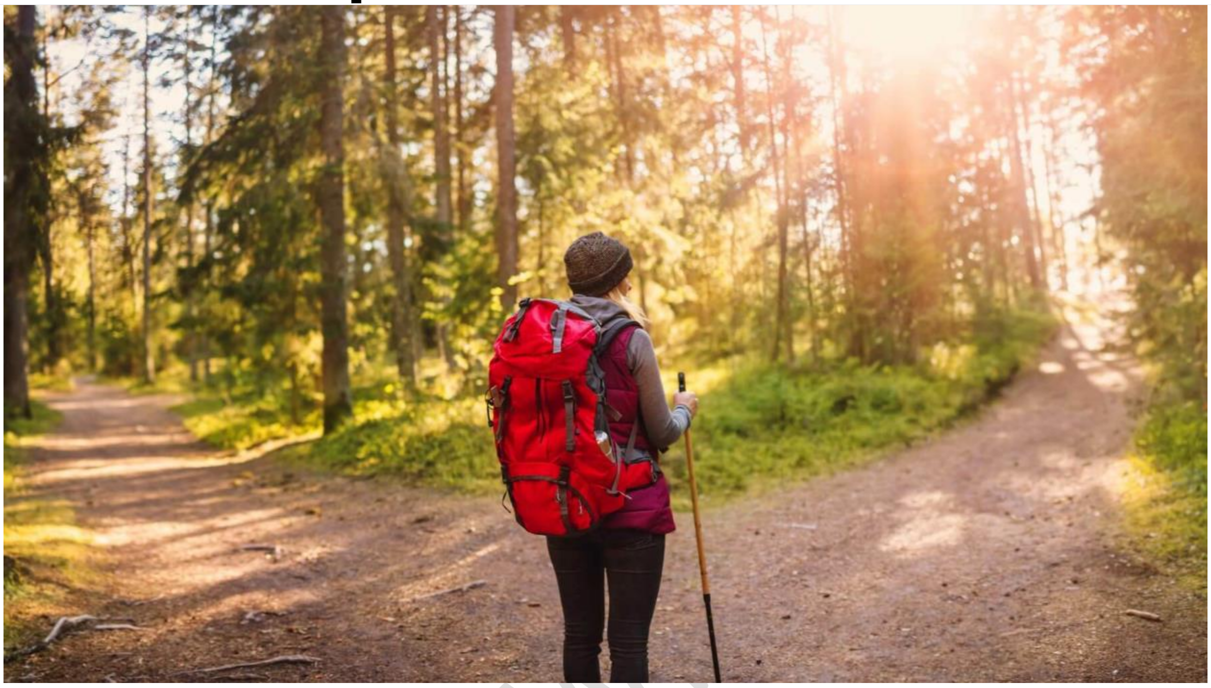
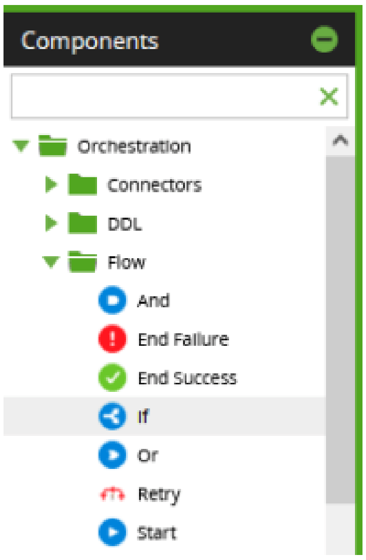
Flow Control in Matillion ETL: The If/Else Component

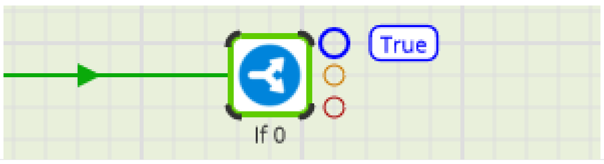


This guide will help you implement if/else flow control – known as conditional branching – in Matillion ETL Orchestration Jobs.

The **If component** allows you to supply a logical expression that determines the branching. Look for "If" among the Flow components while you are editing an Orchestration Job.



Whenever an If component runs, its condition expression is evaluated, and the flow of control passes to exactly **one** of its three output connectors.



- True – control passes along the top, blue colored path only
- False – it passes along the middle, orange path only
- Failed to evaluate the expression – it passes along the bottom, red path only

You can use all three output connectors if you want, but only **one** of them will ever actually be chosen at runtime.

Prerequisites

The prerequisites for implementing conditional branching in Matillion ETL are:

- Access to a **Matillion ETL** instance
- Permission to edit and run **Matillion ETL Jobs**
- An **Orchestration Job**, to manage and control the workflow
- For Advanced Mode, you must be familiar with JavaScript expressions

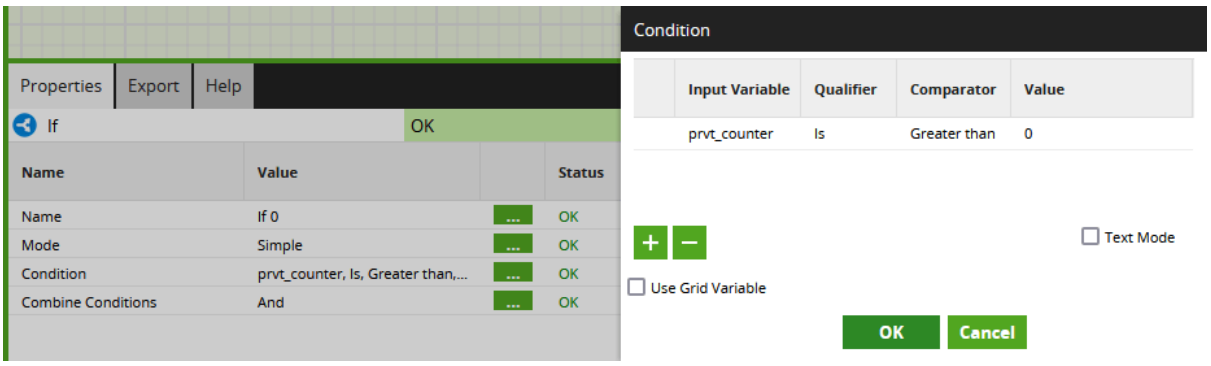
Conditional branching needs to be dynamic and data-driven, so you will almost always **use variables** rather than hardcoded values.

How to use Matillion variables in Simple Mode

An If component can reference **Environment variables** and **Job variables**, including **automatic variables** such as environment_name. Use Private Visibility Job Variables as a first preference. They are limited in scope to one job only, which reduces their footprint in the global variable namespace.

Wherever possible, use Private Visibility Job Variables with If components

Variables are referenced by name in Simple Mode conditions. The $ or ${} syntax is not needed.



Name	Value		Status
Name	If 0	...	OK
Mode	Simple	...	OK
Condition	prvt_counter, Is, Greater than,...	...	OK
Combine Conditions	And	...	OK

Input Variable	Qualifier	Comparator	Value
prvt_counter	Is	Greater than	0

The best Orchestration Job design pattern is like this:

1. Initial tasks
2. Set or Export a variable
3. Check the variable with an If component
4. If True, launch further tasks
5. If False or Failure then perform logging or error handling

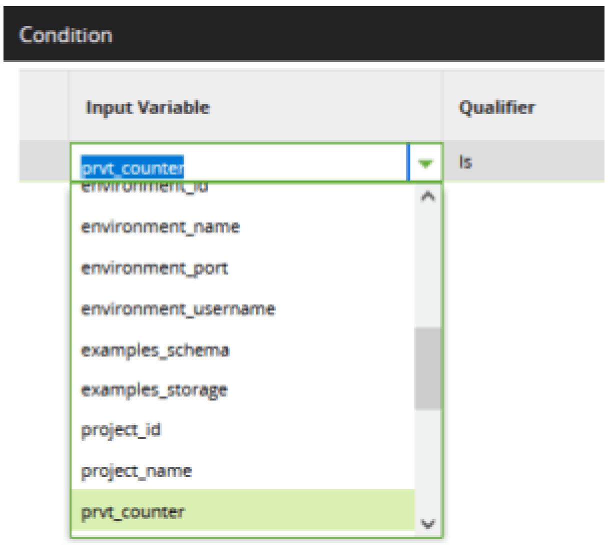
Variables can optionally be defined with a default value. You can leave the default value blank, but that can cause confusion. For instance, comparing an uninitiated numeric against any other value always returns false.

Whenever a Matillion job starts to run, all the variables begin with their default values, regardless of how they finished in previous job runs. For this reason, it is best to choose a default value that makes the job follow the "False" branch.

Give variables a default value that will make the If component evaluate to False.

Check this using Run From Component on the If component.

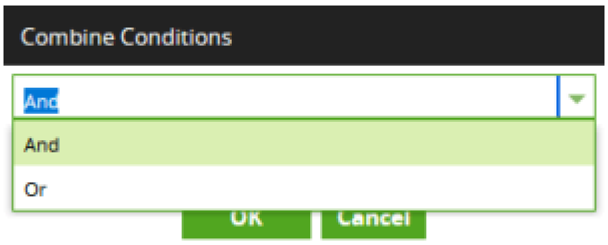
It is best to pick variables using the Simple Mode condition editor, rather than typing their name manually:



When setting up a variable, be careful to choose a data type that is appropriate for the comparison you intend to make. For instance, Text variables always use string comparisons. So if you use a Text variable to store a row count, then – confusingly – 589 will be greater than 10000, and 2000 will be less than 9.

Choose the correct data type for variables

In Simple Mode, you can combine multiple conditions using either “And” or “Or”.



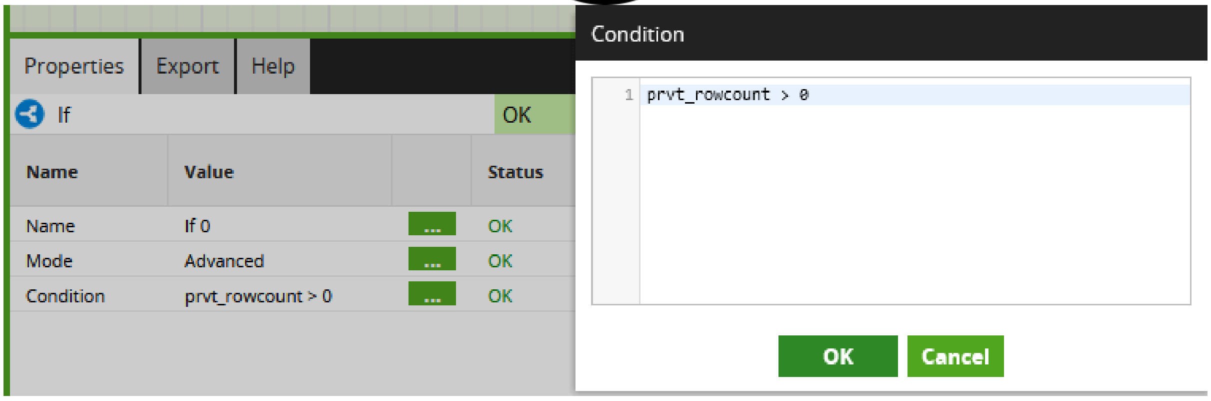
If you need more sophisticated logic, you can switch to Advanced Mode instead.

How to use JavaScript expressions in Advanced Mode

When an If component’s Mode property is set to Advanced, you define the condition as a **JavaScript expression**.

Within the expression, all Matillion Environment variables and Job variables are available as JavaScript variables. Once again the $ or ${} syntax is not needed.

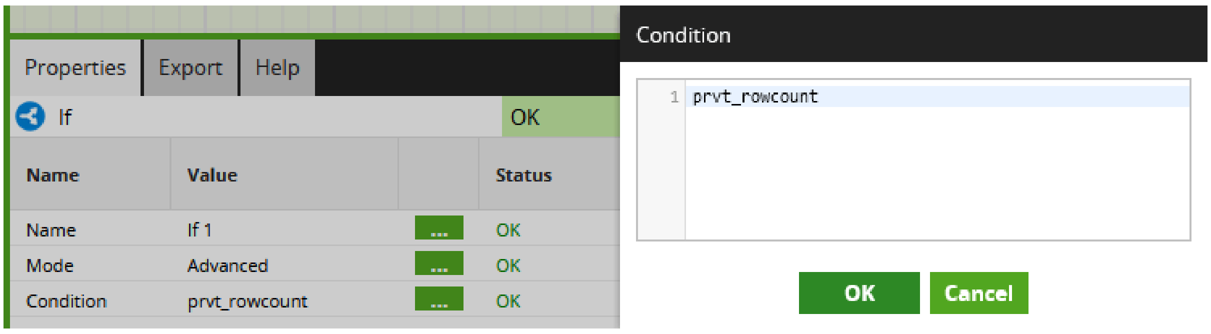
It is simplest to use a comparison operator in the expression, since these always return true or false. The example below is equivalent to the simple mode condition shown earlier:



Name	Value		Status
Name	If 0	...	OK
Mode	Advanced	...	OK
Condition	prvt_rowcount > 0	...	OK

```
prvt_rowcount > 0
```

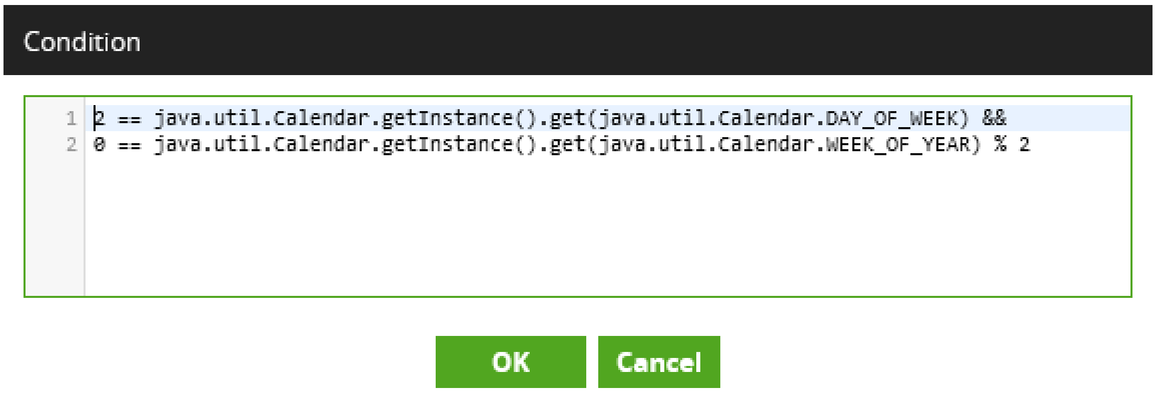
The result of the expression is always coerced into a boolean. So any non-zero numeric and any non-empty string evaluates as “true.” Building upon that concept, the next example below checks that the prvt_rowcount variable is not zero:



Name	Value		Status
Name	If 1	...	OK
Mode	Advanced	...	OK
Condition	prvt_rowcount	...	OK

```
prvt_rowcount
```

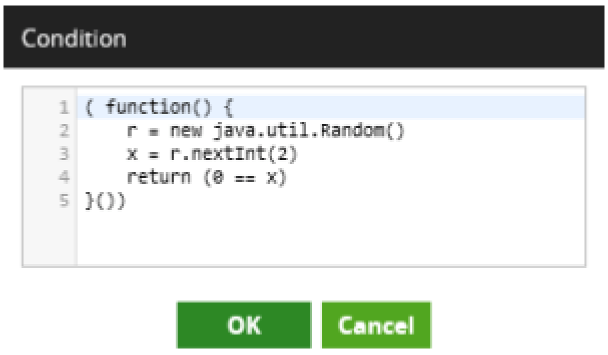
At runtime, the expression is evaluated by the built-in **Nashorn engine**, so you are free to create much more sophisticated expressions. The next example uses the **Java Calendar class** and only evaluates true if it is currently Tuesday in an even-numbered week:



```
2 == java.util.Calendar.getInstance().get(java.util.Calendar.DAY_OF_WEEK) &&
0 == java.util.Calendar.getInstance().get(java.util.Calendar.WEEK_OF_YEAR) % 2
```

This is a good way to implement custom schedules. Have the job run every day, but only actually do the work if the condition is met.

Long JavaScript expressions can be hard to maintain. As an alternative to one single expression, you can switch to using an **immediately invoked function expression** (IIFE) and write actual JavaScript statements. The last example below is an IIFE that acts as a load balancer. It will randomly return either true or false, with a probability of 50 percent each time:



```
( function() {
    r = new java.util.Random()
    x = r.nextInt(2)
    return (0 == x)
}())
```

In Advanced Mode, the condition editor has no syntax checking. If you misspell a variable, or get the JavaScript syntax wrong, the evaluation will hit a generic "Failed to evaluate expression" error at runtime. To find the underlying error you will need to check the **catalina log file**.

Conditional branching in Transformation Jobs

Matillion ETL Transformation Jobs differ fundamentally from Orchestration Jobs in two main respects:

- All of the components in a Transformation Job run at the same time
- All of the input records are processed at the same time, in one single batch-mode pushdown SQL operation

In a Matillion ETL Transformation Job, conditional branching is easy to do at record level, although it is achieved in a different way – using complementary filters. Replicate the incoming records into **Filter components** that have opposite logic from each other. At runtime, every record will pass exactly one of the filters.