



Informatica®

Informatica® Data Integration - Free & PayGo  
November 2024

# Taskflows

Informatica Data Integration - Free & PayGo Taskflows  
November 2024

© Copyright Informatica LLC 2022, 2024

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

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

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

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

The information in this documentation is subject to change without notice. If you find any problems in this documentation, report them to us at [infa\\_documentation@informatica.com](mailto:infa_documentation@informatica.com).

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

Publication Date: 2024-11-15

# Table of Contents

<b>Preface</b> .....	<b>6</b>
Informatica Resources. ....	6
Informatica Documentation. ....	6
Informatica Intelligent Cloud Services web site. ....	6
Informatica Intelligent Cloud Services Communities. ....	6
Informatica Intelligent Cloud Services Marketplace. ....	6
Data Integration connector documentation. ....	7
Informatica Knowledge Base. ....	7
Informatica Intelligent Cloud Services Trust Center. ....	7
Informatica Global Customer Support. ....	7
<b>Chapter 1: Taskflows</b> .....	<b>8</b>
<b>Chapter 2: Taskflow steps</b> .....	<b>9</b>
<b>Chapter 3: Creating a taskflow</b> .....	<b>11</b>
Taskflow templates. ....	11
Basic. ....	12
Parallel Tasks. ....	12
Parallel Tasks with Decision. ....	13
Sequential Tasks. ....	13
Sequential Tasks with Decision. ....	13
Single Task. ....	13
<b>Chapter 4: Setting taskflow properties</b> .....	<b>14</b>
General properties. ....	14
Input fields. ....	15
Output fields. ....	16
Temporary fields. ....	17
Advanced properties. ....	18
Notes. ....	19
<b>Chapter 5: Setting taskflow step properties</b> .....	<b>20</b>
Assignment step. ....	20
Data Task step. ....	20
Notification Task step. ....	25
Email notification examples. ....	27
Rules and guidelines for Notification Task step. ....	28
Command Task step. ....	31
Subtaskflow step. ....	38

Decision step. . . . .	40
Parallel Paths step. . . . .	42
Jump step. . . . .	43
End step. . . . .	43
Wait step. . . . .	44
Throw step. . . . .	44
Throw step properties . . . . .	46
<b>Chapter 6: Runtime parameters. . . . .</b>	<b>48</b>
Parameters in taskflows. . . . .	48
Overriding parameters in a taskflow. . . . .	49
Guidelines and best practices for using parameters in a taskflow. . . . .	51
Example: Overriding parameters with a Data Task step. . . . .	51
<b>Chapter 7: The Expression Editor. . . . .</b>	<b>54</b>
Tips: Using XQuery 3.0 to create expressions. . . . .	56
Keyboard shortcuts. . . . .	57
<b>Chapter 8: Taskflow functions. . . . .</b>	<b>58</b>
addToDate. . . . .	59
base64Decode. . . . .	62
dateDiff. . . . .	62
decode. . . . .	65
getAssetLocation. . . . .	68
getAssetName. . . . .	68
getDatePart. . . . .	69
getInstanceStartTime. . . . .	70
iif. . . . .	71
in. . . . .	73
instr. . . . .	74
isNull. . . . .	77
lastDay. . . . .	77
lpad. . . . .	79
ltrim. . . . .	80
round (Numbers). . . . .	82
rtrim. . . . .	84
toChar (Numbers). . . . .	86
toDate. . . . .	88
toDecimal. . . . .	91
toInteger. . . . .	92
trunc (Dates). . . . .	94
<b>Chapter 9: Using the Validation panel. . . . .</b>	<b>97</b>

<b>Chapter 10: Running a taskflow.....</b>	<b>99</b>
Running a taskflow from the taskflow designer . . . . .	99
Publishing a taskflow. . . . .	99
Scheduling a taskflow. . . . .	100
<b>Chapter 11: Taskflow example.....</b>	<b>102</b>
<b>Index.....</b>	<b>104</b>

# Preface

Use *Taskflows* to learn how to set up dynamic taskflows to run Data Integration tasks in a specific order at the designated time.

## 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](mailto: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](mailto: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>.

# CHAPTER 1

## Taskflows

Use a taskflow to control the execution sequence of a data transfer task or mapping task based on the output of the previous task.

For example, you want to run two mapping tasks in sequence. However, you do not want to run the second mapping task as soon as the first task ends if you get a warning on the first mapping task. Instead, you want to run the second mapping task to run after two hours. You can create a taskflow to orchestrate this scenario.

**Note:** Before you create a taskflow, you must have the task that you want to use ready. You cannot create a task during the taskflow creation process.

You can invoke and run a taskflow from the taskflow designer. You can also run a taskflow on a schedule.



## CHAPTER 2

# Taskflow steps

Use taskflow steps to add and orchestrate data integration tasks. You can add different types of steps to a taskflow. To add a step to a taskflow, drag a step from the palette on the left.

You can add the following steps to a taskflow:

### **Assignment**

Use an Assignment step to set a value for a field. A field is a data holder that carries data around a taskflow. You can use input fields and temporary fields to set a value for a field.

Input fields provide input when you run the taskflow. The taskflow uses temporary fields internally to handle data.

### **Data Task**

Use a Data Task step to add a data transfer task or mapping task to a taskflow. You can configure how the taskflow handles errors and warnings, perform actions based on a schedule, and override runtime parameters.

### **Notification Task**

Use a Notification Task step to send a notification to specified recipients.

You can configure the Notification Task step to send an email notification. For example, you can send an email notification to inform recipients about the number of success rows and error rows that were encountered in a Data Task step of a taskflow.

### **Command Task**

Use a Command Task step to run shell scripts or batch commands from multiple script files on the Secure Agent machine. For example, you can use a command task to move a file, copy a file, zip or unzip a file, or run clean scripts or SQL scripts as part of a taskflow.

You can use the Command Task outputs to orchestrate subsequent tasks in the taskflow.

### **Subtaskflow**

Use a Subtaskflow step to embed and reuse an existing taskflow. You can configure input fields to provide input when you run the taskflow. You can also enable fault handling to determine the reason for a taskflow failure.

### **Decision**

Use a Decision step when you want a taskflow to take different paths based on the value of a specific field or formula.

### **Parallel Paths**

Use a Parallel Paths step when you want a taskflow to run multiple items at the same time. For example, you can run three mapping tasks simultaneously. The taskflow runs all items in the Parallel Paths step and then moves to the next step.

**Jump**

Use a Jump step when you want to jump from one part of the taskflow to another.

**Throw**

Use a Throw step to catch a fault, return the fault details, and prevent the execution of the subsequent steps in a taskflow. The Throw step is an interrupting step, which means that if a fault occurs, the Throw step stops the execution of the taskflow and sets the taskflow status to failed.

**Wait**

Use a Wait step when you want to pause taskflow execution for a specific duration.

**End**

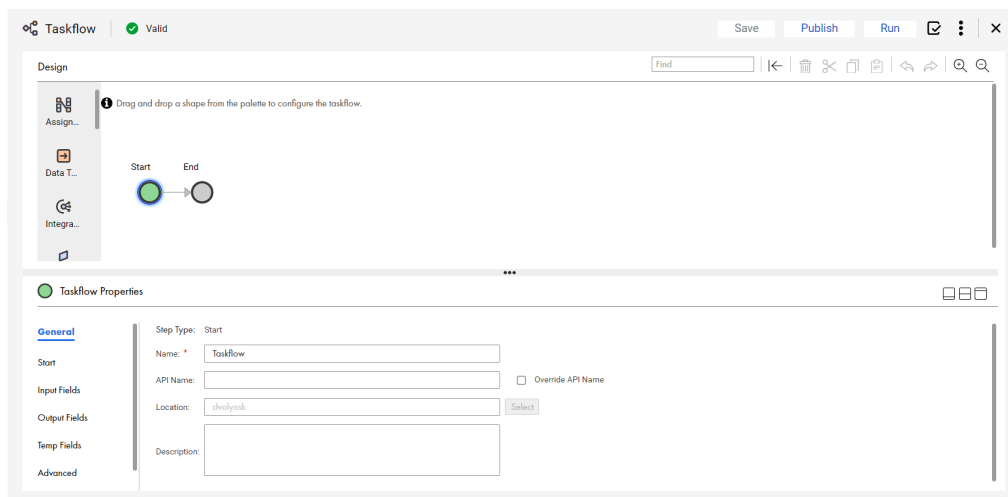
Use an End step to define the HTTP status code that must be used when a taskflow completes.

## CHAPTER 3

# Creating a taskflow

Create a taskflow from scratch or use a taskflow template.

1. In Data Integration, click **New > Taskflows > Taskflow > Create**.



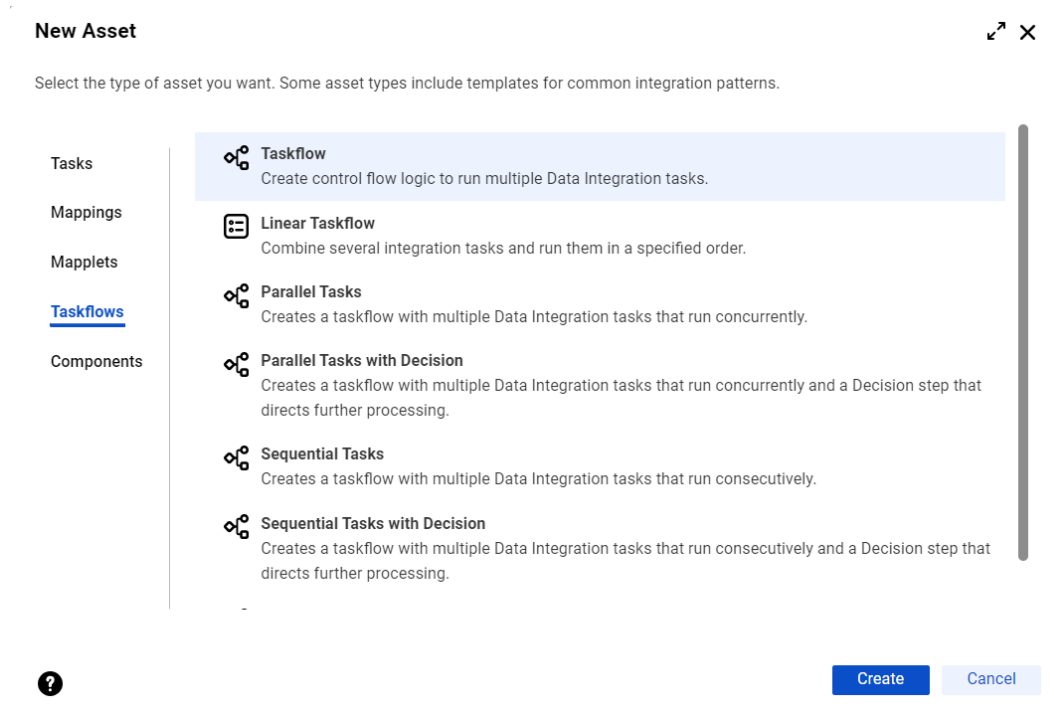
2. Set general properties, input fields, and temporary fields for the taskflow. Optionally, add notes.
3. Add steps to the taskflow.  
For example, you can use the Data Task step to add a data transfer task or mapping task. You can use the Subtaskflow step to embed and reuse an existing taskflow.
4. Validate and save the taskflow. You cannot run a taskflow that is not valid.

## Taskflow templates

Use a taskflow template instead of creating a taskflow from scratch. Select from pre-created templates based on your business needs.

Click **New > Taskflows** to open the **New Asset** dialog box.

The following image shows the **New Asset** dialog box:



When you select a taskflow template in the **New Asset** dialog box, the Data Integration page opens with a copy of the taskflow template.

For example, if you want to run multiple tasks consecutively, use a Sequential Tasks template. After the consecutive tasks, if you then need the taskflow to pause for a time and then take different paths, add a Wait step and a Parallel Paths step to the template.

When you modify a taskflow template, you only modify that instance. The template remains unchanged.

## Basic

Select **Taskflow** when you need a basic canvas with a Start step and an End step. You can create and configure a taskflow that suits your needs.

## Parallel Tasks

Select the **Parallel Tasks** template if your major requirement is to run two or more data integration tasks in parallel. You start with a taskflow that contains a Start step, a Parallel Paths step, and an End step.

You can add other steps at any point on the canvas.

## Parallel Tasks with Decision

Select the **Parallel Tasks with Decision** template if your major requirement is to run two or more data integration tasks in parallel and then make a decision based on the outcome of any task. You start with a taskflow that contains a Start step, a Parallel Paths step, a Decision step, and an End step.

## Sequential Tasks

Select the **Sequential Tasks** template if your major requirement is to run two data integration tasks, one after the other. You start with a taskflow that contains a Start step, two Data Task steps, and an End step.

## Sequential Tasks with Decision

Select the **Sequential Tasks with Decision** template if your major requirement is to run two data integration consecutive tasks and then make a decision based on the output of either task.

You start with a taskflow that contains a Start step, two Data Task steps, a Decision step, and an End step.

## Single Task

Select the **Single Task** template if your major requirement is to run one data integration task on a daily or weekly schedule, for example. You start with a taskflow that contains a Start step, a Data Task step, and an End step.

## CHAPTER 4

# Setting taskflow properties

Taskflow properties define the name and location of the taskflow, the fields that the taskflow uses, and other information required to invoke and run the taskflow.

To set taskflow properties, create a taskflow, click the Start step and access the **Properties** section. Optionally, click the empty area of the canvas to access the **Properties** section.

Define the general properties, taskflow binding and access details, input fields, output fields, temporary fields, advanced properties, and notes for a taskflow.

## General properties

You can specify the following general properties for a taskflow:

Property	Description
Name	Required. A descriptive name to identify the taskflow. Taskflows saved in different folders can have the same name. The name can be up to 255 characters and can't contain the keyword <b>schema</b> as a prefix. To change the name of a published taskflow, you must first unpublish the taskflow. Then, change the taskflow name and republish the taskflow.
Override API Name	Don't select this option. This option is not used.
API Name	Don't enter an API name. This property is not used.
Location	The project and folder in which you want to save the taskflow. Click <b>Select</b> to navigate to a folder. You must create a project and folder before you create a taskflow. You cannot create a project or folder from the taskflow creation page. If the <b>Explore</b> page is currently active and a project or folder is selected, the default location for the asset is the selected project or folder. Otherwise, the default location is the location of the most recently saved asset.
Description	A description of the taskflow.

# Input fields

Use the **Input Fields** section to add fields that a taskflow uses at the beginning of a step.

You can define the input fields that you want to pass when you run a taskflow. You can create input fields of the following types:

## Simple type

Create a simple type field to use common data types such as Checkbox, Date, Date Time, Time, Number, Integer, or Text.

**Note:** You can't use a list of simple type in taskflows.

## Custom type

Create a custom type field to use an object that is added to the taskflow.

Enter the following properties for each input field that you create:

Property	Description
Name	The name of the input field.
Type	The type of the input field. For example, select <b>Checkbox, Date, Date Time, Time, Number, Integer, or Text</b> . You can also add an input field of a simple type or custom type. To do this, select <b>More types</b> , and then in the <b>Edit Type</b> dialog box, select the <b>Category</b> as <b>Simple Types</b> or <b>Custom Types</b> .
Description	A description of the input field.
Required	Indicates whether the input field is required for the taskflow to run.
Initial Value	The value that is assigned to the input field when you start the taskflow execution. You can enter text or use a formula to set the initial value.

## Creating an input field with a custom type

You can create an input field with a custom type using the objects available in tasks in the taskflow. To create an input field with a custom type, you must have one or more steps that contain data integration tasks added to the taskflow.

1. On the **Explore** page, navigate to the taskflow for which you want to create an input field with a custom type.
2. Add steps to the taskflow. For example, add a Data Task step.
3. Click the **Start** step of the taskflow.
4. Click the **Input Fields** tab.
5. Click the **Add** icon.
6. In the **Type** column, select **More types**.  
The **Edit Type** dialog box appears.

The following image shows the **Edit Type** dialog box:

Edit Type



Category: \*

---

Types

---

Data Marketplace Parameters  
 Data Marketplace Path Variables  
 Data Marketplace Query Parameters  
 Runtime Parameter  
▶ Default

Allow a list of objects of this type.

7. From the **Category** list, select **Custom Types**.  
A list of objects available in the taskflow appears in the **Types** section.
8. Select the object that you want to pass as an input and click **OK**.

## Output fields

Use the **Output Fields** section to add output fields to a taskflow.

You can use the output fields of a subtaskflow as variables in the subsequent steps of the parent taskflow. However, if the subtaskflow fails, the output field values of the subtaskflow are not sent to the parent taskflow. When you run the parent taskflow, you can view the output fields of the Subtaskflow step on the **My Jobs** page in Data Integration, and the **All Jobs** page and **Running Jobs** page in Monitor.

You can create output fields of the following types:

### Simple type

Create a simple type field to use common data types such as Checkbox, Date, Date Time, Time, Number, Integer, or Text.

**Note:** You can't use a list of simple type in taskflows.

### Custom type

Create a custom type field to use an object that is added to the taskflow.



Enter the following properties for each output field that you create:

Property	Description
Name	The name of the output field.
Type	The type of the output field. For example, select <b>Checkbox, Date, Date Time, Time, Number, Integer, or Text</b> . You can also add an output field of a simple type or custom type. To do this, select <b>More types</b> , and then in the <b>Edit Type</b> dialog box, select the <b>Category</b> as <b>Simple Types</b> or <b>Custom Types</b> .
Description	A description of the output field.
Initial Value	The initial value for the output field.

## Temporary fields

Create temporary fields for use in a taskflow step. A taskflow uses temporary fields internally. Temporary fields do not appear in the input or output of a taskflow.

For example, you might define temporary fields if the taskflow has a Decision step.

Define the following properties for each temporary field:

Property	Description
Name	The name of the temporary field.
Type	The type of the temporary field. For example, select <b>Checkbox, Date, Date Time, Time, Number, Integer, or Text</b> . You can also add a temporary field of a simple type or custom type. To do this, select <b>More types</b> , and then in the <b>Edit Type</b> dialog box, select the <b>Category</b> as <b>Simple Types</b> or <b>Custom Types</b> . <b>Note:</b> You can't use a list of simple type in taskflows.
Description	A description of the temporary field.
Initial Value	The value that is assigned to the temporary field when you start the taskflow execution. The value can change during the taskflow run. You can use text or a formula to set the initial value for a temporary field.

Some temporary fields appear without you specifically adding them. When you add a task to a taskflow, a corresponding temporary field appears. The **Name** of the temporary field is the name of the Data Task step. The **Type** of the temporary field is the name of the task you add to the Data Task step. Enter a **Description** and select **Required**, if needed.

If you included a Data Task step in a taskflow, the Data Task fields appear on the **Temp Fields** tab of the Start step. The Data Task fields represent the input parameters of the task.

# Advanced properties

You can configure a taskflow to disable concurrent execution. You can also configure a taskflow to suspend on a fault at the taskflow level and send an email notification to specified recipients when it is suspended.

You can define the following advanced properties for a taskflow:

Property	Description
Concurrent taskflow execution	Enables or disables concurrent taskflow execution based on the option that you select from the following list: <ul style="list-style-type: none"><li>- <b>Allow concurrent execution (Default)</b>. When you select this option, the taskflow execution is enabled so that multiple instances of the taskflow can run at the same time. By default, concurrent execution is enabled.</li><li>- <b>Disable concurrent execution</b>. When you select this option, the taskflow execution is disabled. When you select this option, the taskflow does not create a new instance when another instance is still running.</li></ul> <b>Note:</b> This option is not applicable if you use the taskflow in a Subtaskflow step.
Suspend on Fault	Suspends the taskflow on a fault that occurs at the taskflow level. The <b>Suspend on Fault</b> property takes precedence over the following properties that you define in a Data Task step: <ul style="list-style-type: none"><li>- <b>Fail taskflow on completion - if this task fails</b></li><li>- <b>Fail taskflow on completion - if this task does not run</b></li></ul>
Send Email on Suspension	Sends an email notification when the taskflow is suspended on a fault. When you select the <b>Send Email on Suspension</b> option, the <b>Email To</b> , <b>Email Cc</b> , <b>Email Subject</b> , and <b>Email Body</b> fields become available. You can parameterize these fields by using the parameter set, input fields, output fields, and temporary fields. For example, when you move assets from one Point of Deployment (POD) to another, instead of manually entering the recipient Cc email addresses, you can parameterize them to save time.
Email To	Defines the primary recipients for the email notification. Enter one or more valid recipient email addresses. You can also add fields that contain valid email addresses. Separate email addresses and fields with a comma (,) or a semicolon (;). Required if you configure the taskflow to send an email notification on suspension.
Email Cc	Defines the recipients who need to be sent a copy of the email notification. Enter one or more valid recipient email addresses. Separate email addresses with a comma (,) or a semicolon (;).
Email Subject	Specifies a short and descriptive subject that introduces the email.
Email Body	Defines the content that you want to send in the email. Click <b>Edit Content</b> to open a rich text editor and use formatting options such as bold, italics, underlines, lists, indentations, and fonts. You can also insert tables and links.

## Concurrent taskflow execution scenarios

Consider the following scenarios when you configure the **Allow concurrent execution (Default)** option for a taskflow:

- Before you switch from the **Allow concurrent execution (Default)** option to the **Disable concurrent execution** option or vice versa, you must unpublish the taskflow and ensure that the previous taskflow instance is executed. You can then switch to another option in the **Concurrent taskflow execution** list and publish the taskflow.
- When a previous taskflow instance is in a suspended state, even if you run the taskflow again, Data Integration does not run the taskflow instance.

- When a taskflow instance is running, you can run a previous taskflow instance from a step that ran successfully using the **Run from here** option on the **My Jobs** page in Data Integration, and the **All Jobs** and **Running Jobs** page in Monitor.
- The parent taskflow and subtaskflow run independently. For example, you have a taskflow that contains a subtaskflow with the **Allow concurrent execution (Default)** option enabled for both the taskflows. When you run the parent taskflow, the child taskflow runs as a subtask. When you run a taskflow or subtaskflow independently, it runs for the first time without any issue. However, if you run the same taskflow or subtaskflow again while the previous instance is still running, an error occurs.
- When you run a taskflow using an endpoint, if the previous instance of the taskflow is still running, an error occurs. However, if you run the same taskflow several times from the taskflow designer, you see a success message. This is because when you run a taskflow, Data Integration receives a request and sends an immediate response as `200 OK`. Data Integration processes the taskflow run request later. This is for a better user experience because taskflows might take time to run depending on their complexity.

## Notes

Use the **Notes** field to add information that you or other users might need.

The notes that you enter here appear on the Data Integration page. You do not see these notes when you run the taskflow.

For example, use the **Notes** field to add a reminder about a task that you need to complete before you run the taskflow.

## CHAPTER 5

# Setting taskflow step properties

When you add a step to a taskflow, you set properties associated with each step.

**Note:** When you set a value for a field in any taskflow step, you can't enter curly brackets {}.

## Assignment step

When you add an Assignment step, you can configure the following properties:

### Name

The name of the Assignment step. The name can contain only alphanumeric characters, underscores (\_), spaces, and Unicode characters. The name can't contain curly brackets {}.

You select a field name from the list of fields you defined under **Start > Fields**.

### Assignments

The source from which the field takes values. The fields that you see depend on the data type you defined for the **Start > Properties > Input Fields or Temporary Fields**.

For example, you define the data type in the Input Fields as **Date Time**. The following **Value** options appear for that field in the Assignment step:

- Specific date
- Time from now
- Days from today
- Days before/after
- Field
- Formula

You can override runtime parameters in the Assignment step.

For information about runtime parameters, see [Runtime Parameters in Taskflows on page 48](#).

## Data Task step

When you add a Data Task step, you set some properties.

The following sections describes the Data Task step properties:

## General

In the general properties, you can specify a descriptive name for the Data Task step.

The name can contain only alphanumeric characters, underscores (\_), spaces, and Unicode characters. The name can't contain curly brackets {}, +, dot (.), or comma (,).

## Data Task

In the Data Task step properties, select the task from a list of existing tasks that you want to add to the taskflow.

**Note:** You must have an existing task to add to a taskflow. You cannot create a task during the taskflow creation process.

When you add a mapping task to a Data Task step, you see a description, input fields, and output fields. The input fields show the in-out parameters that the mapping task uses.

The output fields show the output fields that the mapping task returns after the taskflow runs.

When you click the **Add** icon on the Data Task step, you see one of the following views:

- If the Data Task step contains a task, a non-editable view of the task opens.
- If the Data Task step does not contain a task, you see a dialog box from which you can choose a task.

When you add a task to a Data Task step, a corresponding taskflow temporary field of type Text is created. When you add a task to the Data Task step, the temporary field type is the name of the task. See [“Temporary fields” on page 17](#) for details.

## Input Fields

The **Input Fields** section appears when you add a task to the taskflow.

In the **Max Wait (Seconds)** field, you can configure the maximum length of time in seconds that the Data Task step waits for the data integration task to complete. Specify a value between 1 and 604800 seconds. Default is 604800 seconds, which is 7 days. If the task is not completed within the maximum time specified in the field, the task stops running and the subsequent task in the taskflow starts running.

**Note:** If the specified value is lesser than 1 or greater than 604800, the maximum wait time is automatically set to 604800 seconds.

If the task contains parameters that you can override, you can add input fields. You can set properties for an input field to override Data Integration runtime parameters. For information about runtime parameters, see [“Overriding parameters or parameter files in a Data Task step” on page 49](#).

If the Data Task step uses a mapping task, you can override the values of input parameters and in-out parameters of the task.

If the Data Task step uses a mapping task, you can perform the following override actions:

- If the mapping task contains a parameter file available on the Secure Agent machine, you can override the parameter file directory and parameter file name.
- If the mapping task contains a parameter file available in a cloud-hosted repository, you can override the parameter file connection and parameter file object. Data Integration supports only the Amazon S3 V2, Azure Data Lake Store Gen2, and Google Storage V2 connection types for mapping tasks.
- If the mapping task uses data formatting options, you can override the data formatting and default precision values of the source data. These options are available in the input fields only if the formatting file is uploaded to the mapping task and not to the mapping. The precision value set in the default precision field takes precedence over the precision set in the data format field or the mapping task. The default precision value is applied to all the columns in the formatting file.

- If the mapping task contains a Lookup transformation, you can override the values of the lookup object and lookup condition.

**Note:** You cannot override the value of an input parameter of type string or text from the parameter file. However, you can override the input parameter value from the taskflow. You can override the connection parameter values from the parameter file.

## Output Fields

The **Output Fields** section is an exhaustive list of output data fields that appear when the task runs.

The following image shows the Output fields you see:

Output Fields:

Name	Type
Run Id	Integer
Task Status	Text
Success Source Rows	Integer
Failed Source Rows	Integer
Success Target Rows	Integer
Failed Target Rows	Integer
Start Time	Date Time
End Time	Date Time
Error Message	Text

If you use a data transfer task, you see the following fields:

### Output Fields

Name	Type
Run ID	Int
Status	String
Success Rows	Int
Failed Rows	Int
Error Message	String

To view the values of each output field, run the taskflow and go to the **Taskflow Instance Detail** page. For more information about the **Taskflow Instance Detail** page, see *Monitor*.

You can use output fields in a Data Decision or Assignment step.

For example, create a temporary field with value Formula and use the following expression to assign data to the field:

```
if (
  ($temp.DataTask1[1]/output[1]/Failed_Target_Rows < 0 or
  $temp.DataTask1[1]/output[1]/Task_Status = '1')
  and
  ($temp.DataTask2[1]/output[1]/Success_Target_Rows > 0
  and $temp.DataTask2[1]/output[1]/Failed_Target_Rows = 0)
  and $temp.DataTask3[1]/output[1]/Success_Target_Rows > 0)
then 'Pass'
else 'Fail'
```

When you use the temporary field in a Decision step, the taskflow takes the `Pass` path if the following conditions are met:

- Data Task 1 has no failed target rows or Data Task 1 runs successfully.
- Data Task 2 has at least one successful target row.
- Data Task 2 has zero failed target rows.

- Data Task 3 has at least one successful target row.

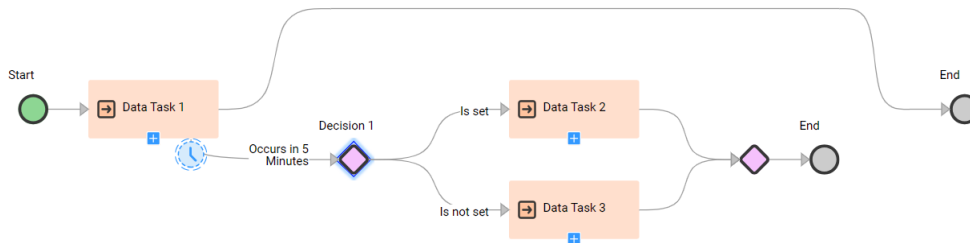
## Timer Events

Enter the following **Events** properties to add timers to a task:

Use a Timer event to perform an action based on a schedule. The action could be either at a specific time or after an interval.

When you add a timer to a Data Task step, a new branch appears. Add an event to this branch and specify whether you want the event to run **At** a specific time or **After** an interval.

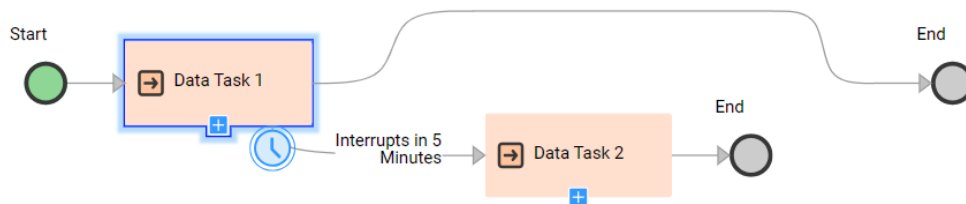
In the following image, the event on the timer branch, a Data Decision step, occurs five minutes after the main data task:



When a timer fires, the taskflow always runs through the entire timer branch. If Data Task 1 finishes before Decision 1, the timer branch is not executed.

Select **Interrupting** if you want the timer to interrupt the main data task. When you set an interrupting timer, the main data task is interrupted and the taskflow only runs the event on the timer set.

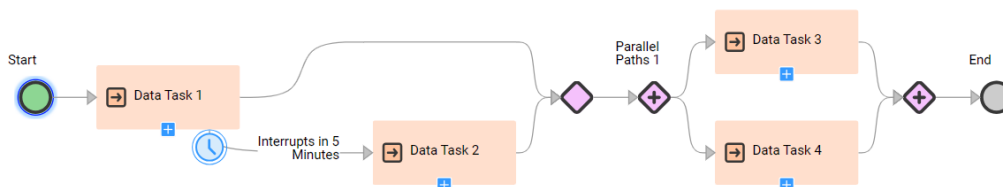
The following image shows an interrupting timer set to occur five minutes after the main data task starts:



When the event on the timer branch, Data Task 2, executes, Data Task 1 is interrupted. The taskflow follows the timer branch. That is, the taskflow runs Data Task 2 and then ends.

If you delete the End step on the timer branch of an interrupting timer, the timer branch rejoins the main branch.

The following image shows an interrupting timer branch with the End step deleted:



The timer event, Data Task 2, executes after 5 minutes and interrupts Data Task 1. The timer branch rejoins the main branch. The taskflow executes Data Task 2, a Parallel Paths step, and then ends.

If you use an interrupting timer, the main data task has no output with respect to this taskflow instance. You see no output fields for the main data task in the job details for the taskflow.

If a Data Task step completes before a timer, interrupting or non interrupting, fires no timer will fire for that Data Task step.

**Note:** When you run a particular step in a timer branch of a taskflow instance, the steps in the alternate branch also get executed. To avoid this issue, add a dummy step after the step that you would like to run.

## Error Handling

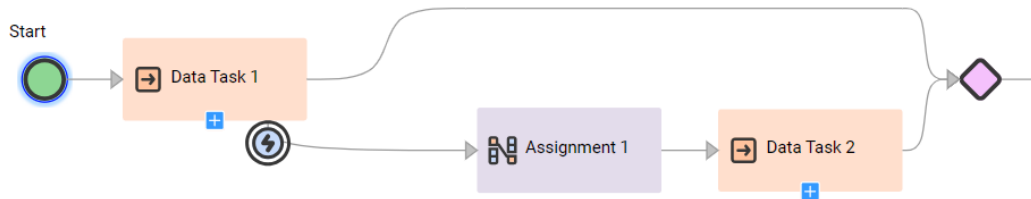
Use the Error Handling section to indicate how you want the taskflow to behave when a Data Task step encounters a warning or an error. You can also configure the taskflow behavior when the task associated with a Data Task step fails or does not run.

After you select a task, enter the following error handling properties:

Property	Description
On Warning	<p>The path that a taskflow takes when it encounters a warning in a Data Task step.</p> <p>A warning occurs when a Data Task step completes incorrectly or incompletely. For example, you see a warning if the Data Task step copies only 20 out of 25 rows from table A to table B.</p> <p>You can choose from the following options:</p> <ul style="list-style-type: none"> <li>- Select <b>Ignore</b> to ignore the warning and move to the next step.</li> </ul> <p><b>Note:</b> If you select <b>Ignore</b> for a Data Task step with a subsequent Notification Task step and the data task fails, the email notification that you receive does not contain the fault details. To get the fault details in the email, select <b>Custom error handling</b>.</p> <ul style="list-style-type: none"> <li>- Select <b>Suspend Taskflow</b> to move the taskflow to the suspended state when it encounters a warning. You can resume the taskflow instance from the <b>All Jobs, Running Jobs, or My Jobs</b> page. The taskflow resumes from the step at which it was suspended. If you know the reason for the warning, correct the issue and then resume the taskflow.</li> </ul> <p>Default: <b>Ignore</b></p>
On Error	<p>The path that a taskflow takes when it encounters an error in a Data Task step.</p> <p>An error occurs when a Data Task step fails. For example, you see an error if the Data Task does not copy table A to table B.</p> <p>You can choose from the following options:</p> <ul style="list-style-type: none"> <li>- Select <b>Ignore</b> to ignore the error and move to the next step.</li> <li>- Select <b>Suspend Taskflow</b> to move the taskflow to the suspended state when it encounters an error. You can resume the taskflow instance from the <b>All Jobs, Running Jobs, or My Jobs</b> page. The taskflow resumes from the step at which it was suspended. If you know the reason for the error, correct the issue and then resume the taskflow.</li> <li>- Select <b>Custom error handling</b> to handle the error in a manner you choose. If you select <b>Custom error handling</b>, two branches appear. The first branch is the path the taskflow follows if no error occurs. The second branch is the custom path the taskflow follows if an error occurs.</li> </ul> <p>Default: <b>Suspend Taskflow</b></p>
Fail taskflow on completion	<p>The taskflow behavior when the task associated with the Data Task step fails or does not run.</p> <p>You can configure a taskflow to fail on its completion if the task associated with the Data Task step fails or does not run. If the task fails or does not run, the taskflow continues running the subsequent steps. However, after the taskflow completes, the taskflow status is set to failed.</p> <p><b>Note:</b> If you configure both the <b>Suspend on Fault</b> taskflow advanced property and the <b>Fail taskflow on completion</b> property, the <b>Suspend on Fault</b> property takes precedence. In this case, if the task associated with the Data Task step fails or does not run, the taskflow is suspended. The taskflow does not run the subsequent steps after the Data Task step.</p>



The following image shows a **Custom error handling** path with an Assignment step and another Data Task step:



## Notification Task step

A Notification Task step sends a notification to specified recipients.

You can configure the Notification Task step to send an email notification. For example, you can send an email notification to inform recipients about the number of success rows and error rows that were encountered in a Data Task step of a taskflow.

You can define properties for the Notification Task step to configure the email recipients and email content. You cannot use distribution lists in the **Email To**, **Email Cc**, and **Email Bcc** fields.

The following sections describe the Notification Task step properties:

### General properties

In the general properties, you can specify a descriptive name for the Notification Task step.

The name can contain only alphanumeric characters, underscores (`_`), spaces, and Unicode characters. The name can't contain curly brackets `{}`.

### Notification Task properties

You can configure the following Notification Task step properties:

#### Notification Method

The type of notification to be sent.

The value of this field is set to **Email** by default. You cannot edit this value. The other values are reserved for future use.

#### Email To

Required. The primary recipients for the email notification.

Use one of the following options to specify the value for this field:

- **Content.** Enter one or more valid recipient email addresses. You can also add fields that contain valid email addresses. Separate email addresses and fields with a comma (,) or a semicolon (;).
- **Field.** Select the field that the Taskflow Designer uses to write the email addresses into this field when this step executes. You can select an input field or a temporary field that was added in any other step in the taskflow.
- **Formula.** Open the formula editor to specify a formula that calculates the value for this field.

Default is **Content**.

### Email Cc

The recipients who need to be sent a copy of the email notification.

Use one of the following options to specify the value for this field:

- **Content.** Enter one or more valid recipient email addresses. You can also add fields that contain valid email addresses. Separate email addresses and fields with a comma (,) or a semicolon (;).
- **Field.** Select the field that the Taskflow Designer uses to write the email addresses into this field when this step executes. You can select an input field or a temporary field that was added in any other step in the taskflow.
- **Formula.** Open the formula editor to specify a formula that calculates the value for this field.

Default is **Content**.

### Email Bcc

The additional recipients who need to be sent a copy of the email notification. The recipients in the **Email To** and **Email Cc** fields will not be able to see the recipients in the **Email Bcc** field. If the field contains more than one recipient, the recipients will not be able to see the other recipients in the **Email Bcc** field.

Use one of the following options to specify the value for this field:

- **Content.** Enter one or more valid recipient email addresses. You can also add fields that contain valid email addresses. Separate email addresses and fields with a comma (,) or a semicolon (;).
- **Field.** Select the field that the Taskflow Designer uses to write the email addresses into this field when this step executes. You can select an input field or a temporary field that was added in any other step in the taskflow.
- **Formula.** Open the formula editor to specify a formula that calculates the value for this field.

Default is **Content**.

### Email Subject

A short and descriptive subject that introduces the email.

Use one of the following options to specify the value for this field:

- **Content.** Enter a subject for the email.
- **Field.** Select the field that the Taskflow Designer uses to write the email subject into this field when this step executes. You can select an input field or a temporary field that was added in any other step in the taskflow.
- **Formula.** Open the formula editor to specify a formula that calculates the value for this field.

Default is **Content**.

### Email Content Type

The type of formatting that you want to use for the email content.

Select one of the following values:

- **HTML.** Select **HTML** to use formatting options such as bold, italics, underlines, lists, indentations, and fonts. You can also insert tables and links.
- **Plain Text.** Select **Plain Text** to add regular text without any formatting and special layout option.

Default is **Plain Text**.

### Email Body

The content that you want to send in the email.

Use one of the following options to specify the value for this field:

- **Content.** Enter content for the email body.  
You can click **Edit Content** to open a rich text editor for formatting the content.  
The email content appears within HTML tags on the **All Jobs, Running Jobs, and My Jobs** pages.
- **Field.** Select the field that the Taskflow Designer uses to write the email body into this field when this step executes. You can select an input field or a temporary field that was added in any other step in the taskflow.
- **Formula.** Open the formula editor to specify a formula that calculates the value for this field.

Default is **Content**.

## Email notification examples

You can define the content of the email body based on the number of data tasks in the taskflow to send an email notification with HTML content.

### Single data task

When you use the Notification Task step for a single data task, you can enter the data task details in the email body with the source type as content.

For example, you might pass the following XQuery expression:

```
Hi,  
Data task with Run Id { $temp.DataTask1[1]/output[1]/Run_Id } started at  
{ $temp.DataTask1[1]/output[1]/Start_Time } and completed at { $temp.DataTask1[1]/  
output[1]/End_Time } with a status of { $temp.DataTask1[1]/output[1]/Task_Status }
```

In the above example, if the data task runs successfully, you will receive the output details in an email notification.

In case the data task fails, you can use the fault fields in the XQuery expression to determine the reason for failure as shown in the following example:

```
Hi,  
Data task with Run Id { $temp.DataTask1[1]/fault[1]/detail[1]/errOutputDetail[1]/  
Run_Id } started at { $temp.DataTask1[1]/fault[1]/detail[1]/errOutputDetail[1]/  
Start_Time } and completed at { $temp.DataTask1[1]/fault[1]/detail[1]/  
errOutputDetail[1]/End_Time } with a status of { $temp.DataTask1[1]/fault[1]/detail[1]/  
errOutputDetail[1]/Task_Status }
```

### Multiple data tasks

When you use the Notification Task step to send an email summary for multiple data tasks, you can use an XQuery expression with the source type as formula.

For example, you might pass the following XQuery expression for a taskflow that contains two data tasks named <DataTask1> and <DataTask2>:

```
util:toXML(<html>  
<head><h3>Taskflow Tasks Status Summary</h3></head>  
<body>  
<table>  
<tr>  
<th>Task Name</th>  
<th>Job Id</th>  
<th>Status</th>  
<th>Start Time</th>  
<th>End Time</th>  
</tr>  
    {  
        let $dataTasks := ($temp.DataTask1, $temp.DataTask2)
```

```

        for $dataTask in $dataTasks
        return
    <tr>
    <td>{ local-name($dataTask) }</td>
    <td>{ $dataTask[1]/output[1]/Run_Id }</td>
    <td>{ $dataTask[1]/output[1]/Task_Status }</td>
    <td>{ $dataTask[1]/output[1]/Start_Time }</td>
    <td>{ $dataTask[1]/output[1]/End_Time }</td>
    </tr>
    }
</table>
</body>
</html>

```

In the above example, if the data task runs successfully, you will receive the output details in an email notification.

In case of failed data tasks, you can use the fault fields in the XQuery expression to determine the reason for failure as shown in the following example:

```

util:toXML(<html>
<head><h3>Taskflow Tasks Status Summary</h3></head>
<body>
<table>
<tr>
<th>Task Name</th>
<th>Job Id</th>
<th>Status</th>
<th>Start Time</th>
<th>End Time</th>
</tr>
    {
    let $dataTasks := ($temp.DataTask1, $temp.DataTask2)
    for $dataTask in $dataTasks
    return
    <tr>
    <td>{ local-name($dataTask) }</td>
    <td>{ $dataTask[1]/fault[1]/detail[1]/errOutputDetail[1]/Run_Id }</td>
    <td>{ $dataTask[1]/fault[1]/detail[1]/errOutputDetail[1]/Task_Status }</td>
    <td>{ $dataTask[1]/fault[1]/detail[1]/errOutputDetail[1]/Start_Time }</td>
    <td>{ $dataTask[1]/fault[1]/detail[1]/errOutputDetail[1]/End_Time }</td>
    </tr>
    }
</table>
</body>
</html>

```

Before sending an email notification, you might want to convert the timezone based on the recipient's location. To do this, you can use the `infa:format` XQuery function in the email subject or email body. For more information about converting the timezone using a formula, see the following community article:

<https://knowledge.informatica.com/s/article/DOC-19342>

## Rules and guidelines for Notification Task step

Certain restrictions apply when you use the **Content** option to specify the email body and the **HTML** option to specify the email content type in a Notification Task step.

Consider the following guidelines:

- If you use HTML content, you may not receive a valid formatted email. You must use a variable to define the HTML content and serialize the variable in the Expression Editor.

For example, if the HTML content is as follows:

```

<html>
    Order {${output.OrderID} } has been submitted for {${input.CustomerEmail}}.

```

```

<br/>
<b>Order details for your records.</b>
<br/><br/>
Item Cost: {$temp.InventoryDetails[1]/ItemCostPrice}
Item Count: {$temp.InventoryDetails[1]/ItemCount }
Item Sell Price: {$temp.InventoryDetails[1]/ItemSellingPrice }
Commission Percentage: {$temp.InventoryDetails[1]/SalesCommissionInPercentage }
<br/><br/>
<b>Margins</b>
Overall Profit: {$output.Calculate_Margin_ServiceResponse[1]/
MarginBeforeCommission}
Sales Commission: {$output.Calculate_Margin_ServiceResponse[1]/SalesCommission}
Profit after Commission: {$output.Calculate_Margin_ServiceResponse[1]/
MarginAfterCommission}
</html>

```

Use the following content in the Expression Editor to define a variable for the HTML content and serialize the variable to receive a valid formatted email:

```

let $doc :=
<html>
Order {$output.OrderID} has been submitted for {$input.CustomerEmail}.
<br/>
<b>Order details for your records.</b>
<br/><br/>
Item Cost: {$temp.InventoryDetails[1]/ItemCostPrice}
Item Count: {$temp.InventoryDetails[1]/ItemCount }
Item Sell Price: {$temp.InventoryDetails[1]/ItemSellingPrice }
Commission Percentage: {$temp.InventoryDetails[1]/SalesCommissionInPercentage }
<br/><br/>
<b>Margins</b>
Overall Profit: {$output.Calculate_Margin_ServiceResponse[1]/
MarginBeforeCommission}
Sales Commission: {$output.Calculate_Margin_ServiceResponse[1]/SalesCommission}
Profit after Commission: {$output.Calculate_Margin_ServiceResponse[1]/
MarginAfterCommission}
</html>
return serialize($doc)

```

- If the HTML content contains valid XML, use the XQuery function `util:toXML` in the Expression Editor to serialize the content into the String format.

For example, if the HTML content is as follows:

```

<html>
<head>TaskDetails</head>
<body>
<table>
<tbody>
<tr>
<td>Task Name</td>
<td>Job Id</td>
<td>Status</td>
<td>Start Time</td>
<td>End Time</td>
</tr>
<tr>
<td>Williams</td>
<td>John</td>
<td>{fn:current-date()}</td>
</tr>
</tbody>
</table>
</body>
</html>

```

Use the following content in the Expression Editor to serialize the content into the String format:

```

util:toXML(<html>
<head>TaskDetails</head>
<body>
<table>

```

```
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task Name | Job Id | Status | Start Time | End Time |
| Williams | John | {fn:current-date() } |  |  |

```

- If the HTML content does not contain valid XML, you must convert the HTML content to valid XML. Sometimes even if the HTML content contains valid XML, you do not receive a valid formatted email. In this case, you must use the String Concat function.

For example, if the HTML content is as follows:

```

<html>
<table>
<tr>
<th>Task Property</th>
<th>Property Value</th>
</tr>
<tr>
<td>Task Name</td>
<td>${temp.DataTask1[1]/output[1]/Object_Name}</td>
</tr>
<tr>
<td>Task Status</td>
<td>${temp.DataTask1[1]/output[1]/Task_Status}</td>
</tr>
<tr>
<td>Error Message</td>
<td>${temp.DataTask1[1]/output[1]/Error_Message}</td>
</tr>
</table>
</html>

```

Use the following content in the Expression Editor with the String Concat function to send a valid formatted email:

```

fn:concat(fn:concat(fn:concat(fn:concat("Task Details: <br/><br/>"),
$temp.DataTask1[1]/output[1]/Object_Name), "<br/><br/>"), $temp.DataTask1[1]/
output[1]/Task_Status), "<br/><br/>"), $temp.DataTask1[1]/output[1]/Error_Message)

```

- If the HTML content does not contain valid XML, you must either convert the HTML content to valid XML or use the String Concat function with an escape character.

The following example shows how to use the String Concat function with an escape character:

```
fn:concat("&lt;html&gt;
&lt;head&gt;TaskDetails&lt;/head&gt;
&lt;body&gt;
&lt;table&gt;
&lt;tbody&gt;
&lt;tr&gt;
&lt;td&gt;Task Name&lt;/td&gt;
&lt;td&gt;Job Id&lt;/td&gt;
&lt;td&gt;Status&lt;/td&gt;
&lt;td&gt;Start Time&lt;/td&gt;
&lt;td&gt;End Time&lt;/td&gt;
&lt;/tr&gt;
&lt;tr&gt;
&lt;td&gt;Williams&lt;/td&gt;
&lt;td&gt;John&lt;/td&gt;
&lt;/tr&gt;
&lt;/tbody&gt;
&lt;/table&gt;
&lt;/body&gt;
&lt;/html&gt;","")
```

## Command Task step

Use a Command Task step to run shell scripts or batch commands from multiple script files on the Secure Agent machine.

For example, you can use a command task to move a file, copy a file, zip or unzip a file, or run clean scripts or SQL scripts as part of a taskflow. You can use the Command Task outputs to orchestrate subsequent tasks in the taskflow.

When you add a Command Task step to a taskflow, you configure the following properties:

### General Properties

In the general properties, you can specify a descriptive name for the Command Task step.

The name can contain only alphanumeric characters, underscores (\_), spaces, and Unicode characters. The name can't contain curly brackets {}.

## Input Fields

You can use the following system variables in the input fields to define the script file name, input argument, and work directory:

- \$PMRootDir
- \$PMLookupFileDir
- \$PMSessionLogDir
- \$PMBadFileDir
- \$PMCacheDir
- \$PMStorageDir
- \$PMTargetFileDir
- \$PMSourceFileDir
- \$PMExtProcDir
- \$PMTempDir
- \$PMWorkflowLogDir

These variables are pre-defined for the Data Integration Server service in Administrator. To use the system variables in the Command Task step, the Common Integration Components and Data Integration Server services must be enabled and up and running on the Secure Agent.

You can use environment variables in the input fields to define the script file name, input arguments, and work directory. To use environment variables, the Secure Agent must have the Common Integration Components service version 14 or later and the command executor package version 140 or later.

Configure the following input fields for a Command Task step:

### Fail task if any script fails

When you use multiple script files, you can configure a Command Task step to fail if any script fails.

When you configure one or more scripts in a Command Task step and select this option, if any script fails, the status of the Command Task step is set to failed and the taskflow does not run further scripts or the subsequent steps.

When you configure one or more scripts in a Command Task step and disable this option, the taskflow runs all the scripts. If any script fails, the status of the failed script is set to failed and the status of the Command Task step is set to warning on the **All Jobs**, **Running Jobs**, and **My Jobs** pages. However, Data Integration treats the Command Task step execution as a success.

This option is disabled by default.

### Runtime Environment

Required. The runtime environment that runs the command.

Use one of the following options to specify the value for this field:

- **Content.** Select a Secure Agent group.
- **Field.** Select the field that the Taskflow Designer uses to write the runtime environment into this field when this step executes. You can select a runtime environment that was added as an input field or temporary field in any other step in the taskflow. The value must be a valid secure agent group name.
- **Formula.** Create an expression for the runtime environment with a valid secure agent group name.



### Max Wait (Seconds)

In the **Max Wait (Seconds)** field, you can configure the maximum length of time in seconds that the Command Task step waits for the task to complete. Specify a value between 1 and 86400 seconds. Default is 86400 seconds. If the task is not completed within the maximum time specified in the field, the task stops running and the subsequent task in the taskflow starts running.

**Note:** If the specified value is lesser than 1 or greater than 86400, the maximum wait time is automatically set to 86400 seconds.

### Scripts

In a Command Task step, you can add multiple scripts. You can configure a script by specifying the script file name, input arguments, and work directory.

To add more scripts, click the **Add** icon from the Scripts panel. If you have multiple scripts in the command task, you can drag and drop the scripts to reorder the scripts.

To run scripts from different Secure Agent groups, you must add separate Command Task steps for different runtime environments.

You can view all the scripts in a taskflow instance as subtasks of the command task from the **All Jobs**, **Running Jobs**, or **My Jobs** page. If a script fails, you can also download the log file to understand the reason for the script failure.

Configure the following input fields for a script:

#### Script File Name

Required. The path and name of the script file that you want to run.

Use one of the following options to specify the value for this field:

- **Content.** Enter the path to the script that you want to run.
- **Field.** Select the field that the Taskflow Designer uses to write the script file name into this field when this step executes. You can select a script file name that was added as an input field, temporary field, or output field in any other step in the taskflow.
- **Formula.** Create an expression for the script file.

You can use system variables to define the script file that you want to run.

For example, to run the script.bat file located in the root directory, you can enter the value as `$PMRootDir/script.bat`.

You can use an environment variable to define the script file path based on the operating system.

For example, you have a java\_script.bat file located in the following directory:

```
C:\Scripts
```

You have created an environment variable named ScriptHome for the directory. To run the java\_script.bat file, you can enter the value as `%ScriptHome%\java_script.bat` for Windows and `$ScriptHome/java_script.bat` for Linux.

You can also create an environment variable for the full path including the script file name.

You can also provide the EFS or NFS mounted directories including the script file name to run the scripts.

For example, you have an aws\_script1.sh file located in the following EFS or NFS directory:

```
mountDir/scripts
```

To run the aws\_script1.sh file, you can enter the value as `/mountDir/scripts/aws_script1.sh`.

For more information about EFS and NFS, see *Runtime Environments*.

### Input Arguments

Optional. The arguments that you want to pass to the script when it is executed.

Use one of the following options to specify the value for this field:

- **Content.** Enter one or more arguments that you want to pass to the script. Enclose each argument and values in double quotes (") and separate arguments with a comma (.). For example, if you want to execute the following command:

```
./ cli.sh runAJobCli -u <username>-p <password> -bu https://dm-  
us.informaticacloud.com/ma -un <sample_value> -t TASKFLOW -w true
```

Enter the input arguments within double quotes as shown in the following example:

```
"runAJobCli", "-u", "<username>", "-p", "<password>", "-bu", "https://dm-  
us.informaticacloud.com/ma", "-un", "<sample_value>", "-t", "TASKFLOW", "-w",  
"true"
```

You must pass `cli.sh` in the script name.

- **Field.** Select the field that the Taskflow Designer uses to write the input arguments into this field when this step executes. You can select an input argument that was added as an input field, temporary field, or output field in any other step in the taskflow.
- **Formula.** Create an expression for the input arguments. Enclose each argument in double quotes (") and separate arguments with a comma (.).

You can define input arguments in the following ways:

#### Using system variables

You can use system variables to define the input arguments that you want to pass to the script when it is executed. For example, to pass the arguments from the root directory or the temp directory, enter the following value:

```
"$PMRootDir", "$PMTempDir"
```

#### Using environment variables

You can use an environment variable to define the input arguments based on the operating system. For example, you have created an environment variable named `ScriptHome` for the following directory:

```
C:\Scripts
```

To pass the arguments from this directory, enter the value as `"%ScriptHome%"` for Windows and `"$ScriptHome"` for Linux.

### Work Directory

Optional. The path to the working directory where the output of the script is saved. By default, the output is saved in the path where the script is saved.

Use one of the following options to specify the value for this field:

- **Content.** Enter the path to the working directory where you want to save the output of the script.
- **Field.** Select the field that the Taskflow Designer uses to write the working directory into this field when this step executes. You can select a work directory that was added as an input field, temporary field, or output field in any other step in the taskflow.
- **Formula.** Create an expression for the work directory.

You can use system variables to define the working directory where you want the output of the script to be saved.

For example, if you want to use the source file directory as the working directory, you can enter the value as `$PMSourceFileDir`.

You can use an environment variable to define the working directory based on the operating system.

For example, you created an environment variable named `ScriptHome` for the following directory:

```
C:\Scripts
```

To use this directory as the working directory, you can enter the value as `%ScriptHome%` for Windows and `$ScriptHome` for Linux.

**Note:** When you use curly brackets `{}` in the script or in the input fields, you must add an additional set of curly brackets `{{}}`. Otherwise, an error occurs. This is because curly brackets are considered as special characters in XQuery.

## Output Fields

When you run a taskflow, the following output fields are generated for the Command Task step:

Output Field	Type	Description
Run Id	Text	The run ID of the command task.
Start Time	Date Time	The start time of the command task.
End Time	Date Time	The end time of the command task.
Exit Code	Integer	The exit code returned after command task execution. The exit code can have one of the following values: <ul style="list-style-type: none"><li>- 0. Indicates that the command task was executed successfully.</li><li>- 1. Indicates that the command task failed.</li></ul>
Execution Status	Text	Displays the status of the command task as successful.
Std Error	Text	Displays the error message.

To view the values of each output field, run the taskflow and go to the **Taskflow Instance Detail** page in Monitor. For more information about the **Taskflow Instance Detail** page, see *Monitor*.

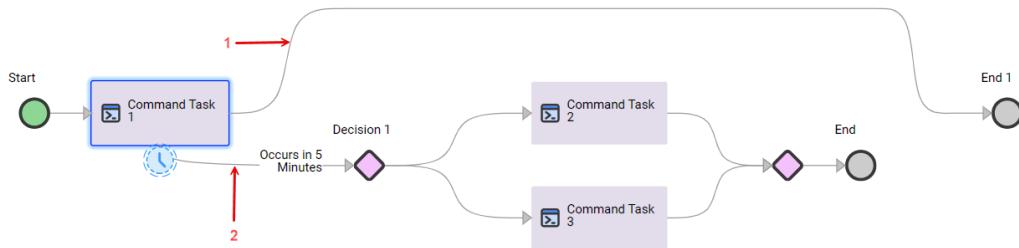
## Events

Configure the **Events** properties to add timers to a command task.

Use a Timer event to perform an action based on a schedule. The action could be either at a specific time or after an interval.

When you add a timer to a Command Task step, a new branch appears. Add an event to this branch and specify whether you want the event to run **At** a specific time or **After** an interval.

In the following image, the event on the timer branch is a Data Decision step (Decision 1) that occurs five minutes after the main command task (Command Task 1) starts:

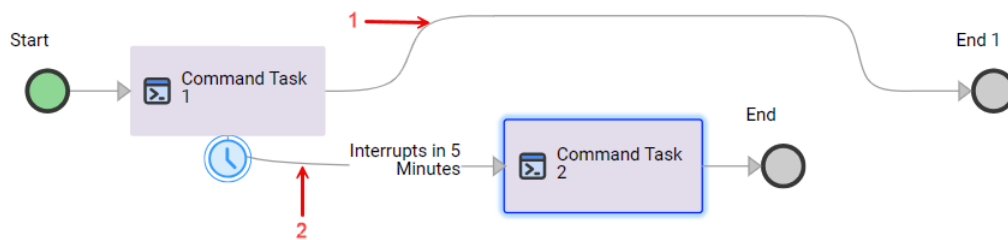


1. Main branch
2. Timer branch

In the example, the timer branch runs five minutes after Command Task 1 starts. If Command Task 1 finishes before Decision 1, the timer branch is not executed.

You can select the **Interrupting** option if you want the timer to interrupt the main command task.

The following image shows an interrupting timer set to occur five minutes after the main command task starts:



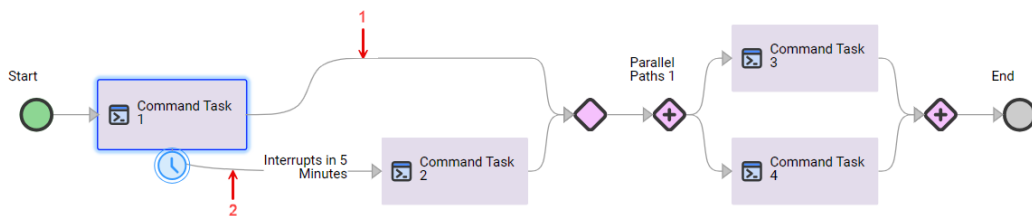
1. Main branch
2. Timer branch

In the example, Command Task 2 executes after five minutes and interrupts Command Task 1. The taskflow executes only Command Task 2 and then ends. Command Task 1 has no output in this taskflow instance. You see no output fields for Command Task 1 in the job details for the taskflow.

If Command Task 1 completes before the timer, the taskflow executes only Command Task 1 and ends.

If you delete the End step on the timer branch of an interrupting timer, the timer branch rejoins the main branch.

The following image shows an interrupting timer branch with the End step deleted:



1. Main branch
2. Timer branch

In the example, Command Task 2 executes after five minutes and interrupts Command Task 1. The timer branch rejoins the main branch. The taskflow executes Command Task 2, a Decision step, a Parallel Paths step, and then ends.

If Command Task 1 completes before the timer, the taskflow executes Command Task 1, a Decision step, a Parallel Paths step, and then ends.

## Error Handling

Use the **Error Handling** tab to indicate how you want the taskflow to behave when a Command Task step encounters an error. You can also configure the taskflow behavior when the Command Task step fails or does not run.

After you select a task, configure the following error handling properties:

### On Error

The path that a taskflow takes when it encounters an error in a Command Task step.

An error occurs when a Command Task step fails. You can choose from the following options:

- Select **Ignore** to ignore the error and move to the next step.

**Note:** If you select **Ignore** for a Command Task step with a subsequent Notification Task step and the command task fails, the email notification that you receive does not contain the fault details. To get the fault details in the email, select **Custom error handling**.

- Select **Suspend Taskflow** to move the taskflow to the suspended state when it encounters an error. You can resume the taskflow instance from the **All Jobs**, **Running Jobs**, or **My Jobs** page. The taskflow resumes from the step at which it was suspended. If you know the reason for the error, correct the issue and then resume the taskflow.
- Select **Custom error handling** to handle the error in a manner you choose. If you select **Custom error handling**, two branches appear. The first branch is the path the taskflow follows if no error occurs. The second branch is the custom path the taskflow follows if an error occurs.

Default is **Suspend Taskflow**.

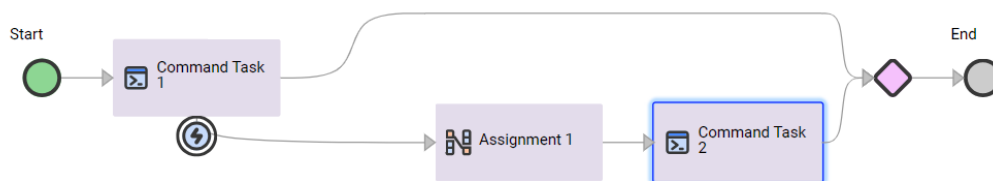
### Fail taskflow on completion

The taskflow behavior when the Command Task step fails or does not run.

You can configure a taskflow to fail on its completion if the Command Task step fails or does not run. If the step fails or does not run, the taskflow continues running the subsequent steps. However, after the taskflow completes, the taskflow status is set to failed.

If you configure both the **Suspend on Fault** taskflow advanced property and the **Fail taskflow on completion** property, the **Suspend on Fault** property takes precedence. In this case, if the Command Task step fails or does not run, the taskflow is suspended. The taskflow does not run the subsequent steps after the Command Task step.

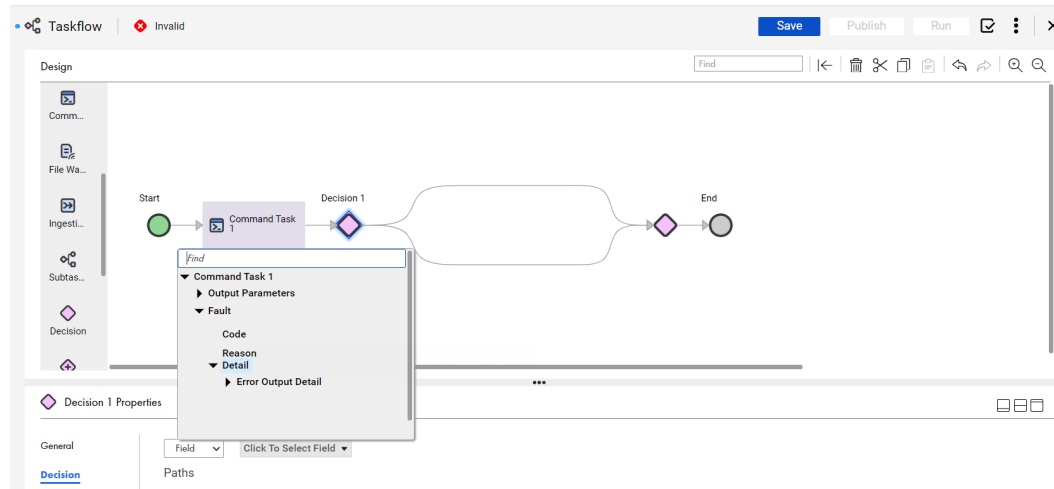
The following image shows a **Custom error handling** path with an Assignment step and another Command Task step:



## Fault

The fault fields are displayed only if the command task fails due to a script failure. The details help you analyze the reason for the fault. You can then take an appropriate action on the faulted command task and proceed with the execution of the taskflow.

You can add the details in fault fields as parameters to the subsequent steps of the taskflow as shown in the following image:



If the command task has faulted, you see the following fault details:

Property	Type	Description
Run Id	Text	The run ID of the command task.
Start Time	Date Time	The start time of the command task.
End Time	Date Time	The end time of the command task.
Exit Code	Integer	The exit code returned after the command task execution. The exit code can have values between <b>0</b> and <b>255</b> . The value <b>0</b> indicates that the command task ran successfully. A failed command returns a non-zero value. The value can be based on the error type or pre-configured code in the shell script.
Execution Status	Text	Displays the status of the command task as failed.
Std Error	Text	Displays the error message.

## Subtaskflow step

When you add a Subtaskflow step, you can embed and reuse an existing taskflow.

You can use a subtaskflow to reuse the same orchestration flow across multiple branches of a taskflow or across different taskflows. You can then invoke the taskflow with different sets of parameters. The subtaskflow is published when you publish its parent taskflow.

When you have a taskflow that contains numerous steps, consider splitting the orchestration logic across multiple smaller taskflows. You can then simplify the design by using the Subtaskflow step to embed the smaller taskflows in the parent taskflow. This not only leads to modular design, but also helps with faster loading when you open the taskflow for editing.

You can define properties for the subtaskflow. The following sections describe the Subtaskflow step properties:

### General properties

In the general properties, you can specify a descriptive name for the Subtaskflow step.

The name can contain only alphanumeric characters, underscores (\_), spaces, and Unicode characters. The name can't contain curly brackets {}.

### Subtaskflow properties

On the **Subtaskflow** tab, select the taskflow that you want to embed. Data Integration displays the location, description, input fields, and output fields for the embedded taskflow.

### Input Fields

The **Input Fields** section appears when you add a subtaskflow to the taskflow.

If the taskflow contains parameters that you can override, you can add input fields. You can set properties for an input field to override Data Integration run-time parameters.

### Fault handling properties

You can configure the following fault handling properties:

#### Catch faults

Select this option to enable fault handling for the Subtaskflow step.

By default, this option is not selected.

#### Fault field name

Required if you select the **Catch faults** option.

The name of the field that captures the fault information.

Default is **faultInfo**.

The name can't start with `temp.<name>`, `input.<name>`, `output.<name>`.

#### Fail taskflow on completion

Defines the behavior when the subtaskflow associated with the Subtaskflow step fails or does not run.

By default, when the subtaskflow associated with the Subtaskflow step fails, the parent taskflow also fails.

To configure a taskflow to fail on its completion when the subtaskflow fails, select the **Catch faults** option and the **If this subtaskflow fails** option. To configure a taskflow to fail on its completion when the subtaskflow does not run, select the **If this subtaskflow does not run** option. In these cases, when the subtaskflow fails or does not run, the taskflow continues running the subsequent steps. However, after the taskflow completes, the taskflow status is set to failed.

**Note:** If you configure both the **Suspend on Fault** taskflow advanced property and the **Fail taskflow on completion** property, the **Suspend on Fault** property takes precedence. In this case, if the subtaskflow associated with the Subtaskflow step fails or does not run, the taskflow is suspended. The taskflow does not run the subsequent steps after the Subtaskflow step.

# Decision step

When you add a Decision step, you set some properties.

You can configure the following Decision step properties:

## Name

The name of the Decision step. The name can contain only alphanumeric characters, underscores (\_), spaces, and Unicode characters. The name can't contain curly brackets {}.

## Decision

The taskflow takes a decision based on the fields, formulae, and paths you define here.

Use one of the following options to specify the path:

- **Field.** Select an input field, output field, or temporary field from the list of fields you define under the **Start** step.

Enter conditions and values that you want the Decision step to base a decision on.

The conditions available depend on the field that you select.

For example, if you select a field of type **Simple > Text**, the following conditions are available:

- Equals
  - Starts With
  - Ends With
  - Starts with any of
  - Contains
- **Formula** Open the formula editor to create a complex expression.  
You can define a path and set appropriate conditions. You can also evaluate simple and complex expressions directly in the Decision step with no dependency on the prior steps.

When you select the **Formula** option and define the expression, the following conditions are available:

- Contains
- Equals
- Starts with
- Ends with
- Starts with any of
- Not equal to
- Less than
- Less than or equal to
- Greater than
- Greater than or equal to

However, you must select the appropriate conditions based on the return type of the functions used in the expression.



The following table describes the supported and not supported conditions based on the return types:

Return type	Supported conditions	Unsupported conditions
String and Boolean	Contains Equals Starts-with Ends with Starts with any of	Not equal to Less than Less than or equal to Greater than Greater than or equal to
Number and Integer	Contains Equals Starts-with Ends with Starts with any of Not equal to Less than Less than or equal to Greater than Greater than or equal to	None
DateTime	Contains Equals Starts-with Ends with Starts with any of	Not equal to Less than Less than or equal to Greater than Greater than or equal to

Default is **Field**.

You can enter text values against the conditions you select.

You can add multiple conditions to a Decision step. Each condition is a potential data path.

For each path that you add, a corresponding branch appears on the UI. Drag branches to rearrange the order in which the branches appear on the UI.

Most Decision steps have an Otherwise path. This path handles execution if no data meets the conditions in your tests.

## Evaluating Paths

A taskflow evaluates conditions based on the criteria you specify. Ensure that you construct paths with non-intersecting conditions.

For example, you create a Data Decision step with the following paths:

- Path 1: Field less than or equal to 100.
- Path 2: Field less than or equal to 75.
- Path 3: Field less than or equal to 25.
- Path 4: Otherwise

If the integer field for which the Data Decision step was created has a value of 25, the Data Decision step takes path 1. This is because 25 is less than 100 and path 1 is the first option.

To ensure that the Data Decision step follows the "Field less than or equal to 25" path, re-create the paths with the following criteria:

- Path 1: Integer between 0 and 25
- Path 2: Integer between 26 and 75.
- Path 3: Integer between 76 and 100.
- Path 4: Otherwise

**Important:** The taskflows evaluates conditions in a top-down manner. Ensure that the Otherwise branch is the last path.

A Decision step can lead to another Decision step. For example, a branch could run if an annual income exceeds \$100,000. The next decision test along the same path could test if the city is Boston, or otherwise. Using this technique, you use Boolean AND logic because you base the test for the second condition on the true branch of the first condition. In this example, you use the Decision step to set the condition "Annual Revenue exceeds \$100,000 AND city is Boston".

Similarly, to support Boolean OR logic, you can add a test for the second condition on any branch.

When the Data Task step of a taskflow fails, you can make decisions based on the output fields of the data task.

You can select the output fields when one of the following conditions are met:

- The **On Error** field is set to **Ignore** or **Custom error handling**.
- The **Fail taskflow on completion** option is set to **If this task fails**.

If you select the field as the entire data task, the Decision step takes the **Is set** path by default.

## Parallel Paths step

When you add a Parallel Paths step, you set some properties.

You can configure the following Parallel Paths step properties:

### Name

The name of the Parallel Paths step. The name can contain only alphanumeric characters, underscores (\_), spaces, and Unicode characters. The name can't contain curly brackets {}.

### Parallel Paths

The paths that you want the taskflow to run in parallel.

Click **Add** to add a new branch.

You can add multiple steps to each branch. To add steps to a branch, drag and drop a step from the palette on the left.

You can run the same mapping task in multiple branches of the Parallel Paths step if the mapping task is configured to run simultaneously.

When you use the Jump step in conjunction with the Parallel Path step, you can only jump to another step on the same Parallel Path branch.

Keep in mind the following restrictions when you use the Jump step and the Parallel Path step together:

- If you are in a Parallel Path step, you cannot jump to a step on another branch of the same Parallel Path step.
- If you are in a Parallel Path step, you cannot jump to any step outside the Parallel Path step.
- If you are outside a Parallel Path step, you cannot jump to any step inside the Parallel Path step.

## Jump step

When you add a Jump step, you configure the **To** field to define the target of the jump. You can select from a list of available steps.

More than one step can jump to the same target step. To see how many Jump steps have a particular step as their target, place the cursor over the arrow next to the target step.

When you use the Jump step in conjunction with the Parallel Path step, you can only jump to another step on the same Parallel Path branch.

Keep in mind the following restrictions when you use the Jump step and the Parallel Path step together:

- If you are in a Parallel Path step, you can't jump to a step on another branch of the same Parallel Path step.
- If you are in a Parallel Path step, you can't jump to any step outside the Parallel Path step.
- If you are outside a Parallel Path step, you can't jump to any step inside the Parallel Path step.
- You can't jump to any step from the fault path. You must merge the fault path back to the main taskflow path to add the Jump step.
- You can't jump to a step inside the fault path if you are outside the fault path.

### Best design practices

Use the following best practices when you use the Jump step in a taskflow:

- Instead of using a Jump step from an outside flow to a step inside the fault path, design the taskflow by replacing the Jump step with the same flow as in the fault path.
- If you use a Jump step for iteration purposes and it points to a Subtaskflow step with fault handling, enclose the Subtaskflow step with fault handling inside another Subtaskflow step.

## End step

An End step indicates the end of the taskflow. When execution reaches this step, the taskflow completes.

You can configure the following End step properties:

### Name

The name of the step. You can edit this value. The name can contain only alphanumeric characters, underscores (`_`), spaces, and Unicode characters. The name can't contain curly brackets `{}`.

### Ending Type

The default value is `End of Process`. You cannot edit this value.

### HTTP Status

The HTTP response status code. The default value is 200 OK. You can edit this value.

## Wait step

When you add a Wait step, you set some properties.

You can configure the following Wait step properties:

### Name

The name of the Wait step. The name can contain only alphanumeric characters, underscores (\_), spaces, and Unicode characters. The name can't contain curly brackets {}.

### Wait

Properties that determine when and for how long the taskflow pauses.

Use the following criteria to decide if you want the taskflow to pause **At a Specific Time** or **After a wait period**:

- Select **At a Specific Time** to pause the taskflow at a particular time. Enter the **Time** you want the taskflow to pause at, and optionally, a **Delay**. The **Delay** value can be an integer or a field that you define.  
For example, set the taskflow to pause at 2:00 am after three days. 2:00 am is the **Time** and three days is the **Delay**.
- Select **After a wait period** to pause the taskflow after a period. The period begins when the taskflow reaches the Wait step. Enter the **Wait Period** that you want the taskflow to pause for. The **Wait Period** value can be an integer or a field that you define.  
For example, set the taskflow to pause for one hour from the time that the taskflow reaches the Wait step.

## Throw step

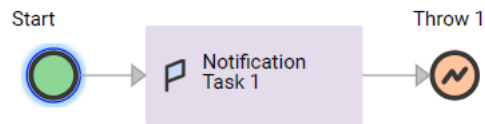
Use a Throw step to catch a fault, return the fault details, stop the execution of the taskflow, and set the taskflow status to failed.

You can use a Throw step for the following use cases:

### To catch faults in a taskflow

You can add a Throw step to the main path of a taskflow to catch faults in a taskflow and return the fault details. You cannot add a step after the Throw step because the Throw step is an interrupting step. If a fault occurs, the Throw step stops the execution of the taskflow and sets the taskflow status to failed.

For example, consider the following sample taskflow:



If a fault occurs in the Notification Task step, the Throw step is executed. The Throw step stops the execution of the taskflow and sets the taskflow status to failed.

**To act as a boundary event for a specific step in a taskflow**

When you enable custom error handling for a taskflow step, you can use a Throw step in the error handling path to act as a boundary event for the step. A boundary event is an event that catches an error that occurs within the scope of the step where it is defined.

You can add a Throw step to the error handling path of the following steps because they support custom error handling:

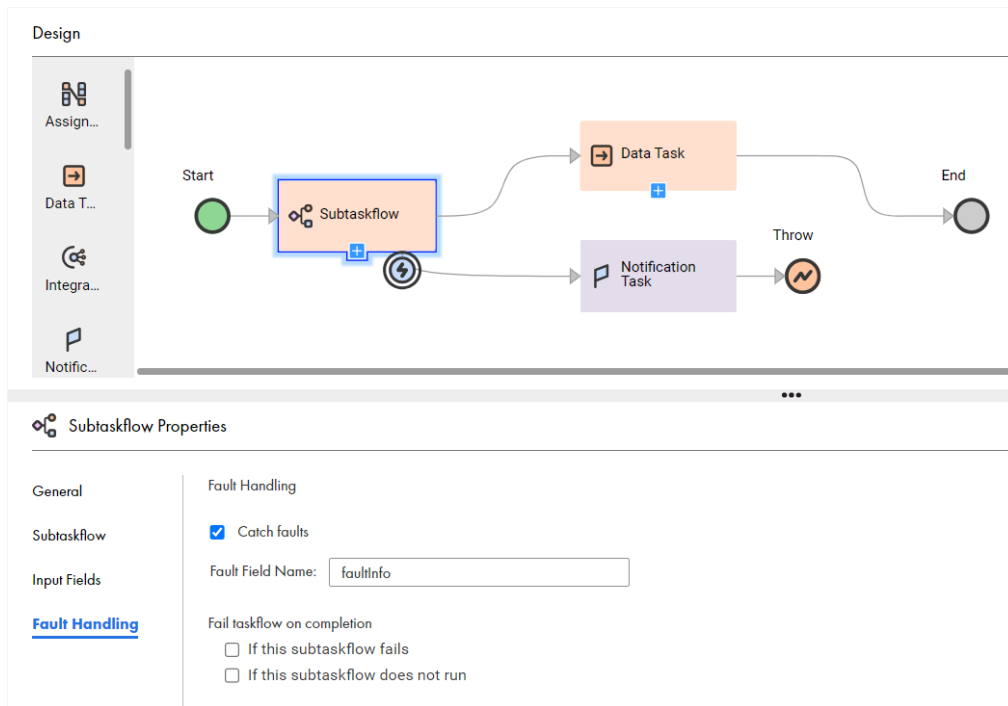
- Data Task
- Command Task

You can also add a Throw step to the error handling path of a Subtaskflow step if you configure the Subtaskflow step to catch faults. If the taskflow that is contained within the Subtaskflow step fails, the parent taskflow also fails. When you view the execution details of the parent taskflow in Monitor, you can click the Throw step that is associated with the Subtaskflow step to understand why the subtaskflow failed.

When you add a Throw step to the error handling path, the error handling path breaks off from the main path of the taskflow. If a fault occurs, the taskflow takes the path that you define for handling the error. For example, in the error path, you might add a Notification Task step to send an email notification followed by a Throw step to catch the fault and stop the taskflow execution.

In the error handling path, you cannot add a step after the Throw step because the Throw step is an interrupting step. If a fault occurs, the Throw step stops the execution of the taskflow and sets the taskflow status to failed. The subsequent steps in the main path of the taskflow that exist after the step that is associated with the Throw step are not executed.

For example, consider the following sample taskflow:



The Subtaskflow step is configured to catch faults. If a fault occurs, an email notification is sent as configured in the Notification Task step. The Throw step then stops the execution of the taskflow, returns the fault details specifying why the subtaskflow failed, and sets the taskflow status to failed. The Data Task step is not executed.

## Throw step properties

The following sections describe the Throw step properties:

### General properties

In the general properties, you can specify a descriptive name for the Throw step.

The name can contain only alphanumeric characters, underscores (\_), spaces, and Unicode characters. The name can't contain curly brackets {}.

### Fault fields

You can configure the following fault fields:

#### Code

Required. Defines the code for the fault.

Use one of the following options to specify the value for this field:

- **Content.** Enter a code for the fault.
- **Field.** Select the field that the Taskflow Designer uses to write the code into this field when this step executes. You can select an input field, temporary field, output field, or fault field that was added in any other step in the taskflow.
- **Formula.** Open the formula editor to specify a formula that calculates the value for this field.

Default is **Content**.

**Detail**

Defines the fault details.

Use one of the following options to specify the value for this field:

- **Content.** Enter the fault details.
- **Field.** Select the field that the Taskflow Designer uses to write the fault details into this field when this step executes. You can select an input field, temporary field, output field, or fault field that was added in any other step in the taskflow.
- **Formula.** Open the formula editor to specify a formula that calculates the value for this field.

Default is **Content**.

**Reason**

Defines why the fault occurred.

Use one of the following options to specify the value for this field:

- **Content.** Enter the reason why the fault occurred.
- **Field.** Select the field that the Taskflow Designer uses to write the fault reason into this field when this step executes. You can select an input field, temporary field, output field, or fault field that was added in any other step in the taskflow.
- **Formula.** Open the formula editor to specify a formula that calculates the value for this field.

Default is **Content**.

## CHAPTER 6

# Runtime parameters

You can configure parameters in the following ways:

### Parameters

Parameters are placeholders that represent values in a mapping task. You can use a taskflow to pass input parameters and in-out parameters to a task.

### Parameter files

A parameter file is a list of user-defined parameters and their associated values.

If the parameter file contains sections and parameters for taskflows and mapping tasks, you can save the parameter file in the location where the Data Integration job runs. For more information about parameter files, see *Mappings*.

## Parameters in taskflows

You can use a taskflow to pass input parameters and in-out parameters to a task.

You can design a mapping with input parameters or in-out parameters. When you add a mapping task to a taskflow, you can override the parameter values. The mapping task passes these parameters to the mapping. You can use the parameterized mapping task in different scenarios.

The following section explains input and in-out parameters and how you can use them in a taskflow:

### Input parameters

An input parameter is a placeholder for a value or values in a mapping task. You define the value of the parameter when you configure the mapping task. For more information about input parameters, see *Mappings* and *Tasks*.

You can use a taskflow to override the following subset of mapping input parameters:

- String. Changes the string value to be used as input for the mapping task.
- Source object. Changes the object that the mapping task reads from.
- Source connection. Changes the connection that the mapping task uses to read from the source.
- Target connection. Changes the connection that the mapping task uses to write to the target.
- Target object. Changes the object that the mapping task writes to.
- Source\_dataFormat. Changes the data format for a precision value or any other value for a particular column in the formatting file.
- Source\_defaultPrecision. Changes the precision value for all the columns in the formatting file.



For example, consider a fully parameterized mapping task. The mapping task uses an SQL connection to read from the `employeeaddress` table and a JDBC connection to write to the `employeedetails` table.

You can create a taskflow and override parameters in the mapping task. For example, you can use a Salesforce connection to read from the `employeeincome` table and a flat file connection to write to the file `income.txt`.

### In-Out parameters

An in-out parameter is a placeholder for a value that you can pass in to or out of a mapping task. Unlike input parameters, an in-out parameter can change each time a task runs. You can use a taskflow to override any type of in-out parameters that a mapping task supports. For more information about in-out parameters, see *Mappings and Tasks*.

For example, consider a mapping that keeps track of how many rows it processes using an in-out parameter named `lastprocessedindex`. Every time you run the mapping, it resumes processing from this index. And also, let us say that every time the mapping is executed, it processes 5000 rows.

You can configure the taskflow such that the mapping task runs till it reaches records number, say, 50000.

**Note:** If the in-out parameters in the selected mapping task are added or updated, you must reselect the mapping task in the taskflow to use the updated mapping task.

## Overriding parameters in a taskflow

Override the parameters of a task when you use it in a taskflow. You can override parameters with the Data Task step or with the Assignment step.

### Overriding parameters or parameter files in a Data Task step

If the Data Task step uses a mapping task with a flat file source, you can override the parameter values that are set in the task. If the Data Task step uses a mapping task, you can override the native data type, default precision, and default scale. If the Data Task step uses a mapping task available on the Secure Agent machine, you can override the parameter file directory or parameter file name set in the mapping task. If the Data Task step uses a mapping task available in a cloud-hosted repository, you can override the parameter file connection and parameter file object set in the mapping task.

Perform the following steps to use a Data Task step to override a Data Integration parameter or parameter file:

1. Create a taskflow and add a Data Task step.
2. Add a task that contains parameters or uses a parameter file to the Data Task step.
3. Go to **Data Task > Input Fields**.
4. Click **Add**.
5. Perform one of the following steps:
  - a. To override the parameter file directory or parameter file name of a mapping task that is available on the Secure Agent machine, expand the **Task Properties Parameters** list. Then, select **Parameter File Directory** or **Parameter File Name**.

**Note:** If the object that you want to override is under a directory, use double forward slashes (`//`) or `%2F` from the third-level onwards in the file path. For example, to override `input1.csv`, which is

located under `infa.bucket/flat_file_dir/parameters/input1.csv`, use one of the following formats:

- `infa.bucket/flat_file_dir//parameters//input1.csv`
- `infa.bucket/flat_file_dir%2F/parameters%2F/input1.csv`

- To override an input parameter or in-out parameter of a mapping task, expand the **Input Parameters** or **InOut Parameters** list. Then, navigate to and select the parameter that you want to override.
- To override the parameter file connection and parameter file object of a mapping task that is available in a cloud-hosted repository, expand the **Task Properties Parameters** list. Then, select **Parameter File Connection** and **Parameter File Object**.

**Note:** When you configure the Data Task step to override the parameter file connection and parameter file object, you must ensure that both the parameters, that is, source object and source connection are passed either through the parameter file or taskflow. Otherwise, an error occurs.

- To override the native data type, default precision, or default scale of the source data of a mapping task, expand the **Input Parameters > Sources > Source** list. Then, select **Source\_defaultNativeDataType**, **Source\_defaultPrecision**, or **Source\_defaultScale** respectively.

**Note:** The values set in the Source list take precedence over the values set in the mapping task and the data format field in the Data Task step. If the values in the Source list are incompatible with the values set in the mapping task and the data format field in the Data Task step, the job might fail. The default precision and scale values are applied to all the columns in the formatting file.

- To override the values in the **Update Columns** field that is defined in the target of the mapping task, expand the **Input Parameters > Targets > Target** list. Then, select **updateColumns**. You can enter multiple column names separated by commas enclosed within or without double quotes.

**Note:** You can override the **Update Columns** field only if the mapping task uses an update or upsert operation.

- Click **Edit**.

The **Edit Value** dialog box appears.

- Under **Source**, select **Content**. For advanced use cases, you might select **Field** or **Formula**.

- Under **Value**, enter the new value that you want to override the default value with.

**Note:** For data format, **Formula** is selected by default, and the payload appears within a single quote in the **Value** field. You can click **f(x)** and override the values in the Expression Editor. Alternatively, you can select **Field** if the field contains the payload. However, the field that you choose must contain the payload using **Formula** within a single quote in the Assignment step.

- Click **OK**.

## Overriding parameters with an Assignment step

Perform the following steps to use an Assignment step to override a Data Integration parameter:

- Create a taskflow and add a Data Task step.
- Add a mapping task that contains input parameters to the Data Task step.
- Drag an **Assignment** step onto the canvas.
- Go to **Assignment Properties > Assignments**.
- Click the **Add** icon, and then navigate to and select the parameter you want to override.
- Under **Value**, select **Content**. For advanced use cases, you might select **Field** or **Formula**.
- Enter or select the object or connection that you want to use to override the default object or connection.

# Guidelines and best practices for using parameters in a taskflow

The following sections describe guidelines and best practices when overriding parameters in a taskflow.

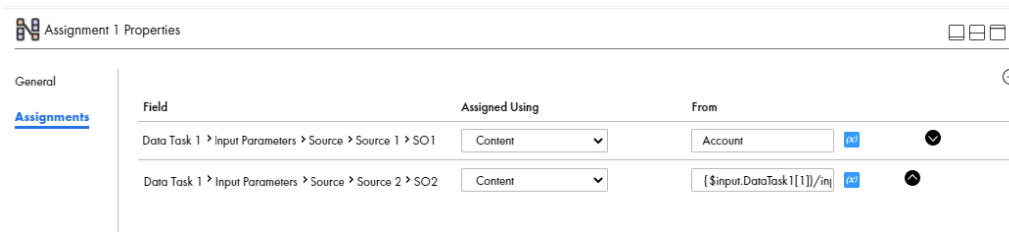
## Guidelines for using input parameters in taskflows

Use the following guidelines when you use input parameters in a taskflow:

- Use the Data Task step to override input parameters. However, in advanced cases, you can override input parameters with an Assignment step.
- If you define a field to override the same parameter in both the Assignment step and the Data Task step, the taskflow considers the value assigned in the Data Task step.
- If you use a Data Task step to assign values to input or in-out parameters, the assignments must be independent of each other. If you use an Assignment step to assign values to input or in-out parameters, an assignment operation can use the result of a previous assignment operation in the same step.

For example, you have two source objects, SO1 and SO2. You want to override SO1 with the value `Account` and override SO2 with the value `SO1`. Use the Assignment step to override the parameters.

The following image shows the source object **SO2** overridden with the value of **SO1**:



- When you configure the Data Task step to override the parameter file connection and parameter file object, you must ensure that both the parameters, that is, source object and source connection are passed either through the parameter file or taskflow. Otherwise, an error occurs.

## Guidelines for using in-out parameters

Use the following guidelines when you use in-out parameters in a taskflow:

- If you define a field to override the same parameter in both the Assignment step and the Data Task step, the taskflow considers the value assigned in the Data Task step.
- Use the Data Task step to override in-out parameters unless your use case specifically demands the usage of the Assignment step.
- You can get the current value of the in-out parameter only after the taskflow executes the Data Task step. You cannot get this value before the taskflow runs the Data Task step.

## Example: Overriding parameters with a Data Task step

You want to override the following input parameters in a mapping task, **MyMT** :

- The source data object parameter, `MySourceObject`, with default value **input.txt**.
- The source connection parameter, `MySourceConnection`, with default value **My File Connection**.
- The target connection parameter, `MyTargetConnection`, with default value **My File Connection**.
- The target data object parameter, `MyTargetObject`, with default value **output5.txt**.

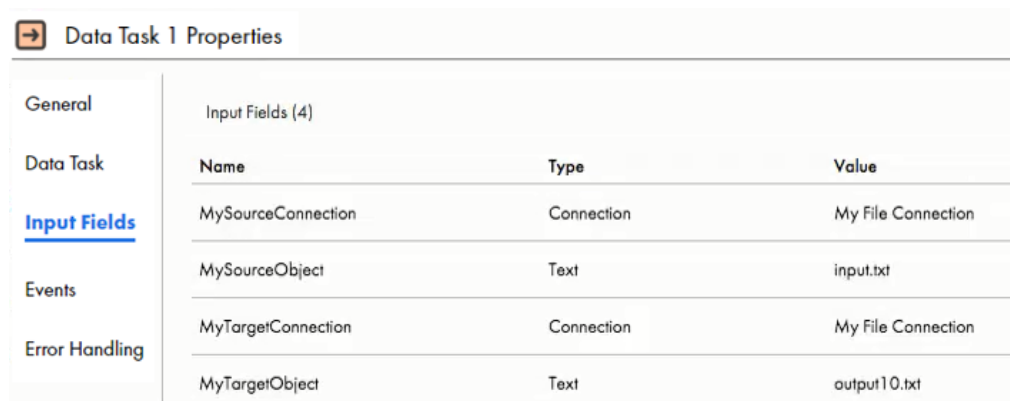
**MyMT** uses **My File Connection** to read from the **input.txt** file and **My File Connection** to write to the **output5.txt** file.

You want **MyMT** to use **My File Connection** read from the **input.txt** file and use table and then use **My File Connection** to write to another output file, **output10.txt** file.

To do this, perform the following steps to override the default value of `MyTargetObject` with a Data Task step:

1. Create a taskflow and add a Data Task step, **Data Task 1**.
2. Go to **Data Task 1 > Data Task** and add the mapping task **MyMT**.
3. Go to **Data Task 1 > Input Fields**.
4. Perform the following steps to add `MySourceConnection` to the **Input Fields** section of **Data Task 1**:
  - a. Click the **Add** icon, and then navigate to and select `MySourceConnection`.
  - b. Click **Edit**. The **Edit Value** dialog box opens.
  - c. In the field next to **Source**, select the **Content**.
  - d. In the field next to **Value**, select **My File Connection**.
5. Perform the following steps to add `MySourceObject` to the **Input Fields** section of **Data Task 1**:
  - a. Click the **Add** icon, and then navigate to and select `MySourceObject`.
  - b. Click **Edit**. The **Edit Value** dialog box opens.
  - c. In the field next to **Source**, select **Content**.
  - d. In the field next to **Value**, enter **input.txt**.
6. Perform the following steps to add `MyTargetConnection` to the **Input Fields** section of **Data Task 1**:
  - a. Click the **Add** icon, and then navigate to and select `MyTargetConnection`.
  - b. Click **Edit**. The **Edit Value** dialog box opens.
  - c. In the field next to **Source**, select **Content**.
  - d. In the field next to **Value**, select **My File Connection**.
7. Perform the following steps to add `MyTargetObject` to the **Input Fields** section of **Data Task 1**:
  - a. Click the **Add** icon, and then navigate to and select `MyTargetObject`.
  - b. Click **Edit**. The **Edit Value** dialog box opens.
  - c. In the field next to **Source**, select **Content**.
  - d. In the field next to **Value**, enter **output10.txt**.

The following image shows the `Input Fields` of `Data Task 1`:



Data Task 1 Properties			
General	Input Fields (4)		
Data Task	<b>Name</b>	<b>Type</b>	<b>Value</b>
<b>Input Fields</b>	MySourceConnection	Connection	My File Connection
Events	MySourceObject	Text	input.txt
Error Handling	MyTargetConnection	Connection	My File Connection
	MyTargetObject	Text	output10.txt

You have overridden the target data object parameter, `MyTargetObject` from the default value **output5.txt** to **output10.txt**.

**Note:** You only need to add parameters that you want to override to the Input Fields section. In this example, you only override `MyTargetObject`. However, you have the option of overriding `MySourceObject`, `MySourceObject`, and `MyTargetConnection` as well.

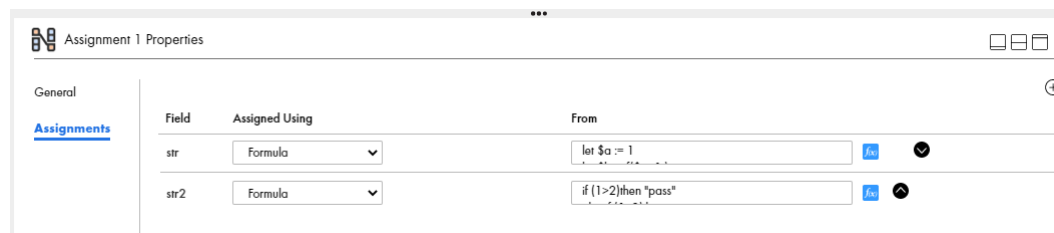
# CHAPTER 7

## The Expression Editor

Use fields, functions, and operators to create expressions in the Expression Editor.

If you assign the value **Formula** to a field, you must create a formula or an expression for the field to take data from. Use the Expression Editor to create expressions.

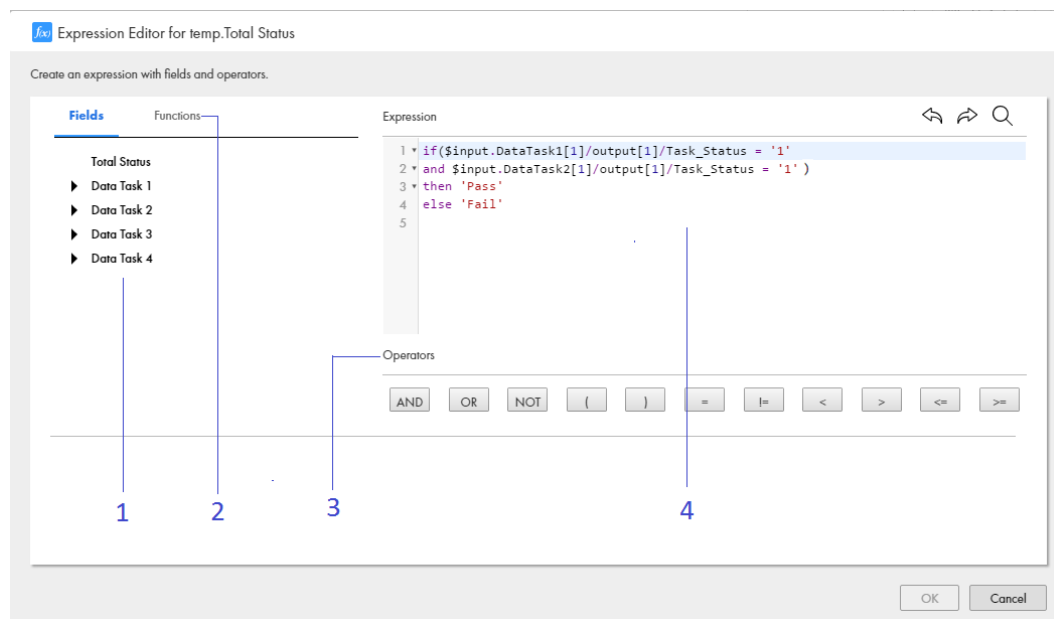
In the following image, fields **str** and **str2** have the value **Formula**:



**Note:** The Expression Editor comes with powerful functions that can invoke operating system features. You must review the content passed into the functions before using them.

To open the Expression Editor, click **f(x)** next to a field with the value **Formula**.

The following image shows the **Expression Editor** dialog box:



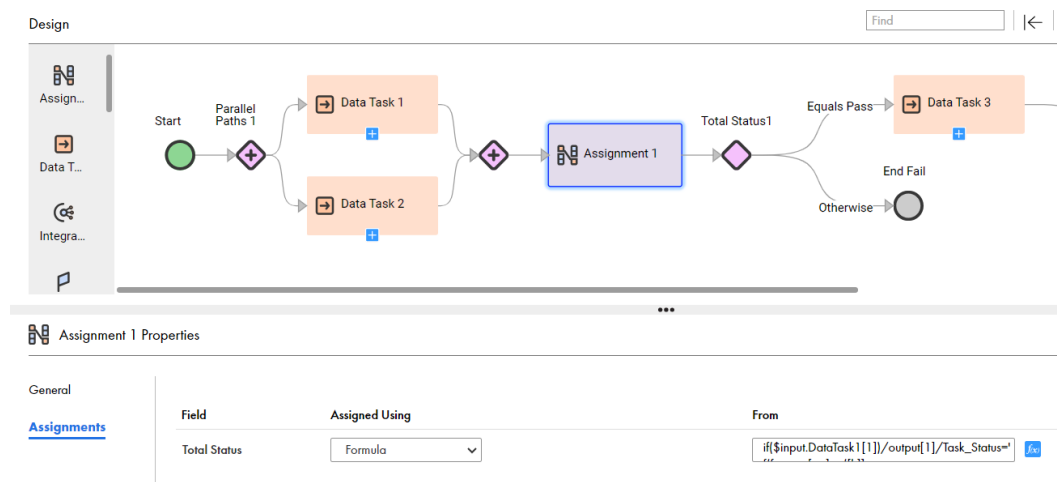
The Expression Editor contains the following sections:

- Section 1, the **Fields** section. A list of input, output, and temporary fields that you define appears here.
- Section 2, the **Functions** section. A list of common XQuery functions appears here. Select a function to view its meaning.
- Section 3, the **Operators** section. A list of operators that you can use to build an expression appears here.
- Section 4, the **Expression** section. The Expression you build appears here. The conditions and operators that you use are case sensitive.

The expression in the image defines a temporary field, **Total Status**, as `Pass` if three tasks that run in parallel succeed.

Next, the taskflow uses **Total Status** in a Data Decision step. If the value of **Total Status** is `Pass`, the taskflow runs another Data Task. If the value of **Total Status** is `Fail`, the taskflow ends.

The following image shows the taskflow that uses **Total Status**:



Use the following options to create an expression:

- To add a field, click the **Fields** tab, drill down to the field that you want to use, and click **Add**.
- To add an operator, click an operator in the **Operators** section. You can also manually enter an operator. For example, manually enter the `If` operator.
- To add a function, click the **Functions** tab, drill down to the function you want to use, and click **Add**.
- To add a comment, enter the comment in the Expression section with the following syntax:  
`(:<comment>:)`.  
For example, enter `(:This is a sample comment:)`.

Use comments to give descriptive information about the expression or to specify a URL to access business documentation about the expression.

The Expression Editor validates the expression as you enter it. You cannot save an expression that is not valid.

# Tips: Using XQuery 3.0 to create expressions

Use XQuery version 3.0 to create expressions in the Expression Editor. The samples in this topic show you the syntax and the elements that you use to construct single statement and multi statement XQuery expressions.

For information about XQuery 3.0, see <https://www.w3.org/TR/xquery-30/>.

## Single statement expression

The following expression is a single statement expression:

```
concat("Hello", " ", $input.n1)
```

The following notes explain the parts of this expression:

- `concat` is a function that joins two or more values into a single string.
- `"hello", " " $input.n1` are the parameters of the function `concat`.
  - `"hello"` is a string. Always include a string within quotes. You may use single or double quotes. However, ensure that you use the same style within an expression.
  - `n1` is an input variable. When you add it to an expression, the Expression Editor converts it to `$input.n1`. Always prefix a variable with a `$`. Do not add quotes around variables.
  - The parameter `" "` denotes space.
- Include parameters within parentheses.
- Separate parameters with commas.

Assume that the value of `$n1` is "World".

If you run:

```
concat("Hello", " ", $input.n1)
```

you get the following output:

```
Hello World
```

## Multi-statement expression

The following expression is a multi-statement expression:

```
let $n1 := number($input.n1)
let $n2 := number($input.n2)

let $r1 := if ($n1 > $n2)
            then "Greater: N1 > N2"
            else if ($n1 < $n2)
            then "Less: N1 < N2"
            else "Same"

return $r1
```

The following notes explain the parts of this expression:

- First, you declare the variables `$n1` and `$n2`. Use the operator `:=` to assign a value.  
**Note:** Even if you defined `$n1` and `$n2` as integers in an Assignment step, you must declare them as numbers in the Expression Editor.
- You can use XQuery keywords such as `let`, `if`, `then`, and `else`. They are case sensitive.  
**Important:** If you start an expression with the keyword `let`, you must end the expression with the keyword `return`.



- We use the Expression Editor to define the value of a third variable \$r1 using the following rules:
  - If the value of \$n1 is greater than the value of \$n2, \$r1 takes the value: Greater: N1 > N2.
  - If the value of \$n1 is less than the value of \$n2, \$r1 takes the value: Less: N1 < N2.
  - If the values of \$n1 and \$n2 are the same, \$r1 takes the value: Same.

Assume that the value of \$n1 is 20 and the value of \$n2 is 250.

If you run:

```
let $n1 := number($input.n1)
let $n2 := number($input.n2)

let $r1 := if ($n1 > $n2)
           then "Greater: N1 > N2"
           else if ($n1 < $n2)
           then "Less: N1 < N2"
           else "Same"

return $r1
```

you get the following output:

```
Less: N1<N2
```

Here, \$r1 now has the value

```
Less: N1<N2
```

## Keyboard shortcuts

You can use keyboard shortcuts when you create an expression.

To use keyboard shortcuts, place the pointer inside the Expression section.

The following keyboard shortcuts are available:

Action	Shortcut
Undo	Ctrl+Z
Redo	Ctrl+Y
Copy	Ctrl+C
Cut	Ctrl+X
Paste	Ctrl+V
Find	Ctrl+F
Indent four spaces	Tab
Show list of available variables	\$
Code Completion, that is, show a list of available insertions. The insertions might be name spaces, functions, fields, or common code fragments.	Ctrl+Space

# CHAPTER 8

## Taskflow functions

You can use several functions in the Taskflow Expression Editor.

Some of the key functions are described below:

### Asset detail functions

You can use the following asset detail functions from the **Miscellaneous** section of the Expression Editor:

- `getAssetLocation`
- `getAssetName`
- `getInstanceStartTime`

### Character functions

You can use the following character functions from the **Strings** section of the Expression Editor:

- `instr`
- `lpad`
- `ltrim`
- `rtrim`

### Conversion functions

You can use the following conversion functions from the **Dates and Times**, **Miscellaneous**, or **String** sections of the Expression Editor:

- `toChar(Numbers)`
- `toDate`
- `toDecimal`
- `toInteger`

### Data cleansing functions

You can use the following data cleansing function from the **Miscellaneous** section of the Expression Editor:

- `in`

### Date functions

You can use the following date functions from the **Dates and Times** section of the Expression Editor:

- `addToDate`
- `dateDiff`
- `getDatePart`

- lastDay
- trunc

#### Numeric functions

You can use the following numeric function from the **Numbers** section of the Expression Editor:

- round(Numbers)
- trunc

#### Organization detail functions

You can use the following organization detail functions from the **Miscellaneous** section of the Expression Editor:

- getDefaultFailureEmailNotification
- getDefaultSuccessEmailNotification
- getDefaultWarningEmailNotification
- getOrganizationName

#### Test functions

You can use the following test functions from the **Miscellaneous** section of the Expression Editor:

- decode
- iif
- isNull

When you use an integer type variable in a function, explicit casting of number is required for best results. For example, rewrite `$output.current` to `number($output.current)` in functions where you pass an integer type variable.

## addToDate

Adds a specified amount to one part of a datetime value, and returns a date in the same format as the date you pass to the function. The `addToDate` function accepts positive and negative integer values. Use `addToDate` to change the following parts of a date:

- **Year.** Enter a positive or negative integer in the *amount* argument. Use any of the year format strings: Y, YY, YYY, or YYYY. The following expression adds 10 years to all dates in the SHIP\_DATE column:
 

```
date:addToDate(xs:dateTime('SHIP_DATE'), 'YY', 10)
```
- **Month.** Enter a positive or negative integer in the *amount* argument. Use any of the month format strings: MM, MON, MONTH. The following expression subtracts 10 months from each date in the SHIP\_DATE column:
 

```
date:addToDate(xs:dateTime('SHIP_DATE'), 'MONTH', -10)
```
- **Day.** Enter a positive or negative integer in the *amount* argument. Use any of the day format strings: D, DD, DDD, DY, and DAY. The following expression adds 10 days to each date in the SHIP\_DATE column:
 

```
date:addToDate(xs:dateTime('SHIP_DATE'), 'DD', 10)
```
- **Hour.** Enter a positive or negative integer in the *amount* argument. Use any of the hour format strings: HH, HH12, HH24. The following expression adds 14 hours to each date in the SHIP\_DATE column:
 

```
date:addToDate(xs:dateTime('SHIP_DATE'), 'HH', 14)
```

- **Minute.** Enter a positive or negative integer in the *amount* argument. Use the MI format string to set the minute. The following expression adds 25 minutes to each date in the SHIP\_DATE column:  

```
date:addToDate(xs:dateTime('SHIP_DATE'), 'MI', 25)
```
- **Seconds.** Enter a positive or negative integer in the *amount* argument. Use the SS format string to set the second. The following expression adds 59 seconds to each date in the SHIP\_DATE column:  

```
date:addToDate(xs:dateTime('SHIP_DATE'), 'SS', 59)
```
- **Milliseconds.** Enter a positive or negative integer in the *amount* argument. Use the MS format string to set the milliseconds. The following expression adds 125 milliseconds to each date in the SHIP\_DATE column:  

```
date:addToDate(xs:dateTime('SHIP_DATE'), 'MS', 125)
```
- **Microseconds.** Enter a positive or negative integer in the *amount* argument. Use the US format string to set the microseconds. The following expression adds 2,000 microseconds to each date in the SHIP\_DATE column:  

```
date:addToDate(xs:dateTime('SHIP_DATE'), 'US', 2000)
```

## Syntax

```
date:addToDate(xs:dateTime('date'), 'format', amount)
```

**Note:** You must manually add the xs:dateTime phrase and enclose the date values within single quotation marks.

The following table describes the arguments:

Argument	Required/Optional	Description
<i>date</i>	Required	Date/Time data type. Passes the values that you want to change. You can enter any valid transformation expression.
<i>format</i>	Required	A format string that specifies the portion of the date value that you want to change. Enclose the format string within single quotation marks, for example, 'mm'. The format string is not case sensitive.
<i>amount</i>	Required	An integer value that specifies the amount of years, months, days, hours, and so on by which you want to change the date value. You can enter any valid transformation expression that evaluates to an integer.

## Return Value

Date in the same format as the date you pass to this function.

NULL if a null value is passed as an argument to the function.

## Examples

The following expressions all add one month to each date in the DATE\_SHIPPED column. If you pass a value that creates a day that does not exist in a particular month, addToDate returns the last day of the month. For example, if you add one month to Jan 31 1998, addToDate returns Feb 28 1998.

Also, addToDate recognizes leap years and adds one month to Jan 29 2000:

```
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'MM', 1)
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'MON', 1)
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'MONTH', 1)
```

The following table shows some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Jan 12 1998 12:00:30AM	Feb 12 1998 12:00:30AM
Jan 31 1998 6:24:45PM	Feb 28 1998 6:24:45PM
Jan 29 2000 5:32:12AM	Feb 29 2000 5:32:12AM <i>(Leap Year)</i>
Oct 9 1998 2:30:12PM	Nov 9 1998 2:30:12PM
NULL	NULL

The following expressions subtract 10 days from each date in the DATE\_SHIPPED column:

```

date:addToDate(xs:dateTime('DATE_SHIPPED'), 'D', -10)
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'DD', -10)
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'DDD', -10)
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'DY', -10)
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'DAY', -10)

```

The following table shows some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:30AM	Dec 22 1996 12:00AM
Jan 31 1997 6:24:45PM	Jan 21 1997 6:24:45PM
Mar 9 1996 5:32:12AM	Feb 29 1996 5:32:12AM <i>(Leap Year)</i>
Oct 9 1997 2:30:12PM	Sep 30 1997 2:30:12PM
Mar 3 1996 5:12:20AM	Feb 22 1996 5:12:20AM
NULL	NULL

The following expressions subtract 15 hours from each date in the DATE\_SHIPPED column:

```

date:addToDate(xs:dateTime('DATE_SHIPPED'), 'HH', -15)
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'HH12', -15)
date:addToDate(xs:dateTime('DATE_SHIPPED'), 'HH24', -15)

```

The following table shows some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:30AM	Dec 31 1996 9:00:30AM
Jan 31 1997 6:24:45PM	Jan 31 1997 3:24:45AM
Oct 9 1997 2:30:12PM	Oct 8 1997 11:30:12PM
Mar 3 1996 5:12:20AM	Mar 2 1996 2:12:20PM
Mar 1 1996 5:32:12AM	Feb 29 1996 2:32:12PM <i>(Leap Year)</i>
NULL	NULL

# base64Decode

Returns the base64-decoded version of the input string provided based on the character set specified in the `charSet` argument. This function is typically used for attachments.

## Syntax

```
util:base64Decode(data, charSet)
```

The following table describes the arguments:

Argument	Required/Optional	Description
<i>data</i>	Required	String data type. Data that you want to decode.
<i>charSet</i>	Optional	Character decoding of the data. Taskflows support the character sets that Azul JDK supports for encoding. For example, US-ASCII, ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, and UTF-16. Default is UTF-8.

## Return Value

Decoded value.

NULL if the input is a null value.

## Example

You encoded MQSeries message IDs and wrote them to a flat file. You want to read data from the flat file source, including the MQSeries message IDs. You can use `base64Decode` to decode the IDs and convert them to their original string value.

# dateDiff

Returns the length of time between two dates. You can specify the format as years, months, days, hours, minutes, seconds, milliseconds, or microseconds. The `dateDiff` function subtracts the second date from the first date and returns the difference.

The `dateDiff` function calculates the value based on the number of months instead of the number of days. It calculates the date differences for partial months with the days selected in each month. To calculate the date difference for the partial month, `dateDiff` adds the days used within the month. It then divides the value with the total number of days in the selected month.

The `dateDiff` function gives a different value for the same period in the leap year period and a non-leap year period. The difference occurs when February is part of the `dateDiff` function. The `dateDiff` function divides the days with 29 for February for a leap year and 28 if it is not a leap year.

For example, you want to calculate the number of months from September 13 to February 19. In a leap year period, `dateDiff` calculates the month of February as 19/29 months or 0.655 months. In a non-leap year period, `dateDiff` calculates the month of February as 19/28 months or 0.678 months. The `dateDiff` function similarly calculates the difference in the dates for the remaining months and the `dateDiff` function returns the total value for the specified period.

**Note:** Some databases might use a different algorithm to calculate the difference in dates.

## Syntax

```
date:dateDiff(xs:dateTime('date'), xs:dateTime('date'), 'format')
```

**Note:** You must manually add the `xs:dateTime` phrase and enclose the date values within single quotation marks.

The following table describes the arguments:

Argument	Required/Optional	Description
<i>date</i>	Required	Date/Time data type. Passes the values for the first date that you want to compare. You can enter any valid transformation expression.
<i>date</i>	Required	Date/Time data type. Passes the values for the second date that you want to compare. You can enter any valid transformation expression.
<i>format</i>	Required	Format string that specifies the date or time measurement. You can specify years, months, days, hours, minutes, seconds, milliseconds, or microseconds. You can specify only one part of the date, such as 'mm'. Enclose the format strings within single quotation marks. The format string is not case sensitive. For example, the format string 'mm' is the same as 'MM', 'Mm', or 'mM'.

## Return Value

Double value. If the first date is later than the second date, the return value is a positive number. If the first date is earlier than the second date, the return value is a negative number.

0 if the dates are the same.

NULL if one (or both) of the date values is NULL.

## Examples

The following expressions return the number of hours between the `DATE_PROMISED` and `DATE_SHIPPED` columns:

```
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'HH')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'HH12')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'HH24')
```

The following table lists some sample values and return values:

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:00AM	Mar 29 1997 12:00:00PM	-2100
Mar 29 1997 12:00:00PM	Jan 1 1997 12:00:00AM	2100
NULL	Dec 10 1997 5:55:10PM	NULL
Dec 10 1997 5:55:10PM	NULL	NULL

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jun 3 1997 1:13:46PM	Aug 23 1996 4:20:16PM	6812.89166666667
Feb 19 2004 12:00:00PM	Feb 19 2005 12:00:00PM	-8784

The following expressions return the number of days between the DATE\_PROMISED and the DATE\_SHIPPED columns:

```

date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'D')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'DD')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'DDD')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'DY')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'DAY')

```

The following table lists some sample values and return values:

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:00AM	Mar 29 1997 12:00:00PM	-87.5
Mar 29 1997 12:00:00PM	Jan 1 1997 12:00:00AM	87.5
NULL	Dec 10 1997 5:55:10PM	NULL
Dec 10 1997 5:55:10PM	NULL	NULL
Jun 3 1997 1:13:46PM	Aug 23 1996 4:20:16PM	283.870486111111
Feb 19 2004 12:00:00PM	Feb 19 2005 12:00:00PM	-366

The following expressions return the number of months between the DATE\_PROMISED and DATE\_SHIPPED columns:

```

date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'MM')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'MON')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'MONTH')

```

The following table lists some sample values and return values:

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:00AM	Mar 29 1997 12:00:00PM	-2.91935483870968
Mar 29 1997 12:00:00PM	Jan 1 1997 12:00:00AM	2.91935483870968
NULL	Dec 10 1997 5:55:10PM	NULL
Dec 10 1997 5:55:10PM	NULL	NULL
Jun 3 1997 1:13:46PM	Aug 23 1996 4:20:16PM	9.3290162037037
Feb 19 2004 12:00:00PM	Feb 19 2005 12:00:00PM	-12



The following expressions return the number of years between the DATE\_PROMISED and DATE\_SHIPPED columns:

```
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'Y')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'YY')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'YYY')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'YYYY')
```

The following table lists some sample values and return values:

DATE_PROMISED	DATE_SHIPPED	RETURN VALUE
Jan 1 1997 12:00:00AM	Mar 29 1997 12:00:00PM	-0.24327956989247
Mar 29 1997 12:00:00PM	Jan 1 1997 12:00:00AM	0.24327956989247
NULL	Dec 10 1997 5:55:10PM	NULL
Dec 10 1997 5:55:10PM	NULL	NULL
Jun 3 1997 1:13:46PM	Aug 23 1996 4:20:16PM	0.77741801697531
Feb 19 2004 12:00:00PM	Feb 19 2005 12:00:00PM	-1

The following expressions return the number of months between the DATE\_PROMISED and DATE\_SHIPPED columns:

```
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'MM')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'MON')
date:dateDiff(xs:dateTime('DATE_PROMISED'), xs:dateTime('DATE_SHIPPED'), 'MONTH')
```

The following table lists some sample values and return values:

DATE_PROMISED	DATE_SHIPPED	LEAP YEAR VALUE (in Months)	NON-LEAP YEAR VALUE (in Months)
Sept 13	Feb 19	-5.237931034	-5.260714286
NULL	Feb 19	NULL	N/A
Sept 13	NULL	NULL	N/A

## decode

Searches a column for a value that you specify. If the function finds the value, it returns a result value, which you define. You can build an unlimited number of searches within a decode function.

If you use decode to search for a value in a string column, you can either trim trailing blank characters with the rtrim function or include the blanks in the search string.

### Syntax

```
util:decode(value, search1, result1, args, default)
```

The following table describes the arguments:

Argument	Required/Optional	Description
<i>value</i>	Required	Passes the values that you want to search. You can enter any valid transformation expression. You can pass any data type except Binary. To pass a NULL value, you must specify an empty sequence in the following format: ()
<i>search1</i>	Required	Passes the values for which you want to search. You can enter any valid transformation expression. You can pass any value with the same data type as the value argument. The search value must match the value argument. You cannot search for a portion of a value. Also, the search value is case sensitive. For example, if you want to search for the string 'Halogen Flashlight' in a particular column, you must enter 'Halogen Flashlight, not just 'Halogen'. If you enter 'Halogen', the search does not find a matching value. To pass a NULL value, you must specify an empty sequence in the following format: ()
<i>result1</i>	Required	The value that you want to return if the search finds a matching value. You can enter any valid transformation expression and pass any data type except Binary. To pass a NULL value, you must specify an empty sequence in the following format: ()
<i>args</i>	Required	Pairs of search values and result values separated by a comma. For example, use the following syntax: <code>util:decode(value, search1, result1, search2, result2, searchn, resultn, default)</code> To pass a NULL value, you must specify an empty sequence in the following format: ()
<i>default</i>	Required	The value that you want to return if the search does not find a matching value. You can enter any valid transformation expression and pass any data type except Binary. To pass a NULL value, you must specify an empty sequence in the following format: ()

## Return Value

*result1* if the search finds a matching value.

*default* value if the search does not find a matching value.

NULL if you omit the default argument and the search does not find a matching value.

Even if multiple conditions are met, decode returns the first matching result.

## decode and data types

When you use decode, the data type of the return value is always the same as the data type of the result with the greatest precision.

For example, you have the following expression:

```
util:decode( CONST_NAME
             'Five', 5,
             'Pythagoras', 1.414213562,
             'Archimedes', 3.141592654,
             'Pi', 3.141592654 )
```

The return values in this expression are 5, 1.414213562, and 3.141592654. The first result is an integer, and the other results are decimal. The decimal data type has a greater precision than the integer data type. This expression always writes the result as a decimal value.

You cannot create a decode function with both string and numeric return values.

For example, the following expression is not valid:

```
util:decode( CONST_NAME
             'Five', 5,
             'Pythagoras', '1.414213562',
             'Archimedes', '3.141592654',
             'Pi', 3.141592654 )
```

When you validate the expression above, you receive the following error message:

```
Function cannot resolve operands of ambiguously mismatching datatypes.
```

## Examples

You might use decode in an expression that searches for a particular ITEM\_ID and returns the ITEM\_NAME:

```
util:decode( ITEM_ID, 10, 'Flashlight',
            14, 'Regulator',
            20, 'Knife',
            40, 'Tank',
            'NONE' )
```

The following table lists some sample values and return values:

ITEM_ID	RETURN VALUE
10	Flashlight
14	Regulator
17	NONE
20	Knife
25	NONE
NULL	NONE
40	Tank

The decode function returns the default value of NONE for items 17 and 25 because the search values did not match the ITEM\_ID. Also, decode returns NONE for the NULL ITEM\_ID.

The following expression tests multiple columns and conditions, evaluated in a top to bottom order for TRUE or FALSE:

```
util:decode( TRUE,
            Var1 = 22, 'Variable 1 was 22!',
            Var2 = 49, 'Variable 2 was 49!',
            Var1 < 23, 'Variable 1 was less than 23.',
            Var2 > 30, 'Variable 2 was more than 30.',
            'Variables were out of desired ranges.')
```

The following table lists some sample values and return values:

Var1	Var2	RETURN VALUE
21	47	Variable 1 was less than 23.
22	49	Variable 1 was 22!
23	49	Variable 2 was 49!

Var1	Var2	RETURN VALUE
24	27	Variables were out of desired ranges.
25	50	Variable 2 was more than 30.

## getAssetLocation

Returns the location where the taskflow that uses the function is stored.

For example, you can use the function to find the taskflow location for debugging purposes.

### Syntax

```
util:getAssetLocation()
```

The `getAssetLocation` function does not use an argument.

### Return Value

Location where the taskflow that uses the function is stored.

### Example

If you use the `getAssetLocation` function in a taskflow that is stored under `Default\Orders`, the function returns the following value:

```
Default\Orders
```

## getAssetName

Returns the name of the taskflow that uses the function.

For example, you can use the function in a Notification Task step to include the taskflow name in a taskflow failure email notification that you send to stakeholders.

### Syntax

```
util:getAssetName()
```

The `getAssetName` function does not use an argument.

### Return Value

Name of the taskflow that uses the function.

### Example

If you use the `getAssetName` function in a taskflow that is named Order Management, the function returns the following value:

```
Order Management
```

# getDatePart

Returns the specified part of a date as an integer value. Therefore, if you create an expression that returns the month portion of the date, and pass a date such as Apr 1 1997 00:00:00, getDatePart returns 4.

## Syntax

```
date:getDatePart(xs:dateTime('date'), 'format')
```

**Note:** You must manually add the xs:dateTime phrase and enclose the date values within single quotation marks.

The following table describes the arguments:

Argument	Required/Optional	Description
<i>date</i>	Required	Date/Time data type. You can enter any valid transformation expression.
<i>format</i>	Required	A format string that specifies the portion of the date value that you want to return. Enclose format strings within single quotation marks, for example, 'mm'. The format string is not case sensitive. For example, if you pass the date Apr 1 1997 to getDatePart, the format strings 'Y', 'YY', 'YYY', or 'YYYY' all return 1997.

## Return Value

Integer representing the specified part of the date.

NULL if a value passed to the function is NULL.

## Examples

The following expressions return the hour for each date in the DATE\_SHIPPED column. 12:00:00AM returns 0 because the default date format is based on the 24 hour interval:

```
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'HH')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'HH12')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'HH24')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Mar 13 1997 12:00:00AM	0
Sep 2 1997 2:00:01AM	2
Aug 22 1997 12:00:00PM	12
June 3 1997 11:30:44PM	23
NULL	NULL

The following expressions return the day for each date in the DATE\_SHIPPED column:

```
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'D')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'DD')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'DDD')
```

```
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'DY')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'DAY')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Mar 13 1997 12:00:00AM	13
June 3 1997 11:30:44PM	3
Aug 22 1997 12:00:00PM	22
NULL	NULL

The following expressions return the month for each date in the DATE\_SHIPPED column:

```
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'MM')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'MON')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'MONTH')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Mar 13 1997 12:00:00AM	3
June 3 1997 11:30:44PM	6
NULL	NULL

The following expressions return the year for each date in the DATE\_SHIPPED column:

```
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'Y')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'YY')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'YYY')
date:getDatePart(xs:dateTime('DATE_SHIPPED'), 'YYYY')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Mar 13 1997 12:00:00AM	1997
June 3 1997 11:30:44PM	1997
NULL	NULL

## getInstanceStartTime

Returns the start date and start time of the running instance of the taskflow that uses the function.

For example, you can use the function to find when a taskflow started running and abort the taskflow if it has been running beyond the expected duration.

## Syntax

```
util:getInstanceStartTime()
```

The `getInstanceStartTime` function does not use an argument.

## Return Value

Start date and start time of the running instance of the taskflow that uses the function.

The return value is in the Coordinated Universal Time (UTC) format as follows:

```
YYY-MM-DDTHH:mm:ss.sssZ
```

## Example

If you use the `getInstanceStartTime` function in a taskflow that was started on January 19, 2021, the function returns the following value:

```
2021-01-19T10:11:21.047Z
```

# iif

Returns one of two values that you specify based on the results of a condition.

## Syntax

```
util:iif(condition, val1, val2)
```

The following table describes the arguments:

Argument	Required/Optional	Description
<i>condition</i>	Required	The condition that you want to evaluate. You can enter any valid transformation expression that evaluates to TRUE or FALSE.
<i>val1</i>	Required	The value that you want to return if the condition is TRUE. The return value is always the data type specified by this argument. You can enter any valid transformation expression, including another <code>iif</code> expression. You can pass any data type except Binary.
<i>val2</i>	Optional	The value that you want to return if the condition is FALSE. You can enter any valid transformation expression, including another <code>iif</code> expression. You can pass any data type except Binary.

The FALSE (*val2*) condition in the `iif` function is not required. If you omit *val2*, the function returns one of the following values when the condition is FALSE:

- 0 if *val1* is a Numeric data type.
- Empty string if *val1* is a String data type.
- NULL if *val1* is a Date/Time data type.

For example, the following expression does not include a FALSE condition and *val1* is a string data type so `decode` returns an empty string for each row that evaluates to FALSE:

```
util:iif(SALES > 100, EMP_NAME)
```

The following table lists some sample values and return values:

SALES	EMP_NAME	RETURN VALUE
150	John Smith	John Smith
50	Pierre Bleu	' ' (empty string)
120	Sally Green	Sally Green
NULL	Greg Jones	' ' (empty string)

## Return Value

*va1* if the condition is TRUE.

*va2* if the condition is FALSE.

For example, the following expression includes the FALSE condition NULL so decode returns NULL for each row that evaluates to FALSE:

```
util:iif(SALES > 100, EMP_NAME, NULL)
```

The following table lists some sample values and return values:

SALES	EMP_NAME	RETURN VALUE
150	John Smith	John Smith
50	Pierre Bleu	NULL
120	Sally Green	Sally Green
NULL	Greg Jones	NULL

## iif and data types

When you use iif, the data type of the return value is the same as the data type of the result with the greatest precision.

For example, you have the following expression:

```
util:iif(SALES < 100, 1, .3333)
```

The TRUE result (1) is an integer and the FALSE result (.3333) is a decimal. The Decimal data type has a greater precision than the Integer data type. Therefore, the data type of the return value is always a decimal value.

## Special uses of iif

Use nested iif statements to test multiple conditions. The following example tests for various conditions and returns 0 if sales is 0 or negative:

```
util:iif(SALES > 0, util:iif(SALES < 50, SALARY1, util:iif(SALES < 100, SALARY2,
util:iif(SALES < 200, SALARY3, BONUS))), 0 )
```



# in

Matches input data to a list of values. By default, the match is case sensitive.

## Syntax

```
util:in(valueToSearch, values, caseFlag)
```

The following table describes the arguments:

Argument	Required/Optional	Description
valueToSearch	Required	Can be a string, date, or numeric value. Input value that you want to match against a comma-separated list of values. To pass a NULL value, you must specify an empty sequence in the following format: ()
values	Required	Can be a string, date, or numeric value. Comma-separated list of values that you want to search for. Values can be columns. There is no limit to the maximum number of values that you can list. To pass a NULL value, you must specify an empty sequence in the following format: ()
caseFlag	Optional	Must be an integer. Determines whether the arguments in this function are case sensitive. You can enter any valid transformation expression. When <i>caseFlag</i> is a number other than 0, the function is case sensitive. When <i>caseFlag</i> is a null value or 0, the function is not case sensitive.

## Return Value

TRUE (1) if the input value matches the list of values.

FALSE (0) if the input value does not match the list of values.

NULL if the input is a null value.

## Example

The following expression determines if the input value is a safety knife, chisel point knife, or medium titanium knife. The input values do not have to match the case of the values in the comma-separated list:

```
util:in(ITEM_NAME, 'Chisel Point Knife', 'Medium Titanium Knife', 'Safety Knife', 0)
```

The following table lists some sample values and return values:

ITEM_NAME	RETURN VALUE
Stabilizing Vest	0 (FALSE)
Safety knife	1 (TRUE)
Medium Titanium knife	1 (TRUE)
	NULL

# instr

Returns the position of a character set in a string, counting from left to right.

## Syntax

```
sff:instr(str, search, start, occurrence, comparison_type)
```

The following table describes the arguments:

Argument	Required/Optional	Description
<i>str</i>	Required	Passes the value that you want to evaluate. The string must be a character string. You can enter any valid transformation expression. The results of the expression must be a character string. Otherwise, instr converts the value to a string before evaluating it.
<i>search</i>	Required	The set of characters that you want to search for. You can enter any valid transformation expression. If you want to search for a character string, enclose the characters that you want to search for within single or double quotation marks. The value must match a part of the string. For example, if you use <code>instr('Alfred Pope', 'Alfred Smith')</code> the function returns 0. The value is case sensitive.
<i>start</i>	Optional	The position in the string where you want to start the search. You can enter any valid transformation expression. The value must be an integer. The default is 1, meaning that instr starts the search at the first character in the string. If the start position is 0, instr searches from the first character in the string. If the start position is a positive number, instr locates the start position by counting from the beginning of the string. If the start position is a negative number, instr locates the start position by counting from the end of the string. If you omit this argument, the function uses the default value of 1.

Argument	Required/Optional	Description
<i>occurrence</i>	Optional	<p>You can enter any valid transformation expression. If the search value appears more than once in the string, you can specify which occurrence you want to search for. For example, you would enter 2 to search for the second occurrence from the start position.</p> <p>You can enter a positive integer that is greater than 0.</p> <p>If you omit this argument, the function uses the default value of 1, which means that <i>instr</i> searches for the first occurrence of the search value. If you pass a decimal value, the function rounds it to the nearest integer value. If you pass a negative integer or 0, the function is not valid.</p>
<i>comparison_type</i>	Optional	<p>The string comparison type, either linguistic or binary.</p> <p>Linguistic comparisons take language-specific collation rules into account, while binary comparisons perform bitwise matching. For example, the German sharp s character matches the string "ss" in a linguistic comparison, but not in a binary comparison. Binary comparisons run faster than linguistic comparisons.</p> <p>You must enter one of the following integer values:</p> <ul style="list-style-type: none"> <li>- 0: <i>instr</i> performs a linguistic string comparison.</li> <li>- 1: <i>instr</i> performs a binary string comparison.</li> </ul> <p>Default is 0.</p>

## Return Value

Integer if the search is successful. Integer represents the position of the first character in the *search* argument, counting from left to right.

0 if the search is unsuccessful.

NULL if a value passed to *instr* is NULL.

## Examples

The following expression returns the position of the first occurrence of the letter 'a', starting from the beginning of each company name:

```
sff:instr( COMPANY, 'a' )
```

The following table lists some sample values and return values:

COMPANY	RETURN VALUE
Blue Fin Aqua Center	13
Maco Shark Shop	2
Scuba Gear	5
Frank's Dive Shop	3
VIP Diving Club	0

Because the *search* argument is case sensitive, it skips the 'A' in 'Blue Fin Aqua Center', and returns the position for the 'a' in 'Aqua'.

The following expression returns the position of the second occurrence of the letter 'a', starting from the beginning of each company name:

```
sff:instr( COMPANY, 'a', 1, 2 )
```

The following table lists some sample values and return values:

COMPANY	RETURN VALUE
Blue Fin Aqua Center	0
Maco Shark Shop	8
Scuba Gear	9
Frank's Dive Shop	0
VIP Diving Club	0

Because the *search* argument is case sensitive, it skips the 'A' in 'Blue Fin Aqua Center', and returns 0.

The following expression returns the position of the second occurrence of the letter 'a' in each company name, starting from the last character in the company name.

```
sff:instr( COMPANY, 'a', -1, 2 )
```

The following table lists some sample values and return values:

COMPANY	RETURN VALUE
Blue Fin Aqua Center	0
Maco Shark Shop	2
Scuba Gear	5
Frank's Dive Shop	0
VIP Diving Club	0

Because the *search* argument is case sensitive, it skips the 'A' in 'Blue Fin Aqua Center', and returns 0.

The following expression returns the position of the first character in the string 'Blue Fin Aqua Center', starting from the last character in the company name:

```
sff:instr( COMPANY, 'Blue Fin Aqua Center', -1, 1 )
```

The following table lists some sample values and return values:

COMPANY	RETURN VALUE
Blue Fin Aqua Center	1
Maco Shark Shop	0
Scuba Gear	0

COMPANY	RETURN VALUE
Frank's Dive Shop	0
VIP Diving Club	0

## isNull

Returns whether a value is NULL.

**Note:** The isNull function evaluates an empty string as FALSE.

### Syntax

```
util:isNull(value)
```

The following table describes the argument for this command:

Argument	Required/Optional	Description
<i>value</i>	Required	Passes the rows that you want to evaluate. You can enter any valid transformation expression. You can pass a value of any data type except Binary.

### Return Value

TRUE (1) if the value is NULL.

FALSE (0) if the value is not NULL.

### Example

The following example checks for NULL values in the items table:

```
util:isNull( ITEM_NAME )
```

The following table lists some sample values and return values:

ITEM_NAME	RETURN VALUE
Flashlight	0 (FALSE)
NULL	1 (TRUE)
Regulator system	0 (FALSE)
' '	0 (FALSE) <i>Empty string is not NULL</i>

## lastDay

Returns the date of the last day of the month for each date in a column.

## Syntax

```
date:lastDay(date)
```

The following table describes the argument for this command:

Argument	Required/Optional	Description
<i>date</i>	Required	Date/Time data type. Passes the date for which you want to return the last day of the month. You can enter any valid transformation expression that evaluates to a date.

## Return Value

Date. The last day of the month for the date value that you pass to this function.

NULL if a value in the selected column is NULL.

## Example

The following expression returns the current date as the last day:

```
date:lastDay(fn:current-dateTime('DATE'))
```

The following table lists some sample values and return values:

DATE	RETURN VALUE
18-04-98 01:00	Apr 18 1998 01:00:00 AM
20-08-99 05:00	Aug 20 1999 05:00:00 AM

The following expression returns the last day of the previous month for each date in the DATE column:

```
date:lastDay(date:addToDate(fn:current-dateTime('DATE','MM',-1))
```

The following table lists some sample values and return values:

DATE	RETURN VALUE
Apr 1 1998 12:00:00AM	Mar 31 1998 12:00:00AM
Jan 6 1998 12:00:00AM	Dec 31 1997 12:00:00AM
Feb 2 1996 12:00:00AM	Jan 31 1996 12:00:00AM
NULL	NULL

You can nest `toDate` to convert string values to a date. `toDate` function always includes time information. If you pass a string that does not have a time value, the date returned will include the time 00:00:00.

The following example returns the last day of the month for each date in the same format as the string:

```
date:lastDay(toDate('DATE', 'MON-DD-YYYY'))
```

The following table lists some sample values and return values:

DATE	RETURN VALUE
'18-NOV-98'	Nov-30-1998 00:00:00

DATE	RETURN VALUE
'28-APR-98'	Apr-30-1998 00:00:00
NULL	NULL
'18-FEB-96'	Feb-29-1996 00:00:00 ( <i>Leap year</i> )

```
date:lastDay(date:toDate("DATE", "YYYY-MM-DD"))
```

The following table lists some sample values and return values:

DATE	RETURN VALUE
'18-NOV-98'	1998-Nov-30 00:00:00
'28-APR-98'	1998-Apr-30 00:00:00
NULL	NULL
'18-FEB-96'	1996-Feb-29 00:00:00 ( <i>Leap year</i> )

## lpad

Adds a set of blank characters to the beginning of a string to set the string to a specified length.

### Syntax

```
sff:lpad(first_string, length, second_string)
```

The following table describes the arguments:

Argument	Required/Optional	Description
<i>first_string</i>	Required	Can be a character string. Passes the string that you want to change. You can enter any valid transformation expression. To pass a NULL value, you must specify an empty sequence in the following format: ( )
<i>length</i>	Required	Must be a positive integer literal. This argument specifies the length that you want for each string. When <i>length</i> is a negative number, lpad returns NULL.
<i>second_string</i>	Optional	Can be any string value. The characters that you want to append to the left-side of the <i>first_string</i> values. You can enter any valid transformation expression. You can enter a specific string literal. However, enclose the characters you want to add to the beginning of the string within single quotation marks, as in 'abc'. This argument is case sensitive. If you omit the <i>second_string</i> , the function pads the beginning of the first string with blank characters. To pass a NULL value, you must specify an empty sequence in the following format: ( )

### Return Value

String of the specified length.

NULL if a value passed to the function is NULL or if *length* is a negative number.

## Example

The following expression standardizes numbers to six digits by padding them with leading zeros.

```
sff:lpad(PART_NUM, 6, '0')
```

The following table lists some sample values and return values:

<b>PART_NUM</b>	<b>RETURN VALUE</b>
702	000702
1	000001
0553	000553
484834	484834

`lpad` counts the length from left to right. If the first string is longer than the length, `lpad` truncates the string from right to left. For example, `lpad('alphabetical', 5, 'x')` returns the string 'alpha'.

If the second string is longer than the total characters needed to return the specified length, `lpad` uses a portion of the second string:

```
sff:lpad(ITEM_NAME, 16, '*..*')
```

The following table lists some sample values and return values:

<b>ITEM_NAME</b>	<b>RETURN VALUE</b>
Flashlight	*..*.Flashlight
Compass	*..*..*.Compass
Regulator System	Regulator System
Safety Knife	*..*Safety Knife

The following expression shows how `lpad` handles negative values for the `length` argument for each row in the `ITEM_NAME` column:

```
sff:lpad(ITEM_NAME, -5, '.')
```

The following table lists some sample values and return values:

<b>ITEM_NAME</b>	<b>RETURN VALUE</b>
Flashlight	NULL
Compass	NULL
Regulator System	NULL

## ltrim

Removes leading spaces or characters from the beginning of a string.



If you do not specify the *trim\_set* argument in the expression, *ltrim* removes both single-byte and double-byte spaces from the beginning of a string.

If you use *ltrim* to remove characters from a string, *ltrim* compares the *trim\_set* to each character in the *str* argument, character-by-character, starting from the left side of the string. If the character in the string matches any character in the *trim\_set*, *ltrim* removes it. The *ltrim* function continues comparing and removing characters until it fails to find a matching character in the *trim\_set*. Then it returns the string, which does not include matching characters.

## Syntax

```
sff:ltrim(str, trim_set)
```

The following table describes the arguments:

Arguments	Required/Optional	Description
<i>str</i>	Required	Any string value. Passes the strings that you want to modify. You can enter any valid transformation expression. Use operators to perform comparisons or concatenate strings before removing characters from the beginning of a string. You must enclose the string value within single or double quotation marks. To pass a NULL value, you must specify an empty sequence in the following format: ()
<i>trim_set</i>	Optional	Any string value. Passes the characters that you want to remove from the beginning of the first string. You can enter any valid transformation expression. You can also enter a character string. You must enclose the trim set value within single or double quotation marks. To pass a NULL value, you must specify an empty sequence in the following format: () The <i>ltrim</i> function is case sensitive. For example, if you want to remove the 'A' character from the string 'Alfredo', you would enter 'A', not 'a'.

## Return Value

String. The string values with the specified characters in the *trim\_set* argument removed.

NULL if a value passed to *ltrim* is NULL. If the *trim\_set* is NULL, *ltrim* returns NULL.

## Example

The following expression removes the characters 's' and '.' from the strings in the LAST\_NAME column:

```
sff:ltrim( LAST_NAME, 'S.')
```

The following table lists some sample values and return values:

LAST_NAME	RETURN VALUE
Nelson	Nelson
Osborne	Osborne
NULL	NULL
S. MacDonald	MacDonald
Sawyer	awyer

LAST_NAME	RETURN VALUE
H. Bender	H. Bender
Steadman	teadman

The `ltrim` function removes 'S.' from S. MacDonald and the 'S' from both Sawyer and Steadman, but not the period from H. Bender. This is because `ltrim` searches, character-by-character, for the set of characters you specify in the `trim_set` argument. If the first character in the string matches the first character in the `trim_set`, `ltrim` removes it. Then, `ltrim` looks at the second character in the string. If it matches the second character in the `trim_set`, `ltrim` removes it, and so on. When the first character in the string does not match the corresponding character in the `trim_set`, `ltrim` returns the string and evaluates the next row.

In the example of H. Bender, H does not match either character in the `trim_set` argument, so `ltrim` returns the string in the `LAST_NAME` column and moves to the next row.

### Tips for `ltrim`

Use `ltrim` with `CONCAT` to remove leading blank spaces after you concatenate two strings.

You can also remove multiple sets of characters by nesting `ltrim`. For example, to remove leading blank spaces and the character 'T' from a column of names, you might create an expression as follows:

```
sff:ltrim( sff:ltrim( NAMES ), 'T' )
```

## round (Numbers)

Rounds numbers to a specified number of digits or decimal places. You can also use `round` to round dates.

The `Round` function behaves as follows:

- Returns the number without the fractional part that is closest to the argument.
- If there are two numbers that are close to the argument, then the number that is closest to the positive infinity is returned.
- If the data type of the argument is one of the four numeric data types, `xs:float`, `xs:double`, `xs:decimal`, or `xs:integer`, then the data type of the result is the same as the data type of the argument.
- If the data type of the argument is a data type derived from one of the numeric data types, then the result is an instance of the base numeric data type.

### Syntax

```
fn:round(arg)
```

The following table describes the argument for this command:

Argument	Required/Optional	Description
<i>arg</i>	Required	<p>Must be a numeric data type. You can enter any valid transformation expression. Use operators to perform arithmetic before you round the values.</p> <p>To pass a NULL value, you must specify an empty sequence in the following format: ()</p> <p>The function rounds to the nearest integer, truncating the decimal portion of the number. For example, round(12.99) returns 13 and round(15.20) returns 15.</p>

## Return Value

Numeric value.

If one of the arguments is NULL, round returns NULL.

### Example

The following expression returns the rounded values in the Price column.

```
fn:round(PRICE)
```

The following table lists some sample values and return values:

PRICE	RETURN VALUE
12.99	13.0
-15.99	-16.0
-18.99	-19.0
56.95	57.0
NULL	NULL

If you want to be more specific with the precision and round the number to the nearest integer or truncate the decimal portion, it is recommended to use round-half-to-even function.

```
fn:round-half-to-even(arg, precision)
```

If you enter a positive precision, the function rounds to this number of decimal places. For example, round(12.99, 1) returns 13.0 and round(15.44, 1) returns 15.4.

If you enter a negative precision, the function rounds this number of digits to the left of the decimal point, returning an integer. For example, round(12.99, -1) returns 10 and rounds(15.99, -1) returns 20.

The value returned is the nearest, that is, numerically closest value to the argument that is a multiple of ten to the power of minus precision. If two such values are equally near, that is, if the fractional part in the argument is exactly .500..., the function returns the one whose least significant digit is even.

### Example

The following expression returns the values in the Price column rounded to three decimal places.

```
fn:round-half-to-even(PRICE, 3)
```

The following table lists some sample values and return values:

<b>PRICE</b>	<b>RETURN VALUE</b>
12.9936	12.994
15.9949	15.995
-18.8678	-18.868
56.9561	56.956
NULL	NULL

You can round digits to the left of the decimal point by passing a negative integer in the precision argument:

```
fn:round-half-to-even(PRICE, -2)
```

The following table lists some sample values and return values:

<b>PRICE</b>	<b>RETURN VALUE</b>
13242.99	13200.0
1435.99	1400.0
-108.95	-100.0
NULL	NULL

If you pass zero in the precision argument, the function rounds to the nearest integer:

```
fn:round-half-to-even(PRICE, 0)
```

The following table lists some sample values and return values:

<b>PRICE</b>	<b>RETURN VALUE</b>
12.99	13.0
-15.99	-16.0
-18.99	-19.0
56.95	57.0
NULL	NULL

## rtrim

Removes blank characters or characters from the end of a string.

If you do not specify a *trim\_set* parameter in the expression, *rtrim* removes both single-byte and double-byte spaces from the end of a string.

If you use *rtrim* to remove characters from a string, *rtrim* compares the *trim\_set* to each character in the *string* argument, character-by-character, starting with the right side of the string. If the character in the string

matches any character in the *trim\_set*, rtrim removes it. The rtrim function continues comparing and removing characters until it fails to find a matching character in the *trim\_set*. It returns the string without the matching characters.

## Syntax

```
sff:rtrim(str, trim_set)
```

The following table describes the arguments:

Argument	Required/Optional	Description
<i>string</i>	Required	Any string value. Passes the values that you want to trim. You can enter any valid transformation expression. Use operators to perform comparisons or concatenate strings before removing blank characters from the end of a string. You must enclose the string value within single or double quotation marks. To pass a NULL value, you must specify an empty sequence in the following format: ( )
<i>trim_set</i>	Optional	Any string value. Passes the characters that you want to remove from the end of the string. You can also enter a text literal. You must enclose the string value within single or double quotation marks. To pass a NULL value, you must specify an empty sequence in the following format: ( ) The rtrim function is case sensitive. For example, if you want to remove the 'o' character from the string 'Alfredo', you would enter 'o', not 'O'.

## Return Value

String. The string values with the specified characters in the *trim\_set* argument removed.

NULL if a value passed to rtrim is NULL. If the *trim\_set* is NULL, rtrim returns NULL.

## Example

The following expression removes the characters 're' from the strings in the LAST\_NAME column:

```
sff:rtrim( LAST_NAME, 're')
```

The following table lists some sample values and return values:

LAST_NAME	RETURN VALUE
Nelson	Nelson
Page	Pag
Osborne	Osborn
NULL	NULL
Sawyer	Sawy
H. Bender	H. Bend
Steadman	Steadman

The rtrim function removes 'e' from Page even though 'r' is the first character in the *trim\_set*. This is because rtrim searches, character-by-character, for the set of characters you specify in the *trim\_set* argument. If the last character in the string matches the first character in the *trim\_set*, rtrim removes it. If, however, the last

character in the string does not match, `rtrim` compares the second character in the `trim_set`. If the second from last character in the string matches the second character in the `trim_set`, `rtrim` removes it, and so on. When the character in the string fails to match the `trim_set`, `rtrim` returns the string and evaluates the next row.

In the last example, the last character in Nelson does not match any character in the `trim_set` argument, so `rtrim` returns the string 'Nelson' and evaluates the next row.

### Tips for `rtrim`

Use `rtrim` with `CONCAT` to remove trailing blank characters after you concatenate two strings.

You can also remove multiple sets of characters by nesting `rtrim`. For example, to remove trailing blank characters and the character 't' from the end of each string in a column of names, you might create an expression similar to the following:

```
sff:trim( sff:rtrim( NAMES ), 't' )
```

## toChar (Numbers)

Converts numeric values to text strings.

### Syntax

```
sff:toChar(xs:double(val))
```

**Note:** After you add the function, you must manually add the phrase `(xs:double)` in the syntax. Otherwise, the taskflow fails.

The following table describes the arguments:

Argument	Required/Optional	Description
<i>val</i>	Required	The numeric value that you want to convert to a string. You can enter any valid transformation expression.

`toChar` converts double values to text strings as follows:

- Converts double values of up to 16 digits to strings and provides accuracy up to 15 digits. If you pass a number with more than 15 digits, `toChar` rounds the number based on the sixteenth digit and returns the string representation of the number in scientific notation. For example, 1234567890123456 double value converts to '1.23456789012346e+015' string value.
- Returns decimal notation for numbers in the ranges  $(-1e16, -1e-16]$  and  $[1e-16, 1e16)$ . `toChar` returns scientific notation for numbers outside these ranges. For example, 10842764968208837340 double value converts to '1.08427649682088e+019' string value.

`toChar` converts decimal values to text strings as follows:

- In low precision mode, `toChar` treats decimal values as double values.
- If you pass a decimal port to the `toChar` function and the input value does not have enough digits to match the scale of the decimal port, the `toChar` function appends zeros to the value.

For example, if the scale of the decimal port is 5 and the value in a row is 7.6901, the `toChar` function treats the input value as 7.69010 and the return value is '7.69010.'

## Return Value

String.

NULL if a value passed to the function is NULL.

## Double Conversion Example

The following expression converts the double values in the SALES port to strings:

```
sff:toChar(xs:double(SALES))
```

SALES	RETURN VALUE
1010.99	'1010.99'
-15.62567	'-15.62567'
10842764968208837340	'1.08427649682088e+019' (rounded based on the 16th digit and returns the value in scientific notation)
236789034569723	'236789034569723'
0	'0'
33.15	'33.15'
NULL	NULL

## Decimal Conversion Example

The following expression converts the decimal values in the SALES port to strings in high precision mode:

```
sff:toChar(xs:double(SALES))
```

The following table lists some sample values and return values when the precision is greater than 38:

SALES	RETURN VALUE
2378964536789761	'2378964536789761'
1234567890123456789012345679	'1234567890123456789012345679'
1.234578945469649345876123456	'1.234578945469649345876123456'
0.99999999999999999999999999999999	'0.99999999999999999999999999999999'
12345678901234567890123456799	'12345678901234567890123456799'
23456788992233456678458934567123465239	'23456788992233456678458934567123465239'
423456789012345678901234567991234567899	'4.23456789012346e+038'

(greater than 38)

The following table lists some sample values and return values when the precision is greater than 28:

SALES	RETURN VALUE
2378964536789761	'2378964536789761'

SALES	RETURN VALUE
1234567890123456789012345679	'1234567890123456789012345679'
1.234578945469649345876123456	'1.234578945469649345876123456'
0.99999999999999999999999999999999	'0.99999999999999999999999999999999'
12345678901234567890123456799 (greater than 28)	'1.23456789012346e+028'

The toChar (Dates) function converts a Date/Time datatype to a string with the format you specify. You can convert the entire date or a part of the date to a string.

**Note:** Use double quotation marks to separate ambiguous format strings, for example D"D"DDD. The empty quotation marks do not appear in the output.

## toDate

Converts a character string to a Date/Time data type. You use the toDate format strings to specify the format of the source strings.

The output port must be Date/Time for toDate expressions.

If you are converting two-digit years with toDate, use either the RR or YY format string. Do not use the YYYY format string.

### Syntax

```
date:toDate(xs:dateTime('date'), 'format')
```

**Note:** You must manually add the xs:dateTime phrase and enclose the date values within single quotation marks.

The following table describes the arguments:

Argument	Required/Optional	Description
<i>date</i>	Required	Must be a string data type. Passes the values that you want to convert to dates. You can enter any valid transformation expression.
<i>format</i>	Required	Enter a valid toDate format string. The format string must match the parts of the <i>date</i> argument. For example, if you pass the date 'Mar 15 1998 12:43:10AM', you must use the format string 'MON DD YYYY HH12:MI:SSAM'.

### Return Value

Date.

The toDate function always returns a date and time. If you pass a string that does not have a time value, the date returned always includes the time 00:00:00.000000000. You can map the results of this function to any target column with a datetime data type.

NULL if you pass a NULL value to this function.



**Warning:** The format of the toDate string must match the format string including any date separators. If it does not, toDate might return inaccurate values or skip the record.

## Examples

```
date:toDate(xs:dateTime('DATE_PROMISED'), 'MM/DD/YY')
```

The following table lists some sample values and return values:

DATE_PROMISED	RETURN VALUE
'01/22/98'	Jan 22 1998 00:00:00
'05/03/98'	May 3 1998 00:00:00
'11/10/98'	Nov 10 1998 00:00:00
'10/19/98'	Oct 19 1998 00:00:00
NULL	NULL

```
date:toDate(xs:dateTime('DATE_PROMISED'), 'MON DD YYYY HH12:MI:SSAM')
```

The following table lists some sample values and return values:

DATE_PROMISED	RETURN VALUE
'Jan 22 1998 02:14:56PM'	Jan 22 1998 02:14:56PM
'Mar 15 1998 11:11:11AM'	Mar 15 1998 11:11:11AM
'Jun 18 1998 10:10:10PM'	Jun 18 1998 10:10:10PM
'October 19 1998'	<i>Error. Integration Service skips this row.</i>
NULL	NULL

The following expression converts strings in the SHIP\_DATE\_MJD\_STRING port to date values:

```
date:toDate(xs:dateTime('SHIP_DATE_MJD_STR'), 'J')
```

The following table lists some sample values and return values:

SHIP_DATE_MJD_STR	RETURN_VALUE
'2451544'	Dec 31 1999 00:00:00.000000000
'2415021'	Jan 1 1900 00:00:00.000000000

Because the J format string does not include the time portion of a date, the return values have the time set to 00:00:00.000000000.

The following expression converts a string to a four-digit year format. The current year is 1998:

```
date:toDate(xs:dateTime('DATE_STR'), 'MM/DD/RR')
```

The following table lists some sample values and return values:

DATE_STR	RETURN VALUE
'04/01/98'	04/01/1998 00:00:00.000000000
'08/17/05'	08/17/2005 00:00:00.000000000

The following expression converts a string to a four-digit year format. The current year is 1998:

```
date:toDate(xs:dateTime('DATE_STR'), 'MM/DD/YY')
```

The following table lists some sample values and return values:

DATE_STR	RETURN VALUE
'04/01/98'	04/01/1998 00:00:00.000000000
'08/17/05'	08/17/1905 00:00:00.000000000

**Note:** For the second row, RR returns the year 2005 and YY returns the year 1905.

The following expression converts a string to a four-digit year format. The current year is 1998:

```
date:toDate(xs:dateTime('DATE_STR'), 'MM/DD/Y')
```

The following table lists some sample values and return values:

DATE_STR	RETURN VALUE
'04/01/8'	04/01/1998 00:00:00.000000000
'08/17/5'	08/17/1995 00:00:00.000000000

The following expression converts a string to a four-digit year format. The current year is 1998:

```
date:toDate(xs:dateTime('DATE_STR'), 'MM/DD/YYYY')
```

The following table lists some sample values and return values:

DATE_STR	RETURN VALUE
'04/01/998'	04/01/1998 00:00:00.000000000
'08/17/995'	08/17/1995 00:00:00.000000000

The following expression converts strings that includes the seconds since midnight to date values:

```
date:toDate(xs:dateTime('DATE_STR'), 'MM/DD/YYYY SSSSS')
```

The following table lists some sample values and return values:

DATE_STR	RETURN_VALUE
'12/31/1999 3783'	12/31/1999 01:02:03
'09/15/1996 86399'	09/15/1996 23:59:59

# toDecimal

Converts a string or numeric value to a decimal value. The toDecimal function ignores leading spaces.

## Syntax

```
util:toDecimal(value, scale)
```

The following table describes the arguments:

Argument	Required/Optional	Description
<i>value</i>	Required	Must be a string or numeric data type. Passes the value that you want to convert to decimals. You can enter any valid transformation expression.
<i>scale</i>	Optional	Must be an integer literal between 0 and 28, inclusive. Specifies the number of digits allowed after the decimal point. If you omit this argument, the function returns a value with the same scale as the input value.

## Return Value

Decimal of precision and scale between 0 and 28, inclusive.

0 if the value in the selected column is an empty string or a non-numeric character.

NULL if a value passed to the function is NULL.

## Example

This expression uses values from the column IN\_TAX. The data type is decimal with precision of 10 and scale of 3:

```
util:toDecimal(IN_TAX, 3)
```

The following table lists some sample values and return values:

IN_TAX	RETURN VALUE
'15.6789'	15.679
'60.2'	60.200
'118.348'	118.348
NULL	NULL
'A12.3Grove'	0

This expression uses values from the column Sales. The data type is decimal with precision of 10 and scale of 2:

```
util:toDecimal(Sales, 2)
```

The following table lists some sample values and return values:

Sales	RETURN VALUE
'1234'	1234

Sales	RETURN VALUE
'1234.01'	1234.01

If you want the return value in 1234.00 format, you can use the following expression:

```
format-number(util:toDecimal('1234', 2), '0.00')
```

The toDecimal function support up to 28 precision. If you pass a value with precision greater than 28, such as 32, you will encounter an issue even after enabling high precision.

If you want a 32-digit value intact from the source to the target, you must define the value as string from source to target with 32 precision. You will be able to load the value to the target database even if that database has a numeric data type and the target definition has a string data type. However, you must ensure that the data type of the column in the target database accepts a precision of 32.

### Decimal overflow

If the size of the number on the left hand side of the decimal point exceeds the precision, the decimal operation overflows.

To resolve this, modify the scale and/or precision of the expression port and connect downstream ports of the mapping to accommodate the size of the input data in the expression.

For example:

If a numeric field is defined to be size 28 with scale of 15, this accepts 13 numbers on the left side of the decimal and 15 on the right. So the following numbers would be valid for a number of 28,15:

1234567890123.11143

13575.123451234567891

However, these numbers would cause a decimal overflow error:

111112222233333.4444

123.1111122222333334

## toInteger

Converts a string or numeric value to an integer. The toInteger syntax contains an optional argument that you can choose to round the number to the nearest integer or truncate the decimal portion. The toInteger function ignores leading spaces.

### Syntax

```
util:toInteger(value, flag)
```

The following table describes the arguments:

Argument	Required/Optional	Description
<i>value</i>	Required	Must be a string or numeric data type. Passes the value that you want to convert to an integer. You can enter any valid transformation expression.
<i>flag</i>	Optional	Specifies whether to truncate or round the decimal portion. The flag must be an integer literal or the constants TRUE or FALSE: <ul style="list-style-type: none"> <li>- toInteger truncates the decimal portion when the flag is TRUE or a number other than 0.</li> <li>- toInteger rounds the value to the nearest integer if the flag is FALSE or 0 or if you omit this argument.</li> </ul>

## Return Value

Integer.

NULL if a value passed to the function is NULL.

0 if a value passed to the function contains alphanumeric characters.

## Example

The following expressions use values from the column IN\_TAX:

```
util:toInteger(IN_TAX, fn:boolean(1))
```

The following table lists some sample values and return values:

IN_TAX	RETURN VALUE
'15.6789'	15
'60.2'	60
'118.348'	118
NULL	NULL
'A12.3Grove'	0
' 123.87'	123
'-15.6789'	-15
'-15.23'	-15

If a bigint column is mapped to a column with integer data type, an issue occurs.

To avoid this issue, you must have the data type of the column same throughout the mapping. If you need to assign bigint to the integer column, ensure that the data being passed does not exceed the range of integer.

Bigint range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Integer range: -2,147,483,648 to 2,147,483,647

# trunc (Dates)

Truncates dates to a specific year, month, day, hour, minute, second, or millisecond. You can also use trunc to truncate numbers.

You can truncate the following date parts:

- **Year.** If you truncate the year portion of the date, the function returns Jan 1 of the input year with the time set to 00:00:00.000000000. For example, the following expression returns 1/1/1997 00:00:00.000000000:

```
date:trunc(xs:dateTime('12/1/1997 3:10:15'), 'YY')
```

- **Month.** If you truncate the month portion of a date, the function returns the first day of the month with the time set to 00:00:00.000000000. For example, the following expression returns 4/1/1997 00:00:00.000000000:

```
date:trunc(xs:dateTime('4/15/1997 12:15:00'), 'MM')
```

- **Day.** If you truncate the day portion of a date, the function returns the date with the time set to 00:00:00.000000000. For example, the following expression returns 6/13/1997 00:00:00.000000000:

```
date:trunc(xs:dateTime('6/13/1997 2:30:45'), 'DD')
```

- **Hour.** If you truncate the hour portion of a date, the function returns the date with the minutes, seconds, and subseconds set to 0. For example, the following expression returns 4/1/1997 11:00:00.000000000:

```
date:trunc(xs:dateTime('4/1/1997 11:29:35'), 'HH')
```

- **Minute.** If you truncate the minute portion of a date, the function returns the date with the seconds and subseconds set to 0. For example, the following expression returns 5/22/1997 10:15:00.000000000:

```
date:trunc(xs:dateTime('5/22/1997 10:15:29'), 'MI')
```

- **Second.** If you truncate the second portion of a date, the function returns the date with the milliseconds set to 0. For example, the following expression returns 5/22/1997 10:15:29.000000000:

```
date:trunc(xs:dateTime('5/22/1997 10:15:29.135'), 'SS')
```

- **Millisecond.** If you truncate the millisecond portion of a date, the function returns the date with the microseconds set to 0. For example, the following expression returns 5/22/1997 10:15:30.135000000:

```
date:trunc(xs:dateTime('5/22/1997 10:15:30.135235'), 'MS')
```

## Syntax

```
date:trunc(xs:dateTime('date'), 'format')
```

**Note:** You must manually add the xs:dateTime phrase and enclose the date values within single quotation marks.

The following table describes the arguments:

Argument	Required/Optional	Description
<i>date</i>	Required	Date/Time data type. The date values that you want to truncate. You can enter any valid transformation expression that evaluates to a date. To pass a NULL value, you must specify an empty sequence in the following format: ( )
<i>format</i>	Required	Enter a valid format string. The format string is not case sensitive. To pass a NULL value, you must specify an empty sequence in the following format: ( )

## Return Value

Date.

NULL if a value passed to the function is NULL.

## Examples

The following expressions truncate the year portion of dates in the DATE\_SHIPPED column:

```
date:trunc(xs:dateTime('DATE_SHIPPED'), 'Y')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'YY')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'YYY')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'YYYY')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 1 1998 00:00:00.000000000
Apr 19 1998 1:31:20PM	Jan 1 1998 00:00:00.000000000
Jun 20 1998 3:50:04AM	Jan 1 1998 00:00:00.000000000
Dec 20 1998 3:29:55PM	Jan 1 1998 00:00:00.000000000
NULL	NULL

The following expressions truncate the month portion of each date in the DATE\_SHIPPED column:

```
date:trunc(xs:dateTime('DATE_SHIPPED'), 'MM')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'MON')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'MONTH')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 1 1998 00:00:00.000000000
Apr 19 1998 1:31:20PM	Apr 1 1998 00:00:00.000000000
Jun 20 1998 3:50:04AM	Jun 1 1998 00:00:00.000000000
Dec 20 1998 3:29:55PM	Dec 1 1998 00:00:00.000000000
NULL	NULL

The following expressions truncate the day portion of each date in the DATE\_SHIPPED column:

```
date:trunc(xs:dateTime('DATE_SHIPPED'), 'D')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'DD')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'DDD')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'DY')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'DAY')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 15 1998 00:00:00.000000000
Apr 19 1998 1:31:20PM	Apr 19 1998 00:00:00.000000000

DATE_SHIPPED	RETURN VALUE
Jun 20 1998 3:50:04AM	Jun 20 1998 00:00:00.000000000
Dec 20 1998 3:29:55PM	Dec 20 1998 00:00:00.000000000
Dec 31 1998 11:59:59PM	Dec 31 1998 00:00:00.000000000
NULL	NULL

The following expressions truncate the hour portion of each date in the DATE\_SHIPPED column:

```
date:trunc(xs:dateTime('DATE_SHIPPED'), 'HH')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'HH12')
date:trunc(xs:dateTime('DATE_SHIPPED'), 'HH24')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:31AM	Jan 15 1998 02:00:00.000000000
Apr 19 1998 1:31:20PM	Apr 19 1998 13:00:00.000000000
Jun 20 1998 3:50:04AM	Jun 20 1998 03:00:00.000000000
Dec 20 1998 3:29:55PM	Dec 20 1998 15:00:00.000000000
Dec 31 1998 11:59:59PM	Dec 31 1998 23:00:00.000000000
NULL	NULL

The following expression truncates the minute portion of each date in the DATE\_SHIPPED column:

```
date:trunc(xs:dateTime('DATE_SHIPPED'), 'MI')
```

The following table lists some sample values and return values:

DATE_SHIPPED	RETURN VALUE
Jan 15 1998 2:10:30AM	Jan 15 1998 02:10:00.000000000
Apr 19 1998 1:31:20PM	Apr 19 1998 13:31:00.000000000
Jun 20 1998 3:50:04AM	Jun 20 1998 03:50:00.000000000
Dec 20 1998 3:29:55PM	Dec 20 1998 15:29:00.000000000
Dec 31 1998 11:59:59PM	Dec 31 1998 23:59:00.000000000
NULL	NULL



## CHAPTER 9

# Using the Validation panel

The Validation panel lists the errors in a taskflow. Use the Validation panel to troubleshoot a taskflow.

1. On the top right of the Taskflow Designer page, click **Validation**.

If the taskflow contains errors, you see a list of errors.

Validation (3)		
▶  SF_Mapping	 1	
▶  FF_Mapping	 1	
▶  Assignment 1	 1	

2. Expand each error to view the details.

## Validation (3)




▼  [SF\\_Mapping](#)  1

The input field Data Task for a Data Task:  
step cannot be empty.

▼  [FF\\_Mapping](#)  1

The input field Data Task for a Data Task:  
step cannot be empty.

▼  [Assignment 1](#)  1

The field contains the following error in the  
expression: The expression depends on the  
context item.

3. Click an error to go to the corresponding step on the UI.
4. Make changes to the taskflow and then click **Save**.

## CHAPTER 10

# Running a taskflow

Use a taskflow to control the execution sequence of multiple Data Integration tasks.

You can invoke and run a taskflow in the following ways:

### From the taskflow designer

To run a taskflow from the taskflow designer, open the taskflow and click **Run** in the upper-right part of the page.

### According to a schedule

To run a taskflow on a schedule, create a schedule in Administrator and associate the taskflow with the schedule.

For more information about schedules, see *Organization Administration*.

## Running a taskflow from the taskflow designer

1. In Data Integration, click **Explore** on the left navigation pane.
2. From the **Explore By** list, select **Asset Types**.
3. Go to **All Assets > Taskflows**.
4. Click the taskflow that you want to run.
5. Click **Run** in the upper-right part of the page.

## Publishing a taskflow

If you want to schedule a taskflow, you need to publish it first.

You can publish a taskflow in any of the following ways:

- Run the taskflow from the taskflow designer. When you run a taskflow, it gets published automatically.
- Click **Publish** in the taskflow designer.
- Navigate to the taskflow from the Explore page and select **Publish** from the **Actions** menu.

If you change a scheduled taskflow, you need to republish it so that Data Integration runs the updated taskflow.

# Scheduling a taskflow

To schedule a taskflow, associate the taskflow with an existing schedule or create a new schedule.

You can create a new schedule in Data Integration and Administrator. For more information about creating a schedule in Administrator, see *Organization Administration*.

Before you schedule a taskflow, you must publish the taskflow. A scheduled taskflow runs only if the taskflow was published at least once.

If you schedule an outdated taskflow that contains unpublished changes, Data Integration schedules the last published taskflow version.

If you unpublish a scheduled taskflow, the scheduled taskflow jobs will not run.

If the scheduler fails to trigger the taskflow job for the first time, it tries to trigger the taskflow job 3 more times.

You can view the scheduled jobs by selecting **Scheduled Jobs** from the **Actions** menu.

**Note:** To delete a scheduled job, select the job and click **Delete**.

1. In Data Integration, select **Explore**.

The **Explore** page opens. You can filter the page by using the **Projects and Folders**, **Asset Types**, or **Tags** option.

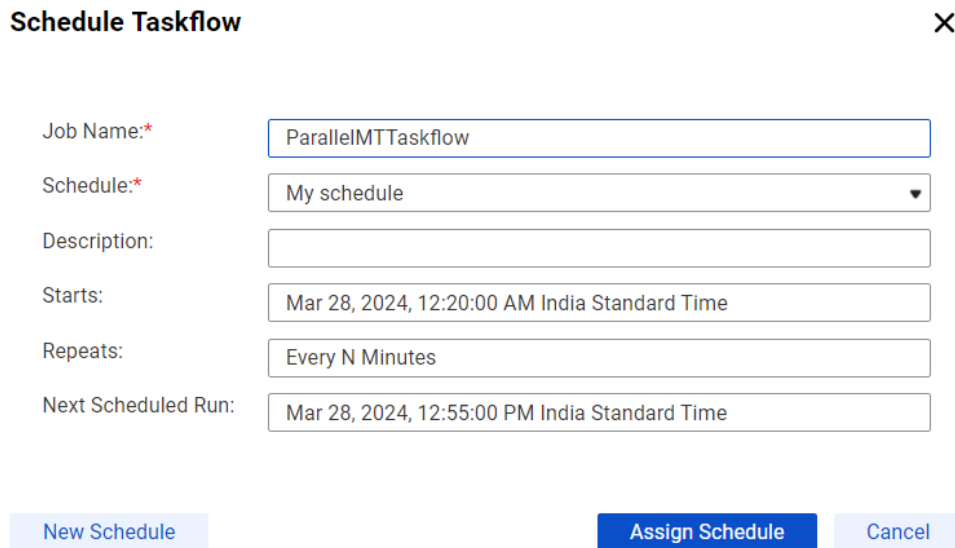
2. Navigate to the taskflow that you want to schedule and click **Actions**.

The **Actions** menu appears.

3. From the **Actions** menu, select **Schedule**.

The **Schedule Taskflow** dialog box appears.

The following image shows the **Schedule Taskflow** dialog box:



4. In the **Job Name** field, enter a name for this combination of taskflow and schedule.

The name can contain alphanumeric characters, spaces, and the following special characters: `_ . + -`

5. Perform one of the following steps:

- To assign an existing schedule, select a schedule from the **Schedule** list and click **Assign Schedule**.

- To create a schedule, click **New Schedule**, enter the schedule details, and click **Save**. The schedule that you created is selected in the **Schedule** list. Click **Assign Schedule** to assign the schedule to the taskflow.

**Note:** If you remove a taskflow from a schedule as the taskflow runs, the job completes. Data Integration cancels any additional runs associated with the schedule.

# CHAPTER 11

## Taskflow example

You can add multiple data integration tasks to a taskflow and run them sequentially or in parallel.

The following example shows how to create a taskflow to run two mapping tasks in parallel. The example assumes that you know how to create a connection, a mapping, and a mapping task.

This example uses the following sample mapping tasks:

- A mapping task that maps a flat file database to a flat file database.
- A mapping task that maps a MySQL database to a flat file database.

Perform the following steps to create a taskflow and monitor the progress of the taskflow:

1. From the top menu bar, select **Data Integration**.
2. Click **New > Taskflows > Taskflow > Create**.
3. Select **Start** and enter the following General properties:

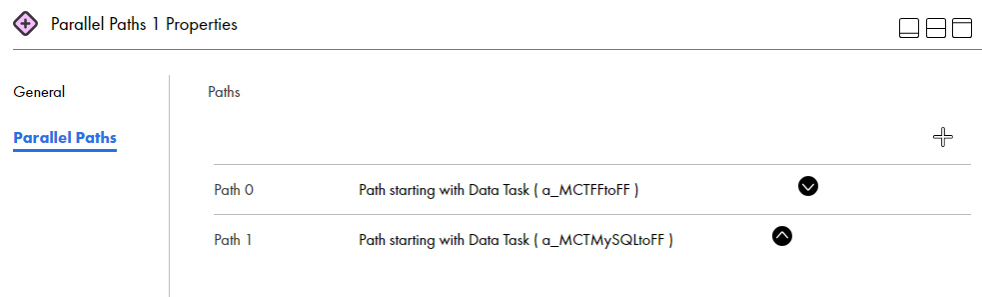
Property	Value
Name	Taskflow2MT
Location	Click <b>Select</b> and browse to the location where you want to save the taskflow.
Description	Taskflow to run two mapping tasks in parallel

The screenshot shows the 'Taskflow2MT Properties' dialog box with the 'General' tab selected. The 'Step Type' is set to 'Start'. The 'Name' field contains 'Taskflow2MT'. The 'API Name' field is empty, with an 'Override API Name' checkbox to its right. The 'Location' field contains 'Default' and has a 'Select' button next to it. The 'Description' field contains 'Taskflow to run two mapping tasks in parallel'. On the left side, there is a vertical menu with options: 'General' (selected), 'Start', 'Input Fields', 'Output Fields', 'Temp Fields', 'Advanced', and 'Notes'.

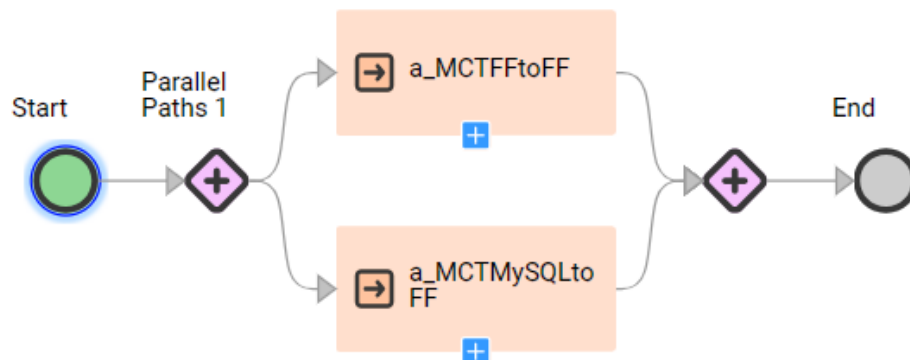
For this use case, you do not need to enter **Input Fields**, **Temp Fields** or **Notes** properties.

4. Drag a Parallel Path step onto the canvas.  
You see a Parallel Path step with two branches.

5. To add Data Task steps, drag Data Task steps from the left palette to each branch of the Parallel Path step.
6. Select the first Data Task step and then perform the following steps:
  - a. Go to **Properties > Data Task** and click **Select**. You see a dialog box with a list of mapping tasks. Select the task you want to use in the first Data Task step and then click **Select**.
  - b. Go to **Properties > General** and enter a name for the Data task step in the **Name** field.
7. Repeat the preceding step for the Data Step in the second Parallel Path branch.
8. Select the Parallel Path step and go to **Properties > General**, and enter a name for the step in the **title** field.
9. Go to **Parallel Path Properties > Parallel Paths**. You see that the Taskflow has set path 0 and path 1 of the Parallel Path step:



10. Click **Save** from the upper-right corner of the page.  
The following image shows the complete taskflow:



11. If you do not see any validation errors, click **Run**.  
**Note:** If you see validation errors, use the Validation panel to trouble shoot the taskflow.
12. To verify that the taskflow is complete, check the **All Jobs** page in Monitor or the **My Jobs** page in Data Integration.

# INDEX

## A

addToDate function  
description [59](#)

## C

character functions  
instr [74](#)  
ltrim [80](#)  
rtrim [84](#)

character strings  
converting to dates [88](#)

characters  
adding to strings [79](#)  
removing from strings [80](#), [84](#)

Cloud Application Integration community  
URL [6](#)

Cloud Developer community  
URL [6](#)

conversion functions  
toChar (numbers) [86](#)  
toDate [88](#)  
toDecimal [91](#)  
toInteger [92](#)

## D

data cleansing functions  
in [73](#)

Data Integration community  
URL [6](#)

date functions  
addToDate [59](#)  
dateDiff [62](#)  
getDatePart [69](#)  
lastDay [77](#)  
trunc [94](#)

date/time values  
adding [59](#)

dateDiff function  
description [62](#)

dates  
truncating [94](#)

decimal values  
converting [91](#)

decode function  
description [65](#)

## F

format  
from character string to date [88](#)

## G

getAssetLocation function  
description [68](#)  
getAssetName function  
description [68](#)  
getDatePart function  
description [69](#)  
getInstanceStartTime function  
description [70](#)

## I

iif function  
description [71](#)  
in function  
description [73](#)  
Informatica Global Customer Support  
contact information [7](#)  
Informatica Intelligent Cloud Services  
web site [6](#)  
instr function  
description [74](#)  
integers  
converting other values [92](#)  
isNull function  
description [77](#)

## L

lastDay function  
description [77](#)  
lpad function  
description [79](#)  
ltrim function  
description [80](#)

## M

maintenance outages [7](#)  
month  
returning last day [77](#)

## N

Notification Task step  
configuring properties [25](#)  
rules and guidelines [28](#)  
NULL values  
checking for [77](#)  
isNull [77](#)



- numbers
  - rounding [82](#)
- numeric functions
  - round (numbers) [82](#)
- numeric values
  - converting to text strings [86](#)

## R

- round (numbers) function
  - description [82](#)
- rounding
  - numbers [82](#)
- rtrim function
  - description [84](#)

## S

- special functions
  - decode [65](#)
  - iif [71](#)
- status
  - Informatica Intelligent Cloud Services [7](#)
- string functions
  - lpad [79](#)
- strings
  - adding blanks [79](#)
  - adding characters [79](#)
  - character set [74](#)
  - converting character strings to dates [88](#)
  - converting numeric values to text strings [86](#)
  - removing blank characters [80](#)
  - removing blank characters and characters [84](#)
  - removing characters [80](#)
- subseconds
  - processing in trunc function [94](#)
- system status [7](#)

## T

- taskflow functions
  - getAssetLocation [68](#)

- taskflow functions (*continued*)
  - getAssetName [68](#)
  - getInstanceStartTime [70](#)

- taskflows
  - creating a taskflow [11](#)
  - expression editor [54](#)
  - overview [8](#)
  - parameters in taskflows [48](#)
  - running a taskflow [99](#)
  - sample taskflow [102](#)
  - taskflow properties [14](#)
  - taskflow step properties [20](#)
  - taskflow steps [9](#)
  - taskflow templates [11](#)
  - validation panel [97](#)

- test functions
  - isNull [77](#)
- text strings
  - converting numeric values [86](#)
- toChar (numbers) function
  - description [86](#)
- toDate function
  - description [88](#)
- toDecimal function
  - description [91](#)
- toInteger function
  - description [92](#)
- trunc function
  - description [94](#)
  - processing subseconds [94](#)
- truncating
  - dates [94](#)
- trust site
  - description [7](#)

## U

- upgrade notifications [7](#)

## W

- web site [6](#)