:32:55","info","9936","Commons Daemon procrun (1.0.13.0 64-bit)
started"
"2015-06-19","08:32:55","info","9936","Running 'VDS_NODE_65_inw00004001' Service..."
"2015-06-19","08:32:55","info","4156","Starting service..."
"2015-06-19","08:32:56","info","4156","Service started in 1159 ms."
"2015-06-19","16:03:42","info","9704","Console SHUTDOWN event signaled"
"2015-06-19","16:03:42","info","9704","Stopping service..."
"2015-06-19","16:03:48","info","9704","Service stopped."
"2015-06-19","16:03:48","info","9936","Run service finished."
"2015-06-19","16:03:48","info","9936","Commons Daemon procrun finished"
"2015-06-19","16:03:48","error","4156","Failed to set service status"
```

If you need to further parse the data, you can include additional Structure Parser transformations midstream that will parse the output from the preceding parser.

CHAPTER 36

Transaction Control transformation

The Transaction Control transformation is an active transformation that commits or rolls back sets of rows during a mapping run. Use the Transaction Control transformation to commit or roll back transactions from transactional targets such as relational, XML, Amazon Redshift, and REST V2 targets. You can also use the transformation in a mapping to write data to a different flat file each time that Data Integration starts a new transaction.

You might want to use a Transaction Control transformation when you process large amounts of data. You can use the Transaction Control transformation to commit the data at certain intervals to prevent data loss. For example, you run a mapping that processes thousands of records in a table that is sorted by order type. You might want to commit the data each time that the mapping processes a different order type.

In a Transaction Control transformation, a transaction is the row or set of rows bound by commit or roll back rows. A transaction can be based on a group of rows that are ordered on a common key, such as employee ID or order entry date. The number of rows in each transaction can vary.

You define a transaction by specifying the transaction control condition in the transformation. Based on whether the condition is met, you can choose to commit rows, roll back rows, or continue processing data without changing the transaction boundaries.

When you run the mapping task, Data Integration evaluates the transaction control condition for each row that enters the transformation. When it evaluates a commit row, it commits all rows in the transaction to the targets. When Data Integration evaluates a roll back row, it rolls back all rows in the transaction from the targets.

If the mapping has a flat file target that is created at run time, you can generate an output file each time Data Integration starts a new transaction. You can dynamically name each target flat file.

Example

You want to use transaction control to write order information based on the order type. You want to ensure that all orders of a specific type are written to a different target file. To accomplish this, you create the following mapping:
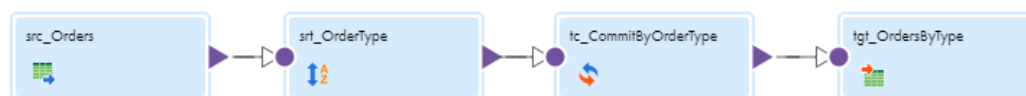
The mapping contains the following transformations:

Source transformation

Configure the connection to the source ORDERS table.

Sorter transformation

Configure the sort condition to sort the source data by ORDER_TYPE.

Transaction Control transformation

Create the following transaction control condition to commit data when the Integration Service encounters a new order entry date:

Property	Value
Transaction Control Condition	If Field Value Changes
Field	ORDER_TYPE (string)
Then	Commit Before
Else	Continue Transaction

Target transformation

Create a new file target at run time and specify a dynamic file name. Use the following expression for the target name to create a different target file for each order type:

```
'Orders_ '||ORDER_TYPE||'.csv'
```

Transaction control condition

Configure the transaction control condition on the **Transaction Control** tab. The transaction control condition tests each row to determine when to commit rows, roll back rows, or continue without any transaction changes.

The following image shows the **Transaction Control** tab:
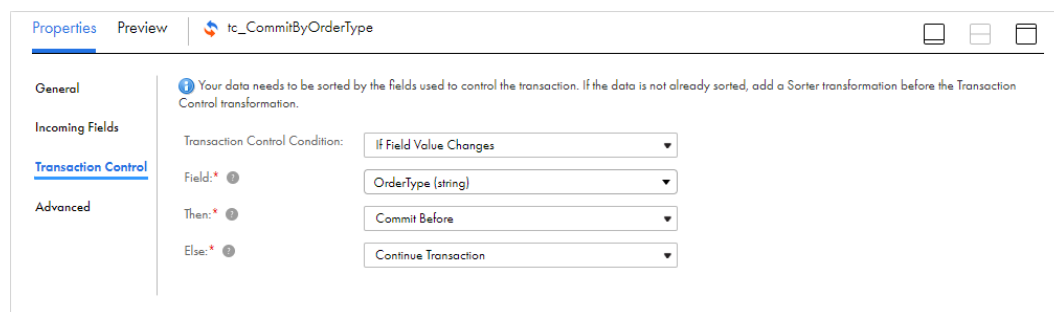
Note: Before you specify the transaction control condition, verify that the incoming data is sorted. Incoming data must be sorted by the fields that you use in the transaction control condition.

You can use one of the following types of conditions to test the row data:

If Field Value Changes

Use an If Field Value Changes condition when you want to test whether a field value changes. For example, if ORDER_DATE changes, then commit the transaction before writing the current row to the target. Otherwise, continue processing the data.

Select the field to test from the **Field** list.

Advanced

Use an advanced condition when you want to use an expression to test the row data. For example, if NEW_FILE_FLAG='Y' AND DEPT_ID>1000, then commit the transaction but include the current row in the next transaction. Otherwise, continue processing the data.

Configure the expression in the **If** part of the condition. The expression can use fields, parameters, built-in functions, and user-defined functions.

Parameterized

You can use an expression parameter to represent the condition. Enter the parameter value when you run the mapping task or enter the value in a parameter file.

You can specify the following actions for the **Then** and **Else** parts of the condition based on the test results:

Action	Description
Continue Transaction	Data Integration does not perform any transaction change for this row.
Commit Before	Data Integration commits the transaction, begins a new transaction, and writes the current row to the target. The current row is in the new transaction.
Commit After	Data Integration writes the current row to the target, commits the transaction, and begins a new transaction. The current row is in the committed transaction.
Rollback Before	Data Integration rolls back the current transaction, begins a new transaction, and writes the current row to the target. The current row is in the new transaction.
Rollback After	Data Integration writes the current row to the target, rolls back the transaction, and begins a new transaction. The current row is in the rolled back transaction.

The **Then** and **Else** parts of the condition must contain different actions.

Using Transaction Control transformations in mappings

Transaction Control transformations are transaction generators. They define and redefine transaction boundaries in a mapping. They drop any incoming transaction boundaries from an upstream active source or transaction generator, and they generate new transaction boundaries downstream.

Transaction Control transformations can be effective or ineffective for the downstream transformations and targets in the mapping. The Transaction Control transformation becomes ineffective for downstream transformations or targets if you put a transformation that drops incoming transaction boundaries after it. This includes any of the following active sources or transformations:

- Aggregator transformation with the All Input level transformation scope
- Java transformation with the All Input level transformation scope
- Joiner transformation with the All Input level transformation scope
- Rank transformation with the All Input level transformation scope

- Sorter transformation with the All Input level transformation scope
- SQL transformation that uses a saved or user-entered query and has the All Input level transformation scope
- Transaction Control transformation
- Any multiple input group transformation that is connected to multiple upstream transaction control points

Although a Transaction Control transformation may be ineffective for a target, it can be effective for downstream transformations. Downstream transformations with the Transaction level transformation scope can use the transaction boundaries defined by an upstream Transaction Control transformation.

The following image shows a valid mapping with a Transaction Control transformation that is effective for a Sorter transformation, but ineffective for the target:
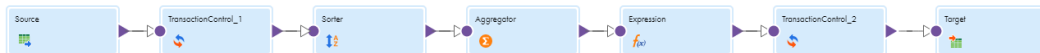
In this example, TransactionControl_1 is ineffective for the target, but effective for the Sorter transformation. The transformation scope for the Sorter transformation is Transaction. It uses the transaction boundaries defined by TransactionControl_1. The transformation scope for the Aggregator transformation is All Input. It drops transaction boundaries defined by TransactionControl_1. Transaction control transformation TransactionControl_2 is an effective Transaction Control transformation for the target.

Sample transaction control mappings with multiple targets

In a mapping with multiple targets, a Transaction Control transformation can be effective for one target and ineffective for another target. If each target is connected to an effective Transaction Control transformation, the mapping is valid. If one target in the mapping is not connected to an effective Transaction Control transformation, the mapping is invalid.

The following image shows a valid mapping with both an ineffective and an effective Transaction Control transformation:
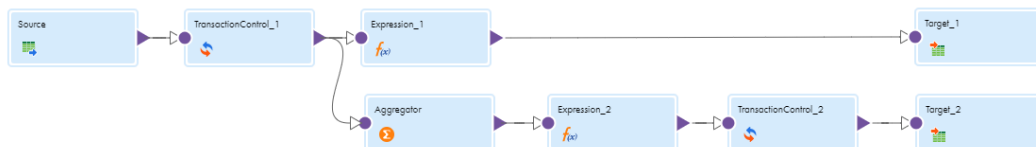
Data Integration processes TransactionControl_1, evaluates the transaction control expression, and creates transaction boundaries. The mapping does not include a transformation that drops transaction boundaries between TransactionControl_1 and Target_1, making TransactionControl_1 effective for Target_1. Data Integration uses the transaction boundaries defined by TransactionControl_1 for Target_1.

However, the mapping includes a transformation that drops transaction boundaries between TransactionControl_1 and Target_2, which makes TransactionControl_1 ineffective for Target_2. When Data Integration processes the Aggregator transformation, with transformation scope set to All Input, it drops the transaction boundaries defined by TransactionControl_1 and outputs all rows in an open transaction. Then Data Integration evaluates TransactionControl_2, creates transaction boundaries, and uses them for Target_2.

If a roll back occurs in TransactionControl_1, Data Integration rolls back only rows from Target_1. It does not roll back any rows from Target_2.

The following image shows an invalid mapping with both an ineffective and an effective Transaction Control transformation:
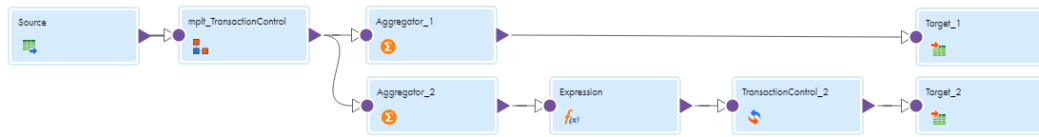
The mapplet, mplt_TransactionControl, contains a Transaction Control transformation. It is ineffective for Target_1 and Target_2. The transformation scope for Aggregator_1 is All Input. It is an active source for Target_1. The transformation scope for Aggregator_2 is All Input. It is an active source for Target_2. TransactionControl_2 is effective for Target_2.

The mapping is invalid because Target_1 is not connected to an effective Transaction Control transformation.

Guidelines for using Transaction Control transformations in mappings

Consider the following guidelines when you use a Transaction Control transformation in a mapping:

- Incoming data must be sorted by the fields that you use in the transaction condition. Place a Sorter transformation upstream of the Transaction Control transformation or use a sorted data source.

- Configuring the transaction control condition to perform frequent commits can affect performance.

- If the mapping includes an XML target, and you choose to append or create a new document on commit, the input groups must receive data from the same transaction control point.

- Transaction Control transformations that are connected to any target that does not support batch or transaction real-time processing are ineffective for those targets.

- You must connect each target instance to a Transaction Control transformation.

- You can connect multiple targets to the same Transaction Control transformation.

- You can connect only one effective Transaction Control transformation to a target.

- You cannot place a Transaction Control transformation in a pipeline branch that starts with a Sequence Generator transformation.

- If you use a dynamic Lookup transformation and a Transaction Control transformation in the same mapping, a rolled-back transaction might result in unsynchronized target data.

- Either all targets or no targets in the mapping should be connected to an effective Transaction Control transformation.

Advanced properties

You can configure advanced properties for a Transaction Control transformation. The advanced properties control the tracing level for session log messages.

You can configure the following property:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

CHAPTER 37

Union transformation

The Union transformation is an active transformation that you use to merge data from multiple pipelines into a single pipeline.

For data integration patterns, it is common to combine two or more data sources into a single stream that includes the union of all rows. The data sources often do not have the same structure, so you cannot freely join the data streams. The Union transformation enables you to make the metadata of the streams alike so that you can combine the data sources in a single target.

The Union transformation merges data from multiple sources similar to the UNION ALL SQL statement. For example, you might use the Union transformation to merge employee information from ADP with data from a Workday employee object.

You can add, change, or remove specific fields when you merge data sources with a Union transformation.

At run time, the mapping task processes input groups in parallel. It concurrently reads the sources connected to the Union transformation and pushes blocks of data into the input groups of the transformation. As the mapping runs, it merges data into a single output group based on the field mappings.

Comparison to Joiner transformation

A Union transformation can merge data from multiple sources but does not combine data based on a join condition like a Joiner transformation.

The following table identifies some key differences between the Union transformation and Joiner transformation, which also merges data from multiple sources. Factor these differences into your mapping design:

Requirement	Union transformation	Joiner transformation
Combine records based on a join condition	No. The Union Transformation is equivalent to a UNION ALL statement in SQL, which combines data vertically from multiple sources.	Yes. The Joiner transformation supports Normal, Right Outer, Left Outer, and Full Outer JOINs.
Include multiple input groups	Yes. You can define multiple input groups and one output group.	Yes. You can define two input groups, Master and Detail.
Include heterogeneous sources	Yes	Yes

Requirement	Union transformation	Joiner transformation
Merge different data types	All of the source columns must have similar data types. The number of columns in each source must be the same.	At least one column in the sources to be joined must have the same data type.
Generate transactions	No	Yes

Planning to use a Union transformation

When you use a Union transformation in a mapping, consider the following guidelines:

- Before you add a Union transformation to a mapping, add all Source transformations and include the other upstream transformations that you want to use.

- You can use a Sequence Generator transformation upstream from a Union transformation if you connect both the Sequence Generator and a Source transformation to one input group of the Union transformation.

Input groups

By default, a Union transformation has two input groups. If you want to merge data from more than two sources, add an input group for each additional source. Each group can have different field rules for each upstream transformation.

The input groups have the following characteristics:

- The Union transformation initializes its output fields based on fields in the first source that you connect to an input group.

- Each input group can use a different field mapping mode or parameter.

- You can parameterize the field mappings or define the field mapping for each input group.

To add an input group, in the Mapping Designer, connect an upstream transformation to the "New Group" group of the Union transformation. You can also add input groups on the **Incoming Fields** tab of the Union transformation.

You can rename input groups. You can also delete input groups as long as there are at least two remaining input groups. Rename and delete input groups on the **Incoming Fields** tab.

To delete an input group, hover over the input group name on the **Incoming Fields** tab. Click the **Delete** icon that appears. When the transformation has only two input groups, the **Delete** icon won't appear because Union transformations require at least two input groups.

Output fields

When you connect upstream transformations to a Union transformation, you initialize the output fields. The initial output fields are an exact copy of the input fields in group Input1.

When you define output fields, note the following:

- After you initialize the output fields, you cannot change the output fields by connecting or disconnecting the input group.
- You can manually add output fields if you add them before you connect one of the Union transformation input groups.
- When you add an output field, you define the field name, data type, precision, scale, and optional description. The description can contain up to 4000 characters.
- If you connect the Union transformation to an upstream transformation that does not pass in any fields, the output fields are not initialized.
- At run time, the mapping passes null values to output fields that are not in a field mapping.

Field mappings

The Union transformation can merge data from multiple source pipelines. The sources can have the same set of fields, have some matching fields, or use parameterized field mappings.

When you work with field mappings in a Union transformation, note the following:

- You must use input groups where the fields have the identical name, type, precision, and scale.
- You can edit, remove, or manually add some of the output fields.
- As part of the field mapping, you choose an input group and specify the parameter from the input group.
- You can use parameters for fields in all input groups.
- You can parameterize the field mapping or map by field name for each input group. At run time, the task adds an exact copy of the fields from the input group as output fields.

If you want Data Integration to automatically link fields with the same name and you also want to manually map fields, select the **Manual** option and click **Automap**.

You can map fields in the following ways:

- Exact Field Name. Data Integration matches fields of the same name.
- Smart Map. Data Integration matches fields with similar names. For example, if you have an incoming field `Cust_Name` and a target field `Customer_Name`, Data Integration automatically links the `Cust_Name` field with the `Customer_Name` field.

You can use both Exact Field Name and Smart Map in the same field mapping. For example, use Exact Field Name to match fields with the same name and then use Smart Map to map fields with similar names.

You can undo all automapped field mappings by clicking **Automap** > **Undo Automap**. To unmap a single field, select the field to unmap and click **Actions** > **Unmap**.

Data Integration highlights newly mapped fields. For example, when you use Exact Field Name, Data Integration highlights the mapped fields. If you then use Smart Map, Data Integration only highlights the fields mapped using Smart Map.

Advanced properties

You can configure advanced properties for a Union transformation. The advanced properties control settings such as the tracing level for session log messages and whether the transformation is optional or required.

The properties that are available vary based on the mapping mode.

You can configure the following properties:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.
Optional	Determines whether the transformation is optional. If a transformation is optional and there are no incoming fields, the mapping task can run and the data can go through another branch in the data flow. If a transformation is required and there are no incoming fields, the task fails.
	For example, you configure a parameter for the source connection. In one branch of the data flow, you add a transformation with a field rule so that only Date/Time data enters the transformation, and you specify that the transformation is optional. When you configure the mapping task, you select a source that does not have Date/Time data. The mapping task ignores the branch with the optional transformation, and the data flow continues through another branch of the mapping.

Union Transformation example

You have demographic data about employees from two flat file sources and you want to merge that data.

You receive the following data in a .txt file:

```
employee_ID,first_name,last_name,location,email,phone
1211,John,Davis,Redwood City,jdavis@infa2.com,555-555-4444
0233,Miles,Simone,Barcelona,msimone@infa2.com,555-555-6666
1045,Billie,Coltrane,Philadelphia,bcoltrane@infa2.com,555-555-7777
0987,Django,Holiday,Paris,dholiday@infa3.com,444-444-4444
1199,Nina,Reinhardt,New York,nreinhardt@infa3.com,444-555-5555
```

A second file contains the following data:

```
ID,first,last,dept,e-mail,phone
0456,Joni,Smith,Marketing,j_smith@infa4.com,333-333-3333
1325,David,Mitchell,R&D,dmitchell@infa4.com,222-222-2222
1101,David,Harry,R&D,dharry@infa5.com,777-777-7777
0623,Debbie,Byrne,HR,dbyrne@infa5.com,888-888-8888
0777,Patti,Bowie,Sales,pbowie@infa5.com,999-999-9999
```

You want to merge those records into a single dataset in MySQL with the following columns:

- id
- last
- first
- email
- phone

Note: Remember that the data to be merged with a Union transformation must have the same data type, precision, and scale.

To merge the files with a Union transformation, complete the following steps:

1. Ensure that the source files reside in a location accessible to your Secure Agent.

2. Define a connection to access the .csv files.

3. Create a mapping in the Mapping Designer.

4. Add two Source transformations to the mapping to connect to data in the .csv files.

5. Add a Union transformation and connect the Source transformations to it.

6. In the Union transformation Properties, perform the following steps for each input group:

   a. In the Field Rules section, click the group you want to configure.

   b. (Optional) For the incoming fields, select the fields you want to merge in the output.

   The following image shows the selected fields in the first input group:
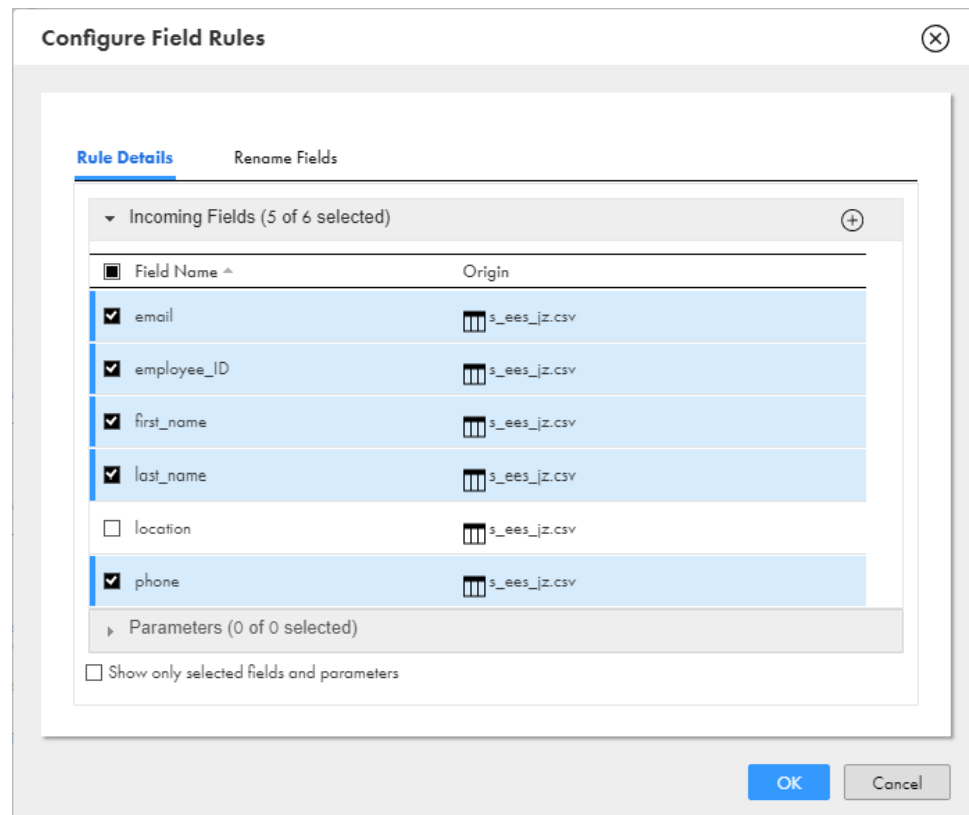
The following image shows the selected fields in the second input group:
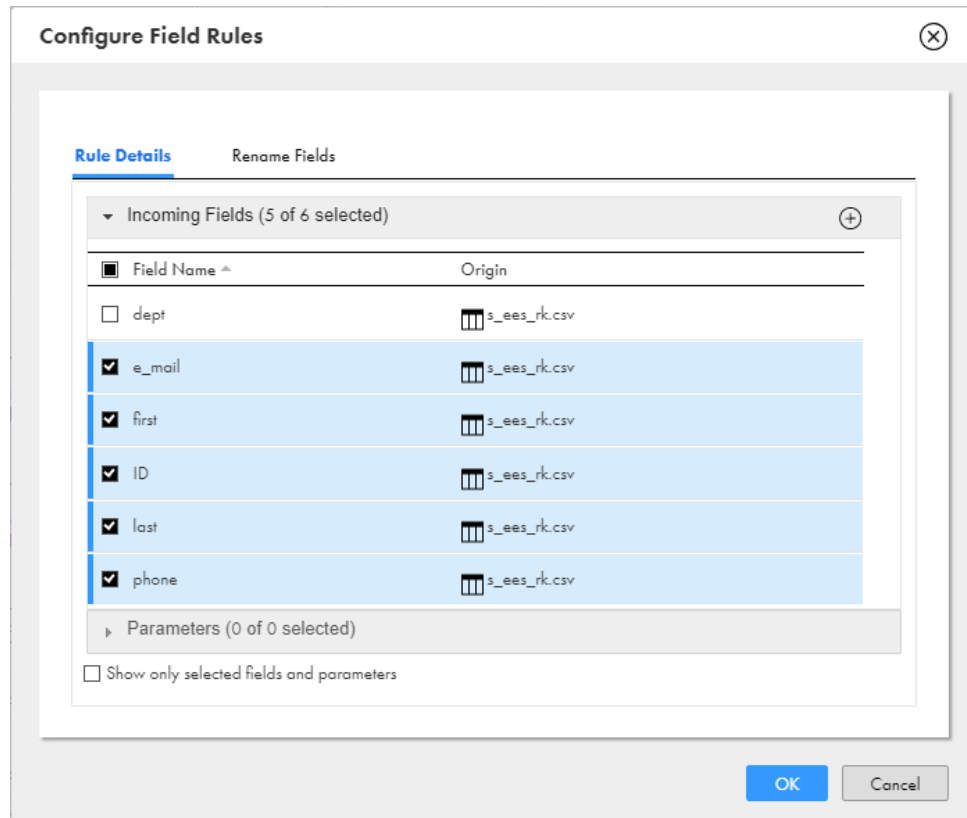
If you do not specify a rule to exclude fields, at run time, the task ignores any fields that you do not map to the output fields.

c. Edit the Output field names in the Union transformation, to correspond to the field names that you want in the target:
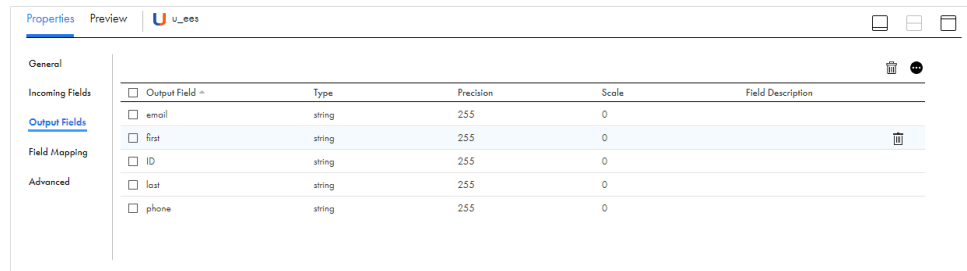
Note: You can also select fields, change metadata, add other fields, or convert the field types, for example, from integer to number.

7. In the Field Mapping of the Union transformation, ensure that the fields are correctly mapped for each input group:
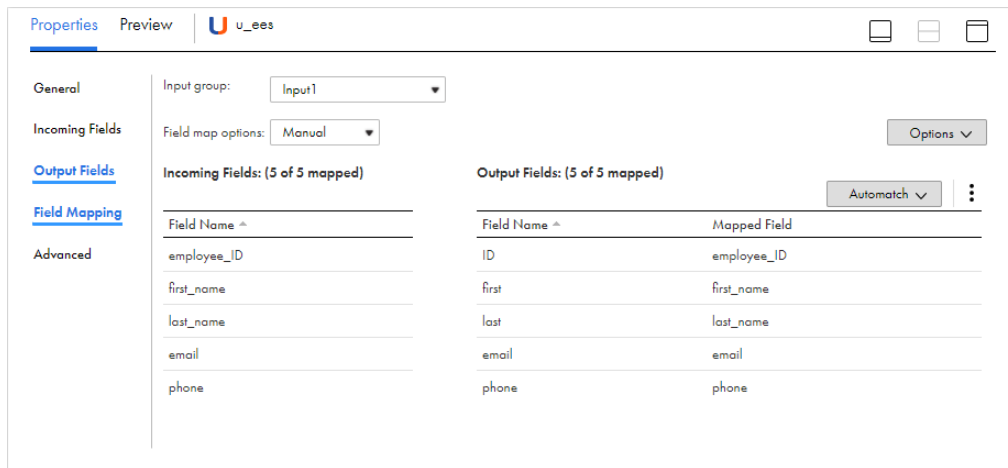
8. Add a Target transformation to the mapping.

9. Connect the Union transformation to the Target transformation.

10. In the Target transformation field mapping, select automatic field mapping:
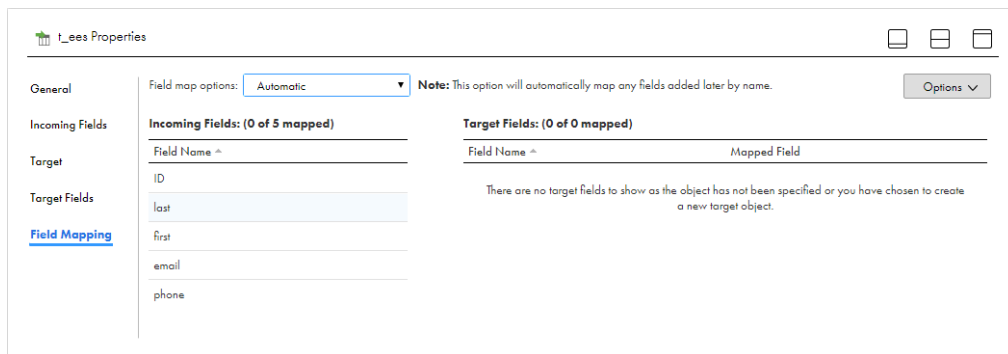
When complete, the mapping appears similar to the following image:

CHAPTER 38

Vector Embedding transformation

In advanced mode, you can use a Vector Embedding transformation to generate vector embeddings for input text, capturing the semantic meaning of the text in a vector format.

Before using a Vector Embedding transformation, use a Chunking transformation to split the text into chunks. Then, the Vector Embedding transformation can generate vector embeddings for each chunk of text using an embedding model like Word2Vec or BERT. For more information about the Chunking transformation, see Chapter 7, "Chunking transformation" on page 112.

To create an identifer for each vector, you can use either the UUID_STRING function in an Expression transformation or a Sequence Generator transformation:

- If you use the UUID_STRING function in an Expression transformation, use the function without passing any arguments. The function returns a globally unique ID that can be stored in a string field with a precision of 100.

    Note: UUID_STRING is an internal function that you can use only in advanced mode. Using it to create identifiers for other use cases might produce unexpected results.

- If you use a Sequence Generator transformation, create a shared sequence to use across all mappings that load data to the same index in the vector database.

A Target transformation can write the vectors to a vector database.

Note: The Vector Embedding transformation can't run in a serverless runtime environment, on an advanced cluster on Google Cloud, or on GPUs. If the transformation runs on a GPU-enabled cluster, GPUs are disabled and the transformation consumes CPUs.

Vector embedding techniques

Use a vector embedding technique to create a vector embedding for input text. You can choose a technique based on the pre-trained model you want to use to convert the text to a vector.

A vector embedding represents the text as an array of numbers. Each element in the array represents a different dimension of the text. To create vector embeddings, select an input column for embedding and then select one of the following vector embedding techniques:

Word embedding

Convert each word to a vector using the Word2Vec Gigaword 5th Edition model with 300 dimensions (word2vec_giga_300). Useful for text classification and sentiment analysis.

BERT embedding

Convert each sentence to a vector using the Smaller BERT Embedding (L-2_H-768_A-12) model with 768 dimensions (small_bert_L2_768). Useful for text classification and semantic search.

Consider the following rules and guidelines for vector embeddings:

- Vector embeddings created by different embedding models can't be compared even if they have the same dimensions. If you switch between embedding models, rerun the mapping, including all Source, Chunking, Vector Embedding, and Target transformations, to create embeddings for all documents using the new model.

- Because the Vector Embedding transformation is a passive transformation that produces one output row for each input row, input columns that contain null or empty strings return an empty output vector. If the vector is empty, vector databases like Pinecone might drop the row.

- The Vector Embedding transformation can process only English text.

Vector embedding output fields

The Vector Embedding transformation creates a Vector output field that stores the vector embedding.

The Vector output field is an array of doubles with a precision that is equal to the number of dimensions in the vector. For example, a vector embedding technique that uses 300 dimensions will output an array of 300 elements. You can use a Target transformation to write the vector to a vector database.

CHAPTER 39

Velocity transformation

Use the Velocity transformation in a mapping to convert hierarchal input from one format to another without flattening the data. The transformation can convert JSON or XML data to JSON, XML, or text output such as plain text, email, or HTML.

You might want to use the Velocity transformation to convert hierarchical data to comply with a downstream API call or for use in a downstream application such as a campaign management system or machine learning algorithm. You can also use the Velocity transformation to process row data such as a JSON BLOB in a database.

To convert the data, the Velocity transformation uses an Apache Velocity script that you provide. The script can contain Velocity Template Language (VTL) statements, Data Integration built-in functions, and Data Integration unconnected lookup expressions.

You can pass data to the Velocity transformation in one of the following ways:

Pass the data directly to the transformation.

You can pass data directly to the Velocity transformation through the input field. The data must be a JSON BLOB or string or an XML BLOB or string.

Pass the name and location of the file that contains the data you want to convert.

If you want the Velocity transformation to convert data stored in a flat file such as an XML or JSON file, pass the file name and location to the transformation through the input field. The data must be a string that contains the file path and file name. The file must be a delimited flat file that is accessible from the Secure Agent machine.

The transformation returns the converted data in one string field.

Velocity transformation input format

You configure the format of the incoming data on the **Input Format** tab. Configure input format properties such as the input field and type, the format of the data that you want to convert, and the variable name that

refers to the data in the template. If the data to be converted is binary data or is in a file, you also select the code page.

The following table describes the file input format properties:

Property	Description
Input Field	Field in the upstream transformation that contains the input for the Velocity transformation. The field that you select must contain either the data to be converted or the file path and name of the flat file that contains the data. Since the input to the transformation must be character or binary data, this field displays only string, text, and binary fields from the upstream transformation.
Input Type	Type of data in the selected input field. Select one of the following input types: - Buffer. Select this option when the input field contains the data that you want to convert such as a JSON BLOB or XML string. - File. Select this option when the input field contains the path to a flat file that contains the data to be converted.
Format Type	Format type of the data that you want to process, either JSON or XML.

Variable Name in Template	The variable that you define in the template to refer to the data to be processed. Do not include a leading dollar sign or other special character.
	
	For example, in the following template, the variable name "root" is used to refer to the data to be processed:
	
	`<xml>`
	`#foreach ($child in $root.getRootElement().getChildren() )`
	`    <$child.getChild("name").getText()>`
	`        <id>$child.getChild("id").getText()</id>`
	`        <size>$child.getChild("size").getText()</size>`
	`    </$child.getChild("name").getText()>`
	`#end`
	`</xml>`

The following table describes the file input format advanced property:

Property	Description
Code Page	Code page of the data that you want to convert. This field is enabled if the input type is File or if the input field contains binary data. Select a code page if the code page of the data that you want to convert differs from the code page of the Secure Agent machine. Otherwise, select **Default**.

Source configuration for file sources

If you configure the Velocity transformation to read data from a file, configure the Source transformation to read data from a single, delimited flat file and configure the formatting options. The flat file must contain the file path on one line.

For example, the following text shows the contents of a source file that reads data from an XML file named Products.xml:

```
C:\IICS_XML_SourceFiles\Products.xml
```

To configure the Source transformation formatting options, complete the following steps:

1. On the **Source** tab, select the file that contains the file path as the source object, and then click **Formatting Options**.

2. In the **Formatting Options** dialog box, ensure that the flat file type is **Delimited**, and configure the following properties:

Property	Value
Text Qualifier	None
Field Labels	Auto-generate
First Data Row	1

3. Click **OK**.

Velocity template

You create the Velocity template on the **Velocity** tab. To create the template, enter or paste the template in the template editor. Then click **Validate** to validate the syntax.

The following image shows the template editor:
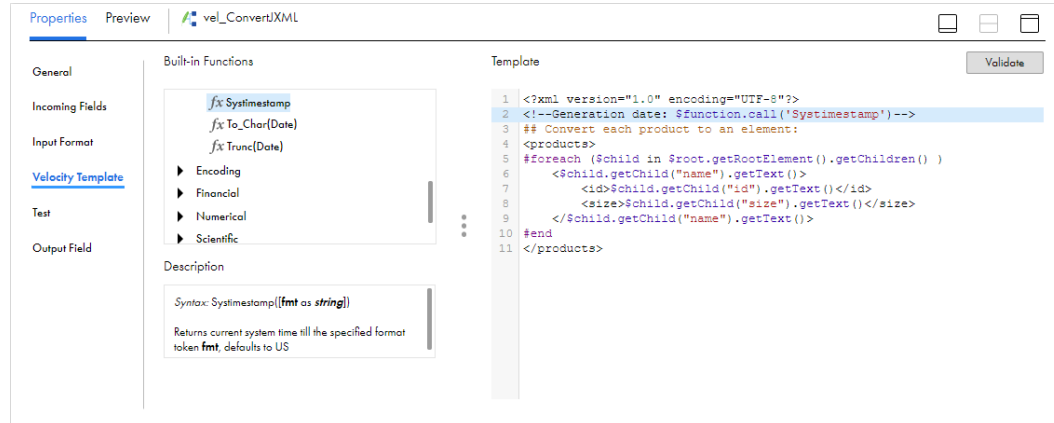
The template can contain VTL statements, Data Integration built-in functions, and Data Integration unconnected lookup expressions. For more information about the Velocity Template Language, see the Apache Velocity documentation.

To call a built-in function, use the following syntax:

```
$function.call('<function name>',<argument>,<argument>,...)
```

For example, the following template code returns the system time stamp in an XML comment:

```
<!--Generation date: $function.call('Systimestamp')-->
```

The following function call passes the property $vendor.name as an argument to the INITCAP function to capitalize the first letter of each word in the vendor name:

```
$function.call('InitCap',$vendor.name)
```

Tip: You can select a function in the **Built-in Functions** list to copy the function call syntax into the template at the cursor position.

To call an unconnected lookup, use the following syntax:

```
$function.call(':LKP.<Lookup transformation name>)',<argument>,<argument>,...)
```

For example, the following expression passes the properties $item.id and $item.category as arguments to an unconnected lookup Transformation named lkp_ItemPrices:

```
$function.call(':LKP.lkp_ItemPrices',$item.id,$item.category)
```

To validate the syntax, click **Validate**. Data Integration checks the template and displays any syntax errors. If the template syntax is invalid, you can save the mapping but the mapping is invalid.

Validation does not test the template output or find runtime errors. You can test the template output on the **Test** tab.

Testing the template

You can test the template on the **Test** tab. The **Test** tab contains fields for sample input and output so you can test the template as you create it.

The following image shows the **Test** tab:
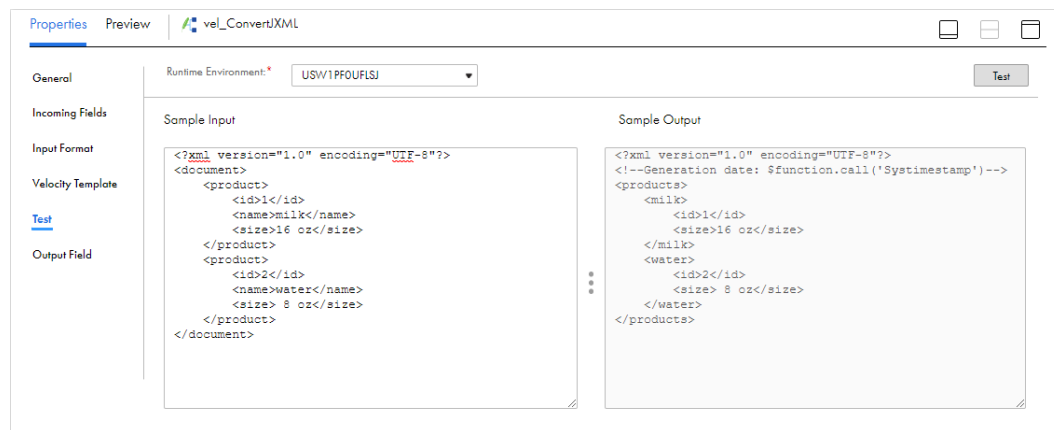
To test the template you created, select a runtime environment, enter or paste a portion of the incoming data into the **Sample Input** field, and then click **Test**. You can select any runtime environment in which the Data Integration Server is enabled except the Hosted Agent.

Sample input can contain up to 1000000 characters. For best results, ensure that the sample input is a representative sample of the data that you want to convert.

Sample output appears in the **Sample Output** field. When you test a template, Data Integration does not process built-in function or lookup expression calls. In the sample output, these expressions appear exactly as they were entered it in the template. If the template does not generate valid output, an error message appears in the **Sample Output** field.

Velocity transformation output

The Velocity transformation returns converted data in a string field named VELOCITYOUT. You cannot change the output field name or type, but you can change the precision. Change the output field precision on the **Output Field** tab.

The precision is the maximum number of characters that the returned character string can contain. Data Integration truncates the returned character string at the precision value. By default, the output field precision is 1000000 characters. To change the output field precision, enter a different value in the **Precision** field.

Target configuration for file targets

You can write the data that is converted by the Velocity transformation to a target file. If the target file is an XML or JSON file that you create at run time, you might want to specify no header and no text qualifier in the output file.

To specify no header in the output file, open the **Target** tab of the Target transformation. In the Advanced properties, set the header options to **No Header**.

To specify no text qualifier, on the **Target** tab, click **Formatting Options** next to the **Object** field. In the **Formatting Options** dialog box, set the text qualifier to **None**.

Velocity transformation parsers

The Velocity transformation uses different parsers to parse JSON and XML data.

The Velocity transformation uses the following parsers based on the type of data you want to process:

JSON data

To parse JSON data, the Velocity transformation uses the Java package org.json. The transformation passes the data that you want to process to the `JSONObject(java.lang.String source)` constructor to construct a JSONObject object within Java. For more information about the JSONObject constructor, see javadoc.io.

XML data

To parse XML data, the Velocity transformation uses the Java package org.jdom2.input. The transformation uses the SAX parser that this package provides and creates a SAXBuilder object in Java. For more information about the SAX parser, see the JDOM v2.0.6 API Specification.

Examples

The following examples show how to use the Velocity transformation to convert XML data from one format to another and to convert and augment JSON data.

XML conversion example

You have product data in an XML file that you want to convert to XML of a different format.

The XML file that you want to convert, products.xml, contains data in the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <product>
        <id>1</id>
        <name>milk</name>
        <size>16 oz</size>
    </product>
    <product>
        <id>2</id>
        <name>water</name>
        <size> 8 oz</size>
    </product>
</document>
```

You want to convert the XML so that each product is in its own element, for example, `<milk>...</milk>`. You also want to append a timestamp in a comment inside the XML file.

To convert the file, first create a source file that contains the path to the XML file that you want to convert so that the mapping can read data from the XML file. Then, create the mapping.

Create a text file called "filepath.txt" that contains the following line:

`C:\XMLSources\products.xml`

After you create the source file, create the mapping. The following image shows the mapping:
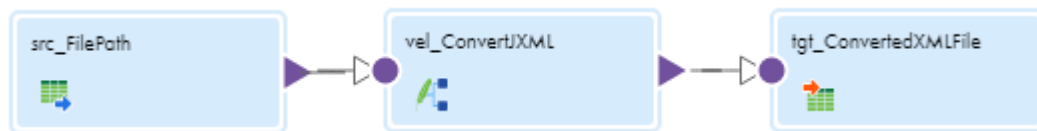
Configure the transformations in the following ways:

Source transformation

Configure the Source transformation to read data from the file that contains the file path, and configure the source formatting options.

On the **Source** tab, select a connection that can access the source file, set the source type to **Single Object**, and select filepath.txt as the source object.

Click **Formatting Options** and then configure the following properties:

Property	Value
Flat File Type	Delimited
Text Qualifier	None
Field Labels	Auto-generate
First Data Row	1

Velocity transformation

Configure the Velocity transformation input and create the template.

On the **Input Format** tab, configure the following properties:

Property	Value
Input Field	Incoming string field from the Source transformation.
Input Type	File
Format Type	XML
Variable Name in Template	root
Code Page	UTF-8

On the **Velocity Template** tab, enter the following template in the template editor and validate the syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Generation date: $function.call('Systimestamp')-->
## Convert each product to an element:
<products>
#foreach ($child in $root.getRootElement().getChildren() )
    <$child.getChild("name").getText()>
        <id>$child.getChild("id").getText()</id>
        <size>$child.getChild("size").getText()</size>
    </$child.getChild("name").getText()>
#end
</products>
```

Target transformation

Configure the Target transformation to write data to a flat file target created at run time. To ensure that the target file contains no header and no text qualifier character, configure the formatting options.

On the **Target** tab, select the connection and set the target type to **Single Object**. Click **Select** next to the **Object** field. Select **Create New at Runtime**, and enter the file name "products_converted.xml."

Click **Formatting Options** and set the text qualifier to **None**.

In the Advanced properties, set the header options to **No Header**.

When you run the mapping, the target file, products_converted.xml, contains the following XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Generation date: 06/17/2020 17:45:41.341526-->
<products>
    <milk>
        <id>1</id>
        <size>16 oz</size>
    </milk>
    <water>
        <id>2</id>
        <size> 8 oz</size>
    </water>
</products>
```

JSON conversion example

You have vendor data from an online review service that is stored in a JSON BLOB in a database. You want to filter the records, convert the data to a different format, and write it to a JSON file. You also want to augment the data with additional attributes.

The database contains a JSON BLOB that contains data in the following format:

```
{"items":[
{"business_id":"1SWheh84yJXfytovILXOAQ","name":"Rancho Golf Club","address":"1200 E
Camino Acequia
Drive","postal_code":"85016","stars":3.0,"review_count":5,"is_open":0,"attributes":
{"GoodForKids":"False"},"categories":"Golf, Active Life","hours":null},
{"business_id":"QXAEGFB4oINsVuTFxEYKFQ","name":"Garden Blessing Chinese
Restaurant","address":"25 Eglinton Avenue W","postal_code":"L5R
3E7","stars":2.5,"review_count":128,"is_open":1,"attributes":
{"RestaurantsReservations":"True","GoodForMeal":"{'dessert': False, 'latenight': False,
'lunch': True, 'dinner': True, 'brunch': False, 'breakfast':
False}","BusinessParking":"{'garage': False, 'street': False, 'validated': False, 'lot':
True, 'valet':
False}","Caters":"True","NoiseLevel":"u'loud'","RestaurantsTableService":"True","Restaura
ntsTakeOut":"True","RestaurantsPriceRange2":"2","OutdoorSeating":"False","BikeParking":"F
alse","Ambience":"{'romantic': False, 'intimate': False, 'classy': False, 'hipster':
False, 'divey': False, 'touristy': False, 'trendy': False, 'upscale': False, 'casual':
True}","HasTV":"False","WiFi":"u'no'","GoodForKids":"True","Alcohol":"u'full_bar'","Resta
urantsAttire":"u'casual'","RestaurantsGoodForGroups":"True","RestaurantsDelivery":"False"
},"categories":"Specialty Food, Restaurants, Dim Sum, Imported Food, Food, Chinese,
Ethnic Food, Seafood","hours":
{"Monday":"9:0-0:0","Tuesday":"9:0-0:0","Wednesday":"9:0-0:0","Thursday":"9:0-0:0","Frida
y":"9:0-1:0","Saturday":"9:0-0:0","Sunday":"9:0-0:0"}},
{"business_id":"gnKjwL_1w79qoiV3IC_xQQ","name":"Fujiyama Japanese
Cuisine","address":"111 Johnston Rd, Ste
15","postal_code":"28210","stars":4.0,"review_count":170,"is_open":1,"attributes":
{"GoodForKids":"True","NoiseLevel":"u'average'","RestaurantsDelivery":"False","GoodForMea
l":"{'dessert': False, 'latenight': False, 'lunch': True, 'dinner': True, 'brunch':
False, 'breakfast':
False}","Alcohol":"u'beer_and_wine'","Caters":"False","WiFi":"u'no'","RestaurantsTakeOut"
:"True","BusinessAcceptsCreditCards":"True","Ambience":"{'romantic': False, 'intimate':
False, 'touristy': False, 'hipster': False, 'divey': False, 'classy': False, 'trendy':
False, 'upscale': False, 'casual': True}","BusinessParking":"{'garage': False, 'street':
False, 'validated': False, 'lot': True, 'valet':
False}","RestaurantsTableService":"True","RestaurantsGoodForGroups":"True","OutdoorSeatin
g":"False","HasTV":"True","BikeParking":"True","RestaurantsReservations":"True","Restaura
ntsPriceRange2":"2","RestaurantsAttire":"'casual'"},"categories":"Sushi Bars,
Restaurants, Japanese","hours":
{"Monday":"17:30-21:30","Wednesday":"17:30-21:30","Thursday":"17:30-21:30","Friday":"17:3
0-22:0","Saturday":"17:30-22:0","Sunday":"17:30-21:0"}}
...
```

You want to filter the records so that they only include restaurants with three or more stars. You also want to add the city to each record based on the postal code.

The postal codes and corresponding cities exist in a flat file with the following format:

```
postal_code|city
15090|Wexford, PA
15102|Bethel Park, PA
15206|Pittsburgh, PA
15317|Canonsburg, PA
28012|Belmont, NC
28027|Concord, NC
...
```

You want the target file to contain JSON data in the following format:

```
{ "Date": "<date>",
"vendors":[

    {
    "name": "<restaurant name>",
    "location": "<city, state/province>",
```

```
    "desc": "<description>",
    "stars": <number of stars>
    }
,
...
```

To convert the data, create a mapping with a Velocity transformation that converts the JSON data and an unconnected Lookup transformation that returns the city based on the postal code.

The following image shows the mapping:
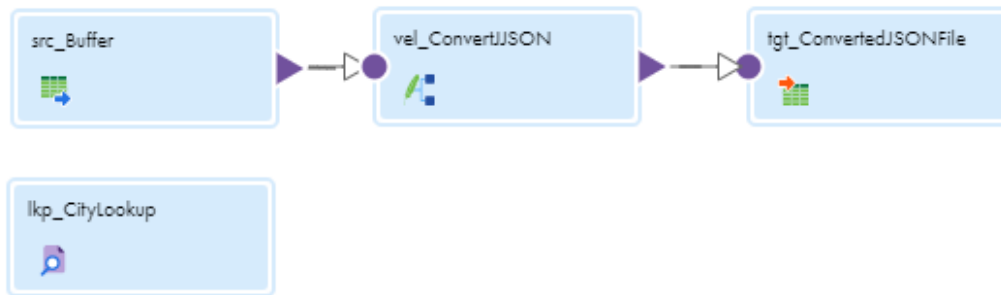
Configure the transformations in the following ways:

Source transformation

Configure the Source transformation to read data from the database table that contains the JSON BLOB.

Velocity transformation

Configure the Velocity transformation input and create the template.

On the **Input Format** tab, configure the following properties:

Property	Value
Input Field	Incoming string field from the Source transformation that contains the JSON BLOB.
Input Type	Buffer
Format Type	JSON
Variable Name in Template	inputRoot
Code Page	Default

On the **Velocity Template** tab, enter the following template in the template editor and validate the syntax:

```
#set($comma ="")
{ "Date": "$function.call('Systimestamp')",
"vendors":[
#foreach($vend in $inputRoot.items)
    #if($vend.categories.toString().contains("Restaurants") && ($vend.stars > 3))
$comma
    {
    "name": "$vend.name",
    "location": "$function.call(':lkp.lkp_CityLookup', $vend.postal_code)",
    "desc": "$vend.categories",
    "stars": $vend.stars
    }
    #set($comma =",")
    #end
```

```
            #end
            ]
            }
```

Unconnected Lookup transformation

Configure the Lookup transformation to return the city based on the postal code. The Lookup transformation must be an unconnected Lookup transformation so that you can call it from the template in the Velocity transformation.

On the **General** tab, select **Unconnected Lookup**.

On the **Incoming Fields** tab, create an incoming field for the postal code called `in_postal_code`.

On the **Lookup Object** tab, select the text file that contains the postal codes and cities. Then, click **Formatting Options** and configure the following properties:

Property	Value
Flat File Type	Delimited
Delimiter	Other: |
Text Qualifier	None
Field Labels	Import from Row 1
First Data Row	2

On the **Lookup Condition** tab, configure the following lookup condition:

`postal_code = in_postal_code`

On the **Return Fields** tab, select `city` as the return field.

Target transformation

Configure the Target transformation to write data to a flat file target created at run time. To ensure that the target file contains no header and no text qualifier character, configure the formatting options.

On the **Target** tab, select the connection and set the target type to **Single Object**. Click **Select** next to the **Object** field. Select **Create New at Runtime**, and enter the file name "vendors.json."

Click **Formatting Options** and set the text qualifier to **None**.

In the Advanced properties, set the header options to **No Header**.

When you run the mapping, the target file, vendors.json, contains the following data:

```
{ "Date": "07/08/2020 12:33:44.647037",
"vendors":[

    {
    "name": "Fujiyama Japanese Cuisine",
    "location": "Charlotte, NC",
    "desc": "Sushi Bars, Restaurants, Japanese",
    "stars": 4.0
    }
,
    {
    "name": "D'Amico's Pizzeria",
    "location": "Mentor-on-the-Lake, OH",
    "desc": "Italian, Restaurants, Pizza, Chicken Wings",
    "stars": 4.0
    }
```

```
,
    {
    "name": "Villa Borghese",
    "location": "Las Vegas, NV",
    "desc": "Restaurants, Italian",
    "stars": 4.0
    }
,
    {
    "name": "3-Rivers Diner",
    "location": "Pittsburgh, PA",
    "desc": "Sandwiches, Salad, Restaurants, Burgers, Comfort Food",
    "stars": 4.0
    }
,
...
```

CHAPTER 40

Verifier transformation

The Verifier transformation adds a verifier asset that you created in Data Quality to a mapping.

A verifier asset defines a template for input and output address data that you can connect to the input and output fields on the Verifier transformation. Connect the fields in your source data or in upstream transformations to the corresponding input ports on the Verifier transformation. Connect the output ports on the Verifier transformation to downstream transformations in the mapping or to the mapping target.

The Verifier transformation performs the following operations on the input address data:

- The transformation compares the address records in the source data to address reference data.
- It fixes errors and completes partial address records. To fix an address, the transformation must find a positive match with an address in the reference data. The transformation copies the required data elements from the address reference data to the address records.
- It writes output addresses in the format that the verifier asset specifies. You define a verifier asset in Data Quality to create address records that suit your business needs and that conform to the structure that the mail carrier requires.
- It can report on the deliverable status of each address and the nature of any error or ambiguity that the address contains.
- It can provide suggestions for any ambiguous or incomplete address.

For more information about the types of address information that the Verifier transformation can read and write, including address status information, consult the verifier asset documentation in the *Data Quality* online help.

A Verifier transformation is similar to a Mapplet transformation, as it allows you to add address verification logic that you created elsewhere to a mapping. Like mapplets, verifiers are reusable assets. A Verifier transformation shows incoming and outgoing fields. It does not display the address data that the verifier contains or allow to you edit the verifier. To edit the verifier, open it in Data Quality.

Address Reference Data

To verify the quality of the addresses in your source data, the Verifier transformation compares the addresses to the data in one or more reference data files.

When you run a mapping with a Verifier transformation, the Secure Agent evaluates the input data and downloads the reference data files that you need. Each reference data file is specific to a single country. The Secure Agent downloads one or more files for each county that the input address data specifies.

You do not need to take any action to download the files. If the current reference data files already exist on the system, the Secure Agent does not download them again.

Each reference data file requires a license. You buy the license from Informatica. You enter the license key information as a system configuration property on the Secure Agent that runs the mapping. Find the Secure Agent properties in the Administrator service.

For more information on reference data properties, consult the *Verifier Guide* in the Data Quality online help.

Verifier transformation configuration

When you configure a Verifier transformation in a mapping, you first select the verifier asset that contains the address data to include in the mapping. Next, you configure the incoming fields and the field mappings. Then, you verify the output fields.

To configure the transformation, complete the following tasks:

1. Connect the Verifier transformation to a Source transformation or other upstream object.

2. On the **Verifier** tab, select the verifier that you want to include in the transformation.

   The following image shows the options that you use to select the verifier:
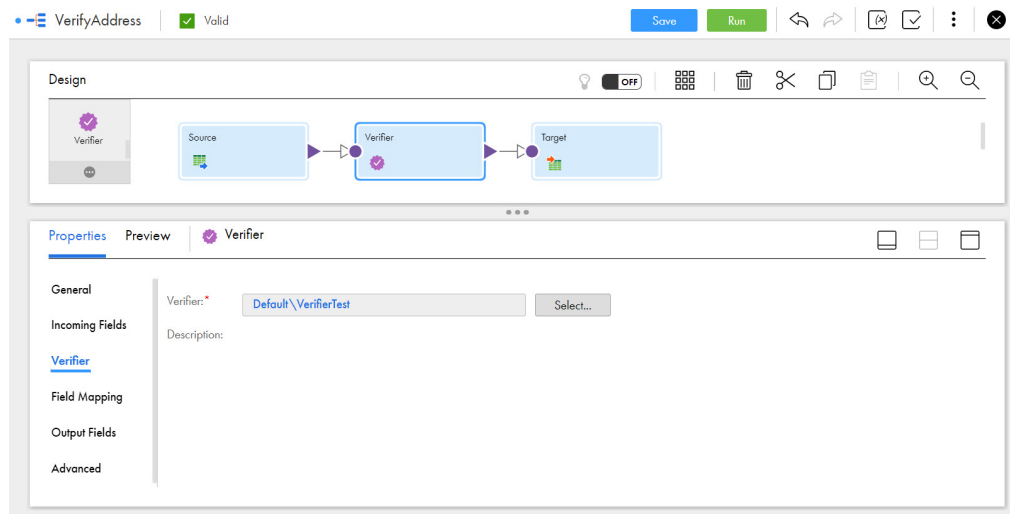
3. On the **Incoming Fields** tab, configure the incoming fields.

   By default, the transformation inherits all incoming fields from any connected upstream object in the mapping. You can define a field rule to limit or rename the incoming fields.

4. On the **Field Mapping** tab, configure the options to connect the data from incoming fields to the target fields on the verifier asset.

   The verifier inputs might already reflect the names of the input fields in the data. If so, you can use the **Automap** options to connect the fields.

   For more information about connecting to upstream object fields, see "Verifier transformation field mappings" on page 453.

5. Verify the verifier output fields on the **Output Fields** tab.

6. You can optionally rename the Verifier transformation and add a description on the **General** tab. You can also update the tracing level for the transformation on the **Advanced** tab. The default tracing level is Normal.

Note: If you update an asset in Data Quality after you add it to a transformation, you may need to synchronize the asset version in the transformation with the latest version. For more information about data quality asset synchronization, see "Synchronizing data quality assets" on page 116.

Verifier transformation field mappings

Configure field mappings to define how data moves from an upstream transformation to the Verifier transformation. Configure field mappings on the Field Mapping tab of the **Properties** panel.

You can configure the following field mapping options:

Field map options

Method of mapping fields to the transformation.

Select one of the following options:

- Manual. Manually link incoming fields to transformation input fields. Removes links for automatically mapped fields.
- Automatic. Automatically link fields with the same name. Use when all of the fields that you want to link share the same name. You cannot manually link fields with this option.
- Completely Parameterized. Use a parameter to represent the field mapping. In the task, you can configure all field mappings.
  Choose the Completely Parameterized option when the verifier in the transformation is parameterized or any upstream transformation in the mapping is parameterized.
- Partially Parameterized. Configure links in the mapping that you want to enforce and use a parameter to allow other fields to be mapped in the mapping task. Or, use a parameter to configure links in the mapping, and allow all fields and links to display in the task for configuration.

Parameter

Select the parameter to use for the field mapping, or create a new parameter. This option appears when you select Completely Parameterized or Partially Parameterized as the field map option. The parameter must be of type *field mapping*.

Do not use the same field mapping parameter in more than one Verifier transformation in a single mapping.

Options

Controls how fields are displayed in the **Incoming Fields** and **Target Fields** lists.

Configure the following options:

- The fields that appear. You can show all fields, unmapped fields, or mapped fields.
- Field names. You can use technical field names or labels.

Automap

Links fields with matching names. Allows you to link matching fields and to manually configure other field mappings. The Automap options appear when you select the Manual or Partially Parameterized field map option.

You can map fields in the following ways:

- Exact Field Name. Data Integration matches fields of the same name.

- Smart Map. Data Integration matches fields with similar names. For example, if you have an incoming field `Post Code` and a target field `Postal Code`, Data Integration automatically links the `Post Code` field with the `Postal Code` field.

You can use both Exact Field Name and Smart Map in the same field mapping. For example, use Exact Field Name to match fields with the same name and then use Smart Map to map fields with similar names.

You can undo all automapped field mappings by clicking **Automap** > **Undo Automap**.

To unmap a single field, select the field to unmap and click **Actions** > **Unmap** on the context menu for the field. To unmap one or more fields that you selected, click **Unmap Selected** on the Target Fields context menu.

To clear all field mappings from the transformation, click **Clear Mapping** on the Target Fields context menu.

Understanding input and output mappings

The verifier asset that you add to the transformation is an address template with a preconfigured set of input and output fields. Each input and output field on the asset represents a different type of address information.

The data on the transformation input fields that you map to the verifier asset must reflect the types of information that the asset inputs expect. If the fields do not correspond, the mapping cannot evaluate the input data with full accuracy.

The asset inputs may expect a single element of address data, or they may expect multiple elements organized in a single field. Likewise, the asset outputs may write a single element of address data, or they may write multiple elements to a single field. If necessary, work with the asset designer to determine the meaning of the inputs.

Verifier transformation output fields

A Verifier transformation displays an output field for each output on the verifier asset. View the output fields on the **Output Fields** tab of the **Properties** panel.

The tab displays the name, type, precision, and scale for each output field. The output field names are the names of the output fields on the asset.

Advanced properties

You can configure advanced properties for a Verifier transformation. The advanced properties control the tracing level for session log messages.

You can configure the following property:

Property	Description
Tracing Level	Detail level of error and status messages that Data Integration writes in the session log. You can choose terse, normal, verbose initialization, or verbose data. Default is normal.

CHAPTER 41

Web Services transformation

Use the Web Services transformation in the Mapping Designer to make a web service request and to map the web service response to a target.

A web service integrates applications and uses open standards, such as SOAP, WSDL, and XML. SOAP is the communications protocol for web services. Web Services Description Language (WSDL) is an XML schema that describes the protocols, formats, and signatures of the web service operations. Web service operations include requests for information, requests to update data, and requests to perform tasks.

A Web Services transformation connects to a web service as a web service client to access, transform, or deliver data. The web service client request and the web service response are SOAP messages. The mapping task processes SOAP messages with document/literal encoding. The Web Service transformation does not support RPC/encoded or document/encoded WSDL files.

For example, the Web Services transformation sends a SOAP request to a web service to run a web service operation called getCityWeatherByZIP. The Web Services transformation passes zip codes in the request. The web service retrieves weather information and then returns the information in a SOAP response.

SOAP request messages and response messages can contain hierarchical data, such as data that follows an XML schema. For example, a web service client sends a request to add customer orders to a sales database. The web service returns the following hierarchy in the response:

```
Response
    Order
        Order_ID
        Order_Date
        Customer_ID
        Product
            Product_ID
            Qty
            Status
```

The response has information on orders, including information on each product in the order. The response is hierarchical because within the Order element, the Product element contains more elements.

To use the Web Services transformation, you need the appropriate license.

To use the Web Services transformation, perform the following steps:

1. Create a Web Services Consumer connection and use a WSDL URL and an endpoint URL.
2. Define a business service. A business service is a web service with configured operations.
3. Configure the Web Services transformation in a mapping in the Mapping Designer.

Create a Web Services consumer connection

Connect to a web service using Web Services Description Language (WSDL) and an endpoint URL. You can also enable security for the web service.

1. In Administrator, select **Connections**, and click **New Connection**.
2. Enter the connection details. For the **Type**, select **WSConsumer**.

   If the WSConsumer connection is not in the list of connections, in Administrator, open the **Add-On Connectors** page to install the connector for your organization.

   Note: Ensure that you select WSConsumer, not Web Service. Web Service connections are used for mapplets or PowerCenter tasks.

3. Enter the connection properties:
   a. Select the runtime environment for the web service connection.
   b. For **Authentication**, select one of the following options:
      - Select **Username Token** to enter the user name and the password to authenticate the web service. You can choose to encrypt the password by sending a digest that is derived from the password in the security token.
      - Select **Other Authentication** to enter the WSDL URL and endpoint URL to access the web service. If the web service is a private web service, enter the HTTP user name and password.

4. Enter the authentication connection properties. The properties that appear depend on the authentication that you selected.

   The following table describes the properties to configure:

Property	Description
WSDL URL	The URL provided by the web service. If you are using a WSDL URL that has HTTP Basic authentication, download the WSDL to the machine where the Secure Agent runs. For the WSDL URL, enter the file path. For example, you can enter the following directory path: `C:\WSDL\Human_Resources.wsdl`
Endpoint URL	Endpoint URL for the web service. The WSDL file specifies this URL in the location element.
Username	Applicable to username token authentication. User name to authenticate the web service.
Password	Applicable to username token authentication. Password to authenticate the web service.
Encrypt Password	Applicable to username token authentication. Enables the PasswordDigest property which combines the password with a nonce and a time stamp. The mapping task applies a SHA hash on the password, encodes it in base64 encoding, and uses the encoded password in the SOAP header. If you do not select this option, the PasswordText property is enabled and the mapping task does not change the password in the WS-Security SOAP header.
HTTP Username	User name used to access the web service.
HTTP Password	Password to access the web service.

5. Save the connection.

Define a business service

A business service is a web service with configured operations. Define a business service to add operations to the Web Services transformation in the Mapping Designer.

1. Click **New** > **Components** > **Business Services** and then click **Create**.

2. Enter a name for the business service and select the location to save it.

3. Select the connection that you want to use or create a new one.

4. Optionally, to refresh the connection metadata every time you run the task, enable dynamic refresh.

5. Select the operation you want to use from the web service.

6. If necessary, configure the operation to specify the choice elements and derived type elements for the request and the response.

    If operation components include choice elements or complexType elements where the abstract attribute is true, then you must choose one or more elements or derived types when you configure the operation mapping.

    Optionally, for a complexType element where the abstract attribute is false, you can also select a derived type for a complexType element.

    a. For the operation you want to configure, click **Configure**.

    b. From the **Configure Operation** window, click the **Request**, **Response**, or **Fault** tab and navigate to the node you need to configure.

        Note: If the WSDL uses the anyAttribute element, the element will not appear for the request or the response.

        You can click the icons at the top to navigate to the nodes that you need to configure:

        1. Expand all nodes.
        2. Collapse all nodes.
        3. Show only nodes that you must configure. If you select this icon and no nodes expand, then you do not
            have to choose any choice elements or derived types.
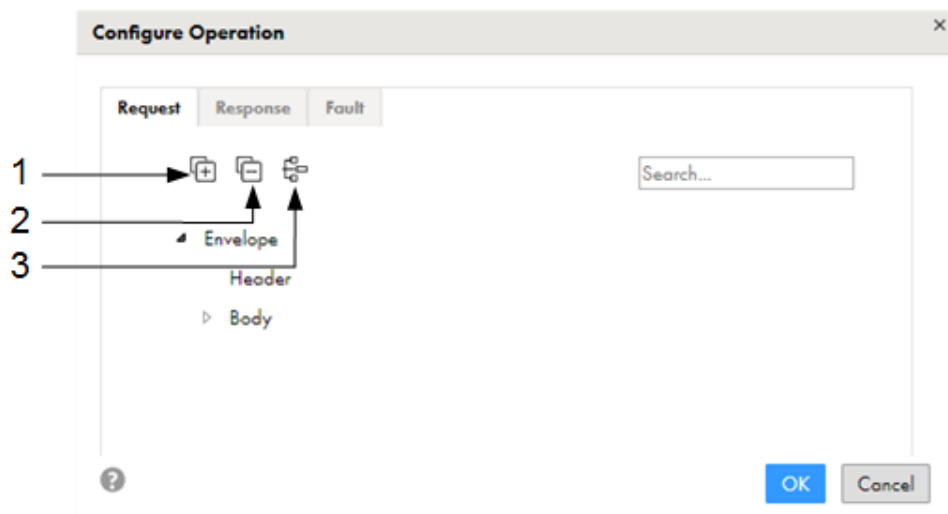
    c. Select the choice elements or derived types.

        Ensure that you configure the request and the response for the operation by clicking the relevant tabs.

      d.   Save the configured operation.

7. Optionally, add more operations and configure the operations if necessary.

8. Save the business service.

    If you have not configured all the required choice elements and derived types, then you cannot save the business service.

Configure the Web Services transformation

After you have defined the business service, you can configure the Web Services transformation in a mapping to access or transform data from a web service. The requirements and available options depend on whether you are connecting the Web Services transformation to one source object or to more than one source object.

Configuring the transformation

When you configure the web services transformation, you connect a source object, configure properties for the transformation, map incoming fields to requested fields for the web service, and map the response to output fields to create one or more success groups. Data Integration creates a fault group automatically but you can choose whether to map it to the output fields.

1. Create a mapping and add the source objects you want to work with.

2. Add a Web Services transformation to the canvas.

3. Connect the source to the Web Services transformation.

4. Select the business service and operation in the **Web Service** tab.

5. On the **Request Mapping** and **Response Mapping** tabs, create the field mappings between the source fields and the web service request.

    For an illustration of the mapping process, see "Web Services transformation example" on page 465.

6. On the **Output Fields** tab, review the success groups, fault group, and field details. You can edit the field metadata, if needed. The success groups contain the SOAP response from the web service. The fault group contains SOAP faults with the fault code, string, and object name that caused the fault to occur.

7. Define the advanced properties.

8. Save and run the mapping.

For additional information about the mapping process, see the following sections:

Advanced properties

The following table describes the properties available for the Web Services transformation from the **Advanced** tab:

Property	Description
Cache Size	Memory available for the web service request and response. If the web service request or response contains a large number of rows or columns, you might want to increase the cache size. Default is 100 KB.
Allow Input Flush	The mapping task creates XML when it has all of the data for a group. When enabled, the mapping task flushes the XML after it receives all of the data for the root value. When not enabled, the mapping task stores the XML in memory and creates the XML after it receives data for all the groups. **Note:** You cannot select the option to allow input flush if you are connecting to multiple source objects.
Transaction Commit Control	Control to commit or roll back transactions based on the set of rows that pass through the transformation. Enter an IIF function to specify the conditions to determine whether the mapping task commits, rolls back, or makes no transaction changes to the row. Use the transaction commit control if you have a large amount of data and you want to control how it is processed. **Note:** You cannot configure a transaction commit control if you are connecting to multiple source objects.

Mapping incoming fields

When you define the request mapping, you can configure relationships between more than one source object. You configure relationships between the response mapping and the output fields separately.

When you map incoming fields, note the following guidelines:

- If you need to apply an expression to incoming fields, use an Expression transformation upstream of the Web Services transformation.
- To ensure that a web service request has all the required information, map incoming derived type fields to fields in the request structure.

You can map the incoming fields to the request mapping as shown in the following image:
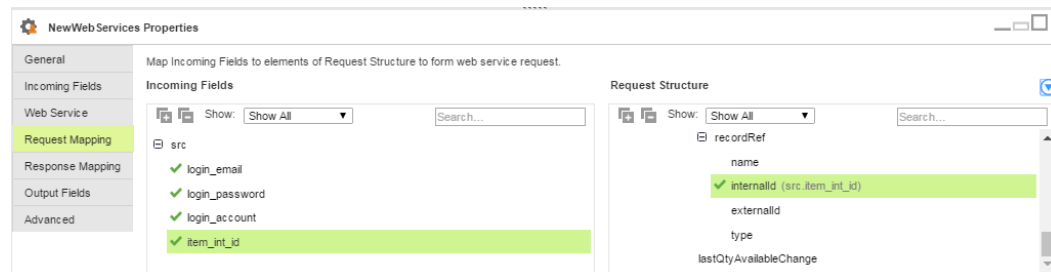
Drag each incoming field onto the node in the request structure where you want to map it.

Working with multiple source objects

If you have multiple sources, note the following requirements:

- Any source fields you want to designate as primary key and foreign key must use the data type Bigint or String. If needed, you can edit the metadata in the Source transformation.

  Note: If the Bigint data type is not available for a source, you can convert the data with an Expression transformation upstream of the Web Services transformation.

- Ensure that the source data is sorted on the primary key for the parent object and sorted on the foreign key and primary key for child objects.

- Map one of the fields or a group of fields to the recurring elements. In the incoming fields, you can see where each recurring element is mapped.

- Map at least one field from each child object to the request structure.

- You must map fields from the parent object to fields in the request structure that are higher in the hierarchy than fields from the child object.

- For child objects, select a primary key and a foreign key.

  - On the **Incoming Fields** tab, select the source object you want to designate as the parent object.

  - Right-click on an incoming field in the tree to designate the primary key and foreign key.

  - For the foreign key, specify the parent object.

- Do not choose a foreign key for the parent object.

Mapping outgoing fields

On the **Response Mapping** tab, you map the response structure to the output fields you want to use. You can choose relational or denormalized format for the output fields.

When you choose **Relational**, the transformation generates the following output groups:

- One output group for the parent element.

- One output group for each element in which the cardinality is greater than one.

- FaultGroup, if it is supported by the connection type you are using.

When you choose **Denormalized**, the transformation generates the following output groups:

- Output group for the parent element. In denormalized output, the element values from the parent group repeat for each child element.

- FaultGroup, if it is supported by the connection type you are using.

Right-click the node in the node of the response where you want to map the response to output fields. You can choose to map all descendants or only the immediate children, as shown in the following image:
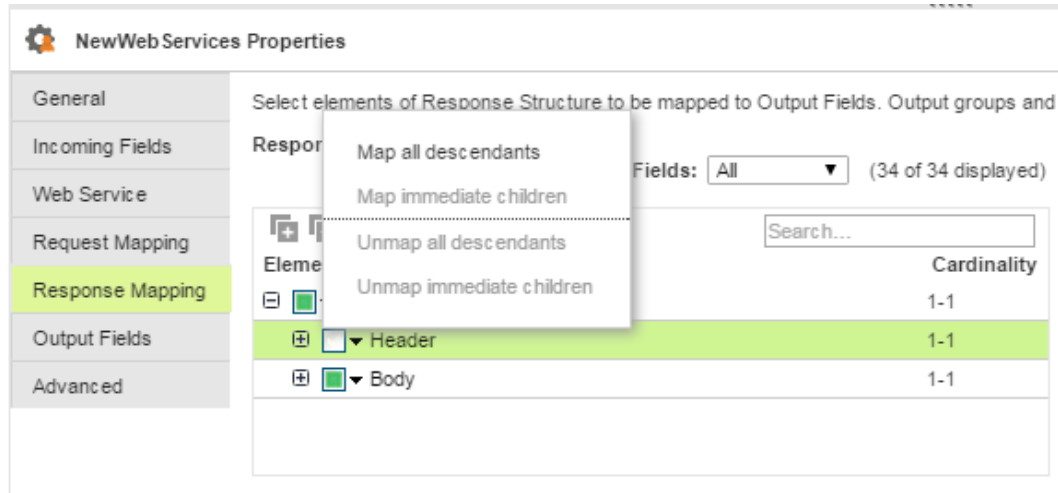
Transaction commit control

The transaction commit control lets you control commit and roll back transactions based on a set of rows that pass through the transformation.

Enter an IIF function to specify the conditions to determine whether the mapping task commits, rolls back, or makes no transaction changes to the row. When the mapping task issues a commit or roll back based on the return value of the expression, it begins a new transaction.

Note: You cannot configure a transaction commit control if you are connecting to multiple source objects.

Use the following syntax for the expression:

```
IIF (condition, value1, value2)
```

Use the following built-in variables when you create a transaction control expression:

- TC_CONTINUE_TRANSACTION. The mapping task does not perform any transaction change for this row. This is the default value of the expression.

- TC_COMMIT_BEFORE. The mapping task commits the transaction, begins a new transaction, and writes the current row to the target. The current row is in the new transaction.

- TC_COMMIT_AFTER. The mapping task writes the current row to the target, commits the transaction, and begins a new transaction. The current row is in the committed transaction.

- TC_ROLLBACK_BEFORE. The mapping task rolls back the current transaction, begins a new transaction, and writes the current row to the target. The current row is in the new transaction.

- TC_ROLLBACK_AFTER. The mapping task writes the current row to the target, rolls back the transaction, and begins a new transaction. The current row is in the rolled back transaction.

If the transaction control expression evaluates to a value other than commit, roll back, or continue, the mapping is invalid.

Example

You want to use transaction commit control to write order information based on the order entry date. You want all orders entered on a given date to be committed to the target in the same transaction.

You create a field named New_Date and populate it by comparing the order date of the current row to the order date of the previous row. If the orders have different dates, then New_Date evaluates to 1.

You then use the following expression to commit data when the mapping task encounters a new order date:

```
IIF(New_Date = 1, TC_COMMIT_BEFORE, TC_CONTINUE_TRANSACTION)
```

Viewing incoming and request fields

When you review incoming fields and request fields, you might want to reduce the number of fields that appear on the page.

You can view and locate fields using the following methods:

Expanding and collapsing nodes

To display the parent and child nodes in a tree, you can expand all nodes. To display only the parent nodes, collapse the nodes. Use the **Expand All Nodes** and **Collapse Nodes** icons on the Request Mapping or Response Mapping tabs.

Searching for fields

To search for a particular field, type the field name in the **Search** text box.

Filtering the fields

You can filter the incoming fields view to show all fields, keys, mapped fields, or unmapped fields. You have a similar option in other tree views:
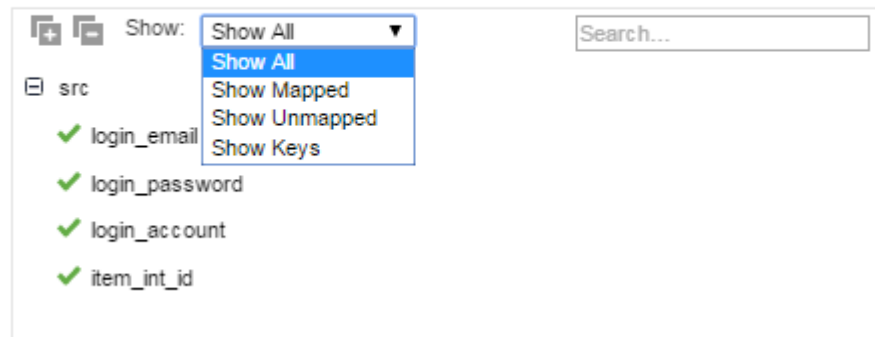
View the field and field mapping details

For each node in a hierarchy, you can view the field and field mapping details. Right-click on the node to show the properties:

Clear the field mappings and keys

In the Request Structure tree, you can clear the mapping, clear the keys, or map and unmap selected fields:
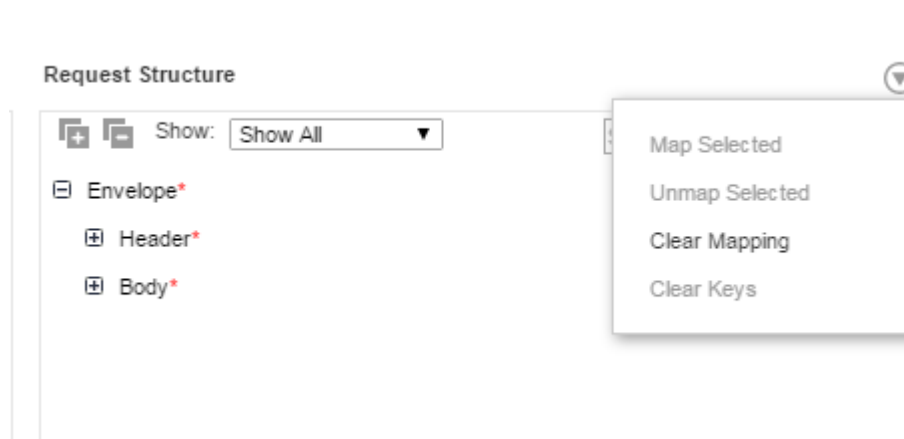
Pass-through fields

Pass-through fields are response fields that you do not need for the current transformation but you might want to use later in the mapping.

For example you might only need the primary key from a source object for the Web Service transformation. However, you might want to use the other fields in the source object in a downstream transformation.

Incoming fields and response fields pass through the Web Services transformation to the success groups and fault group. However, when the mapping contains multiple sources, only the fields from the primary source pass through. The source that is connected to the Web Service transformation first is the primary source. For example, in the following image, NewSource is the primary source and NewSource1 is the secondary source:
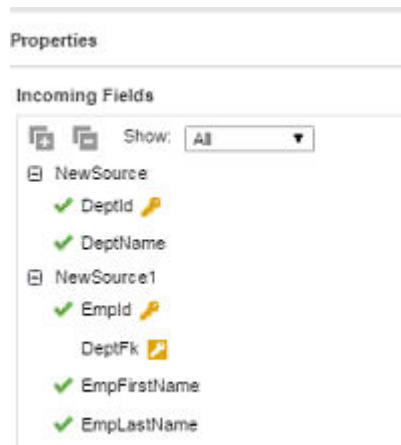
Because NewSource1 is a secondary source, fields from this source do not pass through to the success groups and fault group.

To determine the source of a field in the Success or Fault group, choose the **Incoming Fields** tab. The **Origin** column shows the source for each field.

Web Services transformation example

The following example demonstrates how to use the Web Services transformation to interact with NetSuite to get product availability details.

You can review this example to learn how to configure a Web Services transformation to structure a SOAP request and response using the NetSuite operation, getItemAvailability. The web service client passes the item ID in the request. The web service then returns information on the latest item availability in a SOAP response.

First, to connect to the web service, create a WSConsumer connection. You add a WSDL that describes the request and response web service messages and configures the necessary security. For this example, we use the following NetSuite connection:

Second, define a business service using the connection. In this case, we use the connection to define a business service and access the getItemAvailability operation:

Third, create a mapping that uses the Web Service transformation. The example mapping includes the following configuration options:

1. The source is a simple .csv file that includes four fields with the login details and an item ID:

2. On the Web Service tab, select the business service and operation previously defined:

3. After you add a Web Services transformation, you can connect the source and see four incoming fields. You can filter your view of the incoming fields to show all fields, mapped fields, unmapped fields, or keys:
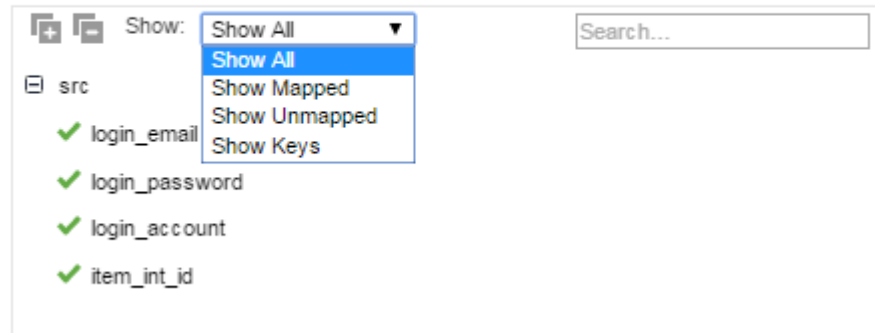
4. In the Request Mapping, the incoming fields are mapped to the SOAP message. The Envelope contains the credentials and the Body contains the item ID, as shown in the following image. You drag incoming fields onto items in the request structure to create the mapping:
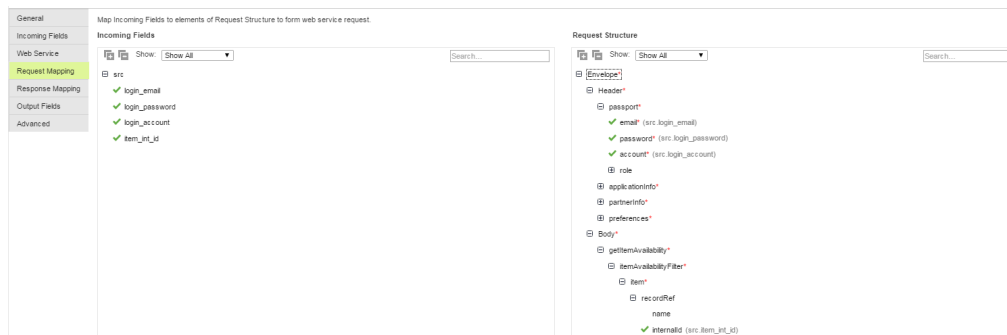
5. On the Response Mapping tab, you map the SOAP message to the output fields you want to use. You can choose Relational or Denormalized format for output fields. This example uses Relational format:
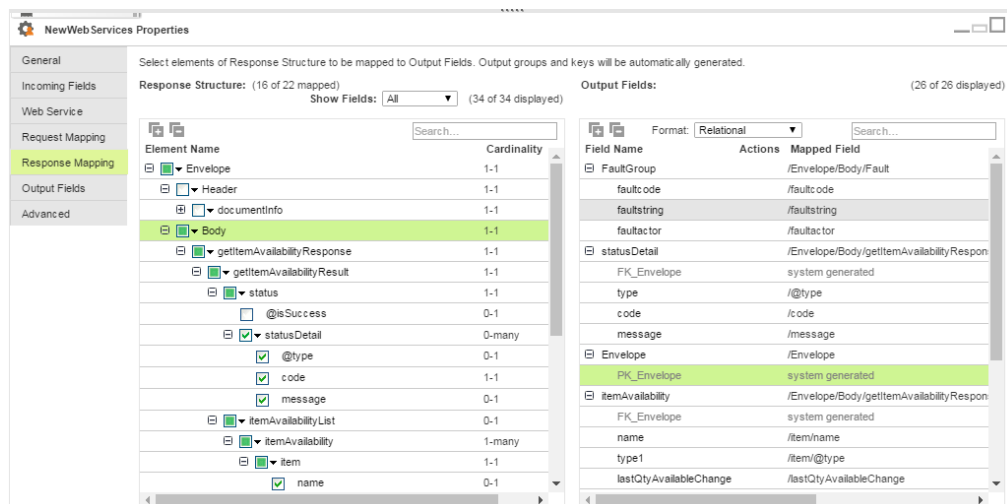
The system generates these output field groups automatically:

- FaultGroup, if it is supported by the connection type you are using.

- Envelope, which contains the documentinfo, if you map the header.

- One group for each element you have mapped where the cardinality is greater than one.

The example above has three groups. In the response, you can choose to map or unmap objects and fields by selecting a node in the hierarchy and choosing an option to map (or unmap), as shown in this image. Click the down arrow to display the mapping options for a specific node:
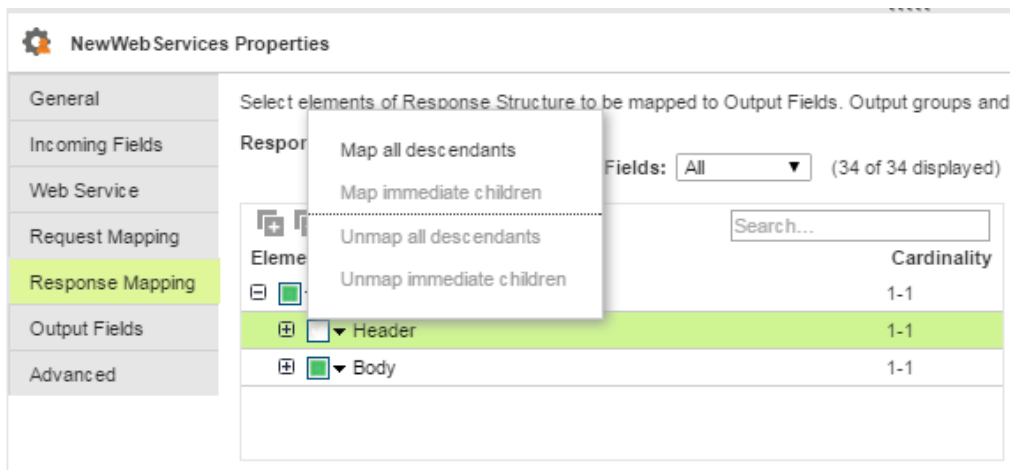
6. On the Output Fields tab, you can see each group. If needed, you can edit the type, precision, and scale of the fields:
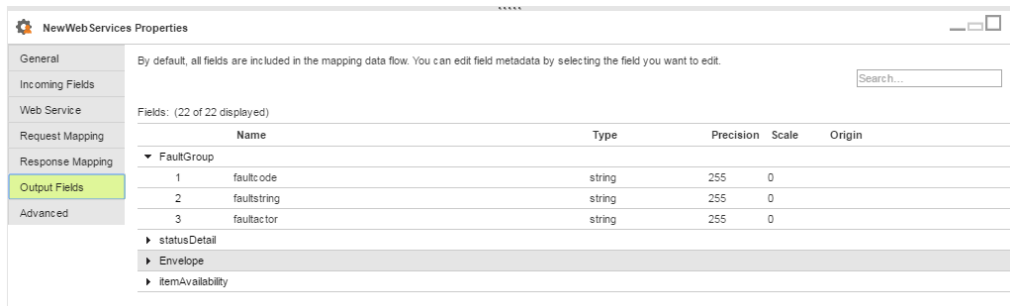
7. The mapping includes three targets: one for the status, one for the availability, and one for the FaultGroup data:

When the mapping runs, it returns the item availability and status code. If it cannot successfully run, it creates a record in the default target that contains the fault information.

Configuration for multibyte hierarchical data

If a mapping includes a transformation that processes hierarchical data and the data uses multibyte characters, configure the Secure Agent machine to use UTF-8.

On Windows, create the INFA_CODEPAGENAME=UTF-8 environment variable in Windows System Properties.

On Linux, set the LC_LOCALE variable to UTF-8.

Index

H

I

J