
Building pipelines

Common patterns & best practice

Updated 26 February 2024

support@quantemplate.com
help.quantemplate.com

Stage and operation naming.....	3
Stage naming.....	3
Operation naming.....	3
Group similar operations, minimise number of stages.....	4
Importing files.....	5
Map to an interim schema.....	7
What is an interim schema?.....	7
Use it as early as possible in the pipeline.....	7
Deploying an interim schema in a pipeline.....	7
Using existing mappings.....	7
Bringing in a new file.....	7
Updating a master schema.....	8
Copying a master schema.....	8
Adding original row number.....	9
Populating fields with default values.....	10
Basic text and number cleansing.....	12
Trim and Change Case.....	12
Strip out non-alphanumeric characters.....	12
Combine Initials.....	13
Remove accented characters.....	13
Add leading zeros before a number.....	14
Cleanse and split names of people.....	14
Dates.....	18
Extract a date from a filename.....	18
Joins.....	20
Preparing join points.....	20
Validating joins.....	20
Validation and amendment.....	21
Validating against a Reference Dataset.....	21
Correct invalid values by fuzzy matching to a reference table.....	22
Aggregating (pivoting).....	23
Transposing columns into rows.....	24
Preparing Outputs.....	27
Output column headers.....	27
Output rounding.....	27
Simplify the number of outputs.....	27
Performance optimisation.....	28
Automap Values.....	28
Aggregate.....	28
One-to-many joins.....	28
Partition.....	28
Generating outputs.....	28
Productivity tips.....	29
Bulk Copy-paste Calculations or Validations.....	29
Test operation output without running the whole pipeline.....	29
Quickly previewing values without running the pipeline.....	29
Updating Reference datasets.....	29
Filters keyboard shortcut.....	29








Stage and operation naming






Stage naming

Stages are easier to understand when similar operations are grouped into logical sections. Stage names should reflect the operations being performed within them. This makes it easier for other members of your team to quickly identify critical functions of the pipeline. Compare the two examples below:




Bad Example

1	First Stage	
2	First Calcs	
3	Second Calcs	
4	Combined Calculations	
5	Final Calculations	

Good Example

1	Map Headers & Tag Data	
2	Date Calculations	
3	Exposure Calculations	
4	Premium & Rate Calculations	
5	Final Validations	

It is also helpful to clearly identify outputs that will be downloaded (Stage 11 below), published to the data repository (Stage 12), or reviewed without publishing or downloading (Stage 13).

11	Output: Current Dataset	
12	Publish: Combined Dataset	
13	Checksum: Total Premium by State	

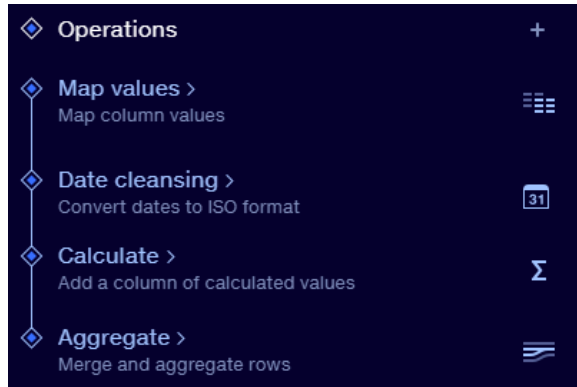
Operation naming

Like stages, operations should have names and descriptions that make it easy to tell what they are doing, and if possible, any columns being created as the result. This makes it easier to review the pipeline.

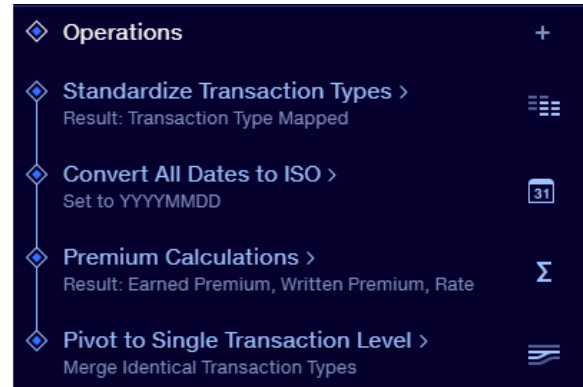


Both column names and operations do not need to have technical names, which can create confusion (e.g. CL_ID vs. Claim ID). Clear, easy to understand column and operation names eliminate confusion about the function being performed or what value the column represents.

Bad Example



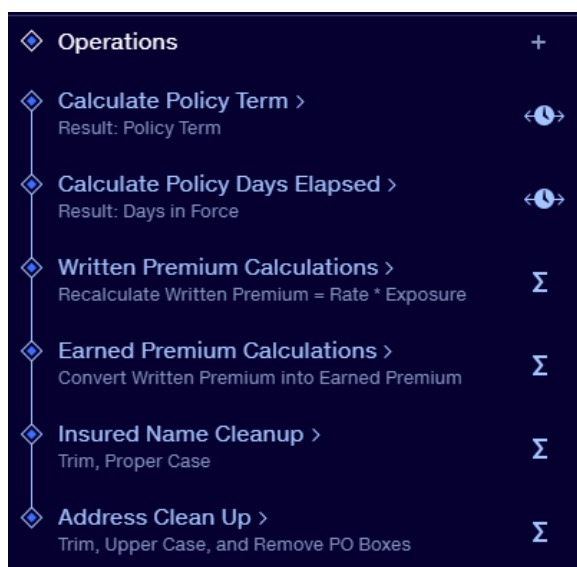
Good Example



Group similar operations, minimise number of stages

Group similar operations together within a stage, to make it easier to review how values are being manipulated or determine where they are being created.

Try to keep the number of stages to a minimum – in a complex pipeline, it's better to have fewer stages to click into to reveal operations inside.



Importing files

This is the standard pattern for importing, then processing files.

Stage 1: Import, Map and Cleanse

1. Transform to a single-header table
2. Map headers to an interim schema
3. Append filename, basic cleansing and validations

Stage 2: Union, to automate flow-through of new files

Stage 3: Calculations, etc.

Stage 1: Import, Map and Cleanse

Inputs: Uploaded bordereaux, 'REF: Master Headers' from data repo

1.Transform to a single-header table

Usually this requires two steps:

1. [Remove rows](#) with less than [3] columns: if there is data above or below the main table, this operation will strip it out. The column filter function in Remove Rows can also be included to strip out subtotals (e.g. remove rows where column 'unnamed' contains 'total').
2. [Detect headers](#): define the number of rows which will be used as headers, combining double or multiple headers into a single header row.

Occasionally, there may be files which require additional steps:

- *Multiple data grids placed next to each other on the same sheet:*
Use [Stack Grids](#) in combination with [Fill Down](#). [Watch the tutorial](#).
- *Tables need to be flattened or transposed:*
Use [Swing Down](#). See the [Transposing data](#) section below.

2. Map Headers to interim schema

Use Map Column Headers to map the incoming data to an interim schema. More details in the [Interim schema](#) section below.

3. Append filename, basic cleansing and validations

- [Date cleanse](#): converts dates to QT format.



- Optional: Use [Append metadata](#) to add columns such as the input filename, pipeline name, ID, run time, run number. These can be used to trace the source of a row after the data has been unioned.
- Optional: Initial [validation](#): apply data quality checks (columns are not blank, claim values are not negative, inception date is after expiry date, etc).

Stage 2: Union, to automate flow-through of new files

Inputs: All Stage 1 outputs.

Set [Linked Stages ON](#) on the Stage 1 outputs. When new files are brought in to Stage 1 they will flow through to Stage 2 automatically.

The output ID from a Union Stage is immutable, so stages referencing it are not affected by changes to the Union's input files. Using a Union early makes the pipeline faster to Trace and more resilient to configuration changes.



Map to an interim schema

What is an interim schema?

An interim schema is a common schema (set of headers) which uses easy to understand terminology and is independent of any specific input or output header names.

Use it as early as possible in the pipeline

Incoming bordereaux should be mapped to the interim schema as early as possible, before any calculations, validations or cleansing steps are carried out. This means changes to the incoming header names need only be dealt with once, in the mapping stage, rather than in all downstream operations relying on that column name.

Likewise, at the end of the pipeline, the data can be mapped from the interim schema to the output format(s). This means changes to the output header names need only be dealt with once, in the mapping stage, rather than in all upstream operations relying on that column name

Deploying an interim schema in a pipeline

To deploy an interim schema in a pipeline, store the headers as a dataset in the Data Repo. The dataset should have **zero rows**, to avoid additional rows being injected into the bordereaux data.

Add the reference headers dataset as a stage input and it will appear as a schema in the Map Column Headers operation, where you can [define a master](#) from it.

Using existing mappings

If your organisation has a set of existing approved mappings, these can be used to seed the Map Column Headers Automap function, so that approved mappings always have a 100% rating.

Bringing in a new file

When a new bordereau file is brought in, if the schema is identical to a previous input, the mappings will be remembered. Note that it may take a few seconds for the new input to trace through the pipeline and for the mappings to be applied.



If there are variations in the schema, the mappings will need to be reapplied. Automap will quickly reapply previously made mappings.

Updating a master schema

Updating the interim schema dataset will update it as an input in all pipelines it is used in. However, it won't automatically update the Master Schema in Map Column Headers.

The best way to update the Master Schema, depends on how big the update is:

- If it's just one column name changing, that can be updated manually in the master schema.
- If multiple columns are being added, then add them to master from the Master Headers input, one-by-one.
- If many column names are changing, it may be best to use 'Define a master from this dataset'. This will copy over all items but may undo mappings that had previously been made.

Copying a master schema

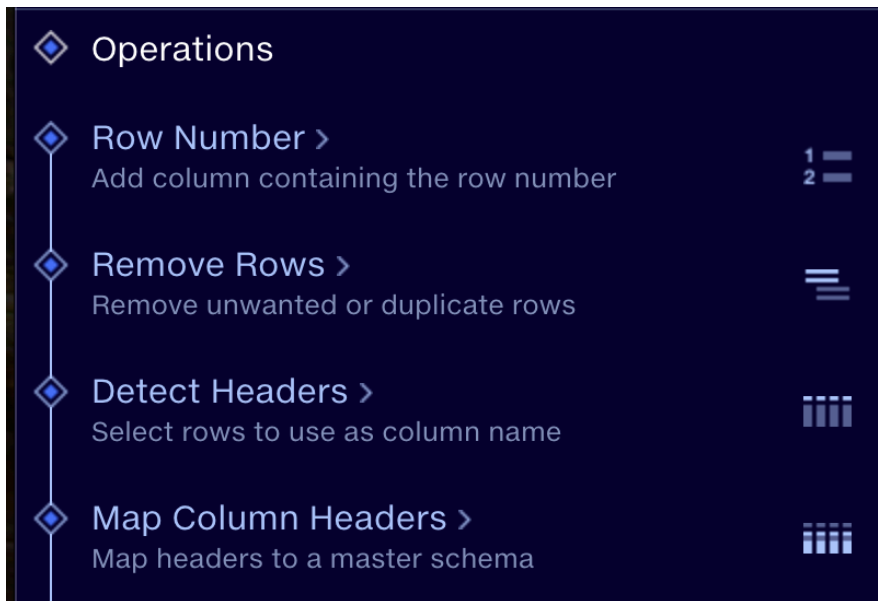
To copy a master schema from Map Column Headers:

1. View the output data from that stage.
2. Copy the data by clicking in the top left of data grid to select all items, then using ctrl/cmd c.
3. Paste the items into a spreadsheet.
4. Remove all rows other than the headers.
5. Save the spreadsheet then upload to the data repo.

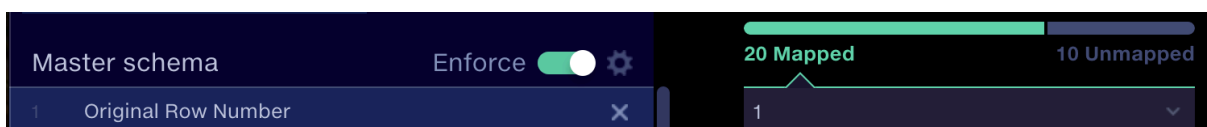


Adding original row number

Appending the uploaded files' original row numbers can help when manually reviewing or resolving validation errors. It's often easier to say 'there's a missing expiry date in row 17', rather than 'there's a missing expiry date in policy number x, certificate number y, transaction number z'. To append the original row number, add the following operations as the first items in the first stage.



1. **Row Number**
This writes a column of row numbers
2. **Remove Rows**
Remove subtotals or metadata above the main data
3. **Detect Headers**
Disable the 'Ignore inputs with named headers option', since the Row Number operation will introduce a named header.
4. **Map Column Headers**
Detect Headers has discarded the header for the Row Number column and replaced it with the first row number remaining after Remove Rows. So, if there were no rows removed above the data, the header would be called '1'. Map this to the name 'Original Row Number' in Map Column Headers.



Populating fields with default values

It's often necessary to fill unpopulated columns with default values. There are a few ways to do this:

1. Add fields in Calculate

Enter the column names as new fields and the desired value within the field.

- a. If desired, an IF statement can be used to determine if the column is already populated and return a default value if not.
- b. The whole array can be copy-pasted to another Calculate operation in another pipeline if desired.
- c. Note that there are a maximum of 51 calculations in a single Calculate operation.

2. Map Column Headers ['populate'](#) function

- a. If the column does not exist in the source data, this will add it and populate it with a value. It can either populate all new blank columns at once with the same value, or add different values to specific columns within a source input.
- b. If a new source file is brought in, then populated values for that file will be lost, so it needs to be done at a point in the pipeline with consistent inputs, e.g. when mapping to an output schema.

3. Append If Missing operation

This operation adds and populates a single column. It's useful where one column is sometimes missing in a group of otherwise identical input files. Using Append If Missing prior to map column headers means only one schema needs to be mapped in.

4. Bring in the values from a reference dataset

One way to manage default values at scale is to have a central reference table with all the default values for all the data producing parties, so 'ABC-MGA' would have one row, 'Acme-MGA' another. The dataset can be centrally maintained, or connected via API. It requires a couple of additional steps prior to the bordereaux import stage, as follows:

Acme-MGA pipeline

Stage 1: Prep Defaults

Input: 'REF: Defaults table' (this should have same headers as interim schema)



Remove Rows where 'MGA name' column does not equal "Acme-MGA"

Stage 2: Import and map BDX

Inputs: Monthly BDX spreadsheets, 'REF: Master Headers'

Remove rows and detect headers and map to interim schema as normal

Stage 3: Union

Inputs: Stage 1 and Stage 2 Outputs

You will now have a single file, consistent with interim schema. Columns with default values have the default value in the first row only.

If only one input file will be processed at once, then this Union stage can be move to Stage 2. See [Import pattern 2: single files](#) above.

Stage 4: Fill Down and clean up

Inputs: Stage 3 Outputs

[Fill Down](#) the columns which have default values, to fill out the rest of the column.

Use a Remove Rows to strip out the single row taken from the Default Values file (e.g. Remove rows where "Policy number" is empty').

This approach works best when the default value columns are not present in the uploaded bordereaux. If some of the columns are present, their values will be retained, though any blanks in these columns will be filled down from the values above. Therefore, in this situation the reference dataset for default values should be brought in using a [join](#) and the values populated from the joined-in values using an IF statement in Calculate.



Basic text and number cleansing

Trim and Change Case

Standardize text strings by using the functions [UPPER](#), [LOWER](#), or [PROPER](#) to set the case, along with [TRIM](#) to eliminate any extra whitespace.

```
[REF]Company Name - TrimUpper  
UPPER ( TRIM ( [REF]Company Name - Raw ) )
```

Strip out non-alphanumeric characters

Use the following [regex](#):

```
[^\w\s\d]
```

[← More Key Fields](#) | [Swing to Per Word](#)

◇ Extract

Fuzzification 3: Strip to Alphanumerics

Column [REF]Company Name - Fuzzy

[^\w\s\d]

ⓘ For more information on how to use a regex,
see the help centre

Output

[REF]Company Name - Fuzzy

Configure output value

☒ Remove matched value

☐ Advanced



Combine Initials

Useful in company name matching. Use the following [regex](#):

```
\b(\S)\s+(?=\S\b)
```

Extract

Fuzzification 5: Combine Initials

Column [REF]Company Name - Fuzzy

\b(\S)\s+(?=\S\b)

i For more information on how to use a regex,
see the help centre

Output

[REF]Company Name - Fuzzy

Configure output value

☐ Remove matched value

☒ Advanced \$1

Remove accented characters

Run a series of [SUBSTITUTE](#) operations in Calculate to replace accented characters with a non-accented alternative. Contact support@quantemplate.com to have this set-up pre-built and dropped into one of your pipelines.

```
[REF]Company Name - Fuzzy
(SUBSTITUTE (SUBSTITUTE (SUBSTITUTE (SUBSTITUTE (SUBSTITUTE (SUBSTITUTE ([REF]Company Name - Fuzzy ,"Á","A"),"Â","A"),"Ã","A"),"Ä","A"),"Å","A"),"Ä","A"),"Å","A")

[REF]Company Name - Fuzzy
(SUBSTITUTE (SUBSTITUTE (SUBSTITUTE (SUBSTITUTE ([REF]Company Name - Fuzzy ,"É","E"),"Ê","E"),"Ë","E"),"È","E")

[REF]Company Name - Fuzzy
(SUBSTITUTE (SUBSTITUTE (SUBSTITUTE (SUBSTITUTE (SUBSTITUTE (SUBSTITUTE ([REF]Company Name - Fuzzy ,"Ó","O"),"Ô","O"),"Õ","O"),"Ö","O"),"Ø","O"),"Ø","O")

[REF]Company Name - Fuzzy
(SUBSTITUTE (SUBSTITUTE (SUBSTITUTE (SUBSTITUTE ([REF]Company Name - Fuzzy ,"Ú","U"),"Û","U"),"Ü","U"),"Ü","U")

[REF]Company Name - Fuzzy
(SUBSTITUTE (SUBSTITUTE (SUBSTITUTE (SUBSTITUTE (SUBSTITUTE (SUBSTITUTE (SUBSTITUTE ([REF]Company Name - Fuzzy ,"Æ","AE"),"Ç","C"),"Ñ","N"),"ß","SS"),"Ð","TH"),"Ý","Y"),"Ý","Y")
```



Add leading zeros before a number

For example, format a 'Row Number' column from 1, 2, 3, 4 to 00001, 00002, 00003, 00004. Use the following formula in Calculate:

```
RIGHT(CONCATENATE("00000", 'Row Number'), 5)
```

Cleanse and split names of people

Scenario: you have names of people in a single column. The names need to be cleansed to remove Mr, Mrs, etc and split into a two columns of Last Name and First Name.

Contact support@quantemplate.com to have a copy of these steps placed into a pipeline in your organisation.

Input data:

- All in one column
- Various prefixes and suffixes
- Multi-word last names

	A	B	C
1	Name		
2	Leonardo da Vinci		
3	Mr Joe Gargery III		
4	Eric B Wopsle Jr.		
5	Mrs. Johanna Smith		
6	James Lee Johnson Sr		
7	Amr ibn Abd al-Wud		
8	Ms Catherine Havilland		
9	Dr. Barnabas Herbert Pumblechook		
10	Miss Estella Havilland		
11	Chris Paul Jaggery II		
12	Ursula von der Leyen		
13			

Output data:

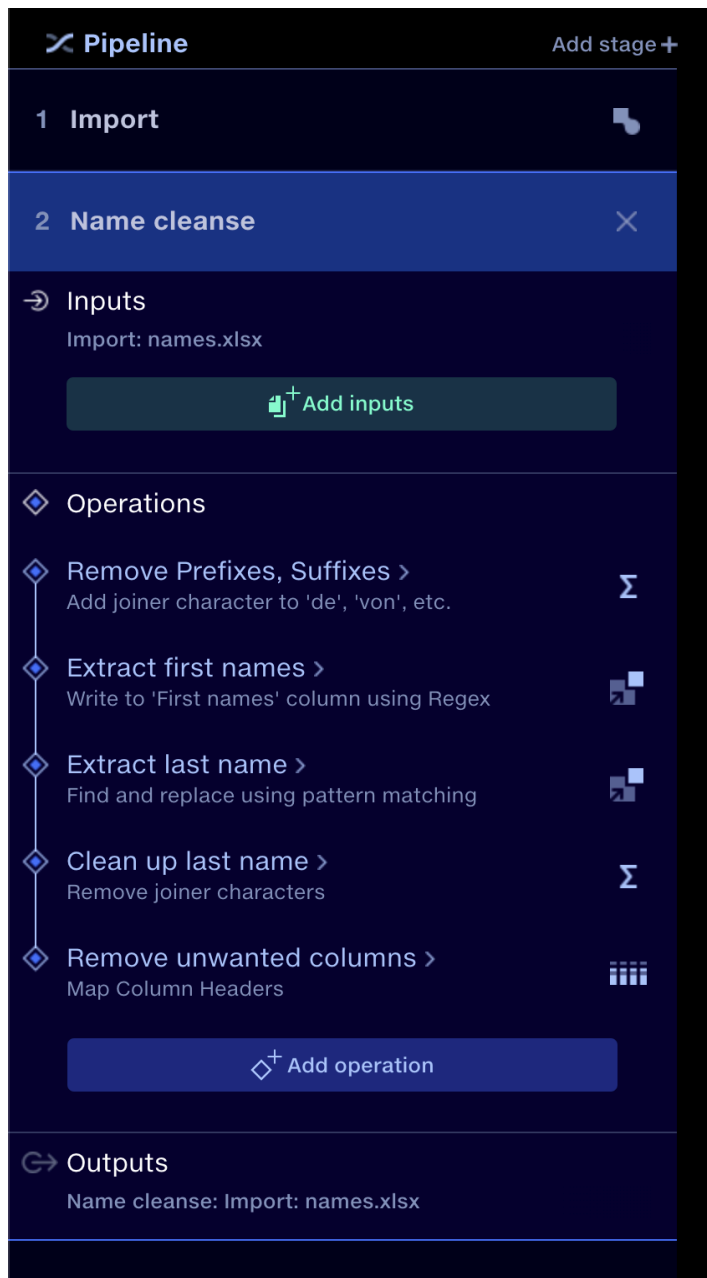
- Suffixes and prefixes removed
- Split into columns for first names and last name. The first names column also includes any middle names submitted. The process could be adapted to split these into a separate column if needed.
- Multi-word last names are split correctly



	First names	Last name
1	LEONARDO	DA VINCI
2	JOE	GARGER
3	ERIC B	WOPSLE
4	JOHANNA	SMITH
5	JAMES LEE	JOHNSON
6	AMR IBN ABD	AL-WUD
7	CATHERINE	HAVILLAND
8	BARNABAS HERBEI	PUMBLECHOOK
9	ESTELLA	HAVILLAND
10	CHRIS PAUL	JAGGERS
11	URSULA	VON DER LEYEN

Let's walk through the steps:





- **Remove prefixes and suffixes** [Calculate](#) operation performing a cleanse and prep:
 - [Uppercase](#) the names. This reduces the number of permutations required to remove suffixes etc. You could keep it mixed case if required and add the additional permutations:

```
UPPER('Name')
```

- Remove dots from Mr. Jr. etc:

```
SUBSTITUTE('Name', ". ", "")
```

- Remove common prefixes and suffixes (Mr, Dr, Sr, III, etc):

```
IF(LEFT('Name',3)="MR " or LEFT('Name',3)="DR " or  
LEFT('Name',3)="MS ", RIGHT('Name', LENGTH('Name')-3), 'Name')
```



```
IF(LEFT('Name',4)="MRS ", RIGHT('Name',  
LENGTH('Name')-4), 'Name')
```

- Last names comprising multiple words separated by a space are typically 'van', 'von der', 'de', etc. To deal with them, we identify the types individually and [substitute](#) the space with a '#' character. This allows us to treat the last name as a single word. The # is removed later:

```
SUBSTITUTE('Name',"VON DER ", "VON#DER#")
```

- **Extract first names** Using a [Regex](#), taking first group

```
(.*) ([^ ]+)$
```

- **Extract last name** Using the same Regex, taking second group
- **Clean up last name** Calculate operation to remove the # character

```
SUBSTITUTE('Last name',"#"," ")
```

- **Clean up columns** Remove the unwanted 'name' column using [Map Column Headers](#). This could be done as part of a more general column clean-up or mapping to output schema later in the pipeline if you wished.



Dates

Quantemplate uses the Basic ISO date format `yyyymmdd` for all date calculations and transformations.

Columns formatted as dates in Excel files (`.xlsx`, `.xlsm`, `.xls`) are automatically converted to Basic ISO format on uploading to Quantemplate (these can then be checked in the Date Cleanse operation). All other date formats, including dates from CSV file imports, need to be converted using the Date Cleanse operation.

See [Working with Dates](#) for guidance on how to use and process dates in Quantemplate.

Extract a date from a filename

A common scenario is that an important date is contained within a filename or tab name of an upload. This needs to be extracted and written to a column. Sometimes, the dates in the filename are in multiple formats.

In the import stage (usually the first stage) configure these operations:

- 1. Append Metadata**

Use the [Append Metadata](#) operation to add the Stage Input Name. This will write the filename to a column

- 2. Date Cleanse (to extract dates from filenames)**

Use the [Date Cleanse](#) operation to identify and extract dates within the filename column. Because the operation is searching for a date within a string, a specific date format should be [specified](#) – ‘Auto’ and ‘Auto US’ options won’t work in this use case.



If there are two or more date formats, perform the above operations, then:

3. In the Date Cleanse operation, add a [new group](#) for each date format. Write each format to a different new column. Disable the validation report for this operation, since we'd always expect some rows to not match the format and trigger a warning.



4. Calculate

Combine the two new date columns into a single using a conditional statement in Calculate:

```
IF('Date full US'="", 'Date short US', 'Date full US')
```

5. Map Column Headers

Now perform your import stage Map Column Headers to map all the incoming files to the [interim schema](#). The additional columns created in the Date Cleanse above can be dropped, along with the filename column if no longer needed.

6. Date Cleanse (general)

Perform a general Date Cleanse to check other incoming date columns, with validation enabled. The new column of extracted dates should be added here. This will report if a new date format is encountered in the filenames which results blanks in this column.



Joins

Join points must be identical between datasets, including case and whitespace. When a join “fails”, a blank value for that row is returned. If all join points fail, the column name will appear but contain no values.

Preparing join points

To increase the likelihood of a match between datasets, it may be necessary to clean up the join points using the [basic text cleansing](#) techniques.

Free form text values may also benefit from being standardized using [Auto Map Values](#), matching incoming data to a known good value that exists in the corresponding reference table.

Joins cannot be performed on blank cells. Blank entries on the joined column should be set to a value such as “N/A” in both the reference file and the pipeline.

In the join, configuring a new output to show “unmatched from left” or “unmatched from right” will help troubleshoot each row that fails the join.

Validating joins

This method will add rows which have been missed from joins to the validation report.

1. Input: unmatched values from any joins
2. Append Metadata: Stage input name. This will help identify the join that has failed.
3. Map values: map the file name to an error code (e.g. to flag if the join to ‘REF: US States’ fails, map the filename to "Unmatched State" and write the values to a new column "Failed Reason"
4. Validation rules: Failed Reason != "Unmatched State" or Failed Reason != "Unmatched Line of Business"



Validation and amendment

Validating against a Reference Dataset

Scenario: Confirm that every value in a column is using a value from a table of valid values, e.g. validate a 'policy_transaction_type' column

- Use a [Join stage](#) to join to your reference table.
 - Set the source data as your left dataset, your ref data as your right dataset.
 - Use the 'policy_transaction_type' column in both the input data and the reference data as the join point
 - Configure two outputs:
 - Output 1: Matched rows + Unmatched from left (this will return all the values in the source data)
 - Output 2: Unmatched from left (this will return only the invalid rows from the input data)

The screenshot shows the configuration for a 'Join stage'. It is set to a 'Left join' where the 'Left dataset' is 'Stage 1: transaction type sample data.xlsx' and the 'Right dataset' is 'transaction type ref'. The join points for both datasets are 'Transaction type'. Below this, the 'Outputs' section is configured with two outputs. 'Output 1' is named 'All Stage 1: transaction type sample data.xlsx with matched transaction type ref' and has three options: 'Matched rows' (checked), 'Unmatched from left' (checked), and 'Unmatched from right' (unchecked). 'Output 2' is named 'Unmatched Stage 1: transaction type sample data.xlsx' and has three options: 'Matched rows' (unchecked), 'Unmatched from left' (checked), and 'Unmatched from right' (unchecked).



- You can additionally create an entry in the [Validation Report](#) to visualise the proportion of failed rows:
 - In a subsequent Transform stage, take Output 1 from the Join as input.
 - Add a [Validate operation](#) and enter a validation condition to return a warning or failure when 'policy_transaction_type' column from the source data does not match the 'policy_transaction_type' from the joined-in reference data.
 - The failed rows will be reported on in the Validation Report. You can view and download them directly from there – this means you can dispense with Output 2 if desired.

Correct invalid values by fuzzy matching to a reference table

Invalid values identified by the process above can be corrected using fuzzy matching. This is accomplished using [Automap Values](#).

In this case you would:

- Create a Transform stage taking Output 2 from the Join (unmatched rows) as its input.
- Insert a Map Values operation:
 - Select 'policy_transaction_type' column as the 'map from' value
 - In the 'Settings' column on the right:
 - select whether to write the values to a new column (recommended) or to overwrite the existing column
 - Select a value to return when the value is unmapped, or leave it unchanged.
 - Select 'Automap'
 - Select the reference dataset by clicking on the 'Select reference dataset' text. Choose the reference dataset with the approved values, then on the line below select the column from the reference data.
 - We recommend leaving the match strength at zero for your first run and leaving [waypointing](#) deactivated. At this point you can try running the pipeline and seeing what kind of results you are getting. A fuzzy match will be performed against the ref dataset with the results reported on in the [Mapping Report](#). You can review these results, adjust match strength accordingly to filter out weak matches, and perform remappings.
- If desired, you can now join the cleansed values back into the main data. Note that the Automap Values could equally be performed on Output 1 from the join (returning a 100% match for the correct values). The advantage of this is that it simplifies the pipeline by not requiring an additional step to join cleansed values back in. The disadvantage is that Automap values may take longer to run if it is analysing every row, rather than just the rows we know to be invalid.



Aggregating (pivoting)

The [Aggregate](#) operation compresses rows with identical dimensions together. Similar to Joins, Aggregate Dimensions are case and whitespace sensitive. Dimensions should be standardised with [Basic text cleansing](#) techniques to ensure proper roll-up.

The Aggregate operation has a built-in validation tool to check that dimensions have correctly aggregated. The 'key' value automatically generates a validation failure and highlights rows that are not rolling up as expected. [More information.](#)



Transposing columns into rows

Example

The input file presents coverage types as columns, so there is one row per policy number and two columns for every coverage type coverage columns, since premium and commission is stated for each coverage type.

Input

Policy ID	Corporate Property Premium	Retail Property Premium	Corporate Property Commission	Retail Property Commission
P-123	5000	1000	500	50

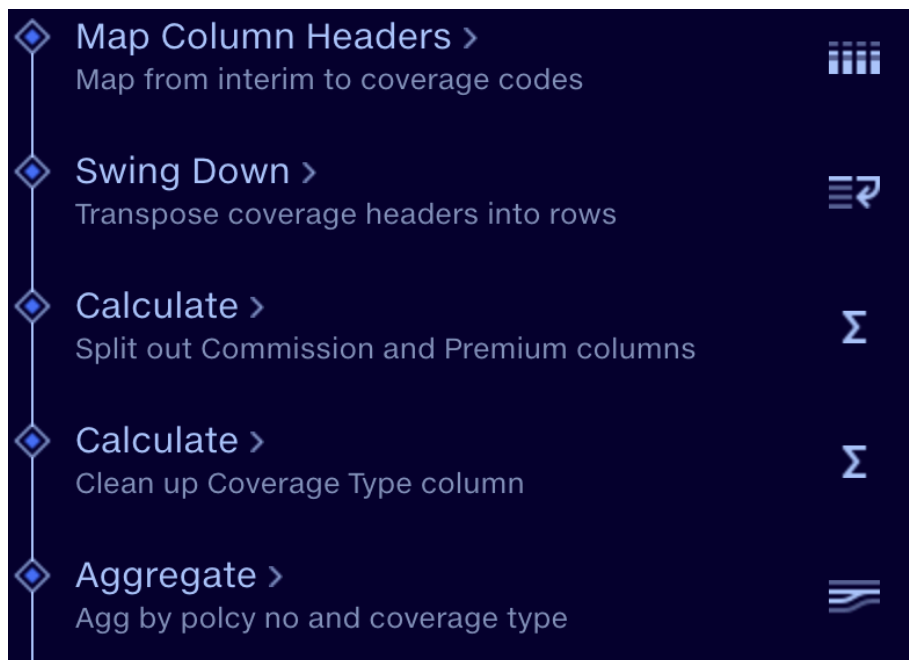
The desired output table presents coverage codes as rows with a premium and commission value for each.

Output

Policy ID	COVERAGE_ID	Premium	Commission
P-123	CORP_PROP	5000	500
P-123	RET_PROP	500	50



Method



1. Map the Coverage header names to coverage codes, appended with 'PREM' or 'COMM'.
2. [Swing down](#) all the Coverage columns in one operation:
 - a. Swing Down columns 'CORP_PROP PREM' 'CORP_PROP COMM', etc.
Output column name 'COVERAGE_ID'.
 - b. Value column: 'Value'
3. Premium and Commission values are now all in one column, so split out into separate columns using a pair of IF statements in Calculate. Optionally, an additional pair of IF statements can replace blanks with zero in the Premium and commission columns



Commission

```
IF ( RIGHT ( COVERAGE_ID , 4 )="COMM", Value ,"" )
```

Premium

```
IF ( RIGHT ( COVERAGE_ID , 4 )="PREM", Value ,"0")
```

Commission

```
IF ( Commission ="" , 0 , Commission )
```

Premium

```
IF ( Premium ="" , 0 , Premium )
```

4. Clean up the coverage names to remove the 'PREM' or 'COMM' suffix (and whitespace!)

COVERAGE_ID

```
LEFT ( COVERAGE_ID , LENGTH ( COVERAGE_ID )- 5 )
```

5. Aggregate back up to remove the duplicated rows.

COVERAGE_ID	DIM	
Commission	AGG	Max
Premium	AGG	Max



Preparing Outputs

Output column headers

If full granularity is required, map to output column headers using Map Column Headers.
If summary or aggregated values are required, use Aggregate

Output rounding

Calculated values are sometimes returned to many decimal places, eg:

'12345.3300000001'

The [Round](#) function in Calculate can tidy these to the desired decimal places, if required by the downstream system, This should be done at the end of the process to preserve accuracy.

Trying to round values in columns with mixed text and numbers may result in a 'NaN' for the text strings. This can be solved by using an IF and ISNUMBER

Example

Values in the column 'Value' need to be rounded, but the column contains text strings.

Create a column in Calculate called 'Value'

IF(ISNUMBER('Value'),ROUND('Value',2),'Value')



Simplify the number of outputs

Once the output is formatted run it into a Union stage to rename the output, e.g.
'Modelling Output'

[Disable outputs](#) from all other stages so they do not appear in the outputs view. This will also speed up the pipeline run, since these outputs do not have to be created.

Once the outputs have been created by the pipeline they can be exported to the data repo.



Performance optimisation

The most performance-intensive operations are [Automap Values](#) and [Aggregate](#).

Automap Values

- Use only once per stage. If you need to use multiple Automap Values operations, put them in individual stages.
- Reduce the number of potential matches by using [context columns](#) or segmenting the volume of data going in (e.g. GWP >\$Xm)

Aggregate

Only use for Aggregation, don't use to simply rename or subset columns.

The following operations can explode data sizes:

One-to-many joins

A one-to-many join occurs when the items in one dataset are matched multiple times in the other dataset, creating a row for every permutation.

Consider if the full set of exploded values is required. Could the same result be achieved by joining to a simple lookup table? See [Company Nmae Matching solution deep dive video](#), 23:50 onwards.

Partition

Creates a table for each value in the partition column, so use with care. Limited to 5000 tables.

Generating outputs

[Disable outputs](#) from all other stages so they do not appear in the outputs view. This will speed up the pipeline run, since these outputs do not have to be created.



Productivity tips

Bulk Copy-paste Calculations or Validations

Calculated columns can be copied and pasted all at once to another Calculate operation. Validation rules can be copied and pasted all at once to another Validate operation.

To copy: open a Calculate or Validate operation and with no fields selected click ctrl+c (Win) or cmd+c (Mac).

To paste: open a Calculate or Validate operation and click ctrl+v (Win) or cmd+v (Mac). This will add the copied columns below any existing columns.

Test operation output without running the whole pipeline

If you've built a long pipeline, or have a lot of data, it can take a few minutes to run. If you're adding stages to the end of this pipeline, you probably just want to test the effect of changes to this stage without running the whole pipeline.

To do this, export the output from the previous stage to the Data Repo. Bring this output into a new pipeline and construct the new stage in there, allowing you to rapidly test changes to your configuration. When you're done, copy the configuration back into the main pipeline.

Quickly previewing values without running the pipeline

A helpful trick to identifying how a value progresses without executing a pipeline is to add a Map Values operation in various stages of the pipeline. On larger datasets, Map Values may take a few moments to parse all possible values.

Updating Reference datasets

To refresh a reference dataset with new data, drag and drop the new data into the preview of the dataset.

If a reference dataset name is updated, the file name will update automatically across the system, wherever it is used.

Filters keyboard shortcut

Whenever a filterable data grid is shown, press 'F' to toggle between hiding and showing the filter bar.



