



**PUBLIC**

SAP Data Intelligence

2023-06-24

# Modeling Guide

# Content

- 1 Modeling Guide for SAP Data Intelligence. . . . . 7**
- 2 Introduction to the SAP Data Intelligence Modeler. . . . . 8**
  - 2.1 Log on to SAP Data Intelligence Modeler. . . . . 10
  - 2.2 Description of the Modeler Main Screen. . . . . 10
- 3 Using Operators. . . . . 14**
  - 3.1 Operator Details. . . . . 15
  - 3.2 Generation 1 and Generation 2 Operators. . . . . 21
  - 3.3 Customizing the List of Operators. . . . . 21
  - 3.4 Ports and Port Types. . . . . 22
    - Compatible Port Types. . . . . 25
    - Table Messages. . . . . 27
    - Data Types in Operator Ports. . . . . 32
    - Adding Ports to Operators. . . . . 33
  - 3.5 Using Managed Connections in Script Operators. . . . . 34
  - 3.6 Creating Operators. . . . . 36
  - 3.7 Configuring Operators. . . . . 40
  - 3.8 Creating Categories. . . . . 41
  - 3.9 Creating Operator Groups. . . . . 41
  - 3.10 Viewing Operator Versions. . . . . 43
    - Replacing Deprecated Operators. . . . . 44
    - Editing Operator Versions. . . . . 44
    - Creating Operator Versions. . . . . 45
  - 3.11 Editing Operators. . . . . 45
  - 3.12 Error Handling in Generation 2 Operators. . . . . 46
  - 3.13 Batch Header. . . . . 47
  - 3.14 State Management. . . . . 48
    - Examples: Operator States. . . . . 50
  - 3.15 Dockerfile Library for Runtime Environment. . . . . 55
- 4 Using Graphs (Pipelines). . . . . 56**
  - 4.1 Creating Graphs. . . . . 57
  - 4.2 Error Recovery with Generation 2 (Gen2) Pipelines. . . . . 59
  - 4.3 Graph Snapshots and Operator States. . . . . 61
  - 4.4 Delivery Guarantee for Generation 2 (Gen2) Graphs. . . . . 66
  - 4.5 Validate Graphs. . . . . 69
    - Graph Validation Warnings and Errors. . . . . 70

4.6	Running Graphs. . . . .	72
	Automatic Graph Recovery. . . . .	74
	Parameterize the Graph Run Process. . . . .	78
	Debug Graphs. . . . .	82
	Schedule Graph Executions. . . . .	84
4.7	Maintain Resource Requirements for Graphs. . . . .	87
	Resource Requirements for a Graph in JSON. . . . .	88
	Configure Resources for a Graph. . . . .	90
4.8	Create Data Types in Graph. . . . .	91
	Use Data Types in Graph. . . . .	92
	Exporting and Importing Graphs with Data Types. . . . .	93
4.9	Groups, Tags, and Dockerfiles. . . . .	94
4.10	Execution Model. . . . .	99
4.11	Monitoring Graphs . . . . .	100
	Monitor the Graph Execution Status. . . . .	101
	Activate Trace Messages. . . . .	108
	Downloading Diagnostic Information for Graphs. . . . .	109
4.12	Native Multiplexing for Gen2 Pipelines. . . . .	115
	Multiplexing Scenarios. . . . .	117
<b>5</b>	<b>Using Git Terminal. . . . .</b>	<b>120</b>
5.1	Git Credential Handling Using Standard Git Credential Helper. . . . .	121
5.2	Create a Local Git Repository. . . . .	122
5.3	Clone a Remote Git Repository. . . . .	123
<b>6</b>	<b>Using Scenario Templates. . . . .</b>	<b>125</b>
6.1	ABAP with Data Lakes. . . . .	125
6.2	Data Processing with Scripting Languages. . . . .	126
6.3	ETL from Database. . . . .	128
6.4	Loading Data from Data Lake to Database (SAP HANA). . . . .	128
<b>7</b>	<b>Using Graph Snippets. . . . .</b>	<b>130</b>
7.1	Importing Graph Snippets. . . . .	130
7.2	Creating Graph Snippets. . . . .	131
7.3	Editing Graph Snippets. . . . .	132
<b>8</b>	<b>Working with the Data Workflow Operators. . . . .</b>	<b>133</b>
8.1	Workflow Trigger and Workflow Terminator . . . . .	136
8.2	Run an SAP BW Process Chain Operator. . . . .	137
8.3	Run a HANA Flowgraph Operator. . . . .	139
8.4	Run an SAP Data Intelligence Pipeline. . . . .	141
8.5	Run an SAP Data Services Job. . . . .	143
8.6	Transfer Data . . . . .	146

	Transfer Data from SAP BW to Cloud Storage. . . . .	146
	Transfer Data from SAP HANA to Cloud Storage. . . . .	153
8.7	Control Flow of Execution. . . . .	156
8.8	Send E-Mail Notifications. . . . .	157
<b>9</b>	<b>Working with Structured Data Operators. . . . .</b>	<b>159</b>
9.1	Data Transform. . . . .	159
	Configure the Projection Node. . . . .	161
	Configure the Join Node. . . . .	163
	Configure the Aggregation Node. . . . .	167
	Configure the Union Node. . . . .	169
	Configure the Case Node. . . . .	170
9.2	Structured Consumer Operators. . . . .	171
	SAP Application Consumer. . . . .	171
	Structured File Consumer. . . . .	172
	Structured SQL Consumer. . . . .	174
9.3	Structured Producer Operators. . . . .	175
	SAP Application Producer. . . . .	175
	Structured File Producer. . . . .	176
	Structured Table Producer. . . . .	177
9.4	Custom Editor. . . . .	178
9.5	Resiliency with Structured Data Operators. . . . .	179
<b>10</b>	<b>Operator Metrics. . . . .</b>	<b>181</b>
<b>11</b>	<b>Replicating Data. . . . .</b>	<b>182</b>
11.1	Create a Replication Flow. . . . .	183
	Create Tasks. . . . .	187
11.2	Validate the Replication Flow. . . . .	193
11.3	Deploy the Replication Flow. . . . .	193
11.4	Run the Replication Flow. . . . .	194
11.5	Cloud Storage Target Structure. . . . .	195
11.6	Kafka as Target . . . . .	198
11.7	ABAP Cluster Table Replications with Delta Load. . . . .	199
11.8	Edit an Existing Replication Flow. . . . .	200
11.9	Undeploy a Replication Flow. . . . .	200
11.10	Delete a Replication Flow. . . . .	202
11.11	Clean Up Source Artifacts. . . . .	202
<b>12</b>	<b>Monitoring SAP Data Intelligence . . . . .</b>	<b>204</b>
12.1	Log in to SAP Data Intelligence Monitoring. . . . .	205
12.2	Using the Monitoring Application. . . . .	205
<b>13</b>	<b>Integrating SAP Cloud Applications with SAP Data Intelligence. . . . .</b>	<b>213</b>

<b>14</b>	<b>Service-Specific Information.</b>	<b>216</b>
14.1	Alibaba Cloud Object Storage Service (OSS).	216
14.2	Amazon Simple Storage Service (AWS S3).	220
14.3	Google Cloud Storage (GCS).	223
14.4	Hadoop Distributed File System (HDFS).	225
14.5	Microsoft Azure Data Lake (ADL).	226
14.6	Microsoft Azure Blob Storage (WASB).	228
14.7	Local File System (/file).	230
14.8	WebHDFS.	230
<b>15</b>	<b>Changing Data Capture (CDC).</b>	<b>233</b>
<b>16</b>	<b>Subengines.</b>	<b>234</b>
16.1	Working with the C++ Subengine to Create Operators.	235
	Getting Started with the C++ Subengine.	237
	Creating an Operator.	237
	Logging and Error Handling.	239
	Port Data.	240
	Setting Values for Configuration Properties.	243
	Process Handlers.	243
	API Reference.	245
16.2	Create Operators with the Python Subengine.	257
	Normal Usage.	259
	Advanced Usage.	261
16.3	Working with the Node.js Subengine to Create Operators.	274
	Node.js Operators and Operating System Processes.	274
	Use Cases for the Node.js Subengine.	275
	The Node.js Subengine SDK.	276
	Node.js Data Types.	279
	Node.js Safe and Unsafe Integer Data Types.	279
	Create a Node.js Operator	280
	Node.js Project Structure.	282
	Node.js Project Files and Resources.	282
	Node.js Subengine Logging.	285
16.4	Working with Flowagent Subengine to Connect to Databases.	286
<b>17</b>	<b>Creating Dockerfiles.</b>	<b>291</b>
17.1	Dockerfile Inheritance.	292
17.2	Referencing Parent Docker Images.	294
<b>18</b>	<b>Creating Configuration Types.</b>	<b>298</b>
<b>19</b>	<b>Security and Data Protection.</b>	<b>301</b>

**20 Using Data Types. . . . . 302**  
20.1 Creating Global Data Types. . . . . 303  
20.2 Creating Local Data Types. . . . . 305

# 1 Modeling Guide for SAP Data Intelligence

The Modeling Guide contains information about using the SAP Data Intelligence Modeler.

The SAP Data Intelligence Modeler helps create data processing graphs and provides a runtime and design-time environment for data-driven scenarios. The tool reuses existing coding and libraries to orchestrate data processing in distributed landscapes.

The following tasks require some expertise and programming skills:

- Creating operators
- Creating types
- Creating Dockerfiles
- Working with the subengines in the SAP Data Intelligence Modeler

If you're new to these tasks, or to modeling, we recommend that you start your learning journey by creating graphs with only the built-in (predefined) operators that the modeler provides.

## Related Information

[Introduction to the SAP Data Intelligence Modeler \[page 8\]](#)

## 2 Introduction to the SAP Data Intelligence Modeler

The SAP Data Intelligence Modeler application is based on the Pipeline Engine, which uses a flow-based programming model to run graphs that process your data.

### Data Ingestion and Transformation

The Modeler offers advanced data ingestion and transformation capabilities using computation graphs. In computation graphs, nodes represent operations on the data and edges represent the data flow. The following are common use cases for data ingestion and transformation in the Modeler:

- Ingest data from source systems like the following:
  - Database systems, such as SAP HANA.
  - Message queues, such as Apache Kafka.
  - Data storage systems, such as Hadoop Distributed File System (HDFS) or Amazon Simple Storage Service (Amazon S3).
- Cleanse data.
- Transform data to target schemas.
- Store data in target systems for consumption, archiving, or analysis.
- Replicate large datasets.

### Modeler Graphical Capabilities

Use the graphical capabilities of the Modeler to create graphs, and use the runtime component to run graphs in a containerized environment that runs on Kubernetes. Construct graphs in the Modeler using predefined operators, which provide for many productive business use cases. These operators help define graphs, including nonterminating, nonconnected, or cyclic graphs.

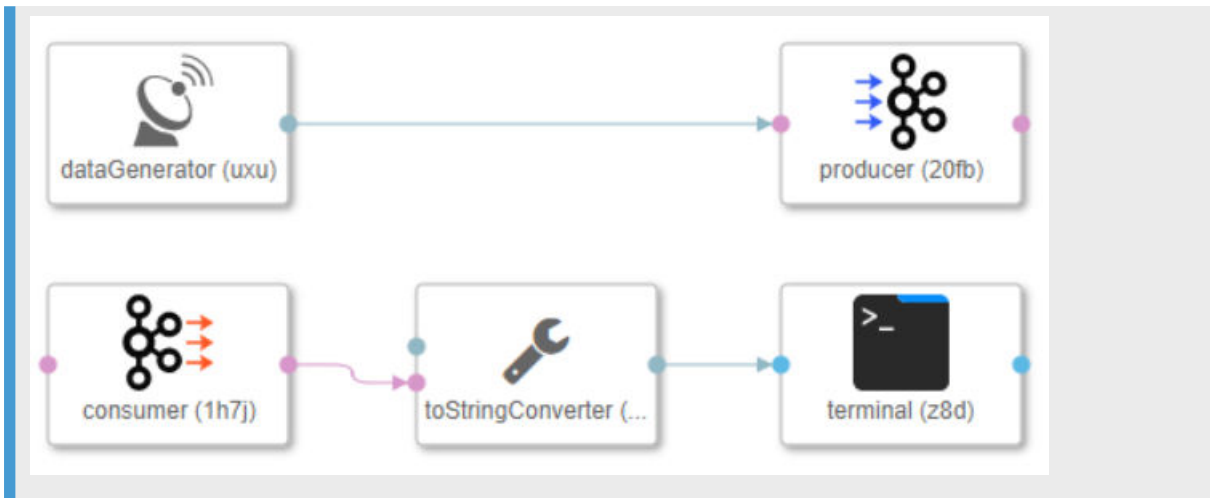
#### ❖ Example

##### A simple interaction with Apache Kafka

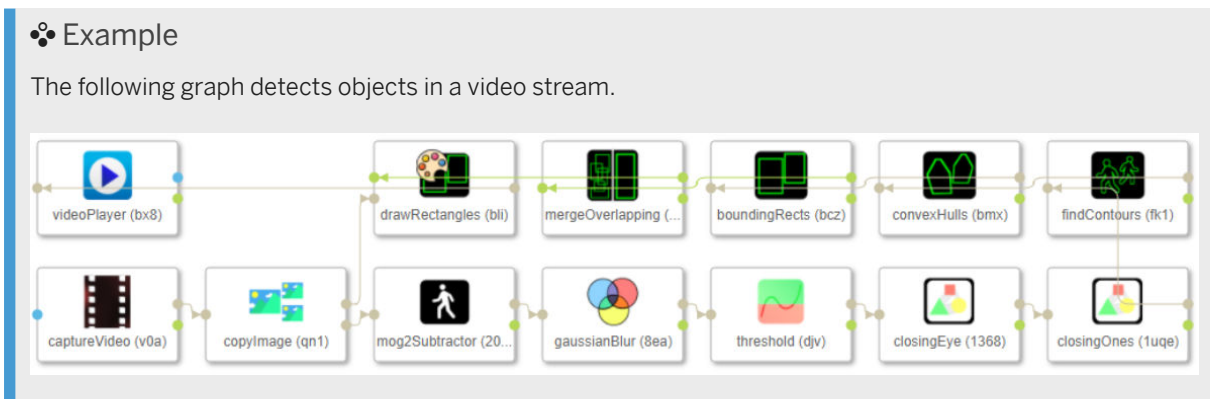
The following graph consists of two subgraphs:

- The first subgraph generates data and writes the data into a Kafka message queue.
- The second subgraph reads the data from Kafka, converts it to string, and prints the data to a terminal.





Also use the Modeler to create generic data processing graphs, as shown in the following example.



## Related Information

[Log on to SAP Data Intelligence Modeler \[page 10\]](#)

[Description of the Modeler Main Screen \[page 10\]](#)

## 2.1 Log on to SAP Data Intelligence Modeler

You can access the SAP Data Intelligence Modeler from the SAP Data Intelligence Launchpad or launch it directly with a stable URL.

### Prerequisites

Before you log on to SAP Data Intelligence for the first time, familiarize yourself with the Launchpad from which you open the Modeler application. For details, see the [Launchpad](#) guide. Also, read about user types in [Manage Users](#) in the *Administration Guide* to identify what type of user privileges you have.

### Procedure

1. Enter or select the SAP Data Intelligence Launchpad URL in a browser.
2. Enter your log on credentials for the SAP Data Intelligence Launchpad application in the welcome screen:
  - Tenant ID
  - User name
  - Password

The SAP Data Intelligence Launchpad opens to the home page. The home page displays the applications available in the tenant based on the policies assigned to you.

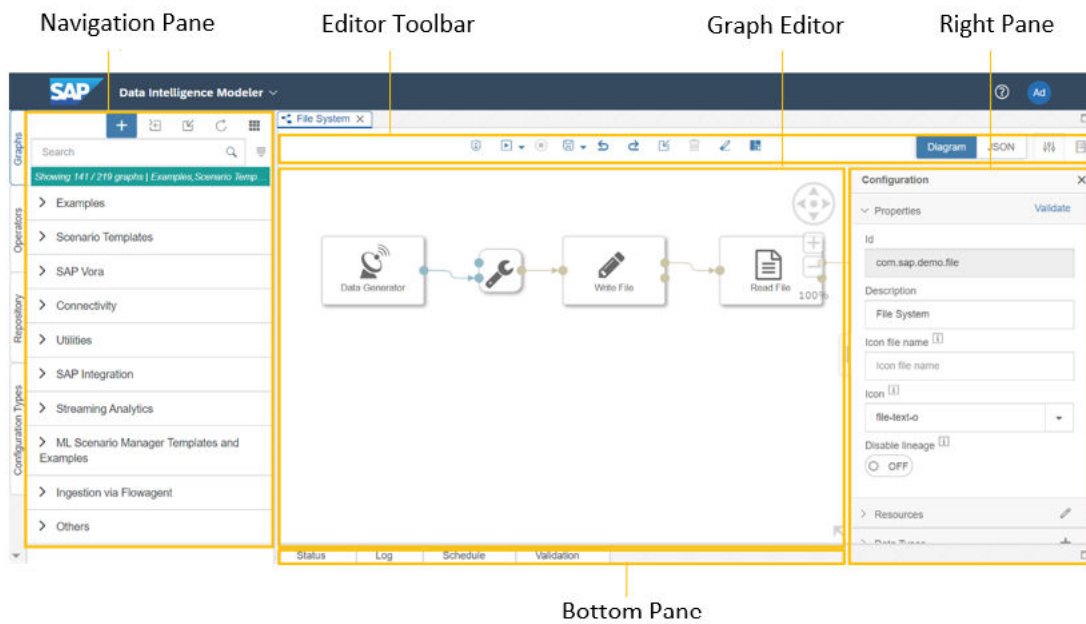
3. Choose the *Modeler* tile.

The Modeler opens to the initial screen.

## 2.2 Description of the Modeler Main Screen

Use the areas of the SAP Data Intelligence Modeler main screen to perform various tasks, such as configuring a graph.

The following image shows the various areas of the Modeler main screen.



The following table describes the areas of the Modeler main screen.

Pane or Toolbar	Description
Graph editor	Area in which you add and connect operators for a graph.

Pane or Toolbar	Description														
Navigation pane	Consists of the tabs described in the following table. <table border="1" data-bbox="804 416 1394 1563"> <thead> <tr> <th data-bbox="810 421 847 443">Tab</th> <th data-bbox="1107 421 1225 443">Description</th> </tr> </thead> <tbody> <tr> <td data-bbox="810 465 884 488"><i>Graphs</i></td> <td data-bbox="1107 465 1374 555">Access built-in or custom graphs, and create custom graphs.</td> </tr> <tr> <td data-bbox="810 577 906 600"><i>Operators</i></td> <td data-bbox="1107 577 1369 667">Access built-in or custom operators, and create custom operators.</td> </tr> <tr> <td data-bbox="810 689 916 712"><i>Repository</i></td> <td data-bbox="1107 689 1390 920">Access and create Modeler objects, such as graphs, operators, and types in your repository. Create new folders, import auxiliary files or solutions, and export folders as .tgz files or vSolution files.</td> </tr> <tr> <td colspan="2" data-bbox="1107 943 1390 1301"> <div style="border-left: 2px solid #0070C0; padding-left: 10px;"> <p><b>i Note</b></p> <p>The Modeler provides individual Dockerfiles to create containerized environments for the operator groups. The Modeler selects the Dockerfiles using a tag-matching mechanism.</p> </div> </td> </tr> <tr> <td data-bbox="810 1323 1007 1346"><i>Configuration Types</i></td> <td data-bbox="1107 1323 1390 1379">Access all type definitions or create new types.</td> </tr> <tr> <td data-bbox="810 1402 922 1424"><i>Data Types</i></td> <td data-bbox="1107 1402 1390 1552">Create data types that define structures for input data streams, which you can use in further processing steps in the pipeline.</td> </tr> </tbody> </table>	Tab	Description	<i>Graphs</i>	Access built-in or custom graphs, and create custom graphs.	<i>Operators</i>	Access built-in or custom operators, and create custom operators.	<i>Repository</i>	Access and create Modeler objects, such as graphs, operators, and types in your repository. Create new folders, import auxiliary files or solutions, and export folders as .tgz files or vSolution files.	<div style="border-left: 2px solid #0070C0; padding-left: 10px;"> <p><b>i Note</b></p> <p>The Modeler provides individual Dockerfiles to create containerized environments for the operator groups. The Modeler selects the Dockerfiles using a tag-matching mechanism.</p> </div>		<i>Configuration Types</i>	Access all type definitions or create new types.	<i>Data Types</i>	Create data types that define structures for input data streams, which you can use in further processing steps in the pipeline.
Tab	Description														
<i>Graphs</i>	Access built-in or custom graphs, and create custom graphs.														
<i>Operators</i>	Access built-in or custom operators, and create custom operators.														
<i>Repository</i>	Access and create Modeler objects, such as graphs, operators, and types in your repository. Create new folders, import auxiliary files or solutions, and export folders as .tgz files or vSolution files.														
<div style="border-left: 2px solid #0070C0; padding-left: 10px;"> <p><b>i Note</b></p> <p>The Modeler provides individual Dockerfiles to create containerized environments for the operator groups. The Modeler selects the Dockerfiles using a tag-matching mechanism.</p> </div>															
<i>Configuration Types</i>	Access all type definitions or create new types.														
<i>Data Types</i>	Create data types that define structures for input data streams, which you can use in further processing steps in the pipeline.														
Bottom pane	Consists of the following tabs: <ul style="list-style-type: none"> <li>• <i>Status</i>: Monitor the status of the graph.</li> <li>• <i>Log</i>: Run trace messages based on severity levels.</li> <li>• <i>Schedule</i>: Monitor graph schedules.</li> <li>• <i>Validation</i>: Validate your graph to find errors.</li> </ul>														

Pane or Toolbar	Description
Editor toolbar	<ul style="list-style-type: none"> <li>• Perform operations on the graph in the graph editor, such as save and run a graph.</li> <li>• Define the configuration parameters for the graph, groups, and operators.</li> <li>• View details about various operators, including example graphs that the application provides.</li> </ul>

## Related Information

[Creating Graphs \[page 57\]](#)

[Creating Operators \[page 36\]](#)

[Creating Dockerfiles \[page 291\]](#)

[Monitor the Graph Execution Status \[page 101\]](#)

[Creating Configuration Types \[page 298\]](#)

[Error Recovery with Generation 2 \(Gen2\) Pipelines \[page 59\]](#)

## 3 Using Operators

Operators represent vertexes in a graph (pipeline) and are components that react to events configured in the graph environment.

Events are messages delivered to the operator through input ports. An operator interacts with the graph environment through its output ports. The operator is unaware of the graph in which it's defined and the source and target of its incoming and outgoing connections.

### i Note

Events can also be internal to the operator, such as clock ticks.

The following image shows an operator with input and output ports. Each port has a type. SAP Data Insight Modeler color codes the ports to indicate compatible port types.



For complete information about operators in SAP Data Intelligence Modeler, see the *Repository Objects Reference*.

#### [Operator Details \[page 15\]](#)

Every operator has an ID (also known as name) and a title (also known as the description). The operator ID is a unique identifier, with a strict format. The operator title is what the graphical interface displays.

#### [Generation 1 and Generation 2 Operators \[page 21\]](#)

SAP Data Insight Modeler offers two generations of operators: Generation 1 (Gen1) and Generation 2 (Gen2).

#### [Customizing the List of Operators \[page 21\]](#)

Limit the list of operator categories in the *Operators* tab to include only the operators that you use.

#### [Ports and Port Types \[page 22\]](#)

The operator uses ports as an interface to communicate between operators in a graph.

#### [Using Managed Connections in Script Operators \[page 34\]](#)

You can use managed connections in operator configurations of the script operators.

#### [Creating Operators \[page 36\]](#)

Use the SAP Data Intelligence Modeler to create your own operators to use in graphs (pipelines).

#### [Configuring Operators \[page 40\]](#)

Each operator has parameters that you can configure based on the business requirements for the graph (pipeline).

### [Creating Categories \[page 41\]](#)

Create a custom category for the operators or graphs that you create.

### [Creating Operator Groups \[page 41\]](#)

Partition a graph into subgroups so that each subgroup runs in a different Docker container assigned to different cluster nodes.

### [Viewing Operator Versions \[page 43\]](#)

You can view all existing versions, modify existing versions, or create additional versions of an operator. You can also replace a deprecated operator with the alternative operator and maintain logs for respective versions of an operator.

### [Editing Operators \[page 45\]](#)

You can modify the existing operators and use them in your graph. The Modeler provides a form-based editor to make changes to the operators.

### [Error Handling in Generation 2 Operators \[page 46\]](#)

The SAP Data Intelligent Modeler reports errors to a dedicated operator through an error output port.

### [Batch Header \[page 47\]](#)

Operators use batch headers to express information about its output batches in a unified way.

### [State Management \[page 48\]](#)

Use the state management feature by implementing a series of functions in the Python3 operator.

### [Dockerfile Library for Runtime Environment \[page 55\]](#)

Operators require a certain runtime environment. For example, if an operator executes some JavaScript code, then the operator requires an environment with a JavaScript engine.

## Related Information

[Ports and Port Types \[page 22\]](#)

[Graph Execution \[page 104\]](#)

[Creating Operators \[page 36\]](#)

## 3.1 Operator Details

Every operator has an ID (also known as name) and a title (also known as the description). The operator ID is a unique identifier, with a strict format. The operator title is what the graphical interface displays.

## Operator Extensions

All the operators available when creating a graph are known as extensions because they “extend” the base operators.

Base operators are visible when you create a new operator. The extension is expressed by a file in the Modeler file system. This file must be named `operator.json` and its folder hierarchy must match its ID. The Modeler names the file with the extension when you create the operator.

Example:

```
ID: 'com.sap.foo.bar'
Filepath: './operators/com/sap/foo/bar/operator.json'
```

## Operator JSON

The `operator.json` file contains the operator definition, including the graphical interface information. It has the structure listed in the following table.

Option	Required	Description
<code>description</code>	No	The operator title.
<code>icon</code>	Yes	The operator icon, expressed as a Font Awesome icon name that is available at <a href="https://fontawesome.com/icons/">https://fontawesome.com/icons/</a> .
<code>iconsrc</code>	Yes	The path to the SVG icon file. The path is relative to the <code>operator.json</code> .
<code>component</code>	Yes	The base operator ID to be extended.
<code>inports</code>	No	An array of input ports.
<code>outports</code>	No	An array of output ports.
<code>config</code>	No	A map of configuration parameters that map a configuration parameter ID to its default value.
<code>config.\$type</code>	Yes	A <code>\$type</code> field that points to its schema.
<code>tags</code>	No	A map of tags that map each tag ID to its default value.
<code>enableportextension</code>	No	A Boolean value that, if set to <code>true</code> , allows adding additional ports and configurations to the operator through the UI.
<code>extensible</code>	No	A Boolean value that, if set to <code>true</code> , allows a base operator to be extended.

### Note

`subenginestags` don't exist in the file system. They're included on the operator JSON for UI purposes.

`icon` and `iconsrc` are mutually exclusive; any field can be derived from the base operator (`component`).



The `operator.json` results in the following structure:

```
{
  "description": "<operator-title>",
  "icon": "<fontawesome-icon>",
  "iconsrc": "<icon-file>",
  "component": "base-operator-id",
  "inports": [
    {
      "name": "<inport1-id>",
      "type": "<inport1-type>"
    },
    ...
  ],
  "outports": [
    {
      "name": "<outport1-id>",
      "type": "<outport1-type>"
    },
    ...
  ],
  "config": {
    "<config-id>": "<config-value>",
    ...
  },
  "tags": {
    "<tag-id>": "<tag-value>",
    ...
  },
  "enableportextension": <true/false>,
  "extensible": <true/false>,
}
```

### ❖ Example

```
{
  "iconsrc": "read.svg",
  "component": "com.sap.storage.read",
  "config": {
    "$type": "http://sap.com/vflow/com.sap.storage.read.schema.json#"
  },
  "tags": {},
}
```

## Documentation

The operator documentation is a README file in markdown format. If the documentation makes sense only for the extension, the file must be named `README.md` and saved in the same folder as the `operator.json` file.

When you have multiple subengine implementations, where there are multiple `operator.json` files, each implementation must have a README in the same folder as the `operator.json` file.

### README structure

The following code shows the README file structure:

```
<operator-title>
===
<introduction>
```

```

<links-to-examples>
Configuration parameters
---
- <configuration-parameter-1>
- <configuration-parameter-2>
- ...
Input
---
- <input-port-1>
- <input-port-2>
- ...
Output
---
- <output-port-1>
- <output-port-2>
- ...

```

If an item list (parameters or ports) is empty, the word "None" must be listed.

### ❁ Example

```

Configuration parameters
---
- None

```

## Introduction

The introduction text must have the following content:

```

- configuration-id
  (type <configuration-type>, default: <configuration-default>)
  <!-- mandatory: only if applicable --> mandatory:
  <!-- brief description --> Lorem ipsum dolor sit amet, consectetur
  adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna
  aliqua.
  <!-- if needed, link to document with further description -->
  (Details are described here)[<link-to-config-docs>].
  - ID: `<configuration-id>`
  - Type: `<configuration-type>`
  <!-- Default: a value must be expressed according to its type formatting,
  e.g.:
    string -> `"value"`,
    int -> `42`,
    object -> `{ "k": ["v1", "v2"] }`,
    ...
  -->
  - Default: `<default-configuration-value>`
  <!-- Possible values: only valid for "enum" type -->
  - Possible values:
    - `<value-1>`
    - `<value-2>`
  <!-- Expected input: only valid when the "pattern" is set -->
  - Expected input: `<pattern-regex>`
  <!-- Additional specification fields may be provided -->

```

The Connection Protocol is mandatory and must have the following protocol in the request to service:

- ID: connProtocol
- Type: string
- Default: "HTTP"
- Possible values:
  - "HTTP"

- "HTTPS"

## Ports

Ports are identified by a unique ID (name). Ports are formatted as follows:

```
- **<port-id>** (type <port-type>): Express the parameter.
  If further is needed, document it in a separate file, and [link]()
  to it in this sentence. External documentation may be linked.
```

## Configuration Schema

When you name parameters, use the same standard you use to name operators.

You must provide a schema for a configuration. The schema contains parameters and further constraints for the UI. You must link a configuration schema in the `operator.json` file as follows:

```
{
  ...
  "config": {
    "$type": "<$id-from-schema>"
  },
}
```

Each parameter in the schema must meet the following criteria:

- Point to its ID with the object's key.
- Have a set title.
- Use the most strict type.
- Have a validation regex in pattern, if applicable.
- Be listed as required, if applicable.

Either write the schema manually or with the help of the [Types](#) panel of the Modeler application. Save the schema in one of the following ways:

- As `configSchema.json` in the same `operator.json` folder. Consider this method first, and when you use the Modeler application to create.
- As `schema.json` in the `/types/<operator-id>/schema.json` directory.

### 🔗 Example

The following code shows a brief example of a configuration schema:

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "http://sap.com/vflow/<operator.id>.schema.json",
  "title": "<schema-title>",
  "description": "<schema-description>",
  "type": "object",
  "properties": {
    "user": {
      "title": "User",
      "type": "string",
      "secure": true
    },
    "password": {
```

```

    "title": "Password",
    "type": "string",
    "secure": true,
    "format": "password"
  },
  "fooConnectionID": {
    "title": "Foo Connection ID",
    "description": "Connection ID used to connected to Foo",
    "type": "string",
    "format": "com.sap.dh.connection.id"
  },
  "isFoo": {
    "title": "Is Foo",
    "type": "boolean",
    "description": "Determines if operator is Foo.",
  },
  "reqMode": {
    "title": "Request Mode",
    "type": "string",
    "enum": [
      "Foo",
      "Bar"
    ]
  },
  "noOfReqs": {
    "title": "Number of Requests",
    "type": "integer",
    "description": "Number of Bar requests to be done.",
    "sap_vflow_constraints": {
      "ui_visibility": [
        {
          "name": "reqMode",
          "value": "Bar"
        }
      ]
    }
  },
  "filepath": {
    "title": "File path",
    "type": "string",
    "description": "File path to save request. Must start with '/'",
    "pattern": "^\\/.*$"
  }
}

```

## Schema Types

The following are the available types for a configuration parameter:

- "array"
- "boolean"
- "integer"
- "number"
- "object"
- "string"

## 3.2 Generation 1 and Generation 2 Operators

SAP Data Insight Modeler offers two generations of operators: Generation 1 (Gen1) and Generation 2 (Gen2).

Gen2 of vFlow operators communicate more efficiently between other Gen2 operators, but can't communicate with Gen1 operators. Therefore, when you create graphs in the Modeler, you first select either Gen1 or Gen2 operators.

Gen2 operators have the following advantages over Gen1 operators:

- More efficient graph (pipeline) recovery from errors.
- New structured types of native streaming of messages between operators.
- Support for statemanagement (snapshot) and auto recovery of failed graphs.
- Improved versions of Gen1 operators, such as the Python3 operator.

The following are the predefined Gen1 and Gen2 operator classifications:

- Operators that connect to messaging systems, such as Kafka, MQTT, NATS, and WAMP.
- Operators that store and read data, such as files, Hadoop Distributed File System (HDFS) and Amazon Simple Storage Service (Amazon S3).
- Operators that connect databases, such as SAP HANA and SAP Vora.
- Operators for the JavaScript engine that manipulate arbitrary data.
- Process operators that run any program (stateful and stateless) for manipulating data in a graph.
- Operators for data type conversion.
- Operators for digital signal processing.
- Operators for machine learning.
- Operators for image processing.

### → Remember

Gen2 operators can't communicate with Gen1 operators. Therefore, you can create graphs with just one generation of operators or the other, but not combined.


## 3.3 Customizing the List of Operators

Limit the list of operator categories in the *Operators* tab to include only the operators that you use.

### Context

The SAP Data Intelligence Modeler groups and displays operators in the navigation pane under specific categories, and initially lists all categories of operators. To control the categories listed so that you see only the operators that you use, perform the following steps:

## Procedure

1. Open the *Operators* tab from the Navigation pane at left.
2. Select  (*Customize Visible Categories*) next to the *Search* text box.

The *Customize Category Visibility* dialog opens showing a list of operator categories.

3. Deselect *Select All*.

This step is only necessary if you haven't edited the category list previously.

4. Check each category to view in the list of operator categories.
5. Click away from the list.

The selected categories and their operators appear in the *Operators* tab.

### → Tip

To display the operators in a specific category, enter the name of the category in the *Search* text box.

To find a specific operator, enter the name of the operator in the *Search* text box.

## 3.4 Ports and Port Types

The operator uses ports as an interface to communicate between operators in a graph.

A port definition includes the elements in the following table.

Element	Description
Purpose	Input or output port.  <b>i Note</b> There are no specific error ports. Use output ports to communicate error messages.
Name	Unique string that consists of alphanumeric characters only.
Type	String with a defined structure. The structure includes a mandatory base type and an optional semantic type. The semantic type can have a hierarchical substructure, separated with periods, and an optional wildcard at the end. The semantic type has the following form: <base type>.<semantic type>.  <b>❖ Example</b> <code>string.com.sap.base64.*</code> <code>com.mycompany</code>

Port types with a wildcard are called incomplete types. The following example shows a general port type specification:

### ❖ Example

```
int64.com.sap.base64.*  
[]blob.com.mycompany
```

You can use the semantics of the type specification to enrich types with additional information, which the owner of the types can use. However, the engine doesn't evaluate beyond the compatibility checks as described in the **Is compatible with type "any"** column in the following base types table.

The **Array** column in the table indicates whether the base type can use arrays. For example, `[]float64` can use arrays, but not `[]message`.

### i Note

Some subengines don't support all array types.

All port types fall into one of the built-in base types listed in the following table.

Type	Description	Is compatible with type "any"	Array
any	generic type	yes	no
string	character sequence	yes	yes
blob	binary large object	yes	yes
int64	8-byte signed integer	yes	yes
float64	8-byte decimal number	yes	yes
byte	single character	yes	yes
message	structure with header and body	no	no
stream	unstructured data stream	no	no

## Use Cases

The following list describes use cases for pipeline-specific types:

- Use the base type "any" when an operator is agnostic of the type and helps to avoid the redefinitions of the operator for each type. An example is the multiplexer operators.
- The base type "message" consists of a message header and the payload stored in the body. Messages have a size limit of 10 MB. The size limit means that larger payloads have to be split into chunks.

### ❖ Example

In the "Read File" operator, the header of the response messages contains the information to interpret the content of the body.

In other scenarios, such as for the “Copy File” operator, the input message triggers an operator to transfer data that is specified by the message, and the output message transfers the result of this operation. The header information can then be used to match the requests with the results. Therefore, it doesn't make sense to include arrays of messages themselves. However, arrays in the body are possible.

- The base type “stream” is special because the other types, including “any” or “message”, have a fixed structure at execution time (elementary type and length). Streams, in general, are unstructured. Typical examples are the IO streams `stdin`, `stdout`, `stderr` of the operating system or data streams generated by sensors.

## Conversions

In general, there are no implicit type conversions or propagations. If the port types are incompatible according to the rules, you can't run a graph. However, there are two exceptions to the rule: Input ports of type “message” and output ports of type “message”.

### Input ports of type “message”

In a graph that flows from left to right, if the output port of the operator feeding into the input port is a nonstream base type, the engine transforms the output into a message automatically. For “message” types, this behavior is obvious and for all other types, the engine generates a minimal message storing the output result in its body.

### Output ports of type “message”

The engine handles the incoming message automatically, but the outgoing message triggers an action in the Modeler application when you try to connect the ports.

#### ❖ Example

- Output port is of type “message”, “incomplete”, or “generic” that allows for a message to be passed.
- Input port of the receiving operator is of type “string”.

In these exception cases, select one of the following two methods to transform the outgoing message to a string:

- Concatenate the string from the serialized header and body of the message.
- Use only the body of the message and output it as a string.

#### i Note

If this body itself is a message, then it's handled as in the first case.

The choice depends on the semantics of the receiving operator. Therefore, there's no one recommended approach.

#### [Compatible Port Types \[page 25\]](#)

You can connect two operators only if the output port of the first operator and the input port of the second operator are compatible.

#### [Table Messages \[page 27\]](#)



A table message is an SAP Data Intelligence Modeler message that represents tabular data. The port type for table messages is `message.table`.

#### [Data Types in Operator Ports \[page 32\]](#)

Before you choose a data type for a new port, consider the type of connection and what the downstream or upstream operator accepts.

#### [Adding Ports to Operators \[page 33\]](#)

Add additional ports to JavaScript, Python, Multiplexer, and other extensible operators.

## Related Information

### 3.4.1 Compatible Port Types

You can connect two operators only if the output port of the first operator and the input port of the second operator are compatible.

The engine performs compatibility checks when you run the graph and when the engine loads the graph. If the port types aren't compatible, the engine fails the graph with a corresponding message in the trace.

The engine checks for compatibility in two steps as follows:

1. For the base types, the engine ensures that one of the following rules is true:
  - The base types of the operators are identical or
  - One of the base types is of type **any** and the other base type is one of the compatible types listed in the table in [Ports and Port Types \[page 22\]](#).
2. The semantic type of one port type is a specialization of the semantic type of the other port type. The semantic type, including the empty one, is a specialization of `*`.

#### **i** Note

Omitting the semantic type or empty semantic type, yields a complete type. This type is different from `<base type>.*`, which is an incomplete type.

The following table shows the compatibility of output and input port types.

Output Port Types	Input Port Types	Compatible	Reason
any	any	yes	Identical base type. No semantic type.
any	any.*	yes	Identical base type. Here, * can be substituted by the empty semantic type. Therefore, any is a specialization of any.*.

Output Port Types	Input Port Types	Compatible	Reason
any	string	yes	Compatible base types and no semantic type.
stream	any	no	Incompatible base types.
float64.*	int64.*	no	Incompatible base types.
any.*	string.*	yes	Compatible base types and identical semantic type.
any.*	string.com.sap	yes	Compatible base types. com.sap is a specialization of *.
any.*	string.com.sap.*	yes	Compatible base types. com.sap.* is a specialization of *.
any.com.sap	any.com.sap	yes	Identical base and semantic types.
string.com.*	string.com.sap.*	yes	Identical base type. com.* is a specialization of com.sap.*.
any	any.com	no	Identical base type, but com isn't a specialization of the empty semantic type. However, both sides are specializations of *.
any	any.com.*	no	Identical base type, but, com.* isn't a specialization of the empty semantic type.

## Related Information

[Ports and Port Types \[page 22\]](#)

## 3.4.2 Table Messages

A table message is an SAP Data Intelligence Modeler message that represents tabular data. The port type for table messages is `message.table`.

### Attributes

All table messages have an attribute named "table" with a value that's an object. The object has the properties described in the following table.

Property	Description
<b>version (required)</b>	Version of the table message type expressed as an integer.
<b>name</b>	Name of the table expressed as a string, such as the name of the database from which it came. If the name is case-insensitive, you must enter it in all uppercase letters. Otherwise, you must use the actual casing.
<b>columns</b>	Objects that describe each column of the table, expressed in an array. Each object has the following properties: <ul style="list-style-type: none"><li>• <b>Name:</b> String containing the name of the column. If the name is unknown, can be the empty string. If the name is case-insensitive, you must enter it in all uppercase letters. Otherwise, you must use the actual casing.</li><li>• <b>Class:</b> String containing the type class of the column. If the type is unknown, must be an empty string or one of the <a href="#">Supported types [page 28]</a>. Class names are always lowercase.</li><li>• <b>Type:</b> Object containing database-specific type information. The keys in this object must be the lower-case name of the database management system (DBMS), and each value a string with the type name.</li><li>• <b>Size:</b> Integer that specifies the column size limit, where applicable.</li><li>• <b>Precision and scale:</b> Integers specifying precision and scale of a decimal column.</li><li>• <b>Nullable:</b> Boolean indicating whether the column accepts NULL values.</li></ul>
<b>primaryKey</b>	Name of the primary key column expressed as a string, or an array of column names for a compound key.

### Body

The message body must be an array of arrays of generic elements using Go language: `[ ] [ ] interface { }`. The data in the body must always be row based. All rows must contain the same number of values.

## Supported Types

The term “class” refers to a group of similar column types commonly found in database management systems and file formats. For example, the string class can be found in the form of SQL types, such as VARCHAR and ALPHANUM.

The following table contains correspondence that is established between classes and the concrete data types used to hold their values.

### i Note

In the **Data type** column of the following table, “int” refers to these data types:

- int8
- uint8
- int16
- uint16
- int32
- uint32
- int64
- uint64
- int
- uint

Type Class	Data Type	String Format
timestamp	string	RFC 3339  RFC 3339 includes a numeric time zone (or Z for UTC) and an optional value of nanoseconds.  <b>❖ Example</b>  2006-01-02T15:04:05.999 999999Z07:00  Columns that store only part of a time-stamp must leave the unused portion at the zero value (midnight for the time and 0000-01-01 for the date).
integer	int	integer

Type Class	Data Type	String Format
decimal	int / float64/ string	<p>Decimal number with a dot as the separator or a fraction (<code>numerator/denominator</code>). You can suffix the int with the letter e followed by an exponent.</p> <div style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> <p>❖ <b>Example</b></p> <p>"1.43e-1" for the value 0.143.</p> </div> <p>Provide the value for a decimal by direct integer/float representation or as a string encapsulating a valid number.</p> <div style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> <p>❖ <b>Example</b></p> <ul style="list-style-type: none"> <li>• <b>Fraction:</b> "numerator/denominator" (for example, "3/4").</li> <li>• <b>Integer:</b> "integer" (for example, "50").</li> <li>• <b>Floating-point:</b> "floating-point" (for example, "2.5", "1.43e-1").</li> </ul> </div>
float	float64	<p>Decimal number with a dot as the separator or a fraction (<code>numerator/denominator</code>). You can suffix the int with the letter e followed by an exponent.</p> <div style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> <p>❖ <b>Example</b></p> <p>"1.43e-1" for the value 0.143.</p> </div>
string	string	N/A
binary	[]byte	Base 64 (RFC 4648, padded)
bool	bool	1, t, T, TRUE, true, True, 0, f, F, FALSE, false, or False

When a column's class is unknown, the engine can leave the operators' values as strings. Later in the graph, if a particular class is expected for that column, it's possible to choose to convert its values. In this case, the operator engine understands that the string is in the format specified under the **String format** column in the table. For example, when the operator is reading from a CSV file to insert its data into an existing database table: Column classes are unknown when parsing CSV, but some database operators require the table schema from the server and treat each column accordingly.

## Encoding

Even if an operator input is a general type of `message.*` or `any.*`, you must set the encoding to `table` so that operators can detect a table message.

## Examples

### Parsing CSV

#### ❖ Example

The following CSV data has a header for the first line (the column names):

```
ID,NAME,BIRTH,INTERNAL
0,John Doe,15-04-1982,no
1,Nancy Milburn,24-11-1991,yes
```

The CSV parser outputs the following table message (represented in JSON):

```
{
  "Attributes": {
    "table": {
      "version": 1,
      "columns": [
        {"name": "ID"},
        {"name": "NAME"},
        {"name": "BIRTH"},
        {"name": "INTERNAL"}
      ]
    }
  },
  "Encoding": "table",
  "Body": [
    ["0", "John Doe", "15-04-1982", "no"],
    ["1", "Nancy Milburn", "24-11-1991", "yes"]
  ]
}
```

Because the formats for timestamp and Boolean don't comply with the formats expected for a table message, you can use an intermediate operator to adapt the format into the following:

```
{
  "Attributes": {
    "table": {
      "version": 1,
      "columns": [
        {"name": "ID"},
        {"name": "NAME"},
        {"name": "BIRTH", "class": "timestamp"},
        {"name": "INTERNAL", "class": "bool"}
      ]
    }
  },
  "Encoding": "table",
  "Body": [
    ["0", "John Doe", "1982-04-15T00:00:00Z", false],
    ["1", "Nancy Milburn", "1991-11-24T00:00:00Z", true]
  ]
}
```

```
}
```

## Querying Database

### ❖ Example

The following table exists on an SAP HANA database:

ID (BIGINT)	SALARY (DECIMAL(10,2))	HIRED (DATE)
0	4560.50	'2003-01-14'
1	8740.50	'2001-07-28'

If an operator runs a `SELECT` statement on this table and outputs the values in a table message, the following format is expected:

```
{
  "Attributes": {
    "table": {
      "version": 1,
      "columns": [
        {"name": "ID", "class": "integer", "type": {"hana": "BIGINT"}},
        {"name": "SALARY", "class": "decimal", "precision": 10, "scale": 2, "type": {"hana": "DECIMAL"}},
        {"name": "HIRED", "class": "timestamp", "type": {"hana": "DATE"}}
      ],
      "primaryKey": "ID"
    }
  },
  "Encoding": "table",
  "Body": [
    [0, 4560.50, "2003-01-14T00:00:00Z"],
    [1, 8740.50, "2001-07-28T00:00:00Z"]
  ]
}
```

When there's a fraction (string) as decimal output format, such as in the 456 0.50 and 8740.50 in the sample code, the values are represented as "9121/2" and "17481/2", respectively.

### 3.4.3 Data Types in Operator Ports

Before you choose a data type for a new port, consider the type of connection and what the downstream or upstream operator accepts.

#### Data Type of None

In addition to the other data types of Global, Local, and Dynamic, you can also select None when you add a port to an operator. The None data type indicates a message with headers and no body. The None data type is meant for transmitting metadata and not data.

#### Any Port Type

The Modeler has provided port types that have input and output ports without any defined Data Type and Data Type ID. We call this type of port “any”. When you view the port details for the Terminal, Wiretap, and Graph Terminator operators, the *Data Type* and the *Data Type ID* options have an asterisk (\*) as a value.

The Modeler doesn't allow you to create an “any” port type.

#### Port Compatibility

The Modeler allows only certain types of ports to connect. The target port has to be at least as general as the source port.

A port with a defined Type ID can't receive a Dynamic type.

##### i Note

A port with a Dynamic scope has an asterisk (\*) as the *Data Type ID*. Dynamic ports are created during runtime and exist only in memory.

The Modeler doesn't allow you to connect ports that have specific Data Type IDs. If you ever require connecting incompatible port types, SAP recommends that you implement a custom solution with a script operator to receive data as one type and send it as another data type.

The following table lists characteristics of an output port with the compatible port characteristics of the input port.

Output port	Compatible input port
Port type: Dynamic	Data Type ID: Any
Data Type ID: Any	



Output port	Compatible input port
Data Type ID: Any	Port type: Dynamic Data Type ID: Any
Port type: Dynamic Data Type ID: Any	Data Type ID: Any
Port type: None	Data Type ID: Any
Port type: None	Port type: Dynamic Data Type ID: Any
Data Type ID: Any	Port type: Dynamic Data Type ID: Any
Port type: Dynamic	Port type: Dynamic

Keep in mind that, in all cases with dynamic port type, the data type of the connected ports have to match.

#### Example

You can connect an operator with a table type port and a dynamic scope to one of the following table ports: any, none, or dynamic.

## 3.4.4 Adding Ports to Operators

Add additional ports to JavaScript, Python, Multiplexer, and other extensible operators.

### Prerequisites

Before you can configure operators, create a new graph or edit an existing graph.

For more information about data types, see [Using Data Types \[page 302\]](#) and [Data Types in Operator Ports \[page 32\]](#).


## Context

To add additional ports to operators in graphs, perform the following steps in the Modeler:

## Procedure

1. Right-click the operator in the graph editor workspace and select *Add Port*.

The *Add Port* dialog box opens.

2. Enter a name in the *Name* text box.
3. Select either *Input Port* or *Output Port* as applicable.
4. Select  (*Browse*) in the *Data Type ID* text box.

The *Select Data Type* dialog box opens.

5. Choose a data type ID and click *Select*.
6. Select *OK*.

## Results

The new port appears on the operator. Hover your mouse pointer over the ports to view the port details.

## 3.5 Using Managed Connections in Script Operators

You can use managed connections in operator configurations of the script operators.

## Context

For this task, we use the Python3 operator as an example.

## Procedure

1. Create a new Python3 operator.

For instruction, see [Creating Operators \[page 36\]](#).

2. Add a new configuration parameter for the operator.

3. Add a new property and switch to *JSON* view.
4. Add a definition of an object-type parameter.

Copy and paste the following script in the JSON view:

### Note

This script is applicable only for the Python3 operator. In this example, we use the *OPENAPI* connection type as shown in the connection list. To return different connection types, change the value of the `connectionTypes` parameter in `properties.connection.connectionID.sap_vflow_valuehelp.url`.

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "http://sap.com/vflow/demos.conn.testme.configSchema.json",
  "type": "object",
  "properties": {
    "codelanguage": {
      "type": "string"
    },
    "script": {
      "type": "string"
    },
    "connection": {
      "title": "Connection",
      "type": "object",
      "properties": {
        "configurationType": {
          "title": "Configuration Type",
          "type": "string",
          "enum": [
            "",
            "Configuration Manager"
          ]
        },
        "connectionID": {
          "title": "Connection ID",
          "type": "string",
          "format": "com.sap.dh.connection.id",
          "sap_vflow_valuehelp": {
            "url": "/app/datahub-app-connection/connections?
connectionTypes=OPENAPI",
            "valuepath": "id",
            "displayStyle": "autocomplete"
          },
          "sap_vflow_constraints": {
            "ui_visibility": [
              {
                "name": "configurationType",
                "value": "Configuration Manager"
              }
            ]
          }
        }
      }
    },
    "required": [
      "connection"
    ]
  }
}
```

5. To use the connection properties, copy and paste the following code in the *Script* tab of the Python3 operator editor.

### ❖ Example

In the following code snippet, the host is read from `managedConnectionProperties`, which is chosen in the operator configuration at design-time.

```
def t1():
    managedConnection = api.config.connection
    managedConnectionProperties = managedConnection['connectionProperties']

    host = managedConnectionProperties['host']
    output = api.Message( host, {})
    api.send("out", output)
    api.add_timer("1s", t1)
```

6. Include the Python3 operator in a graph, and choose the managed connection.

## 3.6 Creating Operators

Use the SAP Data Intelligence Modeler to create your own operators to use in graphs (pipelines).

### Prerequisites

Before you create a new operator, ensure that you choose an existing folder in the repository or create a new folder. Create a new folder in the Modeler as follows:

1. Open the [Repository](#) tab in the Navigation pane at left.
2. Expand the [Operators](#) folder.
3. Right-click the operators folder and choose [Create Folder](#).

### Context

The Modeler provides a form-based editor to create operators. An operator is a reactive component, which means that it reacts to events from the environment. It isn't intended to terminate. The operators that you create in the Modeler are derived from the base operators that the application provides.

### Procedure

1. Right-click the applicable folder choose [Create Operator](#).

The [Create Operator](#) dialog box opens.

2. Enter the fully qualified path and file name in [Name](#).

3. Enter a name by which the operator is listed in *Display Name*.

The Modeler lists the operator by this name in the *Operators* tab. Also, use the name to search for the operator in the Modeler.

4. Choose the applicable base operator from the *Base Operator* list.

The base operators listed are derived from the built-in base operators provided by SAP Data Intelligence.

5. Choose the category in which to create the operator from the *Category* list and select *OK*.

Use the category as a search tool in the navigation pane.

The Modeler opens the operator editor in the main pane. There are several tabs in which you create the operator properties: *Ports*, *Tags*, *Configuration*, *Script*, and

6. Define the operator using the various tabs in the operator editor. The following table describes actions for each tab.

Action	Steps
Define ports	<ol style="list-style-type: none"> <li>1. Open the <i>Ports</i> tab.</li> <li>2. Select <b>+</b> (<i>Add input port or Add output port</i>).</li> <li>3. Enter a port name in <i>Name</i>.</li> <li>4. Choose a type in <i>Data Type</i>.</li> <li>5. Continue adding input or output ports in this manner as necessary.</li> </ol>
Define tags	<p>Tags describe the runtime requirements of operators and are the annotations of Dockerfiles that the application provides. If multiple implementations exist for the operator, the application displays the different subengines in which you can execute the operator. For each subengine, you can further associate them with tags from the database.</p> <ol style="list-style-type: none"> <li>1. Open the <i>Tags</i> tab.</li> <li>2. Select <b>+</b> (<i>Add Tag</i>).</li> <li>3. Choose a tag from <i>Tags</i>.</li> <li>4. Choose a version from <i>Version</i>.</li> <li>5. Continue adding tags in this manner as necessary.</li> </ol>


## Action

## Steps

Define configurations


The options in the *Configuration* tab vary based on the type of operator you're creating.

1. Open the *Configuration* tab.

The original schema name appears under Schema File. You can change the parameters of the schema file by automatic generation or by importing. To enable the automatic or import feature, select  (*Delete*).

2. For automatic generations, select *Generate Config Schema*.


3. To import, select *Import*.

4. To edit the schema, select  (*Edit Schema*) to open the *Edit type* dialog box.


Types allow you to enable semantic type validations on top of the configuration parameters and to define conditions. Create new types or edit existing type definitions in the repository. Reuse existing types to define the operator configuration.

### Note

By default, the type definition of script-based operators such as JavaScript, Python, Go operators, and so on, include a *Script* property. Use this property to view or modify the operator script when creating a graph. To hide the script from users working with this operator, edit the *Script* property in the type schema. Select the property and set the *Visibility* value to *Hidden*.


5. Select a property to edit and view the parameters to the right.
6. Add additional parameters by selecting  (*Add Parameter*). Provide the configuration parameter name, select the parameter type, and provide a parameter value.

### Remember

For some base operators, the application doesn't allow you to define new configuration parameters. If you aren't allowed to define new parameters,  (*Add Parameter*) is disabled, and you can only edit the existing parameter values.

Add scripts

In graphs, use operators that help run script snippets when the graph runs. To include script snippets in the operator definition, perform the following steps:

1. Open the *Script* tab.
2. Enter the script snippet in the *Inline Editor*.
3. To change the default programming language of the editor, choose a different language in the upper right. This option isn't always available.
4. To upload a script file from your system, choose *Upload File* from the list in the top-left corner of the editor and select  (*Upload file*) at right.


Action	Steps
Add operator documentation	<p>Maintain or modify documentation for operators in the <a href="#">Documentation</a> tab. Documentation contains information about the operator configuration parameters, input ports, output ports, and more. Adding documentation helps other users when working with this operator.</p> <ol style="list-style-type: none"> <li>1. Open the <a href="#">Documentation</a> tab.</li> <li>2. Enter the required documentation in the text area.</li> </ol>

→ Tip


SAP recommends that you use the Markdown format to maintain the operator documentation.

7. **Optional:** Select a display icon for the operator.

In the operator editor, the Modeler displays the operator name with a generic icon. You can replace the display icon with any of other icons that the application provides, or you can upload your own icon in Scalable Vector Graphics (SVG) format and use it for the operator display.

- a. Select the operator default icon that is to the left of the operator name.
- b. Select  ([Operator icon/image](#)).

The [Upload Operator Icon](#) dialog box opens.

- c. Select one of the following options based on the location of the icon file:
  - Select [From icon list](#) and choose an icon from the list.
  - Select [Upload SVG](#) and select  ([Upload File](#)).
- d. Select [OK](#).

The application uses the new icon for the operator on the graph editor.

i Note

The size of the icon tile in the operator editor is 80 px x 80 px. To ensure that the uploaded SVG file fits within the icon tile, define the `viewbox` property for the SVG file accordingly.

8. **Optional:** Upload auxiliary files.

You can upload auxiliary files to the repository that the operator can access or execute at runtime. This file can be any type, for example, binary executables such as a JAR file that the application must execute when executing the operator.

- a. In the operator editor toolbar, choose  (Upload Auxiliary File).

These files are stored along with the operators in the operator directory. You can access these files from the SAP Data Intelligence System Management application. Alternatively, you can also upload files from the [Repository](#) tab.

9. **Optional:** Select [JSON](#) in the upper right of the Modeler.

The JSON file opens showing the parameters for the new operator.

## Related Information

[Creating Configuration Types \[page 298\]](#)

## 3.7 Configuring Operators


Each operator has parameters that you can configure based on the business requirements for the graph (pipeline).

### Prerequisites


Ensure that the operator to configure is in a graph.


### Context

### Procedure


1. Select the applicable operator in your graph.
2. Select  (*Open Configuration*).  
The *Configuration* pane opens at right. The parameters in the Configuration pane have default settings based on the operator. To customize the operator, enter new values to the parameters. Based on the operator type, you can specify values to the subengines configuration parameters.

#### i Note

If multiple implementations exist for the operator, select the applicable subengine in which to run the operator. The default value is Main (Pipeline Engine). To select the applicable engine, choose  (*Add subengine*) and choose a subengine from the list.

3. To define new configuration parameters for the selected operator, perform the following substeps:
  - a. Select  (*Add Parameter*) in the *Configuration* pane.

#### i Note

You can define configuration parameters only for some base operators. Therefore,  (*Add Parameter*) is available only for applicable operators.

- b. Enter a name for the new configuration parameter in *Add Property*.
- c. Choose the property type.
- d. Enter a value in *Value*.



- e. Select *OK*.

## 3.8 Creating Categories

Create a custom category for the operators or graphs that you create.

### Procedure

1. Open the Repository tab in the navigation pane at left.
2. Expand the folder named “general” and then expand the subfolder named “ui”.
3. Double-click `settings.json`.

The JSON file opens in the main pane.

4. Add the category information to the array of categories listed.

The category name follows the same operator title naming constraints.

#### Example

```
{
  "name": "Category Name",
  "entities": [
    "com.sap.foo.bar",
    "com.sap.operator.test"
  ]
}
```

## 3.9 Creating Operator Groups

Partition a graph into subgroups so that each subgroup runs in a different Docker container assigned to different cluster nodes.

### Prerequisites

Before you perform the following task, open the applicable graph (pipeline) in which to create operator groups.

For complete information about creating and using operator groups, see [Groups, Tags, and Dockerfiles \[page 94\]](#).

## Context

An operator group can consist of one or many operators. Like individual operators, each group, called a subgraph, is associated with configuration parameters. Provide custom values for the parameters, and define additional configuration parameters for the group.

To create operator groups (subgraphs), open SAP Data Intelligence Modeler and perform the following steps in the applicable graph:


## Procedure

1. Press the `Shift` key and select each operator to include in the operator group.
2. Right-click and choose *Group*.

The Modeler shows the subgraph in a shaded box.

3. Select  (*Show Configuration*) in the editor bar.

The Modeler lists the predefined configuration parameters applicable for the group. The following table describes the parameters that you can define with custom values.

Parameter	Description
<i>Description</i>	Provides a description for the operator group.
<i>Restart Policy</i>	Determines the behavior of the cluster scheduler when a group execution results in a crash. Choose a value from the list. If you don't select a value, the Modeler uses the default value of <i>never</i> : <ul style="list-style-type: none"><li>• <i>never</i>: The cluster scheduler doesn't restart the crashed group and the crash results in the final state of the graph as "dead".</li><li>• <i>restart</i>: The cluster scheduler restarts the group execution. The restart changes the state of the graph from dead to pending and then to running.</li></ul>
	<div style="border-left: 2px solid orange; padding-left: 10px;"><b>⚠ Caution</b> When you restart a group, a single message per inbound connection can be lost.</div>
<i>Tags</i>	Describes the runtime requirement of groups. To define tags for the group, select  ( <i>Add Tags</i> ) and select the required tag and version.  You can assign more than one tag to a group.

Parameter	Description
<i>Multiplicity</i>	<p>Determines the number of executions for this group at runtime.</p> <p>Specify the value as an integer. For example, set to 3 to have the Modeler execute 3 instances of this group at runtime.</p> <p>When the Multiplicity is set to greater than 1, the Modeler sends the data arriving at the group in a round-robin fashion.</p>

4. **Optional:** To define new configuration parameters for the group, select **+** (*Add Parameter*).
  - a. Enter a name for the property in the *Add Property* dialog box.
  - b. Select the property type.

Select Text, JSON, or Boolean based on whether the property value is a string, JSON, or Boolean value.
  - c. Enter a value in *Value*.
  - d. Select *OK*.

## 3.10 Viewing Operator Versions

You can view all existing versions, modify existing versions, or create additional versions of an operator. You can also replace a deprecated operator with the alternative operator and maintain logs for respective versions of an operator.

### Context

See all existing versions of an operator by starting in the navigation pane or the graph editor.

- **Navigation pane:** Open the *Operators* tab, right-click the applicable operator, and choose *Show all versions*.
- **Graph editor:** Right-click the applicable operator and choose *Operator versions*.

### Related Information

[Replacing Deprecated Operators \[page 44\]](#)

[Editing Operator Versions \[page 44\]](#)

[Creating Operator Versions \[page 45\]](#)

## 3.10.1 Replacing Deprecated Operators

You can update a deprecated operator to the replacement operator available.


### Procedure

1. To replace a deprecated operator, in the graph editor, right-click the deprecated operator and choose the *Update* menu option.
2. In the *Replace Operator* dialog, click *Continue* to replace the deprecated operator with the suggested operator.



## 3.10.2 Editing Operator Versions

You can edit an existing version of an operator.

### Procedure


1. You can view the operator versions through the navigation pane or the graph editor:
  - In the navigation pane, choose the *Operators* tab, right-click the operator, and choose *Show all versions* menu option.
  - In the graph editor, right-click the operator and choose *Operator versions* menu option.
2. In the *Operator versions* dialog, choose the  (Edit) option.

The operator version editor opens.

3. To modify the existing configuration of the operator.
  - a. To provide the replacement operator if it's a deprecated operator,
    1. In the version editor of a deprecated operator, choose  (Enter Replacement Info).
    2. Enter the operator name with which you want to replace the deprecated operator.
  - b. To delete a version of the operator, in the *Operator Versions* dialog, select a version and click  (Delete).

#### **i** Note


When you try to delete an operator version that is used in multiple graphs, you will encounter a warning message, as the deletion would result in failure of the graphs in which that version of the operator is used.

- c. To view logs of a version of an operator, in the version editor, choose *View Change Log*.
4. When the changes are complete, in the editor toolbar, choose  (Save) to save the changes to the current version of the operator.


### 3.10.3 Creating Operator Versions

You can create a new version of an operator.

#### Procedure

1. You can view the operator versions through the navigation pane or the graph editor:
  - In the navigation pane, choose the *Operators* tab, right-click the operator, and choose *Show all versions* menu option.
  - In the graph editor, right-click the operator and choose *Operator versions* menu option.
2. In the *Operator versions* dialog, choose the  (Edit) option.

The operator version editor opens.


3. Modify the existing configuration of the operator.
4. When the changes are complete, in the editor toolbar, choose  (Save As New Version) to create a new version of the operator.
5. In the dialog that follows, enter the required details for the new version.
6. Click *Save*.

A new version of the operator is created.

### 3.11 Editing Operators

You can modify the existing operators and use them in your graph. The Modeler provides a form-based editor to make changes to the operators.

#### Procedure

1. To edit an existing operator, in the navigation pane, choose the *Operators* tab.
2. Right-click the operator that you want to edit and choose *Edit* menu option.
3. Modify the existing configuration of the operator.
4. When the changes are complete, in the editor toolbar, choose  (Save) to save the operator.

## 3.12 Error Handling in Generation 2 Operators

The SAP Data Intelligent Modeler reports errors to a dedicated operator through an error output port.

Error handling operators, whether generic or graph specific, are usually scripted. Generally, the errors communicated to an error operator are for internal use only.

The following table describes the options for error handling.

Option	Description
<i>Terminate on error</i>	Unexpected exceptions are logged and the graph is terminated.
<i>Log and ignore</i>	Unexpected exceptions are logged and the operator continues to run.
<i>Propagate to error port</i>	<p>Unexpected exceptions are sent to an <code>error</code> port that is generated at runtime.</p> <p>This port is of type structure (<code>com.sap.error</code>), which contains the following fields:</p> <ul style="list-style-type: none"><li>• <code>code</code>: an error code (<code>int32</code>) that is local to the operator.</li><li>• <code>operatorID</code>: a string with the ID of the operator outputting this error (for example, <code>com.sap.system.terminal</code>).</li><li>• <code>processID</code>: a string with the ID in the graph of the process outputting this error (for example, <code>terminal1</code>).</li><li>• <code>text</code>: a string corresponding to the main error message.</li><li>• <code>details</code>: a table type, which is composed of the fields <code>key</code> and <code>value</code> (both of type <code>string</code>), which can be used to provide further information about the exception.</li></ul> <p>The operator continues to run.</p>

### Error Output Port

Add an error output port to the operator that outputs to the specific error operator. When you create the port, make sure to include "error" in the name, such as `com.sap.error` or `com.sap.error.details`. Error output ports can be either structure or table types.

## 3.13 Batch Header

Operators use batch headers to express information about its output batches in a unified way.

The batch header consists of the following elements:

- **Kind:** `structure`
- **Type ID:** `com.sap.headers.batch`

The following table describes the batch header fields.

Field	Description
<code>index</code>	Integer with the zero-based index of the batch.
<code>isLast</code>	Boolean that indicates whether the batch is the last in the sequence, for example, <code>if batchSize == batchCount - 1</code> .
<code>count</code>	Total number of batches in the sequence. If the operator can't obtain the <code>count</code> because of technical restrictions or because the sequence is open-ended (a stream), for example, operators can omit <code>count</code> .
<code>size</code>	<p>Number indicating the magnitude that delimits one batch from the next. For example, the number of bytes or the number of lines of text the batch contains. When you use other criteria, such as a timeout for the accumulation of each batch, you can omit <code>size</code>.</p> <p>Size doesn't express the size of an individual batch. Rather, <code>size</code> expresses the intended size for batches in general. Size is usually specified by the user.</p>
<code>unit</code>	<p>String containing the unit in which <code>batchSize</code> is measured. If present, <code>unit</code> is expressed as one of the following measures:</p> <ul style="list-style-type: none"><li>• <code>bytes</code>.</li><li>• <code>rows</code>, for table-related operators (including CSV, databases, and so on).</li><li>• <code>lines</code>, for text files.</li></ul>

### Example

#### ❖ Example

You configure a Read File operator to read files in 10-byte chunks. If the file has a total of 87 bytes, the operator outputs the following values for the batch attributes:

- `index` in the range `[0,8]`

- `isLast`: false in all but the ninth and last output (index 8)
- `count`: 9
- `size`: 10
- `unit`: bytes

## 3.14 State Management

Use the state management feature by implementing a series of functions in the Python3 operator.

Use the functions in the following table to support recovery and to store states. SAP Data Intelligence doesn't require that you implement all of the functions to store states.

Function	Description
<code>api.set_initial_snapshot_info(initial_process_info: api.InitProcessInfo)</code>	<p>Sets the initial information for state management before the operator starts. You can call the initial information outside callback functions.</p> <p>The parameter <code>initial_process_info</code> has two attributes:</p> <ul style="list-style-type: none"> <li>• <code>is_stateful</code>: Boolean that specifies that the operator persists states.</li> <li>• <code>outputs_info</code>: Map of port names to <code>api.OutputInfo</code>. <code>api.OutputInfo</code> is a class of one attribute: <code>is_generator</code>. The parameter <code>outputs_info</code> indicates whether output ports of the script operator are generators.</li> </ul>
<code>api.set_restore_callback(callback)</code>	<p>Register function that restores the operator state.</p> <p><b>Arguments:</b> <code>callback</code> (<code>func[str, bytes] -&gt; None</code>).</p> <p>The function expects the following input parameters:</p> <ul style="list-style-type: none"> <li>• Epoch, which uniquely identifies the state that is being recovered.</li> <li>• Serialized operator state.</li> </ul>
<code>api.set_serialize_callback</code>	<p>Register function that returns the operator state.</p> <p><b>Arguments:</b> <code>callback</code> (<code>func[str] -&gt; bytes</code>).</p> <p>The function expects the epoch as a parameter. Epoch uniquely identifies the state that is being recovered. It returns the serialized operator state.</p>



Operators with support to state management can have two extra classes: generator and writer.

### Generator

A generator is a port that produces outputs independent of input port data. All graphs that have data flowing have at least one generator output port. Examples of generators include the following:

- operators reading files with no input connected
- Kafka consumers

### Writers

A writer is an operator that has effects outside the graph (for example, operators writing into databases, or operators writing to a publisher or subscriber queue). If an operator is a writer and it's stateful, it offers an "at-least-once" guarantee. An "at-least-once" guarantee means that there's a guarantee that no data is lost, even though it can be written twice. Being stateful requires implementing the restore and serialize functions. It's also possible to have an exactly once guarantee, which requires the writer to be idempotent and stateful. Idempotency means equivalency when writing the same data several times.

Keep in mind the following information regarding state management:

- To avoid deadlocks, don't block a port callback while it waits for another port callback.
- Before saving a state, the Modeler doesn't pause the shutdown function. Therefore, the shutdown function doesn't change the operator state.
- If the operator has asynchronous behavior, the operator must support pause and resume.

For more information about these functions, see the Python3 Operator V2 section of the *Repository Objects Reference* guide.

## Example

### Sample Code

```
counter = 0
def on_input(msg_id, header, body):
    global counter
    counter += 1
    api.outputs.output.publish(str(counter))
api.set_port_callback("input", on_input)
api.set_initial_snapshot_info(api.InitialProcessInfo(is_stateful=True))
def serialize(epoch):
    return pickle.dumps(counter)
api.set_serialize_callback(serialize)
def restore(epoch, state_bytes):
    global counter
    counter = pickle.loads(state_bytes)
api.set_restore_callback(restore)
def complete_callback(epoch):
    api.logger.info(f"epoch {epoch} is completed!!!")
api.set_epoch_complete_callback(complete_callback)
```

## Related Information

[Examples: Operator States \[page 50\]](#)

[Error Recovery with Generation 2 \(Gen2\) Pipelines \[page 59\]](#)

### 3.14.1 Examples: Operator States

This section contains examples illustrating the use of state management functions in the Python3 operator for graph snapshots and operator states for Gen2 graphs.

#### Exactly Once Graph Structure

The following example uses this exactly-once graph structure:

► [PythonOperator Generator](#) ► [Deterministic Processing](#) ► [Idempotent Writer](#) ►

Assumptions:

- The example doesn't include instances of the processing operator because the processing operator can perform any operation, stateful or not, as long as it's deterministic. Therefore, if the graph is fed with the same inputs, the same outputs are expected.
- The graph has a linear topology, which means that the operators don't have to fan in or out messages.
- The PythonOperator Generator accesses an SAP HANA table to which the system appended its content during the graph execution.

#### ❖ Example

The following code snippet shows the script:

```
''' python
    from hdbcli import dbapi
offset = 0
c = None
api.set_initial_snapshot_info(api.InitialProcessInfo(is_stateful=True, outputs
_info={'output': api.OutportInfo(is_generator=True)}))
# Operator will have batches and it will be Replayable
def prestart():
    conn = dbapi.connect(
        address=api.config.connection['connectionProperties']['host'],
        port=api.config.connection['connectionProperties']['port'],
        user=api.config.connection['connectionProperties']['user'],
        password=api.config.connection['connectionProperties']['password'],
        sslHostNameInCertificate='*',
        sslValidateCertificate=False
    )
    global c
    c = conn.cursor()

def time_callback():
    global offset
    # The order by guarantees the same order when replaying
```

```

c.execute(f"SELECT * FROM test ORDER BY x LIMIT 2 OFFSET {offset};")
test_data = c.fetchall()
c.close()
offset += 2
if len(test_data) > 0:
    t = [[entry for entry in row] for row in test_data]
    api.outputs.output.publish(api.Table(t))
return 0
api.add_timer(time_callback)

api.set_prestart(prestart)
# Since we assume new entries will go to the end of the table, we can only
save the offset
# and not the table.
def serialize(epoch):
    return pickle.dumps(offset)
api.set_serialize_callback(serialize)
def restore(epoch, state_bytes):
    global offset
    offset = pickle.loads(state_bytes)
api.set_restore_callback(restore)

```

## Idempotent Writer Script

### ❁ Example

Continuing with Example 1, the idempotent writer also writes to an SAP HANA table through the following script:

```

from hdbcli import dbapi
offset = 0
c = None
def prestart():
    conn = dbapi.connect(
        address=api.config.connection['connectionProperties']['host'],
        port=api.config.connection['connectionProperties']['port'],
        user=api.config.connection['connectionProperties']['user'],
        password=api.config.connection['connectionProperties']['password'],
        sslHostNameInCertificate='*',
        sslValidateCertificate=False
    )
    global c
    c = conn.cursor()

def on_input(msg_id, header, body):
    data = body.get().body

    # Upsert guarantees idempotency
    sql = 'UPSERT test (x, y, z) VALUES (?, ?, ?) where x=?'
    for row in data:
        row.append(row[0])
        c.execute(sql, row)
    c.close()

api.set_port_callback("input", on_input)
api.set_prestart(prestart)

```

## No Idempotent Guarantee

### ❁ Example

The following example uses the same setup as the Exactly Once Graph Structure example, but the writer isn't guaranteed to be idempotent. To have the exactly once guarantees, the graph has to avoid writing duplicate batches through an auxiliary table in the same SAP HANA database.

The auxiliary table can be another persistence structure. In this example, the auxiliary table is a Write-Ahead Log (WAL). To preserve already seen batches, the table has to survive multiple-graph runs. Therefore, create and delete the WAL outside the graph context.

The following script shows the implementation of the generator and writers as Python Operators:

```
''' python
from hdbcli import dbapi
import pickle
offset = 0
ID = 0
c = None
# Table is broken into messages of 100 rows, and each made of 10 batches fo
10 rows
batch_size = 10
message_size = 100
api.set_initial_snapshot_info(api.InitialProcessInfo(is_stateful=True, outputs
_info={'output': api.OutputInfo(is_generator=True)}))
def prestart():
    conn = dbapi.connect(
        address=api.config.connection['connectionProperties']['host'],
        port=api.config.connection['connectionProperties']['port'],
        user=api.config.connection['connectionProperties']['user'],
        password=api.config.connection['connectionProperties']['password'],
        sslHostNameInCertificate='*',
        sslValidateCertificate=False
    )
    global c
    c = conn.cursor()

final = False
def time_callback():
    global offset, ID
    # The order by guarantees the same order when replaying
    c.execute(f"SELECT * FROM test ORDER BY x LIMIT {batch_size} OFFSET
{offset};")
    test_data = c.fetchall()
    c.close()
    global final
    if len(test_data) > 0 and not final:
        api.logger.info('sending1')
        t = [[entry for entry in row] for row in test_data]
        h = {}
        if len(test_data) < batch_size:
            h['isFinal'] = [True]
            final = True
        else:
            h['isFinal'] = [False]
        h['batchNum'] = [offset // batch_size]
        h['messageNum'] = [offset // message_size ]
        # The last batch has the flag set to true, this is important so the
writer operator can clean up the WAL
        if (offset + batch_size) % message_size == 0 and offset > 1:
            h['isFinal'] = [True]

    h['ID'] = [ID]
```

```

        api.outputs.output.publish(api.Table(t), h)
        api.logger.info('after')
        ID += 1
        offset += batch_size
    return 0
api.add_timer(time_callback)

api.set_prestart(prestart)
# Since we assume new entries will go to the end of the table, we can only
save the offset
# and not the table.
def serialize(epoch):
    return pickle.dumps([offset, ID])
api.set_serialize_callback(serialize)
def restore(epoch, state_bytes):
    global offset, ID
    offset, ID = pickle.loads(state_bytes)

api.set_restore_callback(restore)

```

## Idempotent Guarantee That Accesses SAP HANA Table

### ❖ Example

The writer also accesses an SAP HANA table in the following script:

```

''' python
import pickle
from hdbcli import dbapi
offset = 0
messages_done = {}
c = None
api.set_initial_snapshot_info(api.InitialProcessInfo(is_stateful=True))
def prestart():
    conn = dbapi.connect(
        address=api.config.connection['connectionProperties']['host'],
        port=api.config.connection['connectionProperties']['port'],
        user=api.config.connection['connectionProperties']['user'],
        password=api.config.connection['connectionProperties']['password'],
        sslHostNameInCertificate='*',
        sslValidateCertificate=False
    )
    global c
    c = conn.cursor()

def insert_wal(ID, batch_num, message_num):
    sql = 'INSERT INTO WAL VALUES(?,?,?)'
    c.execute(sql, [ID, batch_num, message_num])
    c.close()

def is_batch_new(batch_num, message_num):
    c.execute(f'SELECT ID FROM WAL WHERE BATCH={batch_num} AND
MESSAGE={message_num}')
    test_data = c.fetchall()
    c.close()
    return len(test_data) == 0

def on_input(msg_id, header, body):
    api.logger.info('Receiving')
    data = body.get().body

    batch_num = header['batchNum'][0]
    message_num = header['messageNum'][0]

    if header['isFinal'][0]:
        api.logger.info('Final message')

```

```

    global messages_done
    messages_done[message_num] = None

    # It will check if messageID and batchID do not exist in the WAL table,
    if they do, message is ignored
    if not is_batch_new(batch_num, message_num):
        return

    api.logger.info('Inserting into test2')
    sql = 'INSERT INTO test2 (x, y, z) VALUES (?, ?, ?)'
    for row in data:
        row.append(row[0])
        c.execute(sql, row)
    c.close()

    api.logger.info('Inserting into wal')
    # Insert into WAL table, and if message is done, insert accordingly
    insert_wal(header['ID'][0], batch_num, message_num)

    api.logger.info('sending to output')
    api.outputs.output.publish(str(data))

api.set_port_callback("input", on_input)
api.set_prestart(prestart)
# Any messages that have been finished by this point are eligible to be
removed from
# WAL when the epoch is completed.
def serialize(epoch):
    global messages_done
    api.logger.info(f'setting epochs {messages_done}')
    for messageID, val in messages_done.items():
        if val is None:
            messages_done[messageID] = epoch

    # The writer has to keep as state the map, since finished messages may
not have had their epochs processed
    return pickle.dumps(messages_done)
api.set_serialize_callback(serialize)
def restore(epoch, state_bytes):
    global messages_done
    messages_done = pickle.loads(state_bytes)
api.set_restore_callback(restore)
# Any message whose corresponding epoch existing could
# be removed as we know the epoch has finished
def complete_callback(epoch):
    global messages_done
    api.logger.info(f'removing from wal {messages_done}')
    for messageID, e in messages_done.items():
        if e == epoch:
            api.logger.info(f'removing message {messageID}')
            c.execute(f'DELETE FROM WAL WHERE MESSAGE=?', [messageID])
            c.close()
            # cleaning up ended epochs
            del messages_done[messageID]
api.set_epoch_complete_callback(complete_callback)
\`

```

## 3.15 Dockerfile Library for Runtime Environment

Operators require a certain runtime environment. For example, if an operator executes some JavaScript code, then the operator requires an environment with a JavaScript engine.

The Modeler provides predefined environments for operators, and these environments are available as a library of Dockerfiles.

When you execute a graph, the application translates each operator in the graph into processes. It then searches the Dockerfiles for an environment suitable for the operator execution and instantiates a Docker image.

### **i** Note

The Docker image with the environment and the operator process is executed on a Kubernetes cluster.

## 4 Using Graphs (Pipelines)

Graphs, also known as pipelines, are a network of operators connected by typed input and output ports for data transfer.

There are two types of graphs based on the type of operators you choose to build the graph. When you create a graph, you must select to use either Generation 1 (Gen1) or Generation 2 (Gen2) operators. Graphs can't contain a combination of Gen1 and Gen2 operators. A graph created with Gen1 operators is a Gen1 graph. A graph created with Gen2 operators is a Gen2 graph.

For more information about operators, see [Using Operators \[page 14\]](#). For more information about Gen1 and Gen2 operators, see [Generation 1 and Generation 2 Operators \[page 21\]](#).

### [Creating Graphs \[page 57\]](#)

A graph (pipeline) consists of operators that you configure to form a specific process and connect using input and output ports.

### [Error Recovery with Generation 2 \(Gen2\) Pipelines \[page 59\]](#)

Gen2 pipelines (graphs) make it possible to recover from errors using specific runtime features.

### [Graph Snapshots and Operator States \[page 61\]](#)

You can configure Generation 2 (Gen2) graphs to take snapshots of their state at regular time intervals so that the operators of the graph send small pieces of data (status) to a central data store.

### [Delivery Guarantee for Generation 2 \(Gen2\) Graphs \[page 66\]](#)

When you enable automatic graph recovery and snapshots for a Gen2 graph (pipeline), your graphs can outlast system failures and system maintenance events.

### [Validate Graphs \[page 69\]](#)

Graph validation is an automatic or manual process that analyzes a graph for correct structure and components, such as operators, ports, groups, and configuration.

### [Running Graphs \[page 72\]](#)

After creating a graph, you can run the graph based on the configuration defined for the graph. The Modeler application runs the operators in the graph as individual processes.

### [Maintain Resource Requirements for Graphs \[page 87\]](#)

Specify compute resource requirements, such as CPU and memory limits, for graph groups in SAP Data Intelligence Modeler.

### [Create Data Types in Graph \[page 91\]](#)

You can create graph-level data types and use them in the graph along with the automatically generated data types.

### [Groups, Tags, and Dockerfiles \[page 94\]](#)

Groups, tags, and Dockerfiles are essential parts of the SAP Data Intelligence environment for running graphs (pipelines) more efficiently. Therefore, you must understand how they work together.

### [Execution Model \[page 99\]](#)

To avoid problems, such as back pressure and deadlocks, SAP Data Intelligence Modeler executes graphs following an execution model.

### [Monitoring Graphs \[page 100\]](#)

After creating and running graphs, monitor graphs and view statistics.



[Native Multiplexing for Gen2 Pipelines \[page 115\]](#)

Connect to multiple ports in a pipeline, such as one to many or many to one, without having to implement multiplexing with a script operator or other predefined operator.

## Related Information

### 4.1 Creating Graphs

A graph (pipeline) consists of operators that you configure to form a specific process and connect using input and output ports.

#### Prerequisites

SAP Data Intelligence Modeler provides Generation 1 (Gen1) and Generation 2 (Gen2) operators. Before you create a graph, determine which group of operators to use. You can build a graph using either Gen1 or Gen2 operators, but you can't combine them in a single graph. For complete information about Gen1 and Gen2 operators, see [Generation 1 and Generation 2 Operators \[page 21\]](#).

A graph can contain a single operator, or a network of operators based on the purpose of the graph.

#### Context

To create a basic graph, open the Modeler application and perform the following steps:

#### Procedure

1. In the navigation pane at left, open the *Graphs* tab.
2. Select the down arrow next to **+** (*Create Graph*) and choose either *Use Generation 1 Operators* or *Use Generation 2 Operators*.

The Modeler opens the *Operators* tab. The tab lists only the operators belonging to the generation you selected. The Modeler also opens an empty graph editor to the right of the navigation pane. Use the graph editor area to create the graph.

#### → Tip


The Modeler groups the operators in the *Operators* tab under specific categories. To customize the list of operators, use the  (*Customize Visible Categories*) icon or use the *Search* bar.

For more information about operator types and categories, see the *Repository Objects Reference*.

3. Double-click the first operator for your graph in the *Operators* tab.

The Modeler adds the operator to the graph editor workspace. Add additional operators as necessary in the same manner.

4. Each operator has default configuration settings. You can change default settings or create additional configuration parameters. To configure each operator based on its purpose, perform the following substeps:

- a. Select the operator in the graph editor workspace.
- b. Select  (*Show Configuration*) next to the operator.

The Modeler opens a *Configuration* pane at right. Based on the operator type, you can also specify values for the subengine configuration parameters.

- c. Configure each operator in your graph in the same manner.

For more information about operator configuration settings, see the *Repository Objects Reference*.

5. **Optional:** Select *Validate* in the *Configuration* pane.

If the operator has a type scheme associated with it, then validate the configuration parameters' values against the conditions defined in the schema. For example, validate mandatory fields, minimum or maximum length, value formats, regular expression, and so on.


### Note

The validation is based on the constraints defined in the schema. The Modeler validates all configuration parameter values and displays validation errors, if any.

6. Connect the operators: Drag and drop your cursor from the output port of one operator to an input port of another operator.


Continue connecting the operators in your graph so that all operators are connected in the order in which to process the data. The Modeler helps you select the allowable input port type by highlighting all input ports based on the output port type.


7. **Optional:** To configure the graph, perform the following substeps:

- a. Ensure that no individual object in the graph is selected, then select  (*Show Configuration*).
- b. Complete the parameters as described in the following table.

Parameter	Description
<i>Description</i>	Enter a description of the graph.
<i>Icon file name</i>	Enter the icon name, such as <code>kafka.png</code> .
<i>Icon</i>	Enter the Font Awesome icon name, or choose an option from the list.  The Modeler uses the icon for display only when you don't provide a value for <i>Icon file name</i> .

Parameter	Description
<i>Disable lineage</i>	Slide the toggle to ON or OFF.  Applicable only for graphs that contain certain operators that support lineage extraction. You can use the Metadata Explorer tool to view data lineage.

8. Choose *Save* from the  (*Save*) list.
9. Enter the fully qualified path and file name for the graph in *Name*, and optionally enter a description in *Description*.
10. Choose the applicable value from the *Category* list, or enter a new category.
11. Select *OK*.

The Modeler saves the graph and operators in a folder structure in the modeler repository, such as `...com/sap/others/<graphname>`. To save another instance of the graph, in the editor bar, choose *Save As* from the  (*Save*) list and provide the applicable information in the *Save* dialog box.

12. **Optional:** To export the graph to the JSON definition after you create and save the graph, perform the following substeps:
  - a. Open the *Graphs* tab in the navigation pane at left.
  - b. Right-click the applicable graph and choose *Export*.

The Modeler creates the JSON file and places it in your Downloads folder.

## Related Information

[SAP Data Intelligence Operators](#)

[Creating Operators \[page 36\]](#)

[Monitor the Graph Execution Status \[page 101\]](#)

## 4.2 Error Recovery with Generation 2 (Gen2) Pipelines

Gen2 pipelines (graphs) make it possible to recover from errors using specific runtime features.

Configure the following features at runtime for your Gen2 graphs to aid in pipeline recovery when errors occur:

- **Auto Restart:** Graphs restart automatically when the pipeline fails or is evicted.
- **Snapshots:** Graphs create periodic snapshots of the current operation. Benefits of snapshots include the following:
  - Recover the operation when there are failures, pauses, or system upgrades.
  - Save information about individual tasks, such as row last read, so that you can view task information when you recover the pipeline.
  - Restart a pipeline at the point of error. Script operators implement an API for saving and restoring the state, which allows you to implement more complex use cases.

## i Note

For more information about snapshots and limitations, see [Graph Snapshots and Operator States \[page 61\]](#).

- **Streaming API:** Use streaming API during data transfer. Loads small chunks of data into system memory instead of data from the entire pipeline. Streaming API has the following benefits:
  - Reduce overall memory usage: Operators send and consume data in small chunks inside the same stream, which reduces memory usage.
  - Combine with batch: Operators send each batch as a stream, and include the metadata for each batch inside the headers.

## → Tip

When you implement runtime features for Gen2 graphs, use standardized error handling so that each operator uses the same methods for processing errors.

To design and operate recoverable Gen2 graphs, use the following methods:

- Gen2 Runtime:

Option	Description
Auto Restart	Configure graphs to restart automatically on failure or eviction. Set the <i>Automatic Recovery</i> options in the <i>Run As</i> dialog box.
Snapshots	Configure graphs to save a snapshot of the current operation periodically. The snapshot allows recovery of the operation when the graph fails, pauses, or when there are system upgrades. Operators save information about individual tasks, such as which row in a table was last updated, and receive that information on recovery, avoiding the need to restart the whole operation from the beginning. There are also functions for script operators that implement an API for saving and restoring state as well, allowing users to implement more complex use cases.

- Streaming: Operators support a streaming API for data transfer that doesn't require loading all information into memory. Operators can send and consume data in small chunks inside the same stream, reducing overall memory usage. Sending data through batches is still possible but it can be combined with streams. Operators can send each batch as a stream with the metadata for each batch being sent inside the headers.
- Error Handling: Each operator has a standardized way of handling errors.

## Related Information

[Graph Snapshots and Operator States \[page 61\]](#)

[State Management \[page 48\]](#)

## 4.3 Graph Snapshots and Operator States

You can configure Generation 2 (Gen2) graphs to take snapshots of their state at regular time intervals so that the operators of the graph send small pieces of data (status) to a central data store.

If the graph fails or pauses, SAP Data Intelligence Modeler uses the last status data to reinitialize to the last state before the failure or pause happened.

### ❖ Example

If a graph is processing a large database table, its operator regularly saves which rows it has already processed. When recovering from a failure, the operator reads its last used state and continues processing from the specific row instead of starting over from the beginning of the table.

If you also enable the snapshot feature, recovery of stateful graphs is more efficient than running with recovery only. Enable snapshots only when the Gen2 graph is conceptually stateful. If a graph doesn't have any state to save, it's more efficient to not capture snapshots.

## Keeping State Size Small

To keep the status data per operator small, design a computation method that incrementally maintains a small state size. Ensure that you design a computation method when you design the operator and before you implement the operator.

### i Note

The size of the stored state depends on the stateful computation by the operator, and not by the snapshot frequency.

### ❖ Example

Calculate the total sales by city and send sales records to the operator in the graph through its input port, and manage the state size. There are two methods to achieve this goal.

#### Method 1

1. Group all records received by each city.
2. After the last record is received, compute the sum of sales in each city group.
3. Return the result.

For Method 1, the size of the state is roughly the size of all records received by the operator, no matter the snapshot frequency.

#### Method 2

Design an incremental version of the Method 1 computation:

1. Maintain a state where, for each city, you compute the current sum of sales amount.
2. After all records are received, the state contains the result that is returned.

For Method 2, the size of the state is roughly the number of cities with their sum, independent of the snapshot frequency.

## Operators That Support Snapshots

Not every Gen2 operator supports the snapshot feature. When you configure and run a Gen2 graph, but some of the operators in the graph don't support the snapshot feature, the graph can't reinitialize to the state before a failure or a pause, and it can lose data. Therefore, ensure that a graph that you run with the [Capture Snapshots](#) option enabled contains only operators that support snapshots.

The following table lists all Gen2 operators, whether they support snapshots, and any conditions that apply.

Category	Operator	Operator ID	Supports Snapshots	Conditions and Remarks
ABAP	Read Data From SAP System	com.sap.abap.reader	Yes	Stores state: list of package IDs in process.
Connectivity	Kafka Consumer	com.sap.kafka.consumer2.v2	With conditions	For details about conditions, see the section <a href="#">Snapshot Support in Kafka Consumer V2</a> .
Connectivity	Kafka Producer	com.sap.kafka.producer.v2	With conditions	For details about conditions, see the section <a href="#">Snapshot Support in Kafka Producer V2</a> .
Connectivity	REST API Client	com.sap.restapi.client	Yes	For details about conditions, see the section <a href="#">Snapshot Support in Rest API Client</a> .
Connectivity (via Flow-agent)	Flowagent SQL Executor	com.sap.dh.ds.sql.executor.v2	No	None
Data Quality	Validation Rule	com.sap.dh.dq.validationrule.v2	No	None
Files	Binary File Consumer	com.sap.file.read.v2	With conditions	Stores state: index of the part file last read.  For details about conditions, see the section <a href="#">Snapshot Support in Binary File Consumer</a> .
Files	Binary File Producer	com.sap.file.write.v2	With conditions	Stores state: See the section <a href="#">State Management Support in Binary File Producer</a> .
Files	List Files	com.sap.file.list.v2	With conditions	Stores state: index of file in the files list.  For details about conditions, see the section <a href="#">Snapshot Support in List Files V2</a> .

Category	Operator	Operator ID	Supports Snapshots	Conditions and Remarks
Processing	Python3 Operator	com.sap.system.python3Operator.v2	Yes	Stores state: depends on user script.
Remote Dataflow	Data Services Transform	com.sap.dataservices.transform.v2	No	None
SAP HANA	Initialize HANA Table	com.sap.hana.initTable.v2	Yes	None
SAP HANA	Read HANA Table	com.sap.hana.readTable.v2	With conditions	Stores state: row index.  For details about conditions, see the section State Management Support in <a href="#">Read HANA Table V2</a> .
SAP HANA	Run HANA SQL	com.sap.hana.runSQL.v2	With conditions	Stores state: last statement executed and the batch index.  For details about conditions, see the section State Management Support in <a href="#">Run HANA SQL V2</a> .
SAP HANA	Write HANA Table	com.sap.hana.writeTable.v2	Yes	None
Structured Data Operators	SAP Application Consumer	com.sap.application.consumer.v3	No	None
Structured Data Operators	SAP Application Producer	com.sap.application.producer.v2	With conditions	Operator supports running with snapshots only when you use it with Structured Data Consumer operators with source partitions. However, the operator doesn't support snapshots when you use it with other consumer operators, like ABAP Readers.

Category	Operator	Operator ID	Supports Snapshots	Conditions and Remarks
Structured Data Operators	SQL Consumer	com.sap.database.sql.consumer.v3	With conditions	<p>Stores state: partition of input data.</p> <p>Operator supports running with snapshots only when it reads data using partitions, and when you use it with Structured Data Producer operators.</p>
Structured Data Operators	Table Consumer	com.sap.database.table.consumer.v3	With conditions	<p>Stores state: partition of database table.</p> <p>Operator supports running with snapshots only when it reads data using partitions, and when you use it with Structured Data Producer operators.</p>
Structured Data Operators	Table Producer	com.sap.database.table.producer.v4	With conditions	<p>Operator supports running with snapshots only when you use it with Structured Data Consumer operators with source partitions. However, the operator doesn't support snapshots when you use it with other consumer operators, like ABAP Readers.</p>
Structured Data Operators	Data Transform	com.sap.datatransform.v2	No	None
Structured Data Operators	Structured File Consumer	com.sap.storage.consumer.v3	With conditions	<p>Stores state: partition of file groupings.</p> <p>Operator supports running with snapshots only when it reads data using partitions, and when you use it with Structured Data Producer operators.</p>



Category	Operator	Operator ID	Supports Snapshots	Conditions and Remarks
Structured Data Operators	Structured File Producer	com.sap.storage.producer.v3	With conditions	Operator supports running with snapshots only when you use it with Structured Data Consumer operators with source partitions. However, the operator doesn't support snapshots when you use it with other consumer operators, like ABAP Readers.
Utilities	Binary to Table	com.sap.table.decode.v2	Yes	None
Utilities	Table to Binary	comsaap.table.encode	Yes	None
Utilities	Graph Terminator	com.sap.util.graphTerminator.v2	Yes	None
Utilities	Terminal	com.sap.util.terminal.v2	With conditions	Data that entered the terminal can be lost when it wasn't processed downstream before a pause or restart.
Utilities	Wiretap	com.sap.util.wiretap.v2	Yes	None

## Limitations

SAP Data Intelligence doesn't support the following scenarios with the snapshot feature:

- Circular pipelines: You can't have a pipeline that has a circular connection, which is when the last operator outputs data to the first operator.
- Debug mode: You can't run graphs in debug mode with snapshots enabled.
- Group multiplicity: You can't have snapshot generation on graphs with groups that have multiplicity.
- Subgraphs: You can't create subgraphs for Gen2 pipelines.

## Related Information

[State Management \[page 48\]](#)

[Examples: Operator States \[page 50\]](#)

## 4.4 Delivery Guarantee for Generation 2 (Gen2) Graphs

When you enable automatic graph recovery and snapshots for a Gen2 graph (pipeline), your graphs can outlast system failures and system maintenance events.

SAP Data Intelligence doesn't persist snapshots continually. Therefore, some already-processed information can be lost, and the Modeler has to reprocess some of the same information after restoring a graph. Regardless, SAP Data Intelligence provides guarantees on the data consistency when there's a chance of losing data or duplicating data when the Modeler recovers the graph.

The following table lists the guarantees that Gen2 graphs with automatic graph recovery and snapshots provide. For descriptions of terms, see [Terminology \[page 66\]](#).

Source	Batch	Writer	Other Conditions	Guarantee
Replayable	No	Idempotent	Pipeline is deterministic	Exactly once
Replayable	Yes	Idempotent	Pipeline is deterministic Discard duplicate messages after source	Exactly once
Any	No	Transactional	None	Exactly once
Replayable	Yes	Transactional	None	Exactly once
Nonreplayable	No	None	Any	At most once
Nonreplayable	Yes	Any	Any	At most once
Replayable	Any	Any	Any	At least once

### Terminology

The terms in this section define the content of the values in the guarantee table.

#### Source

A source is an operator that generates data independently of any signal from an input port or ports. The following table describes the values in the Source column.

## Replayable

Data generated at time "T + x" always contains the data that was generated previously, at moment "T", and in the same order.

The following scenarios apply to replayable generators without batches:

- Reading immutable file.
- Query `SELECT . . . FROM SALESORDER WHERE ORDERDATE < '10-02-2010' ORDER BY ORDERDATE "`, where the comparison guarantees to always include previous information.
- Reading from an event queue starting from an offset.
- Query `"SELECT . . . FROM SALESORDER . . . ORDER BY ORDERDATE "`, where the table is append-only and records can be ordered.

---

## Nonreplayable

Data generated when replayable conditions aren't met.

### i Note

SAP Data Intelligence reads from a directory of files so that when there's a recovery, it doesn't have to start from the beginning.

---

## Any

Either replayable or nonreplayable.

---

## Batch

Data generated and broken into batches. You configure batches typically in the operator configuration.

---

## Yes

Data is broken into batches.

---

## No

Data isn't broken into batches.

---

## Any

Data is either broken into batches or not.

---

## Writer

The writer column contains information about operators that send data outside the graph. The following table describes values in the Writer column.

---

## Idempotent

Writing data multiple times has the same effect as writing data once. An idempotent writer can be guaranteed by the operator's operation, such as an UPSERT or a write-ahead log (WAL).

---

## Transactional

Writing data only when an epoch is finished, where an epoch is the interval between the snapshots. During an epoch to snapshot the graph, the operator doesn't write any data. Applicable only for the Python Operator ('`api.set_epoch_complete_callback`') and Kafka Consumer.

### i Note

Failure happens when the epoch completes, but before persisting data. The data recovery is from the completed epoch, so the writer requires a WAL to determine that the epoch hasn't saved the state yet.

---

## Can discard duplicates

The operator has to guarantee by an independent log that it discards repeated batches before they're propagated to the graph. Discarding repeated batches before propagating to the graph is guaranteed by several operators in the graph. An operator that is directly connected to the generators can have the independent log. Alternately, the source operator can have the independent log.

The **Discard duplicate messages after source** condition is necessary for operators whose internal states are affected by duplicate input messages. Otherwise, there's no risk of having the operators process the data again, and the idempotent writers is enough to prevent external side effects.

---

## Other Conditions

### Pipeline is deterministic

The graph always produces the same data to write and the same internal state for a given set of messages provided by the generators.

Recovering a graph with replayable sources has the same effect as the original run. The generators create the same data of the latest persisted snapshot, which in turn produces the same effects.

### Guarantees

The following table describes the values that appear in the Guarantees column.

Exactly once	Execution is equivalent to failure-free execution.
At most once	If there's a failure, data is lost but not duplicated.
At least once	If there's a failure, data isn't lost and is duplicated.

### i Note

The guarantees are for cases with a single generator or writer, such as a graph that contains the following construction: [▶ Read File](#) [▶ Python Operator](#) [▶ Write HANA](#). [▶](#) The listed guarantees are also applicable to any other configuration, excluding cycles.

## 4.5 Validate Graphs

Graph validation is an automatic or manual process that analyzes a graph for correct structure and components, such as operators, ports, groups, and configuration.

The results of a graph validation show that the graph analyses are successful. Graph validation checks the following components of a graph:

- Graph configuration
- Operator connections
- Tag configuration for groups defined in a graph
- Graph resource requests and limits

Validation doesn't build the graph with the corresponding resources and dependencies. Therefore, validation doesn't take as long as running the graph.

### Types of Graph Validation

There are two types of graph validation based on when the validation is performed:

- **Implicit validation:** Performed when you save or run a graph. Implicit validation is also known as automatic validation.
- **Explicit validation:** Performed when you start graph validation manually. Explicit validation is also known as manual validation.


Start an explicit validation by selecting the  (*Validate*) icon in the *Data Intelligence Modeler* editor toolbar.

### View Validation Results

To view graph validation results, open the *Validation* tab in the bottom pane of the *Data Intelligence Modeler*. It displays either a success message or a list of errors and warnings.

- Success: The graph doesn't have any warnings or errors.
- Warning: There's a problem with the graph. This problem won't cause the graph to fail, but it can result in other issues later.
- Error: There's an issue with the graph. This problem will cause the graph to fail and needs to be fixed.

#### Note

If validation finds any errors or warnings, you can fix some of the issues by selecting the  (help) icon located to the right of the error or warning message.

When the validation results in errors, you can't run the graph until you fix the errors and revalidate the graph.

## Related Information

[Graph Validation Warnings and Errors \[page 70\]](#)

### 4.5.1 Graph Validation Warnings and Errors

When a graph validation isn't successful, the Data Intelligence Modeler creates a list of warnings and errors, with a link to additional information about the warning or error.

The following table describes some of the warnings and errors from a graph validation.


Message	Component	Severity	Quick Fix
There are no operators in the graph.	isGraphEmpty	Warning	Add operators to the graph.
Invalid graph description. Special characters are not allowed.	Validate Graph Description	Error	Check <code>graph.json</code> for special characters in the description.
Group name is either missing or not unique.	Validate Single Group	Error	Check <code>graph.json</code> to add missing group names, and modify identical group names.
The <code>tagdatabase</code> of graph <code>&lt;graph_name&gt;</code> is empty.	Check tags	Error	Check your repository or <code>Tags.json</code> files for errors. Errors occur in the <code>&lt;graph_name&gt;</code> file.
Matching Dockerfile couldn't be found.	Check tags	Error	Check your repository <code>Tags.json</code> files, or other resources, for errors. Also check for missing or incorrect tags in the group configuration.
The graph <code>&lt;graph_name&gt;</code> has a single operator.	Check Operator Connections	Warning	This graph can execute with current configuration. However, consider adding more operators to the graph based on your requirement.
Incompatible ports are connected (source: port %s of operator %s, target: port %s of operator %s).	Check Connection Types	Error	Provide connection between compatible ports only.
Operator <code>&lt;operator_name&gt;</code> either deprecated or beta.	Validate Operator Versions	Warning	This graph can execute with current configuration. However, consider using non-deprecated or nonbeta operators.

Message	Component	Severity	Quick Fix
Operator <code>&lt;operator_name&gt;</code> does not exist in the registry.	Validate Port Names	Warning	Check that the operator <code>&lt;operator_name&gt;</code> is correct and saved successfully.
Some groups are missing resource definitions. If the resource request is missing and the limit is set, the request shall use the smaller value from either the limit set or the default request value.	Validate Graph Resources	Error  The message is an error when you validate the graph manually (explicit validation).	Without selecting any part of the graph in the canvas: <ol style="list-style-type: none"> <li>1. Open the <a href="#">Configuration</a> pane.</li> <li>2. Expand <a href="#">Resources</a>.</li> <li>3. Select the <a href="#">Edit</a> icon.</li> <li>4. Add or edit the resource limits.</li> </ol> <p>For more information about limits, see <a href="#">Maintain Resource Requirements for Graphs [page 87]</a>.</p>
Some groups are missing resource definitions. If the resource request is missing and the limit is set, the request shall use the smaller value from either the limit set or the default request value.	Validate Graph Resources	Warning  The message is a warning when you save or run a graph, and the Modeler validates the graph (implicit validation).	Without selecting any part of the graph in the canvas: <ol style="list-style-type: none"> <li>1. Open the <a href="#">Configuration</a> pane.</li> <li>2. Expand <a href="#">Resources</a>.</li> <li>3. Select the <a href="#">Edit</a> icon.</li> <li>4. Add or edit the resource limits.</li> </ol> <p>For more information about limits, see <a href="#">Maintain Resource Requirements for Graphs [page 87]</a>.</p>
Group restart policy is set to 'restart' for the following groups, while snapshot is enabled: <code>&lt;group_name&gt;</code> .	Validate Restart Policy	Error	Change the restart policy for the following groups: <code>&lt;group_name&gt;</code> .
Operator <code>&lt;operator_name&gt;</code> has a manual connection, which might contain sensitive information, such as ID and password.	Validate Manual Connection	Warning	Use a connection from Connection Manager rather than the manual connection.
Operator <code>&lt;operator_name&gt;</code> is using the "propagate to error port" error handling option but its error port is not connected or not present.	Validate Error Handling Ports	Error	Connect the error port of this operator or change its error handling type.

## 4.6 Running Graphs

After creating a graph, you can run the graph based on the configuration defined for the graph. The Modeler application runs the operators in the graph as individual processes.

### Procedure

1. Open the applicable graph in a graph editor.
2. Select the down arrow next to  (*Run*) in the editor toolbar.
3. Choose one of the run choices.
  - *Run*: Runs the graph immediately.
  - *Run As: Run As*: Runs the graph after you configure additional settings, such as automatic restart, and snapshots. The following table describes the *Run As* options.

#### **i** Note

The *Run* and *Run As* options trigger graph validation before running the graph. The Modeler displays the results in the *Validation* tab. For more information, see [Validate Graphs \[page 69\]](#).

4. Complete the options in the *Run As* dialog box as described in the following table.

Option	Description
<i>Run Graph As</i>	Required. Specifies a unique name for the run.
<i>Trace Level</i>	Sets the trace severity threshold, which determines the trace messages that are sent to the trace server. Choose one of the following options: <ul style="list-style-type: none"><li>• INFO</li><li>• DEBUG</li><li>• ERROR</li><li>• FATAL</li><li>• WARN</li></ul> For complete descriptions of the Trace Level options, see <a href="#">Trace Severity Levels [page 109]</a> .



Option	Description
<a href="#">Configuration Substitutions group</a>	<p>Contains all the configuration substitution parameters for the operators in the graph.</p> <p>If you defined configuration substitution parameters for the operators in the graph, all the configuration substitution parameters display in the <a href="#">Configuration Substitutions</a> group.</p> <p>Provide the required values for the following options as necessary:</p> <ul style="list-style-type: none"> <li>• HANA_DB</li> <li>• file_connection_id</li> <li>• file_path</li> <li>• select_statement</li> </ul> <div data-bbox="821 817 1402 1115" style="background-color: #f0f0f0; padding: 10px;"> <p><b>i Note</b></p> <p>If there are conditional configuration properties for an operator, regardless of the property that is visible in the configuration panel, the <i>Run As</i> dialog box still displays all the configuration substitution parameters. It's optional to provide values for the substitution parameters that are hidden in the Modeler.</p> </div>
<a href="#">Capture Snapshot</a>	<p>Saves a snapshot (intermediate state) of the graph for efficient recovery.</p> <div data-bbox="821 1211 1402 1361" style="background-color: #f0f0f0; padding: 10px;"> <p><b>i Note</b></p> <p>Snapshot configuration is available for graphs only with Generation 2 operators.</p> </div>
<a href="#">Every x second(s)</a>	<p>Specifies the time frequency (in seconds) for the <a href="#">Capture Snapshot</a> option. Frequency is from 1 second to 24 hours.</p>
<a href="#">Automatic Recovery</a>	<p>Restarts the graph automatically if there are failures or system upgrades.</p>
<a href="#">Retry Count</a>	<p>Sets the number of retries for recovery within a specific time period.</p>
<a href="#">Retry Interval</a>	<p>Sets the interval in seconds between retries for recovery.</p>
<a href="#">Retry Threshold Value</a>	<p>Sets the amount of time the system tries to recover the graph before it resets the automatic recovery count.</p>
<a href="#">Remember Configuration Parameters</a>	<p>Saves the <i>Run As</i> values to the configuration substitution parameters for when you run the graph again. If you've saved the configuration for the next graph execution, you can select the configuration for running the graph or update the existing run configuration.</p>

Option	Description
<i>Set as Default Run Configuration</i>	Saves the current settings as the default setting. Applicable when you've defined and saved multiple configurations.

5. Select *OK* to save your settings and to start running the graph.

#### [Automatic Graph Recovery \[page 74\]](#)

Configure any graph to recover from failure automatically, regardless of whether the graph uses Generation 1 or Generation 2 operators.

#### [Parameterize the Graph Run Process \[page 78\]](#)

Parameterize the graph run process using parameters that assume different values based on the values passed in each graph run.

#### [Debug Graphs \[page 82\]](#)

You can start the graph in debug mode to verify the input and output from each operator during execution and analyze or modify the data passing through a connection.

#### [Schedule Graph Executions \[page 84\]](#)

The SAP Data Intelligence Modeler provides capabilities to schedule graph executions.

## Related Information

[Schedule Graph Executions \[page 84\]](#)

### 4.6.1 Automatic Graph Recovery

Configure any graph to recover from failure automatically, regardless of whether the graph uses Generation 1 or Generation 2 operators.

When *Automatic Recovery* is enabled in the *Run As* dialog, the runtime system monitors the graph for failures and maintains a failure counter. As long as the failure counter is lower than the set options in the *Recovery Configuration* group, the runtime system restarts the graph with the same runtime configuration. If the graph fails within the set retry time, the error counter is reset.

#### **i** Note

Early graph failures are often caused for only temporary reasons, such as initial network or resource allocation timeouts. By resetting the failure counter, you prevent unintentional consumption of the automatic retries.

The following table describes the options in the *Recovery Configuration* group.

Option	Description
<i>Automatic Recovery</i>	Select to turn on automatic recovery.

Option	Description
<i>Retry for &lt;n&gt; run(s)</i>	Specifies the number of runs for which to retry the graph before the counter is reset.
<i>within the threshold value of &lt;n&gt; second(s)</i>	Specifies the time limit for retrying the graph before the counter is reset.

## Finally Failed Graphs

When the number of retries exceeds the settings in the *Recovery Configuration* group, the graph finally fails and the automatic restart feature stops. However, a finally failed graph can still be restarted manually at any time, even if the trials have been exceeded.

### i Note

The number for automatic restart trials can't be changed once the graph has started.

## Run Graph with Automatic Recovery

To enable the automatic recovery, check the option *Automatic Recovery* in the *Run As* dialog. To reset the failure counter, complete the *Recovery Configuration* group options.

The following table describes the details that appear in the *Overview* tab in the Modeler after the graph has entered the running state.

Detail	Description
Status	The current status of the graph.
Runtime Handle	The ID of the current running graph graph. Every run has a unique ID.
Restart ID	The ID of the automatic recovery configuration. All recovered graph runs share the same Restart ID.
Run Order	Indicates how often the graph has been recovered. Because the graph can also be restarted manually, the Run Order can exceed the Maximum Automatic Retries.
Current Failure Counter	Shows how often the graph has failed. You can reset this counter if recovery fails within the set retry threshold time.
Run Type	Shows whether the graph has been recovered automatically or restarted manually.
Maximum Automatic Retries	The number of retries.
Retry Threshold Time	The threshold time for resetting the current failure counter for early failures.

## ❖ Example

```
Retry Threshold Time: 30s
Maximum Automatic Retries: 3
Instance: 0
Graph running for: 40s
Increase current failure count: false
Current failure count: 0
Instance: 1
Graph running for: 20s
Increase current failure count: true
Current failure count: 1
Instance: 2
Graph running for: 31s
Increase current failure count: false
Current failure count: 0
Instance: 3
Graph running for: 20s
Increase current failure count: true
Current failure count: 1
Instance: 4
Graph running for: 28s
Increase current failure count: true
Current failure count: 2
Instance: 5
Graph running for: 15s
Increase current failure count: true
Current failure count: 3
Maximum retries reached
```

## The Status Tab

The [Status](#) tab lists all graph runs. The [Status](#) tab has two views, linear list and tree view that groups all runs belonging to the same automatic restart configuration (automatic restart group).

You can perform the following tasks in the [Status](#) tab:

- Download diagnostics information.
- Pause and restart the graph run.
- Archive the graph run.

## Automatic Graph Recovery and System Maintenance

When the system changes to maintenance mode, all graphs eligible for automatic restart enter the “stopped by pause” state. After the system goes back to production mode and the pipeline modeler is running for the respective users, the system restarts the graphs in the “stopped by pause” state automatically.

### i Note

The automatic restart is triggered only when the user who owns that graph logs on to the system and starts the pipeline modeler.

## Pause and Stop Graphs Manually

Use the options in the [Status](#) tab to manually pause, restart, and stop graphs. Pausing and restarting graphs is helpful when you develop and test recoverable graphs.

### i Note

Paused graphs keep allocating resources. If you don't plan to later restart a paused graph, SAP recommends stopping the graph to free resources.

You can restart a graph that has a status of “stopped by pause” at any time.

You can stop a graph manually, or the system stops the graph automatically when it completes its work. When a graph is stopped, it enters the state of “completed” and can't be restarted or recovered.

## Archiving Graphs

You can archive a graph when it enters into any of the following states:

- “completed”
- “stopped by pause”
- “dead”

### ! Restriction

To prevent potential data loss, a graph can't be archived when it's in the “stopped by pause” state and has a valid snapshot.

**Parent topic:** [Running Graphs \[page 72\]](#)

## Related Information

[Parameterize the Graph Run Process \[page 78\]](#)

[Debug Graphs \[page 82\]](#)

[Schedule Graph Executions \[page 84\]](#)

[Managing Cluster Hibernation and Wakeup](#)

## 4.6.2 Parameterize the Graph Run Process

Parameterize the graph run process using parameters that assume different values based on the values passed in each graph run.

When you configure operators and groups in a graph manually during design time, the resulting values are the same for all executions of the graph. But, when you parameterize the graph run process, you can change the results for each run.

Use the following two modes of parameterization through the API:

- Substitution Parameters
- Graph Parameters

### i Note

You can't mix the two types of parameterization in the same graph. A graph, however, can be executed using substitution parameters while another instance of the same graph can use graph parameters.

SAP Data Intelligence restricts both methods to operator configurations and group multiplicity. Therefore, you can change a group's CPU or memory limits only during design time.

## Substitution Parameters

When you use substitution parameters, set values for the parameters at the following times:

- At runtime, when you run the graph.
- At design time, by referencing the value of another configuration from the same operator.  
When you reference the value of another configuration from the same operator, the values must always be strings. The values must be strings even for group multiplicity. The system converts the strings to integers internally.

Define a substitution parameter by providing the value to the configuration in the following format: `${parameter_name}`, where `parameter_name` is the name of the substitution parameter. SAP Data Intelligence handles the substitution parameter value as follows:

- If the name of the substitution parameter (`parameter_name`) matches the name of an operation configuration, then the Modeler uses the configuration value as the substitution parameter value.
- If the name of the substitution parameter (`parameter_name`) doesn't match the name of any configuration parameter of the operator, then you provide the value at runtime.

### 🔗 Example

The operator configuration parameter named "Path" is defined with the value `URL: // ${name}`. In this example, `{name}` is the substitution parameter. You define the value for "Path" by providing a value to `{name}` at runtime.

## ❖ Example

The following snippet shows an excerpt of a graph JSON with a substitution parameter in an operator configuration:

```
{
  "properties": {},
  "description": "",
  "processes": {
    "httpclient1": {
      "component": "com.sap.http.client2",
      "metadata": {
        "label": "HTTP Client",
        "config": {
          "getConnection": {
            "connectionID": "${TO_BE_SUBSTITUTED_CONNECTIONID}"
          },
          "retryPeriodInMS": 500
        }
      }
    },
    ...
  }
}
```

The following code snippet requests the body of an HTTP request that runs a graph named "testSubs", and sets a value for the substitution parameter as "TO\_BE\_SUBSTITUTED\_CONNECTIONID":

```
{
  "src": "test_substitution",
  "name": "testSubs",
  "debug": false,
  "async": true,
  "configurationSubstitutions": {
    "TO_BE_SUBSTITUTED_CONNECTIONID": "TEST_HTTP_CONNECTION"
  }
}
```

## Graph Parameters

Graph parameters support all JSON types. To support graph parameters, SAP Data Intelligence uses the following attributes:

- **parameters:** Use as part of the graph description. Lists all the schemas that appear between the brackets ( $\{\}$ ) and includes the JSON type. Default values are optional. This field contains the schemas of each parameter.
- **parameterMapping:** Use as part of the graph description. Maps an operator configuration property to a value that contains a graph parameter.
- An entry to the request body that JSON uses to start a graph. Instead of the **configurationSubstitutions**, you can use **parameters** that support any type.

## ❖ Example

The following code snippet uses integer and boolean:

```
{
  "src": "test_parameters",
```

```

    "name": "testParams",
    "parameters": {
      "eggs": 100,
      "dis": true
    }
  }
}

```

Where:

- eggs = integer
- dis = boolean

The following example shows the extent of the graph parameter feature and how you can use the graph parameters for operator configurations.

### ❖ Example

```

{
  ...
  "parameters": { // Schemas of the substitution parameters
    "foobar": {
      "type": "number",
      "default": 100 // default value if no value is passed
    },
    "foobar2": {
      "type": "object"
    },
    "foobar3": {
      "type": "number"
    }
  }
  "processes": {
    "pythonOperator1": {
      "configuration": {
        "other": "value",
        "baz": "3",
        "test2": "3",
        "bazarray": [
          {
            "baz": 2
          }
        ]
      }
    }
  }
  "parameterMapping": { // How the parameters will be introduced into the
    configurations, we should always refer to the parameters with ${}
    "processes.pythonOperator1.configuration.baz": "where count < $
    {foobar}", // overwrite existing baz with string "where count < 3".
    "processes.pythonOperator1.configuration.test": "${foobar2}", // create
    configuration that expects any JSON object, not requiring further schema
    "processes.pythonOperator1.configuration.test2": "${foobar3}", //
    overwrites 3 with the parameters integer value.
    "processes.pythonOperator1.configuration.bazrray.0.baz": "${foobar}", //
    overwrites 2 with the parameter value, keep in my the last structure (baz in
    this case) must be an object.
    "processes.pythonOperator1.configuration.test": "foobar2 dummy", // ERROR
    because the value must contain ${}, and the stringification depends only on
    the type in the schema.
  }
  ...
}

```



## Type Validation

SAP Data Intelligence performs type validation on the parameters based on their type definition. An invalid type results in the graph run request to fail with an error code of 400.

The following sample code shows how to parameterize group multiplicities and how SAP Data Intelligence works with the operator configurations.

### ❖ Example

Only integer and number types are allowed. In `parameterMapping`, the system references each group by its index in the graph groups list.

```
{
  "processes": {
    "formatconverter1": {
      "component": "com.sap.util.formatConverter1",
      "metadata": {
        "config": {
        }
      }
    }
  },
  "groups": [
    {
      "name": "group1",
      "nodes": [
        "formatconverter1"
      ],
      "metadata": {
        "description": "Group"
      }
    }
  ],
  "parameters": {
    "foo": {
      "type": "string"
    },
    "bar": {
      "type": "number"
    }
  },
  "parameterMapping": {
    "processes.formatconverter1.configuration.cfg1": "${foo}",
    "groups.0.multiplicity": "${bar}"
  }
}
```

## Limitations

The request to run a graph fails with error "400 bad request" under the following conditions:

- Parameters are too large. The default threshold is 5 MB.
- Graph uses both substitution parameters and graph parameters.
- Parameters aren't valid according to their schema.

Parent topic: [Running Graphs \[page 72\]](#)

## Related Information

[Automatic Graph Recovery \[page 74\]](#)

[Debug Graphs \[page 82\]](#)

[Schedule Graph Executions \[page 84\]](#)

### 4.6.3 Debug Graphs

You can start the graph in debug mode to verify the input and output from each operator during execution and analyze or modify the data passing through a connection.


#### Context

Debug mode enables debugging capabilities on the execution of a pipeline and allows you to investigate the whole pipeline step by step to make it bug free. You can inspect data at a specific stage of a pipeline execution, detect data quality issues, perform a root cause analysis of the detected problems, and try corrective actions.


##### i Note

The debug feature is not supported for generation 1 graphs with structured data operators.

#### Procedure

1. In the graph editor, open the graph that you want to debug.
2. In the editor toolbar, choose the  (Debug) menu option. The graph executes in debug mode.
3. To view the debugging status of the graph, open the *Status* tab in the bottom panel and select the graph to open it in the monitoring/runtime view.

In the runtime view, the *Debug Panel* opens and lists all the breakpoints in the graph. You can select a specific operator in the graph to see the breakpoints associated with that operator.

4. To filter the results in the debug panel, click  and choose from one of the following:
  - *Breakpoint*: Displays only breakpoints.
  - *Hit*: Displays breakpoints and streaming points that are hit.
  - *Streaming*: Displays only streaming points.

**Task overview:** [Running Graphs \[page 72\]](#)

## Related Information

[Add Breakpoints to a Graph \[page 83\]](#)

[Automatic Graph Recovery \[page 74\]](#)

[Parameterize the Graph Run Process \[page 78\]](#)

[Schedule Graph Executions \[page 84\]](#)

### 4.6.3.1 Add Breakpoints to a Graph

Breakpoints and streaming points allow you to inspect the data transformation occurring throughout the pipeline when you run in *Debug* mode.

#### Context

When running in debug mode, streaming points will be set by default in all connections. You will be able to convert streaming points to breakpoints, which stops when data hits them and shows visually. You can modify or resume execution of that connection, which in turn would stop again when data hits the next connection breakpoint.

You can set a breakpoint during design time or run time. The design time breakpoints will not be saved with graph information, which means that design time breakpoints gets reset once you close and open the graph again.

#### Procedure

1. To set a breakpoint,
  - hover over the connection between operators in a graph, you will get an option to click and set the breakpoint.  
or
  - right-click over the connection between operators in a graph and click *Add Breakpoint*.
2. Run the graph in debug mode. For more information, see [Debug Graphs \[page 82\]](#).
3. To add/remove breakpoints, right-click a breakpoint in the debug panel or in the graph and select the appropriate option in the context menu.
4. For a breakpoint, you can:
  - a. inspect and modify data upon hit. Right-click and select *Inspect Data* in the context menu.
  - b. resume inspection. Right-click and select *Resume* in the context menu.
5. For a streaming point, you can:
  - a. open its corresponding debugger page. Right-click and select *Open Streaming UI*.

## 4.6.4 Schedule Graph Executions

The SAP Data Intelligence Modeler provides capabilities to schedule graph executions.


### Context

Scheduling graph execution is useful when you have to schedule graph executions with recurring conditions.

#### i Note

We recommend scheduling executions only for those graphs that, when executed, run for a limited period and finish with the status of *completed* or *dead* (for example, Data Workflows operators).

### Procedure

1. Open the graph that you want to schedule to execute in a graph editor.
2. In the editor toolbar, choose  (Run).
3. Choose the *Schedule* menu option.
4. Schedule a graph execution.

In the *Schedule Graph* dialog box, define the schedule.

- a. In the *Schedule Description* text field, provide a name for the schedule.
- b. Select a schedule property.

The application supports a form-based approach or a cron expression to define the schedule.

Schedule Property


Property	Description
Form	<p>Provides a form-based UI to define a condition that specifies the frequency (or the number of occurrences) for executing the graph.</p> <p>Select a time zone and define the frequency pattern in the schedule properties section to schedule the executions..</p> <p>The system uses the UTC equivalent of the frequency pattern that you specify to schedule the executions.</p> <p>For example, you can define a recurring condition that executes the graph every day at 9:00 AM.</p>

Property	Description
Expression	<p>Defines a cron expression that provides the condition for scheduling a recurring graph execution. The cron expression is a string of five fields separated by white spaces.</p> <p>The syntax for the cron expression is <b>Minute Hour DayOfMonth Month DayOfWeek</b>.</p>

5. Choose *Schedule*.

## Next Steps

After creating a schedule, you can monitor the schedule within the Modeler or use the SAP Data Intelligence Monitoring application to monitor and manage all schedules.

In the bottom pane of the Modeler application, under the *Schedule* tab, you can monitor and manage the schedule. In this tab, you can view the description of the schedule and the graph execution state. If the schedule of the graph is in *active* state, the graph run is triggered. Otherwise, it is suspended. To stop a schedule, select the required graph and choose  (Stop Process).

### i Note

A special scheduling graph running in the SAP Data Intelligence Modeler performs the scheduled operation.

**Task overview:** [Running Graphs \[page 72\]](#)

## Related Information

[Cron Expression Format \[page 86\]](#)

[Automatic Graph Recovery \[page 74\]](#)

[Parameterize the Graph Run Process \[page 78\]](#)

[Debug Graphs \[page 82\]](#)

[Working with the Data Workflow Operators \[page 133\]](#)

[Monitoring SAP Data Intelligence \[page 204\]](#)

## 4.6.4.1 Cron Expression Format

Use cron expressions to define the recurrence condition that the tool uses to schedule the graph execution.

### Cron Format

A cron expression to execute graphs in the Modeler is a string comprised of five fields. The table lists the order of fields (from left to right) in a cron expression and the permitted values for each field.

Cron Field (from left to right)	Description
Minute	Representation for a minute. Permitted: 0 to 59.
Hour	Representation for an hour. Permitted: 0 to 23.
	<b>i Note</b> The system uses the UTC (offset 0) equivalent of the condition specified with the cron expression.
DayOfMonth	Day of the month. Permitted: 1 to 31.
Month	3-letter representation for month. Permitted: jan-dec. Alternatively, numbers 1-12 can be used, where 1 = jan, and so on.
DayOfWeek	3-letter representation for day of week. Permitted: sun, mon, tue, wed, thu, fri, sat. Alternatively, use numbers 0-6, where 0 = sun, and so on.

### Cron Syntax

The table provides the syntax that the application supports to define a cron expression.

Expression	Where Used	Value
.	Anywhere	Any value
*/a	Anywhere	Any a-th value
a-b	Anywhere	Values in range a to b
a-b/c	Anywhere	Every c-th value between a and b
a,b,c	Anywhere	a or b or c

## Cron Expression: Examples

The table lists some examples of cron expressions that you can use.

Expression	Description
* * * * *	Runs the schedule every minute.
* / 5 13 * 12 0	Runs the schedule every fifth minute between 1:00 PM (UTC) and 1:59 PM (UTC) on Sundays in December.
0 9 1-7 * sat,sun	Runs the schedule at 09:00 on every day of the month from 1 through 7, and on Saturday and Sunday.

## 4.7 Maintain Resource Requirements for Graphs



Specify compute resource requirements, such as CPU and memory limits, for graph groups in SAP Data Intelligence Modeler.

For each resource type, specify CPU and memory limits in the *Requests* and *Limits* properties of the *Configuration* pane.

- The *Requests* properties specify the initial resource quantities for memory and CPU that the Modeler requires to start the graph group execution. If the resource availability isn't enough, then the graph execution fails to start. Default settings are as follows:
  - Memory: 256 Mebibytes (Mi)
  - CPU: 0.3 CPU
- The *Limits* properties specify the limits for resource usage for memory and cpu. If the graph group execution violates the limit set for memory, the Modeler terminates the graph execution. If the graph group execution uses excessive CPU, the set CPU limit helps control the CPU consumption by the graph group. Default settings are as follows:
  - Memory: 3 Gibibytes (Gi)
  - CPU: 3 CPU

If you don't specify memory and CPU limits in the *Resources* section of the *Configuration* pane, the Modeler uses the default limits.

When you set resources for *Requests* and *Limits* in the *Configuration* pane, you also select the resource units. SAP uses the same notations as Kubernetes to specify the resource quantity. For more information on memory and CPU resource units, see the following Kubernetes documentation:

- [Memory resource units](#) 
- [CPU resource units](#) 

For more information about CPU and memory size considerations, see [Sizing for Data Pipelines](#) in the *Sizing Guide for Cloud*.

## Errors and Warnings

When a graph or graph group exceeds the limits you set in [Resources](#), the Modeler issues an error and the graph doesn't run.

If you don't set limits in [Resources](#), the Modeler uses the default limits and issues a warning or an error. The message instructs you to adjust the CPU and memory limits for the graph. The following lists when the Modeler issues a warning or an error:

- **Warning:** When you save or run a graph, and the Modeler validates the graph (implicit validation), the Modeler issues a warning.
- **Error:** When you validate the graph manually (explicit validation), the Modeler issues an error.

Adjust the limits either in the Modeler or in the `graph.json` file.

[Resource Requirements for a Graph in JSON \[page 88\]](#)

View and overwrite resource requirements in the JSON file of a graph.

[Configure Resources for a Graph \[page 90\]](#)

You can add and specify resource configuration for a graph or the groups within the graph.

## Related Information

### 4.7.1 Resource Requirements for a Graph in JSON

View and overwrite resource requirements in the JSON file of a graph.

Resource requirements are specified in the `graph.json` file. The `groupResources` property provides the default resource requirements for all groups. For a group (except `default`), you can overwrite requirements in `resources` property in the group definition.

#### ❁ Example

Here is an example of a modified data generator demo graph (unnecessary details are omitted):

#### ≡ Sample Code

```
{
  "groupResources": {
    "requests": {
      "cpu": "0.5",
      "memory": "4M"
    },
    "limits": {
      "cpu": "1.5",
      "memory": "16M"
    }
  },
  "description": "Data Generator",
  "processes": {
    "datagenerator1": {
```



```

        "component": "com.sap.util.dataGenerator",
        ...
    },
    "lgnu": {
        "component": "com.sap.util.terminal",
        ...
    }
},
"groups": [
    {
        "name": "group1",
        "nodes": [
            "datagenerator1"
        ],
        "resources": {
            "requests": {
                "cpu": "1",
            },
        },
        "limits": {
            "cpu": "2",
            "memory": "32M"
        }
    }
],
"connections": [
    {
        "src": {
            "port": "output",
            "process": "datagenerator1"
        },
        "tgt": {
            "port": "in1",
            "process": "lgnu"
        }
    }
]
}

```

In this example graph, there are two groups: default (contains terminal operator) and group1 (contains dataGenerator operator). Resource requirements from groupResources are applied to the default group:

#### ≡, Sample Code

```

{
    "groupResources": {
        "requests": {
            "cpu": "0.5",
            "memory": "4M"
        },
        "limits": {
            "cpu": "1.5",
            "memory": "16M"
        }
    },
    ...
}

```

First the resource requirements from the `groupResources` property is applied to `group1`. Then all resource requirements are overwritten for `group1` but not `memory request` (because it's not specified for `group1`):

#### Sample Code

```
{
  ...
  "groups": [
    {
      "name": "group1",
      "resources": {
        "requests": {
          "cpu": "1"
        },
        "limits": {
          "cpu": "2",
          "memory": "32M"
        }
      }
    }
  ],
  ...
}
```

Parent topic: [Maintain Resource Requirements for Graphs \[page 87\]](#)



## Related Information

[Configure Resources for a Graph \[page 90\]](#)

## 4.7.2 Configure Resources for a Graph

You can add and specify resource configuration for a graph or the groups within the graph.

### Procedure

1. Open the graph for which you want to configure resources, in the graph editor.
2. Click anywhere on the graph or select a group within the graph and choose  (Show Configuration).
3. In the *Configuration* panel, go to the *Resources* section and click  (Edit).
4. In the *Resource Configuration* dialog, add or modify the resource requests and limits as required.
5. Specify the *Resource Type*, *Value*, and *Unit* for the *Requests* and *Limits*, and click *OK*.

### i Note

If you don't provide explicit configuration for a selected group, the graph level configuration is applicable by default.

**Task overview:** [Maintain Resource Requirements for Graphs \[page 87\]](#)

## Related Information

[Resource Requirements for a Graph in JSON \[page 88\]](#)



## 4.8 Create Data Types in Graph

You can create graph-level data types and use them in the graph along with the automatically generated data types.


### Context

To create additional data types to the existing automatically generated data types, perform the following steps in the Modeler:

### Procedure

1. Open the applicable graph in the graph editor.
2. Select  (*Show Configuration*) in the editor toolbar.
3. Expand the *Data Types* section and select  (*Create*).
4. Enter a unique ID for your data type in the *ID* text box.
5. Choose *Structure*, *Table*, or *Scalar* for *Type*.
6. Select *OK*.

The new data type appears in the list of data types in the *Data Types* section.

7. **Optional:** Select  (*Edit*) next to the data type name to set additional properties.

The available properties to set are based on the data type.

## Related Information

[Use Data Types in Graph \[page 92\]](#)

[Exporting and Importing Graphs with Data Types \[page 93\]](#)

### 4.8.1 Use Data Types in Graph

#### Procedure

1. In the navigation pane, choose the *Graphs* tab.
2. In the navigation pane bar, choose **+** (Create Graph).  
The application opens an empty graph editor in the same window, where you can define your graph.
3. In the navigation pane, choose the *Operators* tab.
4. Choose source and target operators that supports addition of ports from the *Structured Data Operators* category.
5. In the context menu of the source, select the *Add Port* option.
6. Do the following in *Add Port* dialog:
  - a. Enter the output port name
  - b. Select data type as Structure, Table, or Scalar
  - c. Select a data type name from the value help, and click *OK*.
7. Drag from this port to the target operator.

If there's a port of compatible data type, the connection is created to this port. Else, a new input port is created on the target operator and is connected. If the port doesn't allow additional ports, the link to the operator is not created.

#### ! Restriction

Only structured data transform operator supports data types.

## 4.8.2 Exporting and Importing Graphs with Data Types

When your graph has added data types, and you want to reuse the graph in another system, you must export and import the graph from the repository instead of copying and pasting the JSON content.

### Prerequisites

Before you perform the following steps, ensure that you complete the following tasks:

- Create a graph: [Creating Graphs \[page 57\]](#).
- Create a new data type: [Create Data Types in Graph \[page 91\]](#).
- Assigned the new data type to a port in the graph: [Use Data Types in Graph \[page 92\]](#).

### Context

SAP Data Intelligence adds additional subfolders to a graph structure to store a data type definition. However, the additional data type definition isn't included in the JSON content. Therefore, if you reuse a graph from one system, such as a test system, to another system, such as a production system, it's important to use the options [Export as a Solution](#) and [Import as a Solution](#).

Perform the following steps in SAP Data Intelligence Modeler:

### Procedure

1. Open the [Repository](#) tab in the navigation pane at left.
2. Select the graph to export in the **Graphs** node.
3. Choose [Export as solution](#) from the [Export selected files or folders](#) list at the top of the navigation pane.

The [Export Solution](#) dialog opens.

4. Complete the export by performing the following substeps:
  - a. Enter a name for the exported graph for the new environment.

```
⌵, Sample Code  
  
"name" : "<graph_name>"
```

- b. Enter a version for the graph.

```
⌵, Sample Code  
  
"version" : "<graph_version>"
```

- c. **Optional:** Complete the remaining attributes as necessary:

#### ≡, Sample Code

```
...format": "2",  
"description": "",  
"dependencies": [ ]  
}
```

- d. Select *Export Solution*.
5. To import the graph, open the target SAP Data Intelligence environment and perform the following steps in the Modeler:
- Open the *Repository* tab.
  - Choose *Import Solution* from the *Import File* list.
  - Select the solution and choose *Open*.

The imported graph appears listed in the **Graphs** node of the *Repository* tab.

## 4.9 Groups, Tags, and Dockerfiles

Groups, tags, and Dockerfiles are essential parts of the SAP Data Intelligence environment for running graphs (pipelines) more efficiently. Therefore, you must understand how they work together.

### Groups

A group is an aggregation of operators in a graph that have similar technical requirements or that run in a common Docker image. When you run a graph with groups, each group runs in a different Docker container. Each Docker container has the possibility of having a different Docker image. The Modeler selects a group's Docker image automatically based on the tags associated with the image. The operators in a group run in the same node. You can configure each group with a different restart policy, tags, or multiplicity.

#### ❁ Example

Assign different restart policies for each group in a graph:

- Group 1 has a restart policy where the container redeploys when the group fails.
- Group 2 has a restart policy where the graph terminates when the group fails.

The most common reason for using groups is to distribute work among many compute nodes. Distribute work by partitioning the graph into many groups and (or) adding multiplicity larger than 1 for a group. Distributing work among many compute nodes can result in better graph throughput and cluster utilization.

#### i Note

Multiplicity determines the number of runs for the group at runtime.

If there's no Dockerfile that satisfies the requirements of a graph, you can partition a graph into groups in such a way that, for each group, there exists at least one Dockerfile that satisfies the graph's requirements.

A graph with no group defined explicitly has only one group, the default group. A default group contains all operators from the graph that haven't been assigned to an explicit group. You can partition the graph by assigning a subset of operators to an explicit group.

### ❖ Example

You have a non-partitioned graph with the following topology:  $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$ . All operators in this graph run in the default group.

You decide to create 2 explicit groups:

- Group 1 contains A and B
- Group 2 contains E

When you include the default group, your graph now has 3 groups:

- Group 1:  $(A \rightarrow B)$
- Group 2:  $(E)$
- Default group:  $C \rightarrow D$

The resulting topology is as follows:  $(A \rightarrow B) \rightarrow C \rightarrow D \rightarrow (E)$ .

## Tags and Dockerfiles

When running a graph, the Modeler selects a Docker image for each group based on the group's tags. View the available tags in the [Tags](#) list in the group properties. If multiple Dockerfiles meet the tag requirements, the Modeler chooses the Dockerfile with the fewest tags. If the Docker image isn't already cached, the Modeler builds the selected Dockerfile during the graph run.

Each tag represents a software component, such as a package or library, that is required at runtime for a group. The Modeler identifies the software component by a pair of values in the following format:

```
{  
  "<resource_id>" : "<resource_version>"  
}
```

### ❖ Example

The following are examples of value pairs for software component:

```
{  
  "python" : "3.9"  
}
```

```
{  
  "tornado" : "6.1.0"  
}
```

```
{
```

```
"opencv" : ""
}
```

### Note

An empty `<resource_version>`, such as "", means that the Modeler can use any version of the component.

The final list of tags for a group is a combination of the following:

- individual tags specified by each operator
- tags from the group configuration

When determining the final list of tags, the Modeler searches for a Dockerfile in your repository directory that meets all requirements. If two operators in the same group require the same resource but different versions, and the versions are compatible, the Modeler uses the more specific version. If the versions aren't compatible, the Modeler issues an error.

### Example

If one operator requires version "1.1" of component "foo" and another operator requires version "1.1.2", the Modeler selects version "1.1.2" because it's more specific. However, if one operator requires version "1.1" and the other requires "2.1.1", the Modeler issues an error because versions "1.1" and "2.1.1" don't share a common prefix and aren't compatible.

When the Modeler searches for Dockerfiles that meet a group's tag requirements, it's essential that it determines whether a specific group tag is fulfilled by a Dockerfile tag. A group is considered satisfied by a Dockerfile tag only when the following conditions are met:

- The resource IDs in both tags are identical.
- The resource versions share a common prefix.
- The resource version of the Dockerfile tag is more specific than the resource version of the group tag.

### Example

The group tag { "foo": "1.1" } is satisfied by the Dockerfile tags { "foo": "1.1" } or { "foo": "1.1.2" }, but not by { "foo": "1" }, { "foo": "" }, or { "bar": "1.1" }.

If multiple Dockerfiles meet a group's tag requirements, the Modeler doesn't use the specific resource versions defined in each Dockerfile to break the tie; it selects the Dockerfile arbitrarily.

To determine the selected Dockerfile for a group, download the graph diagnostics archive and examine the `image` field in the topic [execution.json File \[page 112\]](#). If no Dockerfile meets a group's needs, the Modeler issues an error. To fix the error, you can either split the group into smaller groups that match existing Dockerfiles or create a new Dockerfile that meets all the group's requirements. For information about creating Dockerfiles, see the topic [Creating Dockerfiles \[page 291\]](#).



## Default and Deprecated Tags

### Default Tag

The Modeler gives higher priority over other Dockerfiles to the Dockerfiles that have the `{ "default" : }` tag.

#### ❖ Example

A Docker image named `com.sap.sles.base` has the `{ "default" : }` tag and meets the tag requirements. The Modeler chooses the `com.sap.sles.base` Docker image even when other Dockerfiles meet the requirements and have fewer matching tags, because the other Dockerfiles don't have the `{ "default" : }` tag.

If two Dockerfiles have the `{ "default" : }` tag and meet the tag requirements, the Modeler selects the Dockerfile with the fewer tags.

### Deprecated Tag

Dockerfiles with the `{ "deprecated" : }` tag have a lower priority than other tags.

#### ❖ Example

If the Docker image `com.sap.opensuse.going.zypper` has the `{ "deprecated" : }` tag and meets the tag requirements, the Modeler doesn't choose it over a non-deprecated image, even when it has fewer matching tags.

If the Modeler has to choose between two Dockerfiles, each with the `{ "deprecated" : }` tag, it chooses the Dockerfile with the fewer matching tags.

## Operator Tags

View or edit an operator's tags in the operator editor. Open the editor by right-clicking the operator and choosing [Edit](#).

The operators that run on subengines can have implicit tags. Implicit tags don't appear on the operator editor screen; they're included in the requirements of a group using the operator. Check each subengine's documentation for more information about its implicit tags. The following table lists the implicit tags for all active subengines.

Subengine	Implicit Tags
ABAP	vrep (runs using any image)
Node.js	node
Python 3.9	<ul style="list-style-type: none"><li>'tornado':'6.1.0'</li><li>'sles':""</li><li>'python':'3.9'</li></ul>
C++	Deprecated

## ⚠ Caution

Be careful when you add or change tags for existing operators or groups. Tags affect the selection of the Docker image. Modifying tags can alter the Dockerfile that the Modeler chooses during graph execution, which can cause unexpected results or errors. SAP recommends that you thoroughly review the tag changes and the impact on the Docker image selection process before you complete any modifications.

The following example shows how tags affect the selection of Docker images and the Dockerfile that the Modeler chooses during graph execution.

## ❖ Example

Your repository has the following Dockerfiles and associated tags:

- **com.sap.d1:** { "python36": "", "pandas": "1.2.3", "numpy36": "", "tornado": "1.1.1", "pyarrow": "" }
- **com.sap.d2:** { "python36": "", "corge": "2.2.2" }
- **com.sap.d3:** { "node": "" }

The tags associated with a group in a graph are the union of each operator's tags and the tags specified in the group configuration.

A graph with the topology of (A→B→C)→D→E, has the following groups:

- **Explicit group:** (A, B, C)
- **Default group:** (D, E)

The following table lists the group configurations and operators with their corresponding associated tags.

Group Configuration or Operator	Associated Tags
Explicit group configuration	{ "python36": "", "pandas": "" }
Default group configuration	{ }
Operator A	{ "numpy36": "" }
Operator B	{ "pandas": "1.2", "tornado": "1.1.1" }
Operator C	{ }
Operator D	{ }
Operator E	{ "python36": "" }

The following are the results when you associate the union of the tags with each group results:

- **Explicit group (A, B, C):** { "python36": "", "pandas": "1.2", "numpy36": "", "tornado": "1.1.1" }
- **Default group (D, E):** { "python36": "" }

The Modeler determines a Dockerfile for each group using the following information:

- The aggregation of the tags in the **Explicit group** are satisfied only by the Dockerfile **com.sap.d1**.
  - The Dockerfile **com.sap.d1** has one more tag { "pyarrow": "" } than **Group 1** requires.
- The "1.2.3" version for "pandas" in **com.sap.d1** satisfies the **Explicit group**.
  - The **Explicit group** requires version "1.2", but "1.2.3" is more specific.

- The **Default group** has two Dockerfiles that satisfy the aggregated tags: **com.sap.d1** and **com.sap.d2**.

The Modeler chooses the Dockerfiles with fewer tags:

- `com.sap.d1` for Explicit group
- `com.sap.d2` for Default group

## 4.10 Execution Model

To avoid problems, such as back pressure and deadlocks, SAP Data Intelligence Modeler executes graphs following an execution model.

### Back Pressure

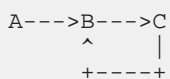
The Modeler executes operators in all graphs concurrently. Operators communicate with each other by sending data to their outports and receiving data from their inports. When “Back-pressure” between two operators occurs, the Modeler blocks the operator that is trying to send data until the receiving operator reads it. This blocking prevents data from accumulating on a region of the pipeline that produces data faster than other parts can consume.

### Deadlocks

Back pressure can cause some graphs with cycles to deadlock. A deadlock happens in graphs when there are at least as many messages in the cycle as there are nodes in the cycle.

#### ❖ Example

The following graph has a cycle between operators A, B, and C:



A smooth execution of a graph with no deadlocks happens when operator A generates just one message and sends it to operator B or operator C. Each time operator B or C receives a message through their inport, they process the message and then send a new message through their outport. This process implies that there's always a single message circulating around the cycle, and no deadlock occurs.

A deadlock can occur when, instead of operator A feeding just one message into the graph, it produces two messages. When operator A produces two messages, the graph deadlocks because of the following activity:

- Operator B is blocked when it tries to send a message to operator C.
- Operator C doesn't read from its inport because it's trying to send the message to operator B.

- Operator B doesn't read from its inport because it's trying to send the message to operator C.

The graph can also deadlock under the following circumstances:

- Operator A generates only one message.
- Operator B outputs two messages for each one message that it receives from Operator A at its input port.

## Sending Data and Mutable Objects

When an operator sends data to another operator, the Modeler doesn't send a copy of the data. Instead, the Modeler sends only a reference to the data. Sending a reference to the data decreases the communication cost.

However, when the graph has mutable objects, it's more secure to send copies of the objects. A mutable object, such as a message data type, is one in which you can modify or edit a value. When you program a script operator and change an object received on input, the object can reflect in other parts of the graph. Therefore, it's safer to make a copy of a mutable object before you change it instead of sending a reference to the data.

## 4.11 Monitoring Graphs

After creating and running graphs, monitor graphs and view statistics.

The following table describes the options for monitoring the graph status in the SAP Data Intelligence Modeler.

Action	Description
Monitor Status of Graph Runs	<p>After you create and run a graph, monitor the status of the graph run in the Modeler.</p> <p>Use the standalone monitoring application that SAP Data Intelligence provides to monitor the status of all graphs executed in the Modeler.</p>
Trace Messages	<p>Trace messages monitor both the system and running graphs to isolate problems or errors that may occur. Trace messages provide an initial analysis of your running graphs so you can troubleshoot potential problems or errors.</p>
Use SAP Data Intelligence Monitoring	<p>SAP Data Intelligence provides the Monitoring application to monitor the status of graphs run in the SAP Data Intelligence Modeler.</p>
Access the SAP Data Intelligence Monitoring Query API	<p>Access the SAP Data Intelligence Monitoring Query API to retrieve application performance metrics for your tenant. For more information, see <a href="#">Accessing the SAP Data Intelligence Monitoring Query API</a>.</p>

Action	Description
SAP Data Intelligence Diagnostics	SAP Data Intelligence Diagnostics deploys one of the most widely used stacks of open-source monitoring and diagnostic tools for Kubernetes. For health and performance monitoring, SAP Data Intelligence Diagnostics provides cluster administrators access to cluster-wide system and application metrics.

### i Note

Developer member users with the sap.dh.monitoring policy can monitor the status of graph runs for all tenant users.

### i Note

By default, the Modeler also performs logging. The log messages are intended for a broader audience with different skills. If you want to view the log messages, start the Modeler, and in the bottom pane, select the [Logs](#) tab.

## Related Information

[Monitor the Graph Execution Status \[page 101\]](#)

[Activate Trace Messages \[page 108\]](#)

[Downloading Diagnostic Information for Graphs \[page 109\]](#)


## 4.11.1 Monitor the Graph Execution Status








After creating and executing a graph, you can monitor the status of the graph execution in the SAP Data Intelligence application.



### Procedure

1. Start the SAP Data Intelligence Modeler.
2. Select the [Status](#) tab in the bottom pane.

The [Status](#) tab provides information on the status of all graphs, including graphs that have completed execution.

Action	Description
Customize the view of status panel	Display the status of all graphs either in  ( <a href="#">List View</a> ) where instances of a graph are displayed separately as a list, or in

Action	Description
	 ( <a href="#">Tree View</a> ) where instances of a respective graph are grouped into one.
<b>Monitor status of subgraphs</b>	<p>When a graph execution triggers the execution of another graph, the other graph is called a subgraph. Monitor the status of various subgraphs that the Modeler has executed by doing the following:</p> <ol style="list-style-type: none"> <li>1. Open the <a href="#">Status</a> tab.</li> <li>2. Enable <a href="#">Show Subgraphs</a>.</li> </ol> <p>You can see subgraphs only in the  (<a href="#">List View</a>).</p>
<b>Download diagnostic information</b>	<p>Download the diagnostic information that the Modeler generates for a graph as a zipped archive.</p> <ol style="list-style-type: none"> <li>1. In the <a href="#">Status</a> tab, next to the required graph, choose  (<a href="#">Download Diagnostic Information</a>).</li> <li>2. Open or save the zipped archive file.</li> </ol> <div data-bbox="783 864 1402 1021" style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> <p><b>i Note</b></p> <p>If a graph has subgraphs or the graph is a subgraph, the archive contains information about all of the graphs in the hierarchy.</p> </div>
<b>Edit a graph</b>	<p>Edit a graph by opening the <a href="#">Status</a> tab next to the applicable graph and selecting  (<a href="#">Open Graph Editor</a>).</p>
<b>Pause and restart a process</b>	<p>Pause the execution of a graph by opening the <a href="#">Status</a> tab next to the required graph and selecting  (<a href="#">Pause process</a>).</p> <p>Restart the paused graph by selecting  (<a href="#">Restart process</a>).</p> <p>Restarting a graph creates a new instance of the graph with previously saved graph, recovery, and snapshot configurations.</p> <p>You can also configure automatic recovery of graph for graph failures. For more information, see <a href="#">Running Graphs [page 72]</a>.</p>
<b>Archive instances</b>	<p>Archive a single graph instance by selecting  (<a href="#">Archive</a>) next to the applicable graph. You can archive a graph in either  (<a href="#">List View</a>) or  (<a href="#">Tree View</a>).</p> <p>To archive multiple graph instances of a group at once, switch to  (<a href="#">Tree View</a>) and select  (<a href="#">Archive</a>) next to the required subgraph.</p> <div data-bbox="783 1715 1402 1944" style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> <p><b>i Note</b></p> <p>The Modeler archives the graph instances with status <a href="#">stopped by pause</a>, <a href="#">dead</a>, and <a href="#">completed</a>. The Modeler doesn't archive graphs whose latest instance is a valid snapshot with status other than completed.</p> </div>

Action	Description
Archive instances by status	<p>Archive graph instances by status by opening the <a href="#">Status</a> tab and selecting <a href="#">Cleanup</a>. Choose one of the following options:</p> <ul style="list-style-type: none"> <li>• <a href="#">Archive only completed instances</a></li> <li>• <a href="#">Archive completed and dead instances</a></li> </ul> <p>To delete any saved snapshots, select the <a href="#">Delete saved snapshots</a> checkbox.</p>
View execution details	<p>View additional details about a graph's execution after execution completes.</p> <p><b>Snapshots enabled:</b> In the <a href="#">Status</a> tab, the symbol  next to the graph name indicates that the snapshots feature is enabled.</p> <p><b>Detailed graph information:</b> To view detailed information about a graph, select the graph name in the <a href="#">Status</a> tab. The Modeler opens the graph in the graph editor area and displays a series of tabs below the graph. For example, view information in the following tabs:</p> <ul style="list-style-type: none"> <li>• <a href="#">Group</a> tab: View more information about the groups in a partitioned graph.</li> </ul> <div data-bbox="831 1077 1396 1263" style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> <p><b>i Note</b></p> <p>If you've defined a multiplicity of a number greater than 1 for a group, such as 3, the Modeler displays three instances for the same group.</p> </div> <ul style="list-style-type: none"> <li>• <a href="#">Process</a> tab: View details of various processes executed in the graph.</li> <li>• <a href="#">Metrics</a> tab: View metrics that the Modeler provides as part of the process.</li> </ul>
Stop execution	<p>Stop the execution of any graph by opening the <a href="#">Status</a> tab, selecting the applicable graph, and choosing  (<a href="#">Stop Process</a>).</p> <p>The status of the graph changes to <a href="#">completed</a> or <a href="#">dead</a> depending on the state of the graph when the execution is stopped.</p>

## Related Information

[Graph Execution \[page 104\]](#)

[Graph Status \[page 104\]](#)

[Process Status \[page 105\]](#)

[Graph Execution Garbage Collection \[page 106\]](#)

### 4.11.1.1 Graph Execution

The Modeler application supports executing a graph and monitoring its status from within the application.

When you schedule a graph for execution, the application translates the graphical representation (internally represented as a JSON document) into a set of running processes. These processes are responsible for the graph execution.

During the graph execution, the application translates each operator in the graph into (server) processes and translates the input and output ports of the operators into message queues. The process runs and waits for an input message from the message queue (input port). When it receives the input message, it starts processing to produce an output message and delivers it to the outgoing message queue (output port). If a message reaches a termination operator, the application stops executing all processes in the graph and the data flow stops.

#### i Note

If a graph does not have a graph terminator operator, it continues to execute all processes until you manually stop the graph execution.

### 4.11.1.2 Graph Status

When you create a graph, each graph is associated with a graph status, which may vary with time and can be manipulated with operations on the graph.

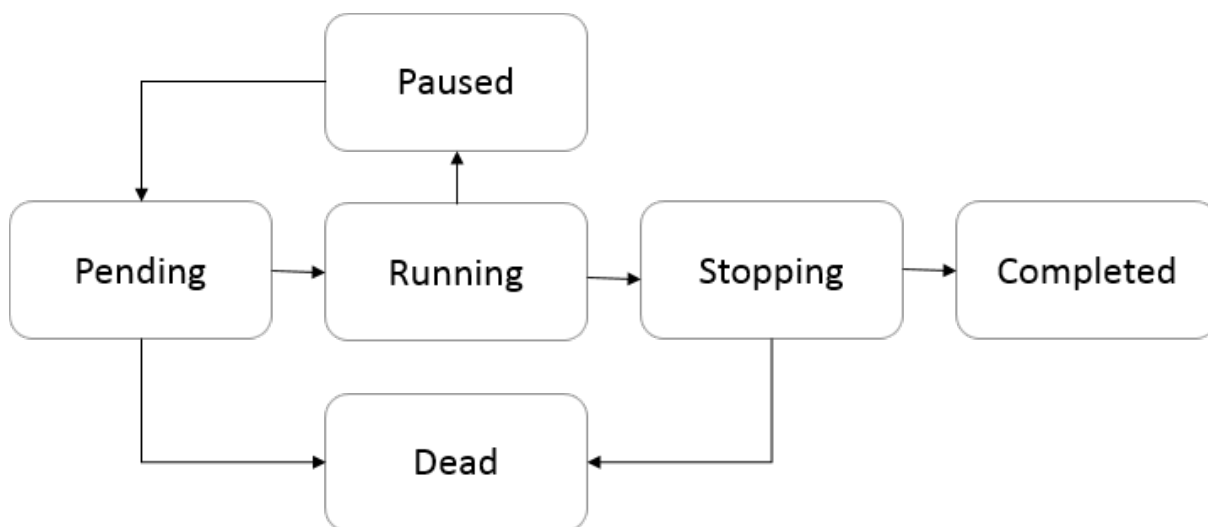
The table lists the possible graph status. You can view the status of the graph in the [Status](#) tab.

Status	Description
Pending	Graph is being prepared for execution. Initial status.
Running	Graph is currently running.
Paused	Graph is currently paused and can be resumed later.
Stopping	Graph execution is stopping.
Completed	Graph terminated successfully.
Dead	Graph terminated abnormally because one or more operators in the graph failed.
Unknown	Status of graph is unknown. Indicates internal problems.

Initially, the graph has pending status, that is, the graph is being prepared for execution. Pending status stays until either an error occurs, or the status of all subgraphs in graph is running.

The following image depicts the potential graph transition status.





If an error occurs during the running status, the graph allocation (and hence the overall graph) changes to dead. If all subgraphs terminate successfully, the graph status changes to completed.

## Related Information

[Monitor the Graph Execution Status \[page 101\]](#)

### 4.11.1.3 Process Status

When you execute a graph, the tool executes each operator in the graph as processes. Each process execution is associated with a status, which may vary with time.

The following table lists the possible status for process execution. Select the *Process* tab in status details to view the status of process execution.

Status	Description
Initializing	Process is starting.
Running	Process is on-going.
Stopping	User called STOP, the process is still running.
Stopped	Process has terminated.
Dead	Process has crashed and is unrecoverable.

## Related Information

[Monitor the Graph Execution Status \[page 101\]](#)

### 4.11.1.4 Graph Execution Garbage Collection

To maintain the cluster, the Modeler employs Graph Garbage Collection using the following strategies: time-based and number-based.

Garbage collection settings are in the System Management properties. For a list of all system management properties, see [System Management Properties](#) in the *Administration Guide*.

The garbage collector collects the following finished graphs:

- completed (always)
- dead (optionally)
- paused (optionally)

For more information, see [Graph Status \[page 104\]](#).

#### Time-Based Strategy

The time-based strategy specifies the time before the garbage collector frees memory used by the finished graphs.

Property	Default Value	Result
Garbage collection time limit for finished graphs.	72 hours	The Modeler collects graphs that have been finished for 3 days.

The lowest value the Modeler system accepts is 1 minute. If you enter a value lower than 1 minute, the Modeler replaces the value with 1 minute. For more information about the format and layout of the duration, see the [Golang.org Parse Duration official documentation](#) .

#### Number-Based Strategy

The number-based strategy limits the number of graphs per user in the Modeler at any time. When the user exceeds the limit, the Modeler removes completed graphs. If the user exceeds the limit and there aren't any finished graphs to be collected, the Modeler waits until finished graphs appear and then collects them.

##### i Note

The number-based strategy prioritizes the removal of completed over dead over paused graphs.

Property	Default Value	Result
Maximum number of graphs per user. Finished graphs will be deleted when exceeded.	50	At most, the Modeler keeps 50 graph executions at a time.

## Collect Dead Graphs

Because dead graphs can collect failure information, you can exclude dead graphs from garbage collection by setting the System Management property *Enable garbage collection for dead graphs*. Use failure information to analyze information about why the graph failed.

If you exclude dead graphs using the time-based strategy, the Modeler ignores dead graphs. In number-based mode, the Modeler doesn't include dead graphs in the graph count.

Property	Default Value	Result
Enable garbage collection for dead graphs.	true	The Modeler collects dead graphs.

## Collect Paused Graphs

By default, the removal of paused graphs is disabled, because pausing a graph signals the intent to resume it later.

However, if users have forgotten them, pausing graphs leads to the accumulation of many paused graphs over time. To prevent this accumulation, you can optionally enable the collection of paused graphs.

If you disable the collection of pause graphs and you use the time-based strategy, paused graphs are ignored. With the number-based strategy, the Modeler doesn't include paused graphs in the graph count.

### Note

Be careful when you turn on the collection of paused graphs because it can cause the Modeler to lose possible valuable data.

Property	Default Value	Result
Enable garbage collection for paused graphs.	false	The Modeler doesn't collect paused graphs.

Changes to the Garbage Collector System Management parameters can take the system up to 5 minutes to update without restarting the application.

## 4.11.2 Activate Trace Messages

Trace message logs contain a complete set of information for monitoring graph execution performance. Trace message logs help you to isolate problems or errors that occur based on different severity threshold levels.

### Prerequisites

Read about trace levels in [Trace Severity Levels \[page 109\]](#).

### Context

You can activate trace message logging to find errors that occur sporadically. The Modeler logs the trace messages and categorizes them based on the severity levels. You can also limit the number of trace files that are written at the same time.

Trace messages are intended for specific users. For example, trace messages are useful for users with development skills and a deep understanding of the Modeler to debug graph executions.

### Procedure

1. Start the SAP Data Intelligence Modeler application.
2. Open the *Trace* tab in the graph's runtime view.  
In the *Trace* tab, you can configure the trace level, download the latest logs, and monitor the trace messages for different severity levels.
3. **Optional:** To change the group and trace level, perform the following substeps:  
By default, a graph's *Trace Level* is INFO.
  - a. Select *Group: default | Trace Level: INFO* at the top right of the *Trace* tab.
  - b. In the *Trace Configuration* dialog, select a different value from the *Trace Level* list.  
  
Trace Level options are as follows:
    - INFO
    - DEBUG
    - ERROR
    - FATAL
    - WARNING
  - c. Select *OK*.
4. **Optional:** Select *Get Latest Logs* at the top right of the *Trace* tab.  
The Modeler displays the latest logs.

## Related Information

[Trace Severity Levels \[page 109\]](#)

### 4.11.2.1 Trace Severity Levels

When streaming the traces for the SAP Data Intelligence Modeler application, you can set a trace severity threshold. Depending on the trace level that you define, the application streams messages accordingly into the trace server.

The application supports the following trace levels.

Trace Level	Description
INFO	The publisher streams trace messages that contain informational text, mostly for echoing what has happened in the application. The trace messages streamed also include messages associated with warnings, errors, and fatal errors.
DEBUG	The publisher streams trace messages that contain useful information for developers to debug and analyze the application. The trace messages streamed also include messages associated with info, error, warning, and fatal error.
ERROR	The publisher streams trace messages that contain information describing the error conditions that may occur when working with the application. The trace messages streamed also include messages associated with the fatal error.
FATAL	The publisher streams trace messages associated with fatal errors that may occur when working with the application.
WARNING	The publisher streams messages associated with warnings and errors that may occur when working with the application. The trace messages streamed also include messages associated with error and fatal error.

### 4.11.3 Downloading Diagnostic Information for Graphs

To help you diagnose graph issues, download a zipped archive of information.

The SAP Data Intelligence Modeler lets you download diagnostic information for graphs in the following ways:

- **For all graphs:** Select your profile icon in the upper right of the Modeler and choose [Download Diagnostic Information](#). The Modeler generates a zip archive of information for all of your graphs.
- **For a specific graph:** In the *Status* tab, select [↓ \(Download Diagnostic Information\)](#) next to a graph to generate a zip archive of information about the graph and its subgraphs.

## Related Information

[Diagnostic Information Archive Structure and Contents \[page 110\]](#)

### 4.11.3.1 Diagnostic Information Archive Structure and Contents

This is the directory structure and contents of the diagnostic information archive.

#### Zip Archive Structure

```
yflow-diagnostic-<timestamp>.zip
├── version.json
├── graphs.json
├── errors.txt
├── api-pods
│   ├── pods.json
│   ├── goroutine.txt
│   ├── <podname>
│   │   ├── goroutine.txt
│   │   ├── logs-<podname>.txt
│   │   └── pod-<podname>.json
├── <graph-source>-<handle>
│   ├── graph.json
│   ├── execution.json
│   ├── <group-instance-id>
│   │   ├── goroutine.txt
│   │   ├── heap.txt
│   │   ├── logs-<podname>.txt
│   │   └── pod-<podname>.json
│   └── ...
├── ...
│   ├── <group-instance-id>
│   │   ├── goroutine.txt
│   │   ├── heap.txt
│   │   ├── logs-<podname>.txt
│   │   └── pod-<podname>.json
├── <graph-source>-<handle>
└── ...
```

#### Related Information

- [version.json File \[page 111\]](#)
- [graphs.json File \[page 111\]](#)
- [<graph-source>-<handle> Folders \[page 112\]](#)
- [graph.json File \[page 112\]](#)
- [execution.json File \[page 112\]](#)
- [events.json File \[page 113\]](#)
- [<group-instance-id> Folders \[page 113\]](#)
- [logs-<pod-name>.txt File \[page 114\]](#)

[pod-<pod-name>.json File \[page 114\]](#)

[goroutine.txt File \[page 114\]](#)

[<heap.txt> File \[page 114\]](#)

[api-pod Folder \[page 114\]](#)

### 4.11.3.1.1 version.json File

The `version.json` file contains version information.

For example:

```
{
  "version": "2.3.30-dev-0828",
  "buildTime": "2018-08-28T17:10:27",
  "gitCommit": "a1897fcf38788b55be1ce177e32bb2ebc733b28c",
  "platform": "linux"
}
```

### 4.11.3.1.2 graphs.json File

The `graphs.json` file contains brief information about executed graphs, including files that are completed and not deleted.

The file can be used, for example, to identify graphs with a status of dead, the last messages of a graph, and so on.

```
[
  {
    "src": "com.sap.demo.datagenerator",
    "name": "com.sap.demo.datagenerator",
    "executionType": "",
    "handle": "9a6eeb59abb242baabb110dba50ae178",
    "status": "running",
    "terminationRequested": false,
    "message": "Graph is currently running",
    "started": "2018-08-29T10:26:41Z",
    "updated": "2018-08-29T10:26:45Z",
    "stopped": "",
    "submitted": "2018-08-29T10:26:40Z"
  },
  ...
]
```

### 4.11.3.1.3 <graph-source>-<handle> Folders

The <graph-source>-<handle> folder contains detailed information about graph execution. For example: `com.sap.demo.datagenerator-9a6eeb59abb242baabb110dba50ae178`.

To identify the files that you want to open, see the `graphs.json` file.

```
[
  {
    "src": "com.sap.demo.datagenerator",
    ...
    "handle": "9a6eeb59abb242baabb110dba50ae178",
    ...
  },
  ...
]
```

### 4.11.3.1.4 graph.json File

The `graph.json` file contains the graph definition.

The content in the `graph.json` file is the same that you get when you export the graph from the Modeler application.

### 4.11.3.1.5 execution.json File

The `execution.json` file contains information about the execution for a graph, such as groups, pods, processes (operator instances in a graph), and so on.

You can use the `execution.json` file to identify which pod or pods failed.

```
{
  "src": "com.sap.demo.datagenerator",
  "name": "com.sap.demo.datagenerator",
  "executionType": "stream",
  "handle": "9a6eeb59abb242baabb110dba50ae178",
  "status": "running",
  "terminationRequested": false,
  "message": "Graph is currently running",
  "remoteExecution": {
    "parent": ""
  },
  "started": 1535538401,
  "updated": 1535538405,
  "stopped": 0,
  "submitted": 1535538400,
  "allocations": [
    {
      "groupName": "default",
      "groupDescription": "",
      "subgraph": "default",
      "container": "vflow-graph-9a6eeb59abb242baabb110dba50ae178-com-sap-
demo-ngq4m",
      "containerIp": "172.17.0.8",
      "host": "minikube",
```



```

    "status": "running",
    "message": "Container is currently running",
    "restartCount": 0,
    "image": "a-docker-registry:5000/vora/vflow-node:2.3.30-dev-0829-
com.sap.debian",
    "destination": "",
    "updated": "2018-08-29T10:26:40Z",
    "processes": [
      {
        "id": "16d1",
        "componentId": "com.sap.system.jsengine",
        "status": "running",
        "processName": "datagenerator (16d1)",
        "engine": "main",
        "timestampPublished": "2018-08-29T10:26:41Z",
        "timestampReceived": "2018-08-29T10:26:41.031657526Z",
        "published": 1535538401,
        "received": 1535538401,
        "message": "Process is running",
        "metrics": null
      },
      {
        "id": "1mnu",
        "componentId": "com.sap.system.terminal",
        "status": "running",
        "processName": "terminal (1mnu)",
        "engine": "main",
        "timestampPublished": "2018-08-29T10:26:41Z",
        "timestampReceived": "2018-08-29T10:26:41.033268298Z",
        "published": 1535538401,
        "received": 1535538401,
        "message": "Process is running",
        "metrics": null
      }
    ]
  }
}

```

### 4.11.3.1.6 events.json File

The `events.json` file contains information about graph events.

For example, when a group is initialized, a group execution is started, or a graph has received a shutdown command, and so on.

### 4.11.3.1.7 <group-instance-id> Folders

For each instance of a group, a folder is created.

To determine which `<group-instance-id>` to open, see the `execution.json` file.

```

{
  "allocations": [
    {
      "subgraph": "<group-instance-id",
      ...
    }
  ]
}

```

---

### 4.11.3.1.8 logs-`<pod-name>`.txt File

The `logs-<pod-name>.txt` file contains logs for a specific pod.

### 4.11.3.1.9 pod-`<pod-name>`.json File

The `pod-<pod-name>.json` file contains a pod description.

The file content is similar to the output of a `kubectl describe pod`.

### 4.11.3.1.10 goroutine.txt File

The `goroutine.txt` file contains `pprof` output for a goroutine profile.

### 4.11.3.1.11 `<heap.txt>` File

The `<heap.txt>` file contains `pprof` output for a heap profile.

### 4.11.3.1.12 api-pod Folder

The `api-pod` folder contains information about vFlow API nodes.

The folder structure is similar to the `<group-instance-id>` folder, except that it does not contain a `heap.txt` file. A `heap.txt` file can become quite large for long-running vFlow API nodes, and it is rarely used.

## 4.11.3.2 Saving Diagnostic Information for Graphs on External Storage

Save a zipped archive of information for your graphs on external storage to help diagnose issues.

The SAP Data Intelligence Modeler allows saving diagnostic information for graphs on external storage. It saves the archive for graphs that reach the final state, depending on the configured System Management properties for the tenant.

- Set the property *Enable diagnostics archive to external storage defined via Connection ID* to `true` to enable the feature.
- Specify the Connection ID for the storage expected to be used on the property *Connection ID for diagnostics storage*. It supports the HDFS, WebHDFS, S3, ADL, GCS, SFTP, OSS, HDL, WASB, and SDL connection types.
- Set the property *Store diagnostics on external storage only for failed graphs* to `true` when it should store diagnostics for failed graphs exclusively. Otherwise, to store the diagnostic archive to any final graph, this property should be `false`.
- Specific graph: In the *Status* tab, select Download Diagnostic Information next to a graph to generate a zip archive of information about the graph and its subgraphs.
- The property *Garbage collector strategy for diagnostics storage* allows to choose whether the graph archive should never be removed from the storage or whether it should be removed when historical graphs are removed. If the `history` option is chosen, the files are removed according to the graph garbage collector parameter `Execution history retention time limit`.
- Data on the storage defined always follows a path pattern: `<rootPath-from-connection>/graph_diagnostics/<tenant>/<user>/graph_<handle>.zip`, where [Diagnostic Information Archive Structure and Contents \[page 110\]](#) is the directory structure and contents of the diagnostic information archive.

## Current Limitations

- If one of the Modeler applications is being downscaled in parallel when a graph reaches its final state, the graph diagnostics archive may not be stored.
- The object store is not automatically cleaned up when a user or tenant is deleted.
- If the underlying storage or connection changes after enabling the feature, the previous storage is not automatically cleaned up.
- Custom time-based cleanup of the stored archives is not supported.

## 4.12 Native Multiplexing for Gen2 Pipelines

Connect to multiple ports in a pipeline, such as one to many or many to one, without having to implement multiplexing with a script operator or other predefined operator.

When you use Generation 1 pipelines, you use Multiplexer operators to connect to an output or input port of one operator with several input or output ports of another operator. In Generation 2 pipelines, multiplexing is natively built into the Python and main subengines. Therefore, you don't need to use additional Multiplexer operators in Generation 2 pipelines.

### i Note

Native multiplexing is available for Generation 2 pipelines only, and only in SAP Data Intelligence Cloud version 2022.21 and higher. Multiplexing works in the Python and main subengines only.

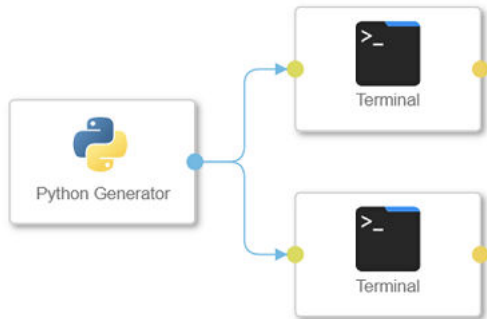
For Generation 2 pipelines, you can still use the Python multiplexer to create a python script for multiplexing when necessary.

## Supported Connections

Native multiplexing for Generation 2 pipelines works for the following types of connections:

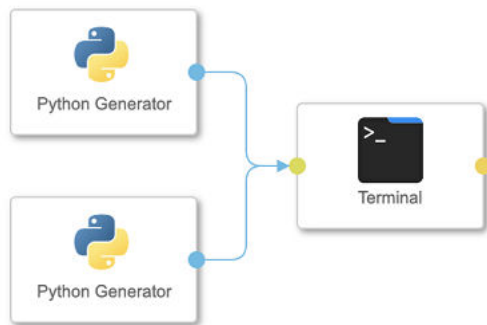
### One to many: 1:n

In the following pipeline, the Python Generator has one output port that leads to two separate Terminal operators. Therefore, the Python Generator has multiple output ports. Each of the terminals has one input connection.



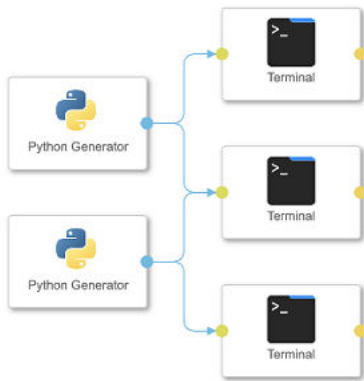
### Many to one: n:1

In the following pipeline, each of the Python Generator operators have one output port. The input port of the Terminal operator receives data from the output ports of two (multiple) Python operators.



### Many to many: m:n

In the following pipeline, each of the Python Generator operators sends data to multiple Terminal operators. The input ports of each of the Terminal operators receives data from the output ports of two (multiple) Python Generator Operators.



## Related Information

[Multiplexing Scenarios \[page 117\]](#)

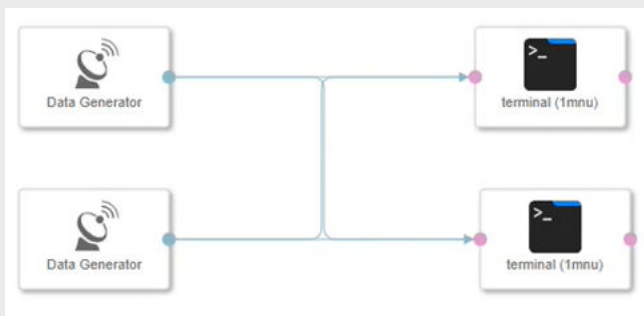
### 4.12.1 Multiplexing Scenarios

When you create Generation 2 pipelines using the native multiplexing feature, you must be aware of situations when multiplexing works, or when it results in errors.

#### Deadlock Error for BLOB Messages

In a scenario where you have multiple generators publishing messages to multiple receivers (m:n), and you have a cross over between the message transmission, the graph can become deadlocked when the message includes blob data types.

##### ❖ Example

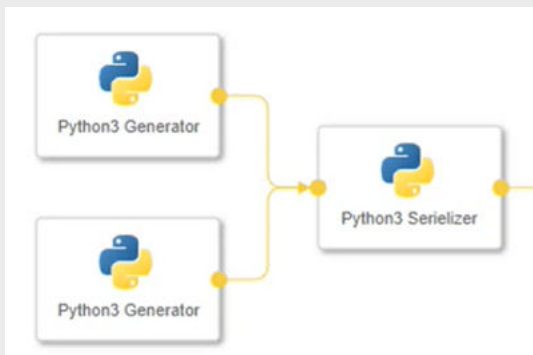


## Multiplexing with Mixed Engine Types

SAP Data Intelligence supports native multiplexing in the Python and main subengines. Multiplexing doesn't work with other subengines, such as the Flowagent subengine.

### ❖ Example

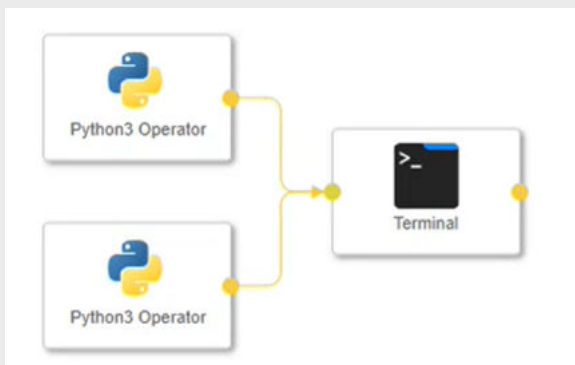
The following diagram shows a portion of a multiplexed pipeline with operators that belong to the Python3 subengine. In this example, the Python subengine processes the multiplexing in this portion of the pipeline.



When a pipeline contains multiplexing portions that involve the Python subengine and the main engine, SAP Data Intelligence switches processing between the subengine and the main engine.

### ❖ Example

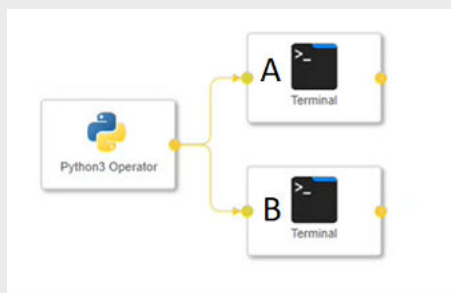
After multiplex processing completes in the Python3 subengine, SAP Data Intelligence switches processing to the main engine. In the following pipeline, the main engine processes the output from the Python3 Operator objects to the terminal.



A pipeline that contains operations for the subengine and main engine can result in an error based on where the multiplexing happens.

### ❖ Example

In the following pipeline, The following pipeline contains operators from three engines.



- Python3 Operator uses the Python subengine.
- Terminal A uses the Flowagent subengine.
- Terminal B uses the main engine.

The output port of the Python3 Operator has multiple connections. The output port sends a message to two Terminal operators, A and B. Each Terminal operator receives a message from one connection. Therefore, the multiplexing is handled by Python subengine.

## 5 Using Git Terminal

Use the Git terminal to integrate SAP Data Intelligence file-based content, such as graphs, operators, docker files, or script code, with an existing Git server.

Keep a version history of changes in Git and work collaboratively with other developers by sharing content files in Git.

Leverage Git capabilities in the SAP Data Intelligence Modeler application, and perform the following basic tasks:

- Maintain Git credentials using standard Git Credential Helper.
- Create a local Git repository.
- Clone a remote Git repository.

### Other Git Features

In addition to the Git commands for the basic configurations, there are other one-line commands and features of the Git command-line client for Linux. For example, use the following one-line codes:

- `git branch`: Creates a new branch.
- `git merge`: Combines a specified branch history with the current branch.
- `git rebase`: Change a series of commits that modify the history of your repository.
- `.gitignore` files:
  - Prevent tracking specific artifacts that are in both the Git repository and the working directory, such as the Modeler workspace.
  - Manipulate the content of the `info/exclude` file in the `.git` directory of the local Git repository.

#### ❁ Example

To ignore all files and track only one `graph.json` file, run the following command:

```
# setup repo to ignore everything
Export GIT_DIR=vhome/.git
echo '# ignore everything' >> $GIT_DIR/info/exclude
echo '*' >> $GIT_DIR/info/exclude
echo '# but allow directories' >> $GIT_DIR/info/exclude
echo '!*/' >> $GIT_DIR/info/exclude
echo '# add include pattern below starting with !' >> $GIT_DIR/info/exclude
echo '!vflow/graphs/demo-graph/**' >> $GIT_DIR/info/exclude
```

### Related Information

[Git Credential Handling Using Standard Git Credential Helper \[page 121\]](#)



[Create a Local Git Repository \[page 122\]](#)

[Clone a Remote Git Repository \[page 123\]](#)

## 5.1 Git Credential Handling Using Standard Git Credential Helper

To avoid entering credentials for each Git command, configure either `git-credential-store` or `git-credential-cache`.

Before you use `git-credential-store` or `git-credential-cache`, set the `HOME` variable accordingly with the following terminal command:

```
export HOME=~
```

The tilde (~) corresponds to a folder named `project` that is located at the same level as the `vhome` folder.

### i Note

While the `vhome` folder is listed in the *Files* tab of the SAP Data Intelligence System Management application, the `project` folder isn't listed in the SAP Data Intelligence System Management application.

### git-credential-store

`git-credential-store` keeps credentials in a file in `${HOME}/.git-credentials`, which requires protection accordingly. For more information about `git-credential-store`, see the Git documentation at <https://git-scm.com/docs/git-credential-store>.

#### ❖ Example

```
git config --global credential.helper store
```

### git-credential-cache

`git-credential-cache` keeps the credentials in memory in a daemon process that is communicating through a socket: `${HOME}/.git-credential-cache/socket`. You can configure `git-credential-cache` timeout. For more information about `git-credential-cache`, see the Git documentation at <https://git-scm.com/docs/git-credential-cache>.

`git-credential-cache` is preferred over `git-credential-store` because `git-credential-store` stores credentials on disk.

#### ❖ Example

The following Git Credential caches credentials for one day:

```
git config --global credential.helper 'cache -timeout=86400'
```

`${HOME}/.git-credential-cache/` and `${HOME}/.git-credentials/` can't be located in `/vhome`.

After the container that runs the git terminal user application is recreated, re-enter the stored or cached credentials.

## 5.2 Create a Local Git Repository

Use the Git Terminal in the SAP Data Intelligence Modeler application to create a local Git repository, then push the graph to your remote Git repository location.

### Prerequisites

Before you perform this task, ensure that you configure credential handling. For information about configuring credential handling, see [Git Credential Handling Using Standard Git Credential Helper \[page 121\]](#).

Ensure that you have the appropriate permissions to create a local Git repository in SAP Data Intelligence.

#### i Note

If you don't see the *Git Terminal* tab in the lower pane of the Modeler, request the applicable permission from your administrator.

### Context

#### ⚠ Caution

The following steps are to introduce the process for a basic creation of a local Git repository. The success of the example command sequences depends on the actual folder structure of the remote Git repository, or additional design time artifacts that are stored in your user workspace.

To create a local Git repository, perform the following steps in the Modeler application:

### Procedure

1. Create a graph using the following folder structure:  
`files/vflow/graphs/demo-graph`
2. Open the *Git Terminal* tab in the lower pane of the Modeler.

A Git terminal opens in the tab.

3. Type the following code in the Git terminal:

```
# init git repository and set the name of the default branch to
# 'main'
git init -b main
# configure user E-Mail and user name for the local git #repository
git config user.email "<replace with email>"
git config user.name "<replace with user name>"
# add the demo-graph to the git staging area
git add vflow/graphs/demo-graph/graph.json
# commit the changes to the local Git repository
git commit -m "add demo graph"
```

This code creates the local Git repository.

4. Type the following code in the Git terminal, replacing `https://<url>/to/remote_repository.git` with the actual address to the remote Git repository:

```
# add the address to the remote Git repository to the local Git
# repository
git remote add origin https://<url>/to/remote_repository.git
# push the local changes to the remote Git repository
git push -u origin main
```

This code pushes the changes (in the local Git repository) to the specified remote repository on the Git server.

## 5.3 Clone a Remote Git Repository

### Prerequisites

Before you perform this task, ensure that you've completed the following tasks:

- Configure credential handling by following the process in [Git Credential Handling Using Standard Git Credential Helper \[page 121\]](#).
- Create a local Git repository by following the process in [Create a Local Git Repository \[page 122\]](#).

Also ensure the following:

- The workspace of the user of the Git terminal application in SAP Data Intelligence Cloud is empty.
- You have the appropriate permissions to access the Git terminal in the Modeler application.

#### **i** Note

If you don't see the *Git Terminal* tab in the lower pane of the Modeler, request the applicable permission from your administrator.

## Context

### ⚠ Caution

The following steps are to introduce the process for a cloning a remote Git repository. The success of the example command sequences depends on the actual folder structure of the remote Git repository, or additional design time artifacts that are stored in your user workspace.

To clone a remote Git repository, perform the following steps:

## Procedure

1. Open the *Git Terminal* tab in the lower pane of the Modeler.  
A Git terminal opens in the tab.
2. Type the following command into the Git terminal:

```
# configure user E-Mail and user name for the local gitrepository  
git clone https://<url>/to/remote_repository.git .
```

### i Note

The dot at the end of this command causes the Git command-line client to clone the repository content to the root folder of the user workspace (`vhome`) instead of cloning the content to a subfolder (`vhome/.remote_repository`).


## Results

Because the folder structure in the remote Git repository is similar to the following structure, SAP Data Intelligence recognizes the `graph.json`, and you can run the graph or manipulate its content directly:

```
vflow/  
  graphs  
    demo-graph  
  operators  
  dockerfiles  
  subdevkits  
  subengines
```

# 6 Using Scenario Templates

SAP Data Intelligence provides common graph scenarios that you can use with operators and graphs.

Find the templates that are shipped as example graphs in the Modeler application Graphs tab in the navigation pane at left. Make sure that you include *Scenario Templates* in the visible categories by selecting  (*Customize Visible Categories*). You can also search for the package `com.sap.scenarioTemplates`.

To learn how to set up and run each scenario template, see “Scenario Templates” in *Repository Objects Reference*.

## Related Information

[ABAP with Data Lakes \[page 125\]](#)

[Data Processing with Scripting Languages \[page 126\]](#)

[ETL from Database \[page 128\]](#)

[Loading Data from Data Lake to Database \(SAP HANA\) \[page 128\]](#)


## 6.1 ABAP with Data Lakes

These graphs show how to ingest ABAP Tables or CDS Views data from SAP S/4HANA and SAP Business Suite systems into a cloud storage.

For both data source types, there are example graphs that showcase how to supply data lakes using a full or delta load mechanism.

A typical template scenario consists of a reader operator (SLT Connector or CDS Reader) that runs on a connected ABAP system. The connected ABAP system streams the data into a pipeline with a file writer or a Kafka producer operator.

### i Note

The ABAP system needs to fulfill the prerequisites documented in [2835207](#)  before it can be connected to SAP Data Intelligence.

## Available Templates

Template	Path	Description
Data Extraction using SLT to a File Store <a href="#">Data Extraction using SLT to a File Store</a>	com.sap.scenarioTemplates.ABAP.SLTtoFile	Connect to an SAP Landscape Transformation Replication Server (SLT) configuration to read table data and write the data to a (cloud) storage or data lake.
Data Extraction using SLT to KAFKA <a href="#">Data Extraction using SLT to KAFKA</a>	com.sap.scenarioTemplates.ABAP.SLTtoKafka	Connect to an SAP Landscape Transformation Replication Server (SLT) configuration to read table data and feed the data into a Kafka pipeline.
Data Extraction from SAP S/4HANA CDS View to a File Store <a href="#">Data Extraction from SAP S/4HANA CDS View to a File Store</a>	com.sap.scenarioTemplates.ABAP.CDSToFile	Connect to an SAP S/4HANA system to read CDS View Data and write the data to a cloud storage or data lake.
Data Extraction from SAP S/4HANA CDS View to KAFKA <a href="#">Data Extraction from SAP S/4HANA CDS View to KAFKA</a>	com.sap.scenarioTemplates.ABAP.SLTtoKafka	Connect to an SAP S/4HANA system to read CDS View Data and feed the data into a Kafka pipeline.

To learn how to set up and run each scenario template, see "Scenario Templates" in the *Repository Objects Reference for SAP Data Intelligence*. [Scenario Templates](#) in the [Repository Objects Reference for SAP Data Intelligence](#).

## 6.2 Data Processing with Scripting Languages

These graphs show how to manipulate data with scripting languages.

A typical scenario consists of the following:

- Reading input data from storage, such as a file or a database table.
- Applying a processing algorithm to the data that is implemented in a scripting language, such as Javascript, Node, Python, or R.
- Writing the results of the data processing to another storage area, which may or may not be the same as the one providing the input data.

### Available Templates

JavaScript

Template	Path	Description
Simple Javascript File Data Manipulation <a href="#">Simple Javascript File Data Manipulation</a>	com.sap.scenarioTemplates.customDataProcessing.simpleF2FJavascript	File data manipulation using JavaScript and writing the manipulated data back to another file.

Template	Path	Description
Simple Javascript File-to-DB Data Manipulation <a href="#">Simple Javascript File-to-DB Data Manipulation</a>	com.sap.scenarioTemplates.customDataProcessing.simpleF2DBJavascript	File data manipulation using JavaScript and writing the manipulated data to an SAP HANA database table.
Simple Javascript DB-to-File Data Manipulation <a href="#">Simple Javascript DB-to-File Data Manipulation</a>	com.sap.scenarioTemplates.customDataProcessing.simpleDB2FJavascript	Manipulation of data from an SAP HANA database table using JavaScript and writing the manipulated data to a file.
Javascript File Data Manipulation <a href="#">Javascript File Data Manipulation</a>	com.sap.scenarioTemplates.customDataProcessing.F2FJavascript	File data manipulation with a custom JavaScript operator and writing the manipulated data back to another file.

#### Node

Template	Path	Description
Simple Node.js File Data Manipulation <a href="#">Simple Node.js File Data Manipulation</a>	com.sap.scenarioTemplates.customDataProcessing.simpleF2FNode	File data manipulation using Node.js and writing the manipulated data back to another file.
File-to-File Custom Node Operator	com.sap.scenarioTemplates.customDataProcessing.F2FNode	File data manipulation with a custom Node.js operator and writing the manipulated data back to another file.

#### Python

Template	Path	Description
Simple Python File Data Manipulation <a href="#">Simple Python File Data Manipulation</a>	com.sap.scenarioTemplates.customDataProcessing.simpleF2FPython	File data manipulation using Python and writing the manipulated data back to another file.
Python File Data Manipulation with Pandas <a href="#">Python File Data Manipulation with Pandas</a>	com.sap.scenarioTemplates.customDataProcessing.F2FPython	File data manipulation with a custom Python operator using the pandas module and writing the manipulated data back to another file.

#### R

Template	Path	Description
Simple R File Data Manipulation <a href="#">Simple R File Data Manipulation</a>	com.sap.scenarioTemplates.customDataProcessing.simpleF2FR	File data manipulation using R and writing the manipulated data back to another file.
Simple R File-to-DB Data Manipulation <a href="#">Simple R File-to-DB Data Manipulation</a>	com.sap.scenarioTemplates.customDataProcessing.simpleF2DBR	File data manipulation using R and writing the manipulated data to an SAP HANA database table.
File-to-File Custom R Operator	com.sap.scenarioTemplates.customDataProcessing.F2FR	File data manipulation with a custom R operator and writing the manipulated data back to another file.

To learn how to set up and run each scenario template, see “Scenario Templates” in the *Repository Objects Reference for SAP Data Intelligence*. [Scenario Templates](#) in the [Repository Objects Reference for SAP Data Intelligence](#).

## 6.3 ETL from Database

These graphs show how to extract, transform, and load data from different databases into file storage or other databases.

### Available Templates

Template	Path	Description
Initial Load from any Table (parallel) <a href="#">Initial Load from any Table (parallel)</a>	com.sap.scenarioTemplates.ETL-FromDB.cdcInitialLoad	Read contents from an Oracle table and load it into files stored on a connected (cloud) storage.
Initial Load + Delta Extraction from AnyDB <a href="#">Initial Load + Delta Extraction from AnyDB</a>	com.sap.scenarioTemplates.ETL-FromDB.cdcGraphGenerator	Use the CDC Graph Generator operator for replication of relational databases. The CDC Graph Generator operator generates the required SQL scripts so users can capture changes from a database source. The graphs generated from this graph capture the changes.

To learn how to set up and run each scenario template, see “Scenario Templates” in the *Repository Objects Reference for SAP Data Intelligence*. [Scenario Templates](#) in the [Repository Objects Reference for SAP Data Intelligence](#).

## 6.4 Loading Data from Data Lake to Database (SAP HANA)

These graphs show how to batch and stream process data.

### Batch Processing

Batch processing graphs show how to load data in a batch from different cloud storages into an SAP HANA database. Batch in this context means that the Modeler loads all data available at runtime and stops the execution when all files are processed.

### Available Templates

Template	Path	Description
Load Files into HANA <a href="#">Ingest Files Into SAP HANA (Incremental Load)</a>	com.sap.scenarioTemplates.datalake-ToDatabase.loadToHana	Load product data from CSV files into an SAP HANA table, offering at-least-once guarantee between multiple graph runs.



## Stream Processing

Stream processing graphs show how to load data in near real-time from different cloud storages into an SAP HANA database. Real-time in this context means that the graphs run continuously and read new files when available by polling repeatedly for changes on the connected file storage.

### Available Templates

Template	Path	Description
Ingest Files into HANA <a href="#">Ingest Files into HANA</a>	com.sap.scenarioTemplates.datalake-ToDatabase.ingestToHana	Ingest product data in parallel from CSV files into an SAP HANA table with a long-running graph, offering at-least-once guarantee between multiple graph runs.

To learn how to set up and run each scenario template, see “Scenario Templates” in the *Repository Objects Reference for SAP Data Intelligence*. [Scenario Templates](#) in the [Repository Objects Reference for SAP Data Intelligence](#).

# 7 Using Graph Snippets

SAP Data Intelligence provides a group of commonly used operators that you can add as a single entity to the graph instead of adding and configuring the operators individually.

A graph snippet is an entity that contains a group of operators and connections. The group performs a single logical function. You can import graph snippets to your graph.

To learn how to set up and run each graph snippet, see [Graph Snippets](#) in the *Repository Objects Reference*.

## Related Information

[Importing Graph Snippets \[page 130\]](#)

[Creating Graph Snippets \[page 131\]](#)

[Editing Graph Snippets \[page 132\]](#)

## 7.1 Importing Graph Snippets

You can import the available graph snippets to your graph.

### Procedure

1. Start the SAP Data Intelligence Modeler.
2. In the navigation pane, select the *Graphs* tab.
3. In the navigation pane bar, search for an existing graph or choose + (Create Graph).  
The application opens the graph editor, where you can import graph snippets.
4. To import graph snippets, right-click the empty area in the graph editor and select *Import Graph Snippet*.  
(Or click the *Import Graph Snippet* icon in the editor toolbar.)
5. In the *Import Snippet* dialog, select the required graph snippet.
6. Click *Proceed*.
7. In the subsequent dialog, fill in the necessary configuration details and click *OK*.

The graph snippet is now imported to your graph. You can configure each operator additionally, if required, in the configuration panel of the operator.

## 7.2 Creating Graph Snippets

You can create your own graph snippet if you don't find a suitable one in the available list of graph snippets.

### Procedure

1. Start the SAP Data Intelligence Modeler.
2. In the navigation pane, select the *Graphs* tab.
3. In the navigation pane bar, search for an existing graph or select **+** (*Create Graph*).
4. To create a graph snippet, select a part of the graph that you want to add as a graph snippet and right-click the selected area.
  - Press **shift** and drag the mouse to select a specific part of the graph.
  - Choose **ctrl** + **a** to select the complete graph.
5. Select the *Create Snippet* option in the context menu.

#### i Note

If your graph snippet consists of ABAP operators, you must select the connection and version for the operator before creating the snippet.

6. In the *Create Graph Snippet* dialog, for each operator or group, do one of the following:
  - Select the properties that are to be configured during import of the graph snippet.
  - Preconfigure some of the properties.
7. **Optional:** To provide better context about the parameters during import of the graph snippet to a graph:
  - a. Click *Add Description* for a selected parameter.
  - b. Provide details about the property and click *Save*.
8. To define parameters for properties and use across several operators, perform the following substeps:
  - a. Click *Add Parameter*.
  - b. Provide the details for the parameter and click *Save*. A parameter is created and you can use this for similar properties in other operators.

#### i Note

- To clear the changes you made to all operators in the graph snippet, click *Reset All*.
- To view all existing parameters in the graph snippet, click *Show Parameters*.

9. When you have finished configuring all required properties, click *Create*.
10. In the *Save Snippet* dialog, provide the necessary information for the graph snippet.
11. Click *Save*.

The graph snippet is created. You can now import the graph snippet that you created.

## 7.3 Editing Graph Snippets

You can modify the properties of existing graph snippets.

### Procedure

1. Start the SAP Data Intelligence Modeler.
2. In the navigation pane, select the *Repository* tab.
3. Navigate to the JSON file of the graph snippet, which is stored under `templates/graphsnippets`.
4. To edit in JSON editor:
  - a. Right-click the JSON file and choose *Open JSON Editor* option in the context menu.
  - b. Modify the existing properties of the graph snippet and click *Save*.
5. To edit in graphical editor:
  - a. Double-click the JSON file to open the graph snippet dialog.
  - b. Modify the existing properties of the graph snippet and choose *Save*.

## 8 Working with the Data Workflow Operators

SAP Data Intelligence Modeler has a category of operators called Data Workflow operators. When used in a graph (pipeline) and executed, the Data Workflow operators run for a limited time and finish with the status of either “completed” or “dead”.

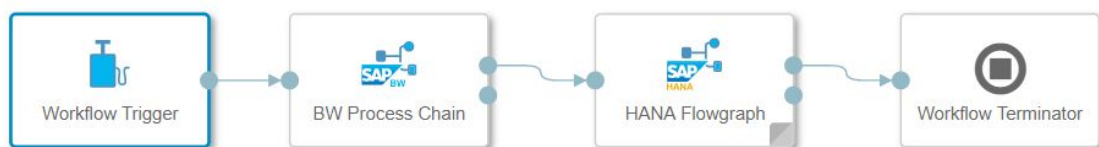
Graphs with Data Workflow operators are known as data workflows. The operators in a data workflow communicate through signals that are transferred at their input and output ports. All Data Workflow operators, except for the Workflow Trigger and Workflow Terminator operators, have an input, an output, and an error port. Thus, the operator begins to execute only when it receives a signal at its input port. The Modeler begins the execution of other connected operators in the data workflow only after the previous operator has finished its execution.

### i Note

SAP recommends that you don't model graphs that have both Data Workflow operators and non-Data Workflow operators.

### Example

In the following data workflow, the Workflow Trigger operator sends a start execution signal to the BW Process Chain operator. Only after the BW Process Chain operator has completed successfully does the SAP HANA Flowgraph operator start to run.



### i Note

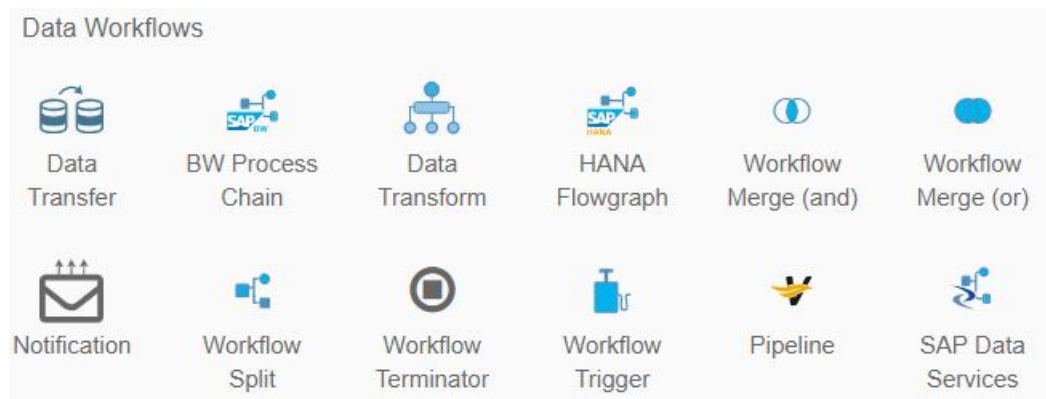
If the graph sends an outgoing signal to the output port, but the output port isn't connected, the graph execution fails.

## Data Workflow Operators

Access the Data Workflow operators in the [Operators](#) tab in the Modeler navigation pane. The Modeler groups these operators under the Data Workflows category. The following Data Workflow operators are supported in the Modeler.

## i Note

For a description of each operator, hover your mouse over each operator icon, or consult the table after the image.



- [Transfer Data from SAP BW to Cloud Storage \[page 146\]](#)
- [Run an SAP BW Process Chain Operator \[page 137\]](#)
- [Run a HANA Flowgraph Operator \[page 139\]](#)
- [Control Flow of Execution \[page 156\]](#)
- [Control Flow of Execution \[page 156\]](#)
- [Send E-Mail Notifications \[page 157\]](#)
- [Control Flow of Execution \[page 156\]](#)
- [Workflow Trigger and Workflow Terminator \[page 136\]](#)
- [Workflow Trigger and Workflow Terminator \[page 136\]](#)
- [Run an SAP Data Intelligence Pipeline \[page 141\]](#)
- [Run an SAP Data Services Job \[page 143\]](#)
- 

Supported Data Workflow Operators

Operator	Description
BW Process Chain	Executes an SAP Business Warehouse process chain in a remote SAP BW system.
Data Transfer	Transfers data from an SAP BW or SAP HANA system to supported cloud storages.
Data Transform	Provides a variety of options for data transformation.
SAP HANA Flowgraph	Executes an SAP HANA flowgraph in a remote SAP HANA system.
Notification	Sends e-mail notifications.

Operator	Description
Pipeline	Executes an SAP Data Intelligence graph on either a remote system or the local system.
SAP Data Services Job	Executes an SAP Data Services job in a remote SAP Data Services system.
Workflow Merge (and)	Combines the output from two Data Workflow operators in a "logical AND".
Workflow Merge (or)	Combines the output from two Data Workflow operators in a "logical OR".
Workflow Split	Duplicates the incoming signal from a Data Workflow operator.
Workflow Terminator	Terminates the current data workflow graph.
Workflow Trigger	Sends a start signal to trigger the execution of a data workflow graph.

## Modeling Data Workflows

A data workflow must have a Workflow Trigger operator as the first operator and a Workflow Terminator operator as the last operator.

- The Workflow Trigger operator sends the start execution signal to the connected operator.
- The Workflow Terminator operator shuts down the data workflow graph.

If you're using operators to perform actions in remote systems, then first create a connection to the remote system using the SAP Data Intelligence Connection Management application. For more information on how to create a connection, see the *Administration Guide*.

## Execution Logic and Data Workflow Status

Each operator in a data workflow starts running after receiving a signal at its input port. After the operator runs successfully, it sends a signal to the connected operator through its output port. If the operator doesn't complete successfully, it sends a signal to its error port.

If the port to which the message is being sent is connected to another Data Workflow operator, the operator that receives the signal begins running. If the port to which the signal is being sent isn't connected to another operator, the overall execution of the data workflow stops with the status of "dead".

### → Tip

To design the data workflow execution to fail when any operator execution fails, model the data workflow so that the respective error ports aren't connected.

## Importing Certificates for Remote Systems

For operators that use HTTPS as the underlying transport protocol (using TLS transport encryption), the upstream system must have a trusted certificate. To import a certificate into the trust chain, obtain the certificates from the target system and import them using the SAP Data Intelligence Connection Management application. The next execution of the graph involving the HTTPS connection picks up any certificate that is present in the trust chain automatically. For more information on how to create a connection, see the *Administration Guide*.

### → Remember

SAP Data Intelligence assumes that the imported certificates are for all of its applications. SAP Data Intelligence doesn't restrict the certificates for use only for the Data Workflow operators.

### i Note

Adding any certificate overwrites the default chain of trust. Thus, the engine can require that you add further certificates for the existing graphs to continue working.

## Related Information

[Workflow Trigger and Workflow Terminator \[page 136\]](#)

[Run an SAP BW Process Chain Operator \[page 137\]](#)

[Run a HANA Flowgraph Operator \[page 139\]](#)

[Run an SAP Data Intelligence Pipeline \[page 141\]](#)

[Run an SAP Data Services Job \[page 143\]](#)

[Transfer Data \[page 146\]](#)

[Control Flow of Execution \[page 156\]](#)

[Send E-Mail Notifications \[page 157\]](#)

[Using SAP Data Intelligence System Management](#)

[Create a Connection](#)

[Manage Certificates](#)

## 8.1 Workflow Trigger and Workflow Terminator

The operators *Workflow Trigger* and *Workflow Terminator* control the beginning and ending of a data workflow graph (pipeline).

Use the *Workflow Trigger* and *Workflow Terminator* operators only with the other operators in the Data Workflow category. The following table contains descriptions for each operator.



Operator	Description
<i>Workflow Trigger</i>	<p>Sends a start message to the next operator in the graph, which starts the data workflow graph. The <i>Workflow Trigger</i> operator has one output port; it doesn't have an input port.</p> <p>Once the graph run is started, the <i>Workflow Trigger</i> operator sends a message to its output port. If the output port is connected to another Data Workflow operator, the next operator starts its execution. However, if the <i>Workflow Trigger</i> output port isn't connected to a Data Workflow operator, the execution of the data workflow graph fails.</p>
<i>Workflow Terminator</i>	<p>Shuts down, or terminates, the execution of the data workflow graph. The <i>Workflow Terminator</i> operator has one input port; it doesn't have an output port.</p> <p>Once the <i>Workflow Terminator</i> receives a message from the upstream operator to its input port, it terminates the data workflow graph with the state, <i>Completed</i>.</p>

## 8.2 Run an SAP BW Process Chain Operator

To run (execute) an SAP Business Warehouse (BW) process chain in an SAP BW system, use the BW Process Chain operator in the SAP Data Intelligence Modeler application.

### Prerequisites

Create a connection to an SAP BW system using the SAP Data Intelligence Connection Management application.

### Context

The SAP BW Process Chain operator contains a sequence of processes that are scheduled to wait in the background for an event. Some of these processes can trigger a separate event that can, in turn, start other processes.

In the SAP BW Process Chain operator, define the start condition of a process chain with the start process type. All other processes in the chain are scheduled to wait for an event. These processes are connected using events that are triggered by an upstream process to start a downstream process.



## Procedure

1. Log into SAP Data Intelligence and select the Modeler tile.
2. Open the *Graphs* tab in the navigation pane at left.
3. Select **+** (*Create Graph*) from the navigation pane toolbar, and select *Use Generation 1 Operators*.

The Modeler opens an empty graph editor at right, and opens the *Operators* tab in the navigation pane.

4. Select the operator by performing the following substeps:
  - a. Enter "BW Process Chain" in the search bar.
  - b. In the search results, double-click the *BW Process Chain* operator.

The Modeler adds the *BW Process Chain* operator to the graph editor workspace.

5. Configure the *BW Process Chain* operator by performing the following substeps:
  - a. Select the *BW Process Chain* operator in the graph editor, and choose  (*Show Configuration*).
  - b. Select  (*Edit Property*) in the *SAP BW Connection* option.
  - c. Enter a connection ID in *Connection ID* that references a connection to a remote SAP BW system, and choose *Save*.

To manually enter the connection details to the remote system, select *Manual* in the *Configuration Type* list and enter the required connection details in *Connection ID*.

- d. Select an SAP BW process chain from the *SAP BW ProcessChain ID* list to run it in the remote system.

The Modeler populates the list based on the connection ID.

### ! Restriction

When the *Configuration Type* for the *SAP BW Connection* is *Manual*, you must enter the *SAP BW ProcessChain ID* manually.

- e. **Optional:** Enter the request timeout in seconds in *Request Timeout (seconds)*.  
The engine waits the set number of seconds for receiving a response from the BW system. The default value is 480 seconds.
- f. **Optional:** Enter the time interval in seconds in *Retry Interval*.  
The engine waits the set number of seconds until checking the status update of the BW Process Chain. The default value is 20 seconds.
- g. **Optional:** Enter the maximum number of attempts in *Retry Attempts*.  
The engine queries the status update for the set number of attempts. The default value is 1080 attempts.

### i Note

The Modeler uses only the value of *Retry Interval* if the BW system sends a successful response (status code 200) when the Modeler checks the status of the executed BW Process Chain.

For long-running Process Chains, in cases when the Process Chain doesn't complete or fails, the maximum time limit for polling the status is 14 days.

If the Modeler receives a response code other than 200, it uses the value set in *Retry Attempts* along with the setting in *Retry Interval* to check the status of the BW Process Chain.

The operator execution fails and the Modeler sends a message to the error output port under the following circumstances:

- The engine doesn't receive a response from the BW system within the number of seconds specified in *Request Timeout (seconds)*.
- The engine reaches the maximum time limit (14 days), which means that the status of the SAP HANA Flowgraph is other than “completed”, “failed” or “canceled”.
- The engine exceeds the number of seconds set in *(Retry Interval) \* (Retry Attempts)*, which means that the BW system never returned a response with status code 200 when it checked the status update of the Process Chain.

6. Save and run the graph.

You can control the start and stop of the graph execution using the Workflow Trigger and Workflow Terminator operators respectively.

→ Tip

You can also schedule the graph execution. For more information, see [Schedule Graph Executions \[page 84\]](#).

## Related Information

[Working with the Data Workflow Operators \[page 133\]](#)

[Create a Connection](#)

## 8.3 Run a HANA Flowgraph Operator

To run (execute) an SAP HANA Flowgraph in an SAP HANA system, use the HANA flowgraph operator in the SAP Data Intelligence Modeler application.

### Prerequisites

Before you perform the following process, ensure that you first create a connection to an SAP HANA system using the SAP Data Intelligence Connection Management application.

### Context

Executing a HANA Flowgraph operator in a graph helps you transform data from a remote source into SAP HANA either in batch or real-time mode. The flowgraphs are predefined in the server and represent a complete data flow. Select the required SAP HANA flowgraph, and all the operations that the flowgraph must perform are defined in the server.

## i Note

The Modeler supports executing only XSC-based HANA flowgraphs.

## Procedure

1. Log into SAP Data Intelligence and select the Modeler tile.
2. Open the *Graphs* tab from the navigation pane at left.
3. Select **+** (*Create Graph*) from the navigation pane toolbar, and select *Use Generation 1 Operators*.

The Modeler opens an empty graph editor at right, and opens the *Operators* tab in the navigation pane.

4. Select the operator by performing the following substeps:
  - a. Enter "HANA Flowgraph" in the search bar.
  - b. Double-click the *HANA Flowgraph* operator in the search results.

The Modeler adds the *HANA Flowgraph* operator to the graph editor workspace.

5. Double-click the *HANA Flowgraph* operator in the graph editor.
6. Complete the *Operator Details* group of options by performing the following substeps:
  - a. Enter the required connection ID in *Connection*.

Alternately, browse and select the required connection.

- b. Browse and select the required SAP HANA flowgraph in *SAP HANA Flowgraph*.
- c. **Optional:** Enter a time interval in seconds in *Retry Interval*.

The *Retry Interval* specifies the time in seconds for the engine to wait until the next status update. The default value is 20 seconds.

- d. **Optional:** Enter a value in *Retry Attempts*.

The *Retry Attempts* specifies the number of attempts to query for the status update. The default value is 10 attempts.

## i Note


If the BW system sends a successful response (status code 200) when the Modeler checks the status of the executed BW Process Chain, the Modeler uses only the value of *Retry Interval*. For long-running Process Chains that don't complete or they fail, the maximum time limit for polling the status is 14 days. If the Modeler receives a response code other than 200, it uses the value of *Retry Attempts* with *Retry Interval* to check the status of the BW Process Chain.

The operator execution fails and the Modeler sends a message to the error output port under the following circumstances:

- A response from the BW system isn't received within the number of seconds specified in *Request Timeout (seconds)*.
- The maximum time limit (14 days) is reached, which means that the status of the SAP HANA Flowgraph is other than "completed", "failed" or "canceled".
- The number of seconds set in  $(\textit{Retry Interval}) * (\textit{Retry Attempts})$  is exceeded, which means that the BW system never returned a response with status code 200 when it checked the status update of the Process Chain.



7. **Optional:** Update the *Variables* section by performing the following substeps:

In the *Variable* section, the Modeler displays the variables associated with the SAP HANA flowgraph, its data type, and default value. You can edit the default value of a variable.

- a. Select  (*Edit*) in the *Variables* section.
- b. Enter new variable values as applicable.

8. **Optional:** Update the *Table Variables* section by performing the following substeps:

In the *Table Variables* section, the Modeler displays the table variables associated with the SAP HANA flowgraph, its data type, and default value. You can edit the default value of a table variable or define new table variables.

- a. Enter (or select) the name of the table variable in the *Table Variables* section under the *Name* column.
- b. Select  (*Edit*) in the *Value* text field and provide the required variable value.
- c. Select  (*Add New*) to define a new table variable for the selected SAP HANA flowgraph.

9. Save and run the graph.

You can control the start and stop of the graph execution using the Workflow Trigger and Workflow Terminator operators respectively.

#### → Tip

You can also schedule the graph execution. For more information, see [Schedule Graph Executions \[page 84\]](#).

## Related Information

[Working with the Data Workflow Operators \[page 133\]](#)

[Create a Connection](#)

## 8.4 Run an SAP Data Intelligence Pipeline

Use the Pipeline operator in the SAP Data Intelligence Modeler application to run (execute) a graph (pipeline) in an SAP Data Intelligence system.



### Context

Use the Pipeline operator to run a graph in a remote SAP Data Intelligence system or in a local system. A graph represents a concrete and complex data flow and helps transform data between elements connected in a series. When you run a graph, it can help process the raw data from multiple sources and make the data available for different use cases.

## Procedure

1. Log into SAP Data Intelligence and select the Modeler tile.
2. Open the *Graphs* tab in the navigation pane at left.
3. Select **+** (*Create Graph*) from the navigation pane toolbar, and select *Use Generation 1 Operators*.  
The Modeler opens the graph editor in the workspace and opens the *Operators* tab automatically.
4. Enter "Pipeline" in the search bar and double-click the Pipeline operator in the search results.

The Modeler adds the Pipeline operator to the graph editor workspace.

5. Configure the operator by performing the following substeps:
  - a. Select  (*Open Configuration*) to the right of the Pipeline operator in the workspace.
  - b. Select  (*Edit Property*) in the *VFlow Connection* box in the *Configuration* pane.

The *Edit Property* dialog box opens.

- c. Enter a connection ID in one of the following ways:
  - If you've already created connections in the SAP Data Intelligence Connection Management application, select the dropdown arrow at the end of the *Connection ID* option and choose the required connection.
  - To enter the connection manually, first choose *Manual* from the *Configuration Type* list.

### i Note

Providing connection details for the Pipeline operator is necessary only to run a graph in a remote SAP Data Intelligence system. If you don't provide a value, you can run only a pipeline from the local system. A pipeline from the local system is one that you're working on currently in the SAP Data Intelligence Modeler.

- d. Select *Save*.

The *Edit Property* dialog box closes.

- e. Choose the SAP Data Intelligence graph (pipeline) to run from the *Graph Name* list in the *Configuration* pane.

The Modeler populates the list with all graphs in the remote system, based on the connection ID, or all graphs available in the local system.

- f. **Optional:** Enter the time interval in seconds in *Retry Interval*.

The engine waits the set number of seconds until the next status update. The default value is 20 seconds.

- g. **Optional:** Enter the maximum number of attempts in *Retry Attempts*.

The engine queries for the status update for the set number of attempts. The default value is 10 attempts.

### i Note

If the Pipeline operator exceeds the settings for both *Retry Interval* and *Retry Attempts*, the graph run fails and the Pipeline operator sends a message to the error output port. The Pipeline operator exceeds the settings when it has requested a status update for the set number of times in *Retry Attempts* and the Modeler hasn't executed the graph, or the graph executed with errors.

- h. Choose a value from the *Running Permanently* list to indicate whether the selected SAP Data Intelligence graph is running permanently.

The following table describes the values for *Running Permanently*.

Value	Description
<i>true</i>	The operator execution checks whether the graph is in a running state. If the graph isn't running, the operator starts the graph execution and terminates immediately (status: completed), while the graph remains in the running state. But, if the task is already running with a different task version, then the already-running instance is stopped, and the new task version is started and then terminated immediately (status: completed).
<i>false</i>	The operator runs the graph once, and the operator execution terminates after the graph execution terminates. <i>False</i> is the default value.

- 6. Save and run the graph.

You can control the start and stop of the graph execution using the Workflow Trigger and Workflow Terminator operators respectively.

→ Tip

You can also schedule the graph execution. For more information, see [Schedule Graph Executions \[page 84\]](#).

## Related Information

[Working with the Data Workflow Operators \[page 133\]](#)

[Create a Connection](#)

## 8.5 Run an SAP Data Services Job

Use the SAP Data Services Job operator in the SAP Data Intelligence Modeler application to run (execute) an SAP Data Services job in a remote system.

### Prerequisites

Create a connection to an SAP Data Services system using the SAP Data Intelligence Connection Management application.

## Context



Running an SAP Data Services job helps you integrate, transform, and improve the data quality. In SAP Data Services, the unit of execution is called a job. Running an SAP Data Services job in SAP Data Intelligence helps you in the following ways:

- Ingesting data into a Hadoop cluster for further processing (natively in a Hadoop cluster).
- Moving data out of SAP Data Intelligence after processing.

## Procedure

1. Log into SAP Data Intelligence and select the *Modeler* tile.
2. Open the *Graphs* tab in the navigation pane at left.
3. Select **+** (*Create Graph*) from the navigation pane toolbar, and select *Use Generation 1 Operators*.  
The Modeler opens the graph editor in the workspace and opens the *Operators* tab in the navigation pane automatically.
4. Select the SAP Data Services Job operator by performing the following substeps:
  - a. Enter "SAP Data Services" in the search bar.
  - b. Double-click the SAP Data Services Job operator under the Remote DataFlow group in the search results.

The Modeler adds the Pipeline operator to the graph editor workspace.

5. Configure the operator by performing the following substeps:
  - a. Select  (*Open Configuration*) to the right of the SAP Data Services Job operator in the workspace.
  - b. Select  (*Edit Property*) in the *SAP Data Services Connection* box in the *Configuration* pane.
  - c. Browse for and choose a connection in *Connection ID*.
  - d. Enter a description for the operator in *Description*.
  - e. Browse for and select the required SAP Data Services job in *Job*.

The Modeler populates the *Repository* field automatically with the repository that contains the SAP Data Services job. The Modeler also displays the global variables and the substitution parameters (if any) that are associated with the job in the Global Variables table.

- f. Choose the required Job Server from the *Job Server* list.

The Modeler uses the selected Job Server to run the job.

### Note

You can also use a Job Server group to run the job. If you select a Job Server group, specify the distribution level (job level, data flow level, or sub data flow level).

- g. Choose the required system configuration details from the *System Configuration* list.



The Modeler uses the system configuration for running the job.

6. **Optional:** Edit global variables by performing the following substeps:  
If the selected job is associated with global variables, in the *Global Variables* section the Modeler displays the global variables, its data type, and default value. You can edit the default value of a global variable associated with the job.



- a. Select  (*Edit*) in the *Global Variables* box.

The Modeler populates values in the *Global Variables* section depending on the version of the SAP Data Services system in which you've created the selected job.

- b. Choose the applicable global variable and select  (*Edit*) in the *Value* text field and provide the required value.
- c. To define a new global variable for the SAP Data Services job, in the *Global Variables* section, choose  (*Add*) and provide the variable and value.

#### Note

Adding new global variables isn't applicable for all Data Services jobs. It depends on the version of the SAP Data Services system in which you've created the selected job.

7. **Optional:** Edit substitution parameters in the same manner as global variables.

In the *Substitution Parameters* section, define the substitution parameters associated with the job, its data type, and default value. You can edit the default value of a substitution parameter or define new substitution parameters.

#### Note

Substitution parameters that the application displays in the dropdown list are the parameters that are associated with the selected system configuration.

#### Note

The Modeler doesn't always auto-populate the global variables or substitution parameters. It depends on the version of the SAP Data Services system in which you've created the selected job. If the global variables aren't auto-populated, enter the information manually.

8. Save and run the graph.

You can control the start and stop of the graph execution using the Workflow Trigger and Workflow Terminator operators respectively.

#### Tip

You can also schedule the graph execution. For more information, see [Schedule Graph Executions \[page 84\]](#).

## Related Information

[Working with the Data Workflow Operators \[page 133\]](#)

[Create a Connection](#)

## 8.6 Transfer Data

The Data Transfer operator allows you to transfer data from specific source systems to specific target systems.

Use the Data Transfer operator in SAP Data Intelligence Modeler in a data workflow for the following tasks:

- Transferring data from SAP Business Warehouse or SAP HANA.
- Loading data to cloud storages, such as Amazon S3, Google Cloud Platform, and Hadoop Distributed File System.

Before you use the Data Transfer operator in the data workflow, create connections to source and target systems using the SAP Data Intelligence Connection Management application. When you configure the Data Transfer operator, browse and select the source and target connections.

### Related Information

[Transfer Data from SAP BW to Cloud Storage \[page 146\]](#)

[Transfer Data from SAP HANA to Cloud Storage \[page 153\]](#)

### 8.6.1 Transfer Data from SAP BW to Cloud Storage

Use the *Data Transfer* operator in a data workflow graph (pipeline) to transfer data from an SAP Business Warehouse (BW) system to cloud storage.

#### Prerequisites

Before you perform the following task, ensure that you've created the following connections using the SAP Data Intelligence Connection Management application:

- Create a connection to an SAP BW system.
- Create a connection to a cloud storage.

#### i Note

SAP Data Intelligence displays decimal numbers from the source with the same scale as shown in other SAP tools, such as SAP Logon or SAP BW Modeling tools. SAP Data Intelligence represents the data in its raw form as generated by the SAP BW system and rounds it down to the nearest number based on the SAP BW backend scale.

## Context

In the Modeler, configure and execute the Data Transfer operator in a graph to transfer data from SAP BW to cloud storage. Run the data workflow using three access layers.

## Procedure

1. Start the SAP Data Intelligence Modeler.
2. Open the *Graphs* tab in the navigation pane.
3. Select **+** (*Create Graph*) in the navigation pane toolbar and choose *Use Generation 1 Operators*.
4. Select the Data Transfer operator.

A graph can contain a single operator or a network of operators based on the business requirement.

- a. Open the *Operators* tab in the navigation pane.
- b. Enter "Data Transfer" in the search bar in the navigation pane toolbar.
- c. Double-click the Data Transfer operator in the search results.

The Modeler adds the Data Transfer operator to the graph editor.

- d. Double-click the Data Transfer operator in the graph editor.

The Modeler opens a form-based editor where you define the source and target for data transfer

5. Provide details of the source dataset for the data transfer operation.
  - a. Open the *Source* tab.
  - b. Browse for or enter a connection ID in the *Connection ID* text box.

A connection ID provides the connection to an SAP BW system.

- c. Browse for or enter the source in the *Source* text box.

If you browse for the source, the *Browse File* dialog box contains all queries, InfoProviders, or DataStores from the SAP BW system used in the connection ID definition. Source types include the following:


- Queries
- InfoProviders
- DataStores


6. **Optional:** Import source dataset (Queries, InfoProviders, or DataStores) from the Metadata Catalog.
  - a. Select *Import Dataset* in the *Source* tab.
  - b. Browse and select the required dataset.
  - c. Select *OK*.

The Modeler populates the connection details automatically based on the selected data set.

7. If you select a query as the source dataset, and if the query is defined with parameters, provide values to the parameters.

The Modeler populates the default values automatically, if any, that are already defined for the parameters.

- a. Select  (*Edit*) in the *Variables* text field.
- b. Select the required parameter and operator type, and provide a value in the *Provide Parameter Values* dialog box.

- c. **Optional:** To view the mandatory parameters only, select  (*Add Filter*) and select *Show Mandatory Only*.
- d. Select *OK*.

8. **Optional:** Specify the transfer mode.

SAP Data Intelligence supports three types of transfer modes. The preferred mode is to use SAP HANA.

- a. If the engine uses the SAP BW OLAP (Online Analytical Processing) Interface Information Access Protocol (INA) to retrieve data from the source, enter a value in milliseconds in *Timeout*  
The Modeler waits for the time that you specify before it times out on the data retrieval. After the timeout period, the graph execution fails. The default timeout value is 60 seconds.
- b. To use an SAP HANA view, mark a specific query or InfoProvider in the query designer to generate an underlying Calculation view. SAP Data Intelligence can query this view to transfer the data to other target systems.

If an external SAP HANA view exists, the Modeler displays the name of the SAP HANA view that it's using to retrieve data from the source.

→ Remember

The engine uses the nonoptimized SAP BW OLAP (INA) provider to retrieve data from the source only if no external SAP HANA view exists.

If you select a DataStore, then the application by default uses SAP BW Operational Data Provisioning (ODP) as the Data Access configuration. In this case, you can't modify the access configuration.

9. **Optional:** If you use SAP BW DataStore as a source, perform the following substeps:

- a. Choose an extraction mode.  
Choose from one of the following extraction modes:
  - *Full*: Extracts all data at once.
  - *Delta*: Extracts only what has changed with each run of the graph.
- b. If you choose *Full*, define a file as the target.

The application supports two modes:

Mode	Description
<i>Overwrite</i>	Choose a file or a file name in the target file system. For each run, the Modeler overwrites the selected file.
<i>Create for each package</i>	Choose a file or pick a file name in the target file system. For each run, the Modeler writes a new file for each data package.

i Note

The Modeler extracts the full data when you run the graph for the first time. After the first run, the Modeler extracts only delta changes for all subsequent runs.

- c. Provide a subscription ID.  
Consider the following information about subscription IDs:
  - The ID must be unique for the same InfoProvider in the same SAP BW system for all clients accessing it.

- For Full extraction mode, the system generates a subscription ID automatically.
- The ID serves as the session for the delta extraction and stores the delta pointer. It ensures that just the changed data is transferred.
- Write the subscription ID so that you can later look it up on your subscriptions.

10. Select the required measures and dimensions from the source dataset to project to the target.

a. Open the *Target* tab.

The Modeler displays all measures and dimensions from the selected source in the *Column Mapping* section.

b. Drag and drop applicable source columns from the Source pane to the Target pane.

### **i** Note

When you map columns from source to target, or if there are pre-existing target objects that have the data types listed below, the Modeler reorders the columns in the target pane automatically based on those data types regardless of the target type.

The columns with the following data types appear at the end of the list in the target pane:

- BLOB
- CLOB
- NCLOB
- BINARY
- VARBINARY
- TEXT
- BINTTEXT

11. Apply filter conditions on dimensions in the source dataset, and project only the filtered values.

a. Open the *Source* tab.

b. Select  (*Edit Filters*) in the *Filters* text box.

### → Remember

You can define filters only after you map columns to the target.

c. Select the dimension, and define the filter condition in the *Provide Filter Values* dialog box.

d. Select *OK*.

12. **Optional:** Define partitions.

To optimize the data transfer operation for SAP HANA views, the Modeler provides capabilities to define a maximum of two partition conditions for columns in the source dataset. It supports the following partition types to define the partition condition: List and Range.

a. Choose *Add Condition* in the *Partition Conditions* section.

b. Select the required partition column and its data type.

c. Select the required partition type from the *Type* list.

d. Define one or more partition values in the *Partition Values* text box.

For range partition type, define only the low boundary value.

### i Note

If data is retrieved using the SAP BW OLAP Interface provider (INA) as the transfer mode, then the Modeler ignores the partitions.

13. Define the cloud storage target in the *Target* pane.

To use any of the supported cloud storages as the target dataset for data transfer, provide details about the required cloud storage.

- a. Enter or browse for a connection ID that provides a connection to the required cloud storage in the *Connection ID* text box.
- b. Enter or browse for the path of the file to which the graph transfers the data in the *Target* text field.

If the selected connection has a root path specified in the connection definition, then the content of this field is relative to the path.

### i Note

The Modeler supports CSV, ORC, or Parquet file formats.

- c. Define a file as a target for *Delta* extraction mode.

The application supports three modes to write deltas to files as described in the following table.

Mode	Description
<i>Append</i>	Choose a file or select a file name in the target file system. Each delta run is appended to the selected file.
<i>Create</i>	Choose a folder in the target file system. Each delta run creates a new file in the selected folder.  In <i>Create</i> mode, you must define the <code>fileNamePattern</code> property that tells how the delta files are to be created inside the folder.
<i>Create for each package</i>	Choose a file or pick a file name in the target file system. Each delta run writes a new file for each data package.

### ❖ Example

`Customer-<date>-<time>`, where, Customer is the name of the source dataset.

### i Note

SAP Data Intelligence supports only the `.csv` file format for delta write mode.

- d. **Optional:** Select *Fetch Metadata* only if you manually entered the file path in the top right of the editor. The *Fetch Metadata* functionality helps to fetch the metadata (schema) from the selected source or target and populates the column details accordingly in the Modeler.

14. Map source and target columns.

- a. Open the *Target* tab.

- b. Select a source column from the source and drag it to the target column in the [Column Mapping](#) section.

Use the mapping editor.

15. **Optional:** After you complete the mapping, preview data.

- a. Open the [Source](#) tab.
- b. Select [Data Preview](#).

The following table describes the areas to preview.

Area	Description
<a href="#">Source Data</a>	Preview remote dataset. Enter variable values to see the data preview.
<a href="#">Adapted Data</a>	Preview the selected columns in the <a href="#">Target</a> tab, the variable values that you entered, and the filters that you provided.

- c. Open the [Target](#) tab.
- d. Select [Data Preview](#).

View data similar to data in the [Source](#) tab.

#### **i** Note

When you run the data transfer graph, only the mapped columns are projected to the target dataset.

16. Save and run the graph.

You can control the start and stop of the graph execution using the Workflow Trigger and Workflow Terminator operators respectively.

#### → Tip

You can also schedule the graph run. For more information, see [Schedule Graph Executions \[page 84\]](#).

## Related Information

[Transfer Modes \[page 151\]](#)

[Working with the Data Workflow Operators \[page 133\]](#)

[Create a Connection](#)

### 8.6.1.1 Transfer Modes

The Data Transfer operator in SAP Data Intelligence supports different modes for retrieving data.

Use the Data Transfer operator to retrieve data using any of the following modes:

- SAP Business Warehouse (BW) OLAP. (Use the OLAP Interface (INA) protocol as the access method.)
- Generated SAP HANA views.
- SAP BW ODP for full or delta data extraction.

For all other cases, the preferred way is to use SAP HANA. To use SAP HANA, mark a specific query, infoprovider, or datastore in the query designer to generate an underlying calculation view. SAP Data Intelligence queries this view to transfer the data to other target systems.

### i Note

The application processes the partition result sets one after the other and not in parallel.

## SAP BW OLAP

SAP BW OLAP is the default mode. SAP BW uses OLAP to access data. The OLAP method ensures that the data you write matches the data from the common SAP BW user interfaces, such as Transaction RSRT2. The OLAP method supports a wide variety of SAP BW functionality.

Because of the online nature of the OLAP access method, SAP doesn't recommend it for large-scale data transfer. With the SAP BW OLAP mode, transfer the complete data in a single result set.

### i Note

SAP BW INA has a maximum number of cells that can be exported.

## Generated SAP HANA Views

Generate designated calculation views on the underlying SAP HANA database for each query or infoprovider in SAP BW on SAP HANA and SAP BW/4HANA. SAP Data Intelligence uses generated SAP HANA views when the following conditions are met:

- You create a connection to an SAP BW system using the SAP Data Intelligence Connection Management application.
- You use SAP BW version 4.2.0 and higher.
- You specify a working connection to the SAP HANA database and referenced it in the SAP BW connection that you use.
- You upload two certificates for SSL access in case the two connections can't be covered by the same root certificate.
- You ensure that the corresponding query or infoprovider has a generated calculation view.
- The projection of chosen dimensions and measures doesn't contain a restricted attribute, such as specified in [2145502](#).
- There are no further errors that occur, such as SAP HANA authorization errors.

If data is retrieved with generated SAP HANA views, then you can specify partitions of data. The application transfers partitions of data separately, which enables large result sets.



## SAP BW ODP

You can use the SAP BW ODP mode only with datastores.

### i Note

You must change the setting in the Data Transfer operator manually to activate querying with SAP HANA. The application uses the non-optimised BW INA provider for data extraction in the following cases:

- If SAP HANA isn't available.
- If the application doesn't generate the necessary calculation view.
- If you've selected the INA option explicitly.

If the application uses the non-optimised BW INA provider, it requires that you provide an appropriate timeout for the BW INA call, depending on the amount of data you expect to be extracted.

In the current version, SAP Data Intelligence supports the following file storage targets for data transfer using the SAP BW ODP mode:

- ADL (Microsoft Azure Data Lake)
- GCS (Google Cloud Storage)
- HDFS (Hadoop Distributed File System)
- OSS (Alibaba Cloud Object Storage Service)
- S3 (Amazon Simple Storage Service)
- SDL (Semantic Data Lake)
- WASB (Microsoft Azure Storage Blob)

## 8.6.2 Transfer Data from SAP HANA to Cloud Storage

Use the Data Transfer operator in a graph to transfer data from an SAP HANA system to an SAP Vora system or to cloud storage.

### Prerequisites

- Create a connection to an SAP HANA system using the SAP Data Intelligence Connection Management application.
- Create a connection to a cloud storage using the SAP Data Intelligence Connection Management application.

### Procedure

1. Start the SAP Data Intelligence Modeler.

2. Open the *Graphs* tab in the navigation pane.
3. Select **+** (*Create Graph*) in the navigation pane toolbar.
4. Choose *Use Generation 1 Operators*.
5. Add The Data Transfer operator to a graph.

A graph can contain a single operator or a network of operators based on the business requirement.

- a. Open the *Operators* tab in the navigation pane.
  - b. Enter "Data Transfer" in the search text box in the navigation toolbar.
  - c. Double-click the Data Transfer operator in the search results to add the operator as a process to the graph editor.
6. Define the source for data transfer.
    - a. In the graph editor, double-click the *Data Transfer* operator.
 

The Modeler opens an editor in the graph editor pane where you define the source and target for data transfer.
    - b. Open the *Source* tab.
 

In the *Source* tab, provide details of the source dataset for the transfer operation.
    - c. Enter or browse for the applicable connection ID in the *Connection ID* text box.
 

The connection ID provides the connection to an SAP HANA system.
    - d. Select **🔍** (*Browse*) in the *Table Name* field and select the required SAP HANA table.
 

The Modeler populates the connection details automatically based on the selected data set.
  7. **Optional:** Browse the Metadata Catalog and import the applicable SAP HANA table.
    - a. Select *Import Dataset* in the *Source* tab toolbar.
    - b. Browse for and select the applicable data set.
    - c. Select *OK*.
 

The Modeler populates the connection details automatically based on the selected data set. For more information about the Metadata Catalog, see the *Data Governance User Guide*.
  8. Provide values to parameters.
 

If the selected SAP HANA objects are defined with parameters, then it's necessary to provide values to those parameters. The application automatically populates the default values, if any, that are already defined for the parameters.

    - a. Select the *Edit* icon in the *Variables* text box and provide applicable values.
    - b. Select the variable and operator type, and provide the required values in the *Provide Parameter Values* dialog box.
    - c. **Optional:** To view only required parameters, select **🔍** (*Add Filter*) and select *Show Mandatory Only*.
  9. Select applicable columns from the target dataset to load to the target.
    - a. Open the *Target* tab.
 

The Column Mapping section lists all columns from the selected source.
    - b. Drag and drop applicable columns to the Target pane.

### **i** Note

When you map columns from source to target, or if there are pre-existing target objects that have the data types listed below, the Modeler reorders the columns in the target pane automatically based on those data types regardless of the target type.

The columns with the following data types appear at the end of the list in the target pane:

- BLOB
- CLOB
- NCLOB
- BINARY
- VARBINARY
- TEXT
- BINTEXT

10. **Optional:** Filter column values in the source dataset to load only filtered values to the target.

#### → Remember

You can define filters only after you map columns to the target.

- Open the *Source* tab.
- Select  (*Edit Filters*) in the *Filters* text box.
- Select applicable columns in *Provide Filter Values* dialog and define the filter condition.
- Select *OK*.

11. Define the cloud storage target table.

- Open the *Target* tab.
- Enter or browse for the connection ID for the cloud storage dataset in the *Connection ID* text box.
- Enter or browse for the path to the cloud storage dataset in the *Target* text box.

If the selected connection has a root path specified in the connection definition, then the content of this field is relative to this path.

12. Map source columns to the target columns.

- Open the *Target* tab.
- Drag and drop each source column in the Source pane to the corresponding target column in the Target pane in the Column Mapping section.

13. After you've mapped all of the source columns to the target columns, preview data in the columns.

- Open the *Source* tab.
- Select one of the following options from the *Data Preview* list:
  - *Source Data*: Input variable values to preview the remote dataset.

#### i Note

The variable values that you enter don't overwrite the existing values of the data transfer operator.

- *Adapted Data*: Preview the columns that you selected in the *Target* tab, the variable values, and the filters that you set.
- To view data in the target table columns, open the *Target* tab and select *Data Preview*.

#### i Note

When you run the data transfer, only the mapped columns are loaded to the target dataset.

14. Save and run the graph.

You can control the start and stop of the graph run using the Workflow Trigger and Workflow Terminator operators respectively.

→ Tip

You can also schedule the graph execution. For more information, see [Schedule Graph Executions \[page 84\]](#).

## 8.7 Control Flow of Execution

SAP Data Intelligence provides the data workflow operators Workflow Merge (or), Workflow Merge (and), and Workflow Split to control the flow of execution in a data workflow.

A typical use case for these data workflow operators is parallel executions within a data workflow.

**i** Note

You can connect the input ports of the operators to other data workflow operators. But, if the operator receives a message at any of its input ports, then leaving the output ports unconnected results in a data workflow execution to fail or stop.

Operator	Description
Workflow Merge (or)	<p>A message is sent to the output port once the operator receives a message at any of its input ports. All further inputs are ignored.</p> <p>The operator has two input ports and one output port.</p>
Workflow Merge (and)	<p>This operator sends a message to the output port once it receives a message on both the input ports. The process shuts down after sending the message.</p> <p>The Workflow Merge (and) operator is intended to control the flow of execution when there are parallel executions within the data workflow.</p> <p>The operator has two input ports and one output port. Leaving the output port unconnected results in a data workflow execution failure once the operator receives an input message.</p>
Workflow Split	<p>This operator helps duplicate an input message into two output messages.</p> <p>The Workflow Split operator is intended to control the flow of execution when there are parallel executions within the graph.</p> <p>The operator has one input port and two output ports. The operator sends the message at the input port to both the output ports.</p>

These operators are listed in the SAP Data Intelligence Modeler under the Data Workflow category.

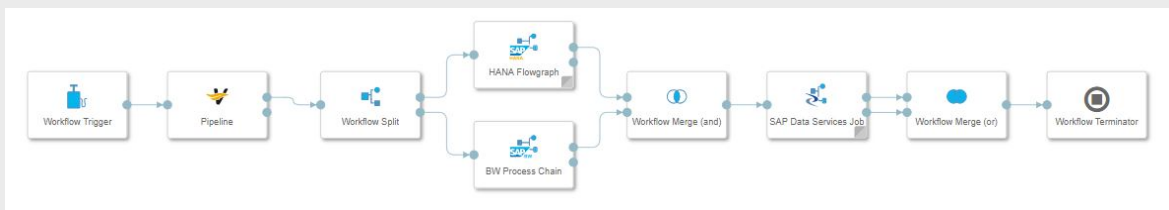
## i Note

Use the operators Workflow Split, Workflow Merge (and), and Workflow Merge (or) operators in a graph with other data workflow operators only.

## Example

### ❁ Example

The following data workflow uses the Workflow Split, Workflow Merge (and), and Workflow Merge (or) operators.



After the engine executes the Pipeline operator successfully, the Modeler duplicates the output message to both the HANA Flowgraph operator and the BW Process Chain operator. Therefore, both the HANA Flowgraph and BW Process Chain operators run in parallel.

The Workflow Merge (and) operator ensures that the data workflow sends a message to the SAP Data Services operator only if both the operators have finished successfully. The Workflow Merge (or) operator takes the first incoming message and forwards it to the Workflow Terminator operator that shuts down the data workflow execution.

## 8.8 Send E-Mail Notifications

Use the Notification operator in the SAP Data Intelligence Modeler to send e-mail notifications to users at certain points during the data workflow execution.


### Prerequisites

Create a connection to an SMTP server using the SAP Data Intelligence Connection Management application.

## Context

A typical use case for using the notification operator is to send an e-mail when an operator writes onto its `error` or `success` port. The message body contains technical information that the notification operators receive at its input port from the previous operator in the data workflow.

## Procedure

1. Start the SAP Data Intelligence Modeler.
2. In the navigation pane, select the *Graphs* tab.
3. In the navigation pane toolbar, choose **+** (*Create Graph*).  
The application opens an empty graph editor in the same window, where you can define your graph.
4. Select the operator.  
A graph can contain a single operator or a network of operators based on the business requirement.
  - a. In the navigation pane, choose the *Operators* tab.
  - b. In the search bar, search for the Notification operator.
  - c. In the search results, double-click the Notification operator (or drag and drop it to the graph editor) to add it as a process in the graph execution.
5. Configure the operator.
  - a. In the graph editor, select the Notification operator and choose  (*Open Configuration*).
  - b. In the *Connection* text field, enter the required connection ID.  
You can also browse and select the required connection.
  - c. If you want to delete the attachments after sending the e-mail, in the *Delete attachments after sending* dropdown list, select *true*.
  - d. In the *Default From Value* text field, enter the value of `email.from` that the Modeler must use when no such value is received through the incoming message.
  - e. In the *Default To Value* text field, enter the value of `email.to` that the Modeler must use when no such value is received through the incoming message.
  - f. In the *Default Subject Value* text field, enter the value of `email.subject` that the modeler must use when no such value is received through the incoming message.

### **i** Note

The following characters aren't supported in message header names: `<`, `>`, `$`, and `{}`.

# 9 Working with Structured Data Operators

You can model a flow and build a transformation pipeline which will move data from the source(s) to the target(s) while transforming the data in the process without having to create custom operators.

## Related Information

[Data Transform \[page 159\]](#)

[Structured Consumer Operators \[page 171\]](#)

[Structured Producer Operators \[page 175\]](#)

[Custom Editor \[page 178\]](#)

[Resiliency with Structured Data Operators \[page 179\]](#)

## 9.1 Data Transform

The Data Transform operator in the SAP Data Intelligence Modeler provides wide variety of options to meet your data transformation needs.


### Context

This operator allows you to perform data transformations, such as projection, union, aggregation, and join. It can connect to other operators from the Structured Data Operators category as sources and targets. There are nodes available in the operator that provide capabilities to meet your data transformation requirements. For example, you can use these nodes to create projections and joins. Configure each node to meet your individual data specifications.


### Procedure

1. Start the SAP Data Intelligence Modeler.
2. In the navigation pane, select the *Graphs* tab.
3. Choose **+** (Create Graph).

The application opens an empty graph editor in the same window where you can define your graph. A graph can contain a single operator or a network of operators based on the business requirement.

- In the *Operators* tab, select any consumer operator. For example, double-click the *Structured File Consumer* operator (or drag and drop it to the graph editor) to add it as a process in the graph execution.
- Select the operator and choose  (Open Configuration) to define the required configurations to the operator.

### **i** Note

- For consumer operators (Structured File Consumer, Table Consumer, and Application Consumer) you can use the *Data Preview* option in the configuration panel to view the content of the selected file or table. If the table is very wide or contains a number of large column types, the result may be truncated in order to avoid out of memory issues.
- The file browsing and data preview is not possible if substitution parameters are used to configure the connection ID or the table name properties.  
For CSV files, you can modify the CSV properties using  *Data Access Configuration* and view data accordingly.

- Add the Data Transform operator to the graph.
- Connect the output port of the *Structured File Consumer* operator to the *Data Transform* operator.
- In the graph editor, double-click the *Data Transform* operator.
- In the *Nodes* tab, drag and drop the required node to the operator editor.  
The Data Transform operator provides different nodes that you can use to define your data transformation requirements.

Node	Description
Projection	Represents a relational selection (filter) combined with a projection operation. It also allows calculated columns to be added to the output.
Join	Represents a relational multiway join operation. It supports multiple input ports.
Aggregation	Represents a relational group-by and aggregation operation.
Union	Represents a relational union operation. It supports multiple input ports.
Case	Specifies multiple paths so that the rows are separated and processed in different ways.

- Double-click the node and define the required node configurations. For more information on configuring the data transform nodes, see the Related Information section.
- In the menu bar, use the breadcrumb navigation to navigate back to the operator configuration editor.
- Add any producer operator. For example, add the Structured File Producer operator to the graph editor.
- Connect the output port of the *Data Transform* operator to the *Structured File Producer* operator.
- Define the required configurations to the Structured File Producer operator.
- To control the start and stop of the graph execution, add the Workflow Trigger and the Graph Terminator operators at the beginning and at the end of the graph, respectively.
- Save and execute the graph.



### → Tip

You can also schedule the graph execution. For more information, see [Schedule Graph Executions \[page 84\]](#).

## Related Information

[Configure the Projection Node \[page 161\]](#)

[Configure the Join Node \[page 163\]](#)

[Configure the Aggregation Node \[page 167\]](#)

[Configure the Union Node \[page 169\]](#)

[Configure the Case Node \[page 170\]](#)

## 9.1.1 Configure the Projection Node

A Projection node represents a relational selection (filter) combined with a projection operation. It also allows calculated attributes to be added to the output.

### Prerequisites

You have defined the operator with a Projection node and connected the previous node to this node.

### Procedure


1. Double-click the *Projection* node.
2. Define the output columns.

If you have connected the previous node to the Projection node, you can map any columns from the input as output columns of the Projection node. You can also add, delete, and rename the columns, as needed.

- a. In the *Mapping* pane, select a *Source* column and drag and drop into the *Drop row here* zone in the *Target* section.

#### i Note


You can also duplicate an output column (map the same source column more than once). Select the required *Source* column and drag and drop into the *Drop row here* zone in the *Target* section.

- b. **Optional:** To automatically map the columns based on column names, in the *Source* section, choose  (Auto Map by Name).

- c. **Optional:** To add new output columns, in the *Target* section, choose + (Add Column) to add new rows and define its data type.

### i Note

For all new output columns, define the output column with a column expression or with a mapping to any of the existing source columns.


- d. **Optional:** To use an expression to define an output column, click the required *Target* column and then define the output column expression in the expression editor.
  - e. **Optional:** To remap an output column with a different source column, right-click the mapping, choose *Remap*, select a new source column, and choose *OK*.
  - f. **Optional:** To edit a column name or its data type, select a *Target* column and choose  (Edit).
  - g. **Optional:** To delete a mapping without deleting the output column, right-click the mapping and choose *Delete*.
3. **Optional:** Define additional configurations for output columns.

In the *Columns* tab toolbar, switch to the *Form* pane to define additional configurations for output columns.

- a. If you have identical records from the previous node, in the toolbar, select the *Distinct* checkbox to output unique records only.

### → Remember

The duplicate records must match exactly; similar records are included to the output. For example, if the only difference between record 1 and record 2 is that the first name is spelled "Jane" and "Jayne" respectively, then both records are output.

- b. If you want to add new output columns by duplicating an existing output column definition, select an output column and in the toolbar, choose .
  - c. Under the *Primary Key* column, select an output column that serves as the primary key. Typically the data in this column is unique.
  - d. Under the *Nullable* column, choose whether the column value can be empty (nullable).
  - e. To reorder the columns, select the column that you want to move and in the toolbar click the up or down arrows.
4. **Optional:** Define filters.

For example, if you want to move all the records that are in Canada for the year 2017, your filter might look like the following: "Filter1\_input"."COUNTRY" = 'Canada' AND "Filter1\_input"."DATE"='2017'

- a. Select the *Filter* tab to compare the column name against a constant value.
- b. In the expression editor, enter the required expression.

### → Tip

Use the *SQL Helper*, choose whether to select the required columns, functions, or operators to the expression editor. Double-click the columns, operators, or functions to include them in the expression editor. For more information on each function, see the "SQL Functions" topic in the *SAP HANA SQL and System Views Reference* guide.

5. Add and connect new nodes.

If you want to configure the Data Transform operator with another node:

- a. In the menu bar, use the breadcrumb navigation to go back to the node editor.

- b. **Optional:** Add new nodes.
  - c. To connect the nodes, select the output port of a node and drag the cursor to an input port of another node.
6. Define data target.

It is necessary to create a data target to the node and create an output port for Data Transform operator.

  - a. Right-click the output port of the node and select *Create Data Target*.

## 9.1.2 Configure the Join Node

A Join node represents a relational multiway join operation.

### Prerequisites

You have configured the operator with the Join node and connected output of one or two previous nodes to the Join node.


### Context

The Join node can perform multiple step joins on maximum of two inputs. Configure the Join node to define the join condition and the output columns of the Join node.

#### **i** Note

The Join node is not available for real-time processing.

### Procedure

1. Double-click the *Join* node.
2. Create a join.
  - a. In the *Definition* tab, select a source.
  - b. Choose  (Create Join) and drag the cursor to another source on the canvas with which you want to create a join.

You can also create a join by selecting a column in the source and dragging the cursor to the required column in a different source. In this case, the application automatically creates a join condition.
3. Define the join.

The *Join Definition* pane, define the join type and the join condition.

- a. In the *Join Type* dropdown list, select a value.

Join Type	Description
Inner Join	Use when each record in the two tables has matching records.
Left Outer	Outputs all records in the left table, even when the join condition does not match any records in the right table.
Right Outer	Outputs all records in the right table, even when the join condition does not match any records in the left table.
Cross	Outputs all possible combinations of rows from the two tables.

- b. In the expression editor, enter a join condition.
- c. **Optional:** To use the join condition that the application proposes, select *Propose Condition*.  
The application analyzes the sources participating in the join and proposes a condition.



#### 4. Define the output columns.

In the *Columns* tab, define the output columns of the join node.

- a. In the toolbar, choose the *Columns* tab.
- b. In the *Mapping* pane, select a *Source* column and drag and drop it into the *Drop row here* zone in the *Target* section.

#### i Note

You can also duplicate an output column (map the same source column more than once). Select the required *Source* column and hold and drag the cursor to the *Drop row here* zone in the *Target* section.

- c. **Optional:** If you want to automatically map the columns based on column names, in the *Source* section, choose  (Auto Map by Name).
- d. If you want to edit a column name, select a *Target* column and choose  (Edit).

#### i Note

You cannot edit the data type of output columns of a join node.

- e. If you want to remap an output column with a different source column, right-click the mapping and choose *Remap*.  
Select a new source column and choose *OK*.

#### 5. **Optional:** Reorder output columns.

In the *Columns* tab toolbar, switch to the *Form* pane to reorder the output columns.

- a. To reorder the columns, select the column that you want to move and in the toolbar click the up or down arrows.

#### 6. Add and connect new nodes.

If you want to configure the Data Transform operator with another node:

- a. In the menu bar, use the breadcrumb navigation to go back to the node editor.

- b. **Optional:** Add new nodes.
  - c. To connect the nodes, select the output port of a node and drag the cursor to an input port of another node.
7. Define data target.

It is necessary to create a data target to the node and create an output port for Data Transform operator.

  - a. Right-click the output port of the node and select *Create Data Target*.

## Related Information

[Join Design Considerations \[page 165\]](#)

### 9.1.2.1 Join Design Considerations

You can control memory and CPU usage used to process joins, improve runtime performance of the joins in the graph, and process more joins within a graph effectively with resource limitations.

#### Join Resource and Performance Control Options

When designing joins within a structured data transform operator, use the following options to help control the resources needed:

- **Rank:** A number whose value determines if a table should be considered inner or outer. If the rank is higher, the table is outer. If the rank is lower, the table is inner.
- **Cache:** A value of yes or no determines whether a table should be cached or not. A value of automatic means that the engine decides.

#### Choosing Rank and Cache Values for Join

- DB – database source
- File – cloud file source
- Any – any output within a data transform operation like projection or join

One source will be referred as left and other one as right.

#### Inner Join

##### DB and DB

- Use low rank for the large table and high rank for the small table
- Cache = no, for both sources

Index must be defined for join columns on the DB table.

Runtime performance option: If index is not defined for the inner table, choose Cache = yes. This setting improves runtime performance but affects memory. Choose this option when there are fewer resources consuming transforms.

#### **File and DB, Any and DB**

- Use low rank for the DB and high rank for the file
- Cache = no, for both sources

Index must be defined for join columns on the DB table.

Runtime performance option: When the number of database rows is less than the number of rows in the file, use a lower rank for the file and set Cache = Yes. This setting improves runtime performance but affects memory. Choose this option when there are fewer resources consuming transforms.

#### **File and File, Any and File, Any and Any**

- Use low rank for the smaller source
- Cache = yes for the low rank source

Index must be defined for join columns on the DB table.

Runtime performance option: When the number of rows is fewer in one source than another source, use low rank for the larger source and set Cache = Yes. This setting improves runtime performance but affects memory. Choose this option when there are fewer resources consuming transforms.

## **Left Outer Join**

In case of Left Outer Joins, the left source is always chosen as outer irrespective of type, and the right source is chosen as inner. The Rank value is not considered.

- Cache = no, for left table
- Cache = no, when right table is DB
- Cache = yes, when right table is File or Any

When the number of rows is fewer in one source than another source, use the smaller source as Left (outer). This setting improves runtime performance but affects memory. Choose this option when there are fewer resources consuming transforms.

- When the Right table is a database, choose Cache = no. When the Right table is a File or Any, choose Cache = yes.

Index must be defined for join columns in the DB table.

## Right Outer Join

In case of Right Outer Join, the right source is always chosen as outer irrespective of type, and the left source is chosen as inner. The Rank value is not considered.

- Cache = no, for right table
- Cache = no, when left table is DB
- Cache = yes, when left table is File or Any

Runtime performance option: When the number of rows is smaller in one source than another source, use the smaller sources as the RIGHT (outer). This setting improves runtime performance but affects memory. Choose this option when there are few resource-consuming transforms.

- When the LEFT table is a database, choose Cache = no. When the LEFT table is a File or Any, choose Cache = yes.

Index must be defined for join columns on the DB table.

## Cache Size Estimate

When any input source is chosen to be cached for join, the amount of cache memory used by the Flowagent engine is roughly estimated using the following formula.

Cache size in bytes = Number of rows \* Number of columns \* 20 bytes (average column size) \* 1.3 (30% overhead)

Number of columns here refers to the number of columns selected in the Join from the table plus the number of join columns.

## 9.1.3 Configure the Aggregation Node

An Aggregation node represents a relational group-by and aggregation operation.

### Prerequisites

You have defined the operator with an Aggregation node and connected the previous node to this node.

### Procedure



1. Double-click the [Aggregation](#) node.
2. Define the output columns.

If you have connected the previous node to the Aggregation node, you can map any columns from the input as output columns of the Aggregation node. You can add, delete, and rename the columns, as needed.


- a. In the *Mapping* pane, select a *Source* column and drag and drop it into the *Drop row here* zone in the *Target* section.

### Note

You can also duplicate an output column (map the same source column more than once). Select the required *Source* column and drag and drop it into the *Drop row here* zone in the *Target* section.

- b. **Optional:** If you want to automatically map the columns based on column names, in the *Source* section, choose  (Auto Map by Name).
  - c. If you want to edit a column name, select a *Target* column and choose  (Edit).
  - d. If you want to remap an output column with a different source column, right-click the mapping and choose *Remap*.  
Select a new source column and choose *OK*.
3. Define the aggregation type.

You can specify the columns that you want to have the aggregate or group-by actions taken upon.

- a. If you want to define an aggregation type, select a *Target* column and choose  (Edit).
- b. In the *Aggregation Type* dropdown list, select a value.

Aggregation Type	Description
<empty>	Specifies a list of columns for which you want to combine output. For example, group sales orders by date to find the total sales ordered on a particular date. This type is the default aggregation type.
Avg	Calculates the average of a given set of column values.
Count	Returns the number of values in a table column.
Max	Returns the maximum value from a list.
Min	Returns the minimum value from a list.
Sum	Calculates the sum of a given set of values.

4. **Optional:** Reorder output columns.

In the *Columns* tab toolbar, switch to the *Form* pane to reorder the output columns.

- a. To reorder the columns, select the column that you want to move and in the toolbar click the up or down arrows.

5. Add and connect new nodes.

If you want to configure the Data Transform operator with another node, follow these steps:

- a. In the menu bar, use the breadcrumb navigation to go back to the node editor.
- b. Add new nodes.
- c. To connect the nodes, select the output port of a node and drag the cursor to an input port of another node.



6. Define data target.

It is necessary to create a data target to the node and create an output port for Data Transform operator.

- a. Right-click the output port of the node and select *Create Data Target*.

## 9.1.4 Configure the Union Node

A Union node represents a relational union operation.


### Prerequisites

You have configured the operator with the Union node and connected the previous nodes to Union node.

### Context

The union operator forms the union from two or more inputs with the same signature. This operator can either select all values including duplicates (UNION ALL) or only distinct values (UNION).

### Procedure

1. In the canvas, select the *Union* node.
2. Choose  (Open Configuration).  
In the *Configuration* pane, under the *Columns* section, you can see the column information from the previous nodes.
3. If you want to merge all of the input data (including duplicate entries) into one output, enable the *Union All* toggle button.
4. Add and connect new nodes.  
If you want to configure the Data Transform operator with another node:
  - a. In the menu bar, use the breadcrumb navigation to go back to the node editor.
  - b. Add new nodes.
  - c. To connect the nodes, select the output port of a node and drag the cursor to an input port of another node.
5. Define data target.  
It is necessary to create a data target to the node and create an output port for *Data Transform* operator.
  - a. Right-click the output port of the node and select *Create Data Target*.

## 9.1.5 Configure the Case Node

The Case node specifies multiple paths so that the rows are separated and processed in different ways.

### Prerequisites

You have defined the operator with a Case node and connected the previous node to the Case node.

### Context

Route input records from a single source to one or more output paths. You can simplify branch logic in data flows by consolidating case or decision making logic in one node. Paths are defined in an expression table. By default, there are two output ports defined. The one ending in *Default* contains any records that do not meet the expression definition in one of the other output ports.

### Procedure

1. Double-click the *Case* node.
2. **Optional:** Select *Output Row Once* to specify whether a row can be included in only one or many output targets.

For example, you might have a partial address that does not include a country name such as 455 Rue de la Marine. It is possible that this row could be output to the tables named *Canada\_Customer*, *France\_Customer*, and *Other\_Customer*. Select this option to output the record into the first output table whose expression returns TRUE. Not selecting this option would put the record in all three tables.

3. **Optional:** Choose + (Add Port) to add more Case output ports and expressions.
4. Click the name under the *Output Port Name* heading to rename the port to be more meaningful. For example, *Canada\_Customer*.
5. Click the link under the *Expression* heading to open the expression editor.
6. In the expression editor, define an expression for the records that you want to include in this output.
7. Define the default port by selecting the required output port and choose *Default*.

The default port contains any records that do not meet the expression definition in one of the other output ports.

8. Add and connect new nodes.

If you want to configure the Data Transform operator with another node:

- a. In the menu bar, use the breadcrumb navigation to go back to the node editor.
- b. Add new nodes.
- c. To connect the nodes, select the output port of a node and drag the cursor to an input port of another node.

9. Define data target.

It is necessary to create a data target to the node and create an output port for *Data Transform*

- a. Right-click the output port of the node and select *Create Data Target*.  
operator.

## 9.2 Structured Consumer Operators

### Related Information

[SAP Application Consumer \[page 171\]](#)

[Structured File Consumer \[page 172\]](#)

[Structured SQL Consumer \[page 174\]](#)

### 9.2.1 SAP Application Consumer


The SAP Application Consumer operator allows you to create pipelines that consume data from SAP and non-SAP sources as modeled in the Pipeline Modeler, and connect them to other structured operators such as the Structured Data Transform operator.

#### Procedure

1. Start the SAP Data Intelligence Modeler.
2. In the navigation pane, select the *Graphs* tab.
3. Choose + (Create Graph).  
The application opens an empty graph editor in the same window, where you can define your graph. A graph can contain a single operator or a network of operators based on the business requirement.
4. In the *Operators* tab, double-click the *SAP Application Consumer* operator (or drag and drop it to the graph editor) to add it as a process in the graph.
5. To open the editor of the *SAP Application Consumer* operator, double-click the operator.
6. Configure the Service, Connection, and Source (dataset) properties. You can delete columns or define filters to orchestrate the data as required.
  - For directories containing part-files, you can select the partition type and choose what happens upon string truncation.
  - For OData service, you can browse and expand the navigation properties by defining the *Depth* value based on which the *Source Columns* section is updated. The depth value can be either 1 or 2.

### i Note

- Use the [Data Preview](#) option to see the original data from the source or to see the configured data. If the table is very wide or contains a number of large column types, the result may be truncated in order to avoid out of memory issues.
- The file browsing and data preview is not possible if substitution parameters are used to configure the connection ID or the table name properties.

7. In the graph editor, select the operator and choose  (Open Configuration) to define the other required configurations of the operator in the configuration panel.
8. Add any producer operator from the Structured Data Operators category to the graph.
9. Connect the output port of the [SAP Application Consumer](#) operator to the producer operator.
10. To control the stop of the graph execution, add the Graph Terminator operator at the end of the graph.
11. Connect the output port of the producer operator to the Graph Terminator operator.
12. Save and execute the graph.

## 9.2.2 Structured File Consumer

Structured File Consumer operator reads from any supported cloud storage. The operator produces structured output, and you need to connect it to other operators from Structured Data Operators category.


### Procedure

1. Start the SAP Data Intelligence Modeler.
2. In the navigation pane, select the [Graphs](#) tab.
3. Choose **+** (Create Graph).

The application opens an empty graph editor in the same window, where you can define your graph. A graph can contain a single operator or a network of operators based on the business requirement.
4. In the [Operators](#) tab, double-click the [Structured File Consumer](#) operator (or drag and drop it to the graph editor) to add it as a process in the graph.
5. To open the editor of the [Structured File Consumer](#) operator, double-click the operator.
6. Configure the Service, Connection, and Source (dataset) properties.
  - For CSV files, you can define various CSV properties.
  - For JSON files, you can mention the flattening level in the JSON properties.
  - For directories containing part-files, you can select the partition type and choose what happens upon string truncation.
7. Delete columns or define filters to orchestrate the data as required.

### i Note

- Use the [Data Preview](#) option to see the original data from the source or to see the configured data. If the table is very wide or contains a number of large column types, the result may be truncated in order to avoid out of memory issues.
- The file browsing and data preview is not possible if substitution parameters are used to configure the connection ID or the table name properties.

8. In the graph editor, select the operator and choose  (Open Configuration) to define the other required configurations of the operator in the configuration panel.
9. Add any producer operator from the Structured Data Operators category to the graph.
10. Connect the output port of the [Structured File Consumer](#) operator to the producer operator.
11. To control the stop of the graph execution, add the Graph Terminator operator at the end of the graph.
12. Connect the output port of the producer operator to the Graph Terminator operator.
13. Save and execute the graph.

## Related Information

[Consuming Excel Files with Structured File Consumer Operator \[page 173\]](#)

### 9.2.2.1 Consuming Excel Files with Structured File Consumer Operator

Use a structured file consumer operator and select an Excel file as a source in a pipeline to transform the Excel format to a table or dataset.

## Context

After you open the structured file consumer operator in your graph, select a connection in [Connection ID](#) and an Excel file source in [Source](#). Then perform the following steps to configure the Excel worksheet:

## Procedure

1. Select [Data Preview](#).

The [Excel Properties](#) pane opens. The Modeler fetches the first sheet in the selected Excel file.

The lower portion of the Excel Properties pane shows the source columns in the table that also includes the qualified names and the data types. You can change the data types, if necessary.

2. **Optional:** Select *Modify* to select a different worksheet. The *Excel Properties* dialog opens.
  - a. Select a sheet number from the *Select Sheet* list.
  - b. Select *Set first row as header* to use the first row of the sheet as the header row in the resulting table.
  - c. Select *OK*.

## Results


The Modeler doesn't transform formulas into the final table or dataset. Therefore, if your Excel sheet contains cells with formulas, the result is an empty cell in the final table or dataset.

## 9.2.3 Structured SQL Consumer

SQL Consumer operator reads from a specified database using native SQL. This operator produces a structured output, and you need to connect it to other operators from Structured Data Operators category.

### Procedure

1. Start the SAP Data Intelligence Modeler.
2. In the navigation pane, select the *Graphs* tab.
3. Choose **+** (Create Graph).

The application opens an empty graph editor in the same window, where you can define your graph. A graph can contain a single operator or a network of operators based on the business requirement.
4. In the *Operators* tab, double-click the *Structured SQL Consumer* operator (or drag and drop it to the graph editor) to add it as a process in the graph.
5. To open the editor of the *Structured SQL Consumer* operator, double-click the operator.
6. Configure the Service and Connection properties.
7. Write an SQL statement in the *SQL Editor* dialog.
8. In the graph editor, select the operator and choose  (Open Configuration) to define the other required configurations of the operator in the configuration panel.
9. Add any producer operator from the Structured Data Operators category to the graph.
10. Connect the output port of the *Structured SQL Consumer* operator to the producer operator.
11. To control the stop of the graph execution, add the Graph Terminator operator at the end of the graph.
12. Connect the output port of the producer operator to the Graph Terminator operator.
13. Save and execute the graph.

## 9.3 Structured Producer Operators

### Related Information

[SAP Application Producer \[page 175\]](#)

[Structured File Producer \[page 176\]](#)

[Structured Table Producer \[page 177\]](#)

### 9.3.1 SAP Application Producer


The SAP Application Producer operator allows you to create pipelines that write data to SAP Business Warehouse and OData targets as modeled in the Pipeline Modeler, and connect them to other structured operators such as the Structured Data Transform operator.

#### Procedure

1. Start the SAP Data Intelligence Modeler.
2. In the navigation pane, select the *Graphs* tab.
3. Choose **+** (Create Graph).  
The application opens an empty graph editor in the same window, where you can define your graph. A graph can contain a single operator or a network of operators based on the business requirement.
4. In the *Operators* tab, double-click the *SAP Application Producer* operator (or drag and drop it to the graph editor) to add it as a process in the graph.
5. Add any consumer operator from the Structured Data Operators category to the graph. To know more about configuring consumer operators, See [Structured Consumer Operators \[page 171\]](#).
6. Connect the output port of the consumer operator to the *SAP Application Producer* operator.
7. To open the custom editor of the *SAP Application Producer* operator, double-click the operator.
8. Configure the Service, Connection, and Target properties. You can map source and target columns if not automatically mapped.

#### Note

- Use the *Data Preview* option to see the output data. If the table is very wide or contains a number of large column types, the result may be truncated in order to avoid out of memory issues.
- The file browsing and data preview is not possible if substitution parameters are used to configure the connection ID or the table name properties.

9. In the graph editor, select the operator and choose  (Open Configuration) to define the other required configurations of the operator.
10. To control the stop of the graph execution, add the Graph Terminator operator at the end of the graph.
11. Connect the output port of the producer operator to the Graph Terminator operator.
12. Save and execute the graph.

## 9.3.2 Structured File Producer


The Structured File Producer operator receives data from any structured data operators and produces a file (CSV, ORC, or PARQUET) in the specified storage.

### Procedure

1. Start the SAP Data Intelligence Modeler.
2. In the navigation pane, select the *Graphs* tab.
3. Choose **+** (Create Graph).  
The application opens an empty graph editor in the same window, where you can define your graph. A graph can contain a single operator or a network of operators based on the business requirement.
4. In the *Operators* tab, double-click the *Structured File Producer* operator (or drag and drop it to the graph editor) to add it as a process in the graph.
5. Add any consumer operator from the Structured Data Operators category to the graph. To know more about configuring consumer operators, See [Structured Consumer Operators \[page 171\]](#).
6. Connect the output port of the consumer operator to the *Structured File Producer* operator.
7. To open the custom editor of the *Structured File Producer* operator, double-click the operator.
8. Configure the Service, Connection, and Target properties.
  - a. To create new target within an existing schema, click **+** *Add Target* in the *Browse* dialog. The columns will be copied from the source consumer operator to the new target.
9. You can map source and target columns if not automatically mapped.

#### Note

- Use the *Data Preview* option to see the output data. If the table is very wide or contains a number of large column types, the result may be truncated in order to avoid out of memory issues.
- The file browsing and data preview is not possible if substitution parameters are used to configure the connection ID or the table name properties.

10. In the graph editor, select the operator and choose  (Open Configuration) to define the other required configurations of the operator.
11. To stop running the graph, add the Graph Terminator operator at the end of the graph.
12. Connect the output port of the producer operator to the Graph Terminator operator.



13. Save and execute the graph.

### 9.3.3 Structured Table Producer

The Structured Table Producer operator receives data from any structured operators, and produces a table in the specified database.

#### Procedure

1. Start the SAP Data Intelligence Modeler.
2. In the navigation pane, select the *Graphs* tab.
3. Choose **+** (Create Graph).

The application opens an empty graph editor in the same window, where you can define your graph. A graph can contain a single operator or a network of operators based on the business requirement.
4. In the *Operators* tab, double-click the *Structured Table Producer* operator (or drag and drop it to the graph editor) to add it as a process in the graph.
5. Add any consumer operator from the Structured Data Operators category to the graph. To know more about configuring consumer operators, See [Structured Consumer Operators \[page 171\]](#).
6. Connect the output port of the consumer operator to the *Structured Table Producer* operator.
7. To open the custom editor of the *Structured Table Producer* operator, double-click the operator.
8. Configure the Service, Connection, and Target properties.
  - a. To create new target within an existing schema, click **+** *Add Target* in the *Browse* dialog. The columns will be copied from the source consumer operator to the new target..
9. Choose a write mode:
  - Append: Add data to an existing table. In this mode, you can choose to perform an upsert. Select the checkbox *Update Records By Primary Key (Upsert)*.
  - Overwrite: Drop the table and create another one according to the source schema.
  - Truncate: Clear the table before inserting data.
  - Delete: Delete records from the target where values of the mapped target columns match with the source column.

#### Caution

Existing column mappings will be cleared once you switch to delete mode from any other mode.


#### Note

Delete mode is not available for HDL database connection.

10. You can map source and target columns if not automatically mapped.

### i Note

- Use the [Data Preview](#) option to see the output data. If the table is very wide or contains a number of large column types, the result may be truncated in order to avoid out of memory issues.
- The file browsing and data preview is not possible if substitution parameters are used to configure the connection ID or the table name properties.

11. In the graph editor, select the operator and choose  (Open Configuration) to define the other required configurations of the operator.
12. To stop running the graph, add the Graph Terminator operator at the end of the graph.
13. Connect the output port of the producer operator to the Graph Terminator operator.
14. Save and execute the graph.

## 9.4 Custom Editor

Use the Custom Editor to update the source dataset and projection and filters are pushed down to the source.

### i Note

The Custom Editor is only for the operators from the [Structured Data Operators](#) and [Connectivity \(via Flowagent\)](#) categories.

## Projection

Use the [Delete Projection](#) button to drop a column and reduce the amount of data read from the source.

### i Note

If you drop a column and need to revert the changes, use [Restore Metadata](#) to get the latest source dataset definition again.

## Filters

Filtering is based on column capabilities. If a source allows filtering on a column, it is shown in the user interfaces as drop-down options.

Supported operators are EQUAL, BETWEEN, >, <, <= and >=.

If the operator user interface doesn't show any filtering, then the source doesn't support it. To filter on these columns, use the Data Transform operator.

### i Note

The operator OR is applicable per column while AND is applicable across columns. For example, a source has two columns: ID and NAME. If you choose multiple conditions on ID and on NAME, the query would look like (ID = 2 OR ID = 3) AND (NAME = "TEST" OR NAME = "TEST2").

## Data Preview

Use Data Preview to preview data in the source system.

Data Preview has two modes: Source and Adapted. Adapted Dataset takes the selected projection and applies a filter in the user interface, while Source Dataset shows the preview of the source object as-is.

## Related Information

[SAP Application Consumer V2](#)  
[Structured File Consumer V3](#)

## 9.5 Resiliency with Structured Data Operators

Table Consumer version 3 (com.sap.database.table.consumer.v3) is part of the Generation 2 set of operators, and you can run pipelines with snapshot enabled.

Here are the requirements:

- Table Consumer must be configured with a Partition Type:
  - *Logical* (user-defined) for all supported sources
  - *Row ID* and *Physical* (Source Based) for Oracle
- Table Consumer must not be in a group with multiplicity greater than one.
  - Snapshot does not support multiplicity greater than one.
  - If you want to run partitions in parallel, then snapshot is not supported.

If your pipeline does not meet the requirements above, you can still run the graph. However, snapshot is not enabled.

For more information about partitioning, see [Partitioning](#).

For more information about graph termination using partition, see [Graph Termination Using Partition](#).

## Resiliency Behavior

On restart, the pipeline skips processing partitions that were successfully consumed previously. For example:

In the first run, partitions 1 and 2 were completed successfully, but the graph crashed while running partition 3 and only some rows were read from partition 3. On restart, the pipeline skips partitions 1 and 2 and then starts processing from partition 3.

Different delivery guarantees are provided depending on the producer and writing modes selected. We recommend that you use a combination that leads to **exactly-once** delivery guarantee whenever possible. The table describes the behavior of the Structured Data Producer Operators.

Operator	Mode	Delivery Guarantee	Note
Table Producer	Append + Upsert	exactly-once	No duplicated data in the target.
Table Producer	Append, Truncate, or Overwrite	at-least-once	Target could have some duplicated data from a partition that was partially run, but it will not lose any data.
Structured File Producer	Overwrite	exactly-once	There is exactly one file per partition index.
Structured File Producer	Append	at-least-once	Target could have some duplicated data from a partition that was partially run, but it will not lose any data.
Structured File Producer	Overwrite or Append + Part-file	exactly-once	Operator ensures that files within a part-file directory are unique per partition index.
Application Producer	Append	at-least-once	Target could have some duplicated data from a partition that was partially run, but it will not lose any data.
Cloud Table Producer	Append or Truncate	at-least-once	Target could have some duplicated data from a partition that was partially run, but it will not lose any data.

## Sample Graph

The sample graph `com.sap.flowagent.gen2.resilient` demonstrates how a resilient pipeline with partition can be modeled.

# 10 Operator Metrics

The operators publish a set of metrics as soon as the graph is executed. Each operator provides a different set of metrics.

## i Note

Operator Metrics is currently relevant only for the operators from the Structured Data Operators and Connectivity (via Flowagent) categories.

## Consumer Operators

- Optimized (Boolean): indicates whether the operator is optimized with other operators from the same engine. When this metric is set to 1, the runtime metrics, such as row count, aren't displayed.
- Row Count (rows): provides the number of rows read from the source.
- Column Count (columns): provides the number of columns read from the source.
- Partition Count (partitions): provides the number of partitions being read when the source is set to use partitions.

## Producer Operators

- Row Count (rows): provides the number of rows written to the target.
- Current row rate (rows/s): provides the number of rows per second written to the target.
- Batch count (batches): provides the number of batches written when the operator is set to write the data in batches.
- Elapsed execution time (seconds): indicates the amount of time the graph has been running.

## Metrics Available in Debug Mode

- Job CPU usage (percentage): indicates the CPU usage for the execution engine.
- Job memory usage (KB): indicates the memory usage for the execution engine.
- Operator CPU usage (percentage): indicates the CPU usage for the operator subengine.
- Job memory usage (KB): indicates the memory usage for the operator subengine.

# 11 Replicating Data

SAP Data Intelligence replication flows let you transfer small or large datasets, in batch or real-time, from a source to a target using full or delta loading.

You create a replication flow by first configuring source and target connections. Then add one or more tasks, each of which consists of a source object, its corresponding target, and any associated configuration options such as filters or custom mappings. Finally, save it in your user space.

## Deploying and Undeploying Replication Flows

After creating and validating a replication flow, you deploy it to the shared (tenant) repository, where you can run it. Deployment also makes the replication flow available to other users of the tenant for viewing, importing, or modification.

You can undeploy your replication flow for various reasons, such as:

- You want to delete that replication flow from the shared repository because it no longer needs to be run.
- You wish to modify the replication flow and redeploy it.

Undeploying deactivates the flow from the replication service and performs some housekeeping of the connected source system and of runtime artifacts of the replication service within SAP Data Intelligence Cloud. For more specific details, see [Undeploy a Replication Flow \[page 200\]](#).

The SAP Data Intelligence Monitoring application provides an interface where you can view and manage replication flow and task processes.

### i Note

To see the known limitations of the replication flow functionality in SAP Data Intelligence, see SAP Note [3223810](#).

## Possible Duplication of Records

If you use replication flows with any target system, the data replication process provides "at-least-once delivery". Therefore, duplicate change records may be delivered to the target system, which results in eventual consistency when the replication is recovering the data load in error situations.

However, when the target system is a database (such as HANA Cloud), the duplicates are eliminated during the replication execution by using the UPSERT mechanism of the databases, thereby providing "exactly-once delivery".

For other targets—for example, Amazon Web Service Storage Services (S3), Microsoft Azure Data Lake Storage Gen2 (ADL\_V2), Google Cloud Service (GCS), or HANA Data Lake (HDL) file, use [Suppress Duplicates](#)

to minimize the occurrence of duplicates during the initial load. For more information about duplicate suppression, see [Create a Replication Flow \[page 183\]](#).

For duplicate records that can't be automatically eliminated in the target cloud storage, use the information provided in the timestamp column to help you identify the currently valid record and ignore potential duplicated records.

Also note that because source rows may be sent multiple times in an effort to maintain data consistency, the replication may present a replication row count that is greater than the actual rows in the target.

## Related Information

[Create a Replication Flow \[page 183\]](#)

[Validate the Replication Flow \[page 193\]](#)

[Deploy the Replication Flow \[page 193\]](#)

[Run the Replication Flow \[page 194\]](#)

[Cloud Storage Target Structure \[page 195\]](#)

[Kafka as Target \[page 198\]](#)

[ABAP Cluster Table Replications with Delta Load \[page 199\]](#)

[Edit an Existing Replication Flow \[page 200\]](#)

[Undeploy a Replication Flow \[page 200\]](#)

[Delete a Replication Flow \[page 202\]](#)

[Clean Up Source Artifacts \[page 202\]](#)

[Monitoring SAP Data Intelligence \[page 204\]](#)

[Replication Flow Connections](#)

## 11.1 Create a Replication Flow

The SAP Data Intelligence Modeler includes an interface for creating and running replication flows.

### Prerequisites

Before you perform the following steps, ensure that the required source and target connections are created and enabled for you in SAP Data Intelligence Connection Management.

For delta loading tasks in a Microsoft Azure SQL database source, the source must include a schema that has the same name as the user name specified for the corresponding AZURE\_SQL\_DB connection. If this schema doesn't exist, a database administrator must create it including the necessary write privileges. This schema is required for storing internal objects for delta replication.

## Procedure

1. Start the SAP Data Intelligence Modeler.
2. Open the *Replications* tab in the navigation pane.

### ! Restriction

Even though the *Schedule* tab appears in the lower panel, scheduling isn't supported in *Replications*.

3. Select **+** (*Create Replication Flow*) in the Navigation pane menu toolbar.
4. Enter a name for the replication flow and select *OK*.
5. **Optional:** Add a description in the *Properties* tab.
6. Choose a source in Source section of the *Properties* tab by performing the following substeps:
  - a. Choose a source connection from the *Source Connection* list.  
 The list is sorted alphabetically. Only connections for which you have authorization are available.
  - b. Add a name or browse for the parent object that contains one or more datasets to replicate in *Container*.
    - To replicate a database table, choose the schema that includes the table.
    - For SAP ECC/SLT sources, browse for and open the SLT folder and choose an SLT configuration.
    - For CDS views, browse for and select the CDS root folder.
7. Repeat the connection and container steps for one or more of the following targets:

### i Note

For cloud storage, the container name is limited to 64 characters. For Kafka, there is no character limit.

- Amazon Web Service Storage Services (S3)
  - Microsoft Azure Data Lake Storage Gen2 (ADL\_V2)
  - Google Cloud Service (GCS)
  - SAP HANA Data Lake (HDL)
  - Kafka
8. Complete the options as described in the following table for a cloud storage target.

Option	Description
<b>Group Delta By</b>	<p>Specifies whether you want to create additional folders for sorting updates based on the date or hour.</p> <p>For Load Type of Initial and Delta, choose one of the following options:</p> <ul style="list-style-type: none"> <li>• <i>None</i></li> <li>• <i>Date</i></li> <li>• <i>Hour</i></li> </ul>
<b>File Type</b>	For CSV:



Option	Description
	<ul style="list-style-type: none"> <li>• <i>File Delimiter</i>: Specifies the character to use as a delimiter for columns in CSV files.</li> <li>• <i>File Header</i>: Specifies whether CSV files include a header row with the column names.</li> </ul> <p>For Parquet:</p> <ul style="list-style-type: none"> <li>• <i>File Compression</i>: Specifies the compression algorithm to use for Parquet files. Applicable algorithms include the following: <ul style="list-style-type: none"> <li>• <i>None</i></li> <li>• <i>GZIP</i></li> <li>• <i>Snappy</i></li> </ul> </li> <li>• <i>Compatibility Mode</i>: Specifies a compatibility mode to use when replicating to object stores in Parquet. Options include the following: <ul style="list-style-type: none"> <li>• <i>None</i>: Uses the default compatible data type when you create a replication flow. For more information, see <a href="#">Data Type Compatibility [page 191]</a>.</li> <li>• <i>Spark</i>: Converts and stores time data type columns to int64 in the Parquet files. The int64 data type represents microseconds after midnight. This conversion allows the columns to be consumed by Spark.</li> </ul> </li> </ul> <p>For JSON:</p> <ul style="list-style-type: none"> <li>• <i>Encoding</i>: Generated JSON files are encoded in UTF-8 format.</li> <li>• <i>Orient</i>: Specifies the internal structure of the produced JSON files. Internal structures include the following: <ul style="list-style-type: none"> <li>• "records": [{column -&gt; value}, ... , {column -&gt; value}]</li> <li>• "values": [value, ... ,value]</li> </ul> </li> </ul> <p>For jsonlines:</p> <p><i>Encoding</i>: Generated JSON Lines files are encoded in UTF-8 format.</p>
<p><b>Suppress Duplicates</b></p>	<p>Options include the following:</p> <ul style="list-style-type: none"> <li>• <i>True</i>: Minimizes the occurrence of duplicates during the initial load. <ul style="list-style-type: none"> <li>• If necessary for replications from ABAP, apply SAP Note <a href="#">3302718</a> to the ABAP system.</li> </ul> </li> </ul>

Option	Description
	<ul style="list-style-type: none"> <li>SAP Data Intelligence minimizes duplicates by generating unique file names from the source partitions and recording only once. For more information, see <a href="#">Cloud Storage Target Structure [page 195]</a>.</li> <li><i>False</i>: SAP Data Intelligence can deliver duplicate change records to the target. When duplicate change records are delivered to the target, the timestamp column in the file name helps you to identify the current valid record and to ignore potential duplicated records.</li> </ul>

9. Complete the options as described in the following table for a Kafka storage target.

Option	Description
<b>Serialization Type</b>	Choose <i>JSON</i> or <i>Avro</i> . The default is JSON.
<b>Compression</b>	Choose one of the following compression types: <ul style="list-style-type: none"> <li><i>No Compression</i></li> <li><i>GZIP</i></li> <li><i>Snappy</i></li> <li><i>LZ4</i></li> <li><i>Zstandard</i></li> </ul>
<b>Partitions</b>	Specifies the number of partitions to split the topic into. The default is 1.
<b>Replication Factor</b>	Specifies how many copies of data to store across several Kafka brokers. The default is 1.

**i Note**

Replication Factor is limited to the number of Kafka brokers.

**i Note**

The replication flow considers messages as written successfully when the message is accepted by all in-sync replicas (`ack=all`).

10. Save the replication flow.

## Next Steps

Next, create one or more tasks in the replication flow. Tasks are the executable components that consist of a source dataset, its corresponding target, and any associated configuration options.

## Related Information

[Create Tasks \[page 187\]](#)

[Using SAP Data Intelligence Connection Management Replication Flow Connections](#)

### 11.1.1 Create Tasks

In a replication flow, a task is an executable component that consists of a source dataset, its corresponding target, and any associated configuration options such as filters, mappings, and load type.

#### Context

A replication flow must contain at least one task.

When you create a task, the target dataset name matches that of the source dataset, but you can change it.

During execution, the system checks if the target dataset exists. For a database, the target dataset is a table; for object stores and data lakes, the target dataset is a folder or set of files with the same prefix. If the target dataset does not exist, the system creates the table or folder, respectively.

#### i Note

If you add a new name for an object store target, do not use the characters dot (.) or forward slash (/) in the name.

#### Procedure

1. Choose the *Tasks* tab.
2. Choose *Create*.

The *Add Datasets* dialog displays.

3. In the *Add Datasets* dialog, navigate through the folder hierarchy, if necessary, or add a dataset name in the search box (required for SLT sources). Choose one or more datasets from the list and choose *OK*.

Note that a given source dataset can't be used in more than one replication flow.

4. Add filters and custom mappings to the task.
5. To change the target to an existing object (table or folder), in the *Target* column, choose the *Select Target* icon next to the target name and browse to the object. It is also possible to manually add a new target name. The name is limited to 64 characters.

### i Note

When you change the target of an object store by choosing an existing folder, the folder must be empty; if not, you must choose the *Truncate* option to enable task execution.

6. The default *Load Type* is *Initial Only*. To enable delta loading for this task, choose *Initial and Delta*.
7. To clear the content of the target before the task runs, choose *Truncate*.
8. Save the replication flow.

## Next Steps

You can now validate the replication flow to ensure it's ready to deploy and run.

## Related Information

[Add a Filter \[page 188\]](#)

[Define the Mapping for a Dataset \[page 189\]](#)

[Data Type Compatibility \[page 191\]](#)

[Validate the Replication Flow \[page 193\]](#)

### 11.1.1.1 Add a Filter

For a given task, you can optionally add one or more filters to a dataset to customize the target.

## Procedure

1. In the *Tasks* list, browse to the replication flow to configure.
2. For the dataset to filter, in the *Source Filter* column, choose the *Filter* icon.
3. In the *Provide Filter Values - <dataset\_name>* dialog, choose a column to filter.
4. Under *Define values for <column\_name>*, choose an operator from the dropdown list and add a value in the field.
5. To add more filters for this or other columns, click the + button to add another row.
  - More than one filter on different columns applies the AND operator.
  - More than one filter on the same column applies the OR operator.

For example, filtering a dataset on the product ID number 123 for the countries United States and Germany results in the following:

```
(PRODUCT_ID = 123) AND (COUNTRY = 'US' OR COUNTRY = 'DE')
```

Columns that have filters applied are marked. You can display only the columns that have filters applied by choosing the checkbox above the list.

6. Choose *OK*.

## Results

The dataset's *Source Filter* column now displays *Filtered*. Hover over the word *Filtered* to view the columns that have filters applied.

### 11.1.1.2 Define the Mapping for a Dataset

For existing target tables, you can edit the mapping of a dataset to customize it. Or, for a target table that doesn't exist yet, SAP Data Intelligence replication flow lets you define the table by specifying column names and data-type information.

## Context

By default, SAP Data Intelligence replication flow copies all columns from the source dataset to the target dataset. Custom mappings let you add, edit, or remove columns.

## Procedure

1. View the *Tasks* list.
2. In the *Mapping* column, choose the *Mapping* icon.  
The *Target Mapping - <dataset\_name>* appears.
3. To filter the list of columns, add a value in the *Search* field.
4. The following table contains information about the various tasks, such as editing or defining the mapping of a target column.

Task	Action
<b>Change the source mapping column</b>	Choose a different column from the <i>Source Mapping</i> list.
<b>Add a custom mapping expression</b>	Enable the <i>Expression</i> field and add one or more values. For example: <ul style="list-style-type: none"><li>• String constant (must be surrounded by single quotes).</li><li>• Numeric constant such as -123, 123, 123.45.</li></ul>

Task	Action
	<p>You can also use the following supported functions:</p> <ul style="list-style-type: none"> <li>• CURRENT_UTCDATE</li> <li>• CURRENT_UTCTIME</li> <li>• CURRENT_UTCTIMESTAMP</li> </ul>
<p><b>Change (promote) the data type of a column</b></p>	<p>Enabled only if the target doesn't exist yet.</p> <p>SAP Data Intelligence replication flow maps the target data type from the source data type in the mapping editor automatically, ensuring the data type doesn't change. Either maintain the same data type as the source data type, or choose a compatible data type from the list.</p> <div data-bbox="826 763 1396 920" style="background-color: #f0f0f0; padding: 5px;"> <p><b>i Note</b></p> <p>The list contains all data types regardless of what's compatible to the source data type.</p> </div> <p>For information about compatible data types, see <a href="#">Data Type Compatibility [page 191]</a>.</p> <div data-bbox="826 1021 1396 1178" style="background-color: #f0f0f0; padding: 5px;"> <p><b>❖ Example</b></p> <p>SAP Data Intelligence maps a source data type of int8 to a target data type of int8 automatically.</p> </div> <p>You can promote values, such as field length, decimal, scale, and precision, based on the data type.</p> <div data-bbox="826 1279 1396 1458" style="background-color: #f0f0f0; padding: 5px;"> <p><b>❖ Example</b></p> <p>Promote integer data types. When a source data type is int8, promote the target to either int16, int32 or int64.</p> </div> <p>SAP Data Intelligence replication flow supports only implicit data type compatibility.</p>
<p><b>Identify the column as a primary key</b></p>	<p>Enabled only if the target doesn't exist yet.</p> <p>Choose the <i>Key</i> checkbox.</p>
<p><b>Add a column to the target</b></p>	<p>Enabled only if the target doesn't exist yet.</p> <p>Choose <i>Create</i>. A new row appears at the bottom of the list.</p> <ul style="list-style-type: none"> <li>• Add the new column name and the desired custom mapping expression.</li> <li>• Alternately, disable the <i>Expression</i> field to enable the <i>Source Mapping</i> drop-down list and choose a different source column.</li> </ul>

Task	Action
<b>Delete a column from the target</b>	Enabled only if the target dataset doesn't exist yet. Choose the column name and choose <i>Delete</i> .
<b>Reorder the columns</b>	Enabled only if the target dataset doesn't exist yet. Choose the column row and choose <i>First, Up, Down, or Last</i> .

- Choose *OK* to confirm the mapping changes.

## Results

The dataset's *Mapping* column now displays *Transformed*. Hover over the word *Transformed* to view the columns that have transformations applied.

### 11.1.1.3 Data Type Compatibility

SAP Data Intelligence replication flow maintains data type compatibility between source and target data types.

The following table specifies compatible data types so that you choose a compatible data type when you create a replication flow.

#### Replication Management Service Data Type Compatibility

Source Data Type	Compatible Target Data Type	Additional Information
uint8	uint8 int16 int32 int64 uint64	N/A
int8	int8 int16 int32 int64	N/A
int16	int16 int32 int64	N/A

Source Data Type	Compatible Target Data Type	Additional Information
int32	int32 int64	N/A
int64	int64	N/A
uint64	uint64 decimal (p,s)	decimal(p,s), where $(p - s) \geq 20$ .
bool	bool uint8 int8 int16 int32 int64	For int and uint data types: <ul style="list-style-type: none"> <li>true = 1</li> <li>false = 0</li> </ul>
decimal16	decimal16 decimal34	N/A
decimal34	decimal34	N/A
float32	float32 float64	N/A
float64	float64	N/A
date	date	N/A
time	time	N/A
timestamp	timestamp	N/A
decimal (p,s)	decimal (p,s)	Verify length, precision (p), and scale (s): <ul style="list-style-type: none"> <li>Target length must be <math>\geq</math> source length.</li> <li>Target s must be <math>\geq</math> source s.</li> <li>Target <math>(p - s)</math> must be <math>\geq</math> source <math>(p - s)</math>.</li> </ul>
string(n)	string(m)	$m \geq n$
binary(n)	binary(n)	Target length must be $\geq$ source length.



## 11.2 Validate the Replication Flow


Validation ensures the minimum requirements have been specified for the replication flow for deployment and execution.

### Context

Validation checks the following criteria:

- Source and target connections and containers have been specified.
- Source and target connection and container combinations must be different.
- At least one task has been configured.
- Source and target datasets must be specified for each task.

### Procedure

1. Open the replication flow to validate.
2. In the toolbar, choose the  (*Validate*) button.
3. Review the validation results that are displayed on the screen.
4. Resolve any errors to enable deployment.

### Next Steps

Deploy the replication flow.

## 11.3 Deploy the Replication Flow

Deploy the replication flow to enable its execution.

### Context

After creating the replication flow and configuring its tasks, you deploy the replication flow so it can be run. Deployment also saves the replication flow in the shared repository where it will be available for other users to view, import, or modify.

Note that once deployed, you cannot change the replication flow's source and target connections or containers.

## Procedure

1. Display the replication flow to deploy.
2. Select the  (*Deploy*) icon in the toolbar.

The *Activity Monitor* displays the status of deployment. You can choose the error message text to display and copy the full error message.

You can also view the *Log* tab for processing details.

## Next Steps

You can now run the deployed replication flow.

# 11.4 Run the Replication Flow

Running replicates the datasets from the source to the target.

## Prerequisites

You have created a replication flow with one or more tasks and saved, validated, and deployed the replication flow to the shared repository.

## Procedure

1. Open the replication flow to run.
2. Choose the *Run* icon in the toolbar.

The *Activity Monitor* displays the status. You can also view the *Log* tab for processing details.

Note that once it is running, you can suspend the execution using the toolbar button. To resume, choose *Run*. These actions are also available on the Monitoring application for both replication flows and individual tasks.

## Results

You can now view the execution status and other details of the replication flows and tasks in the SAP Data Intelligence Monitoring application.

Note that after the target has been populated, to reload the data again follow these steps:

1. In the Modeler, display the replication flow.
2. Choose *Undeploy*.
3. Choose *Deploy*.
4. Choose *Run*.

## Related Information

[Monitoring SAP Data Intelligence \[page 204\]](#)

## 11.5 Cloud Storage Target Structure

Running a replication flow with a cloud storage target creates various files and structures.

For Amazon Web Service Storage Service (S3), Microsoft Azure Data Lake Storage Gen2 (ADL\_V2), Google Cloud Service (GCS), and HANA Data Lake (HDL), the replication flow proceeds as follows:

- When the initial load completes, the system writes a `_SUCCESS` file.
- Downstream applications that have direct access to the object store can use the `_SUCCESS` file to verify that the replication completed successfully without checking the replication flow status.

The `.sap.partfile.metadata` objects include dataset metadata information. The objects exist in the root of each data file directory and are created and referenced as part of replication flow processing. Additionally, you can leverage these objects for interpreting and processing the dataset data files.

You can view the `.sap.partfile.metadata` objects in the Metadata Explorer as follows:

```
 /<container-base-path>/
  .sap.rms.container
  <tableName>/
    .sap.partfile.metadata
    initial/
      .sap.partfile.metadata
      part-<unix_timestamp>-<workOrderID-1>-<delimitationNo-01>.<extension>
      ...
      part-<unix_timestamp>-<workOrderID-M>-<delimitationNo-N>.<extension>
      _SUCCESS
    delta/ (only there in case of delta load)
      <date(time)-optional>/
        .sap.partfile.metadata
        part-<unix_timestamp>-<workOrderID-X>-
<delimitationNo-01>.<extension>
        ...
        part-<unix_timestamp>-<workOrderID-Y>-
<delimitationNo-01>.<extension>
```

## ❁ Example

### 📄 Sample Code

```
/path/  
  to/  
    container1/  
      .sap.rms.container  
      table1.csv/  
        .sap.partfile.metadata  
        initial/  
          .sap.partfile.metadata  
          part-1634738166-a2480cda-2a7f-11ec-8d3d-0242ac130003-01.csv  
          part-1634738167-a2480cda-2a7f-11ec-8d3d-0242ac130003-02.csv  
          part-1634738169-acfe5bca-8086-4f04-9014-c1b92106cb23-01.csv  
          part-1634738170-b89722fa-9ff5-43e9-abdf-3d73827447cb-01.csv  
          _SUCCESS  
        delta/  
          .sap.partfile.metadata  
          20211020/  
            .sap.partfile.metadata  
            part-1634738180-cde25008-53e2-4b41-949e-  
b992f4f0f04d-01.csv  
            part-1634739123-d60a2652-2801-4db4-8d7b-  
f20a6b09ef87-01.csv  
            20211021/  
              .sap.partfile.metadata  
              part-1634824512-  
f2c188d0-2a7f-11ec-8d3d-0242ac130003-01.csv  
            table2.csv/  
              .sap.partfile.metadata  
              initial/  
                .sap.partfile.metadata  
                part-1634738166-2bac0984-3888-11ec-8d3d-0242ac130003-01.csv  
                ...  
            container2/  
              table2.parquet/  
                .sap.partfile.metadata  
                initial/  
                  .sap.partfile.metadata  
                  part-1634738166-  
db1a8efa-3887-11ec-8d3d-0242ac130003-01.parquet  
                  part-1634738167-  
db1a8efa-3887-11ec-8d3d-0242ac130003-02.parquet  
                  ...
```

If you set *Suppress Duplicates* to *true*, the file names in the *initial* folder contain an internal-id consisting of 33 base64URL characters.

The target structure may look like the following:

```
/<container-base-path>/  
  .sap.rms.container  
  <tableName>/  
    .sap.partfile.metadata  
    initial/  
      .sap.partfile.metadata  
      part-<internal-id>.<extension>  
      ...  
      part-<internal-id>.<extension>  
      _SUCCESS  
    delta/ (only there in case of delta load)  
      <date(time)-optional>/  
        .sap.partfile.metadata
```

```

Y>.<extension> part-<unix_timestamp>-<workOrderID-X>-<delimitationNo-
...
Y>.<extension> part-<unix_timestamp>-<workOrderID-Y>-<delimitationNo-

```

The replication flow creates multiple dataset files (part-\*.<extension>) during initial and delta loading. The number and size of the dataset files depends on the following factors:

- Source table size.
- Source table structure.
- Change frequency during delta loading.

Each dataset file contains the source columns. Source columns are defined in the dataset mapping in the replication flow task. In addition, the system appends the column as described in the following table.

Column Name	Description
__operation_type	<p>Identifies the type of target row. The possible values are as follows:</p> <ul style="list-style-type: none"> <li>• <i>L</i>: Written as part of the initial load.</li> <li>• <i>I</i>: After the initial load completed, new source row added.</li> <li>• <i>U</i>: After the initial load completed, after image of an update to a source row.</li> </ul> <div data-bbox="847 1106 1396 1330" style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> <p><b>i Note</b></p> <p>For some sources the system switches the value <i>U</i> to <i>A</i> after you apply SAP Note <a href="#">3044005</a>. The APE_KEEP_UPDATE_OPERATION parameter is described in the SAP Note.</p> </div> <ul style="list-style-type: none"> <li>• <i>B</i>: After the initial load completed, before image of an update to a source row. These records are only sent by some sources (like SAP HANA) and only when the after image of the update is not passing the filters specified in the replication task.</li> <li>• <i>X</i>: After the initial load completed, source row deleted. The only target columns to contain data for this operation code are codes that reflect the source key columns. All other target columns are empty.</li> <li>• <i>M</i>: After the initial load completed, archiving operations.</li> </ul>
__sequence_number	An integer value that reflects the sequential order of the delta row in relation to other deltas. This column is empty for initial load rows and isn't populated for all source systems (for example, ABAP).
__timestamp	The UTC date and time the system wrote the row.

## 11.6 Kafka as Target

In a replication flow, every source dataset is transferred to a respective topic in the Kafka cluster.

The target dataset name is matched to the topic name in the Kafka cluster.

### i Note

You can edit the name of the target topic, so it does not need to be the same as the source dataset.

Schema registries are not supported. If you choose AVRO as the serialization format, the schema is contained in every message. For JSON serialization format, no schema information is provided.

Each record from the source system is transferred into a single message in the selected target topic. The software does not load each batch set of data into a message.

The key of the messages is the combination of all primary key values of the record concatenated by "\_".

Additional headers are set in the messages:

### i Note

For information about configuring Kafka target options like number of partitions and replication factor, see [Create a Replication Flow \[page 183\]](#).

- kafkaSerializationType: AVRO or JSON

### i Note

If you select AVRO, the columnName must consist of only alphanumeric and underscore characters and it must also start with a letter or an underscore

- opType: Identifies the type of target row:
  - *L*: Written as part of the initial load.
  - *I*: After the initial load completed, new source row added.
  - *U*: After the initial load completed, after image of an update to a source row.

### i Note

For some sources the system switches the value *U* to *A* after you apply SAP Note [3044005](#). The APE\_KEEP\_UPDATE\_OPERATION parameter is described in the SAP Note.

- *B*: After the initial load completed, before image of an update to a source row. These records are only sent by some sources (like SAP HANA) and only when the after image of the update is not passing the filters specified in the replication task.
- *X*: After the initial load completed, source row deleted. The only target columns to contain data for this operation code are codes that reflect the source key columns. All other target columns are empty.
- *M*: After the initial load completed, archiving operations.
- Seq: Sequence number, an integer value that reflects the sequential order of the delta row in relation to other deltas. This column is empty for initial load rows and is not populated for all source systems (for example, ABAP).

## i Note

If the Kafka cluster is behind an SAP Cloud Connector (SCC), the Kafka cluster and the SCC must be configured such that the broker addresses advertised by the cluster match the virtual hosts maintained for the brokers in the SCC. The simplest solution is to use the same value for virtual and internal hosts in the SCC and to maintain no dedicated advertised listeners for the Kafka brokers. If advertised listeners are maintained, these must be used as virtual hosts in SCC and as broker addresses in the DI connection definition.

## Related Information

[Create a Replication Flow \[page 183\]](#)

## 11.7 ABAP Cluster Table Replications with Delta Load

Replications from ABAP cluster tables result in more complex operations on the target system if delta load is involved.

There's one large underlying database table (the table cluster) that holds the data of multiple "logical" tables (the cluster tables). The table cluster has a set of key columns that are common to all cluster tables. Every cluster table can have additional key columns. (For more information about cluster tables, see [Cluster Name](#).)

You must always select one of the "logical" cluster tables as the replication source.

When replicating cluster tables that include delta load, any change in the source table results in the following:

1. Deletion of all records in the target that match the respective values of the key columns of the underlying table cluster.
2. Insertion of all records from the source table that match the corresponding values of the keys of the underlying table cluster. This includes the newly inserted or modified records, and excludes the deleted records.

If the target is a cloud storage or a Kafka cluster, the above-mentioned delete using the key values of the underlying table cluster is indicated by a delete record, where the key columns that are only present in the cluster table but not in the underlying table cluster aren't specified. For cloud storage targets in CSV format, a "?" is used to indicate such an unspecified key value.

When replicating a cluster table that includes delta load to a Kafka cluster, the keys of the messages are determined using only the key columns of the underlying table cluster.

## 11.8 Edit an Existing Replication Flow

You can import, edit, and update replication flows that have been deployed by you or other users to the shared repository.

### Context

A replication flow that you create is saved in your user repository until you deploy it to the shared repository. Then, other users can view, import, or update (overwrite) the replication flow.

The left pane of the *Replications* editor lists all of the replication flows available to you. Hover over a name to see details for that replication flow.

### Procedure

1. Right-click a replication flow to see the delete or import options. To import a replication flow, right-click it to display the context menu and choose *Import from shared repository*.

The system copies the replication flow to your user repository, overwriting any existing versions.

2. You can now edit the configuration of the replication flow and save it without affecting the version in the shared repository.

For example, to delete a dataset from the replication flow, on the *Tasks* tab, select the dataset and choose *Delete*.

3. To replace (overwrite) the version in the shared repository, Deploy the replication flow.

A change list displays your changes plus any that might have been made to this replication flow by other users at the same time.

## 11.9 Undeploy a Replication Flow

You can undeploy an SAP Data Intelligence replication flow (remove it from the shared repository) using the Replications editor in the SAP Data Intelligence Modeler.

### Context

You may want to undeploy replication flows for different reasons:

- You want to change the configuration of your existing replication flow; for example, to change the target data set or to apply a filter to a task.



- You want to reset or reinitialize a replication flow, including all of its associated tasks, and remove dependent artifacts on the connected source system.

You can undeploy a replication flow that is available in your personal user repository from the SAP Data Intelligence shared repository by using the Undeploy icon in the modeling screen. Note that the undeploy action can only be executed for replication flows that are already activated or running.

If you undeploy a replication flow, the system stops any running initial load or delta process for this task and it deletes certain runtime objects that belong to your replication flow, but it does not delete the target table even if the replication management system created the target table.

## Procedure

1. To undeploy a replication flow from your user repository, start the SAP Data Intelligence Modeler.
2. In the navigation pane, choose the [Replications](#) tab.
3. Choose the replication flow in the list of available replication flows.
4. Click the [Undeploy](#) button available on the top menu bar.
5. In the confirmation dialog, click [OK](#) to continue with the undeploy action.

### i Note

When undeploying a replication flow that copies data from a connected source system (for example: SAP S/4HANA, SAP System via SLT, or Azure SQL), this removes the subscription to the source table from the source system, which includes the triggers and logging tables. This applies to all of the source systems, including SAP S/4HANA and relational databases, such as Azure SQL. In the case of SAP S/4HANA, the undeploy action also removes the dependent runtime artifacts related to replication when the undeployment process is successfully executed.

In order to not block the undeployment process indefinitely, SAP Data Intelligence retries three times to successfully undeploy each task in a replication flow. Due to connection failure to source systems or some other unexpected reasons, SAP Data Intelligence may not be able to successfully execute the undeployment process in the source system. In case of an error during the undeployment process, it can happen that not all dependent artifacts are deleted in the connected source system. Despite the error, SAP Data Intelligence still deletes the definition of the replication flow, including its runtime data, status, and metrics from its repository in order to complete undeployment requests and allow users to proceed with completing the task of remodeling the replication flow. In this case, see [Clean Up Source Artifacts \[page 202\]](#).

## 11.10 Delete a Replication Flow

Delete SAP Data Intelligence replication flows or their associated tasks using the [Replications](#) editor in the Modeler interface.

### Prerequisites

Before you delete a replication flow from the shared repository, you must undeploy it. For more information, see [Undeploy a Replication Flow \[page 200\]](#).

### Context

You can delete a replication flow from your user repository or from the shared repository. In either case, existing versions of the replication flow remain in the other repository.

If you delete or undeploy a replication flow, the system does not delete the target table or the tasks even if the target table was created by the replication management system.

- To delete a replication flow from your user repository, do the following:
  1. On the [Replications](#) tab of the Modeler, in the left pane right-click the replication flow to delete.
  2. Choose [Delete from user repository](#).
- To delete a replication flow from the shared repository, do the following:
  1. On the [Replications](#) tab of the Modeler, display the replication flow to delete.
  2. From the toolbar, choose the [Undeploy](#) icon.
  3. Confirm the undeployment.

## 11.11 Clean Up Source Artifacts

Some situations, such as when an administrator deletes a tenant or connection, can result in unused source artifacts that you need to delete to avoid performance issues.

For SAP S/4HANA cloud, deactivation of the communication channel (Communication Arrangement) results in automated clean-up.

For Microsoft Azure SQL database:

Situation	Result	Actions
A connection associated with a replication flow is deleted from Connection Management.	The replication flow no longer displays in the Modeler or Monitor.	<p>Re-create the connection.</p> <p>The audit log lists deleted connections with the event ConfigurationChange. See <a href="#">Viewing Audit Logs</a>.</p>
Either the tenant was deleted or a replication task was deleted, but the system cannot connect to the source system after three attempts.	Replication flow can no longer connect to the source.	<p>Run the following statements for each subscribed table to delete replication artifacts where:</p> <p><code>\${&lt;SourceSchema&gt;}</code> is the schema name of the source table.</p> <p><code>\${&lt;SourceTableName&gt;}</code> is the source table name.</p> <p><code>\${&lt;RmsSchema&gt;}</code> is the schema name of the replication artifacts, which is the same as the database user name that was specified when creating a source connection in Connection Management.</p> <p>Drop triggers:</p> <pre>DROP TRIGGER \${&lt;SourceSchema&gt;}.\${&lt;SourceTableName&gt;}_I_RMS_TRIG DROP TRIGGER \${&lt;SourceSchema&gt;}.\${&lt;SourceTableName&gt;}_D_RMS_TRIG DROP TRIGGER \${&lt;SourceSchema&gt;}.\${&lt;SourceTableName&gt;}_U_RMS_TRIG</pre> <p>Drop sequences:</p> <pre>DROP SEQUENCE \${&lt;RmsSchema&gt;}.RMS_\${&lt;SourceTableSchema&gt;}_\${&lt;SourceTableName&gt;}_SEQ1 DROP SEQUENCE \${&lt;RmsSchema&gt;}.RMS_\${&lt;SourceTableSchema&gt;}_\${&lt;SourceTableName&gt;}_SEQ2</pre> <p>Drop stored procedure:</p> <pre>DROP PROCEDURE \${&lt;RmsSchema&gt;}.RMS_\${&lt;SourceTableSchema&gt;}_\${&lt;SourceTableName&gt;}_PROC_V1</pre> <p>Drop shadow table for logging changed data:</p> <pre>DROP TABLE \${&lt;RmsSchema&gt;}.RMS_SHADOW_\${&lt;SourceTableSchema&gt;}_\${&lt;SourceTableName&gt;}</pre>

# 12 Monitoring SAP Data Intelligence

SAP Data Intelligence provides a stand-alone monitoring application to monitor the status of graphs run in the Modeler. The Monitoring application provides capabilities to visualize the summary of graphs run in the SAP Data Intelligence Modeler with relevant charts.

The Monitoring application also allows you to schedule graph runs. For each graph instance, the Monitoring application provides details for processes like the following:

- Graph run status.
- Time of graph run.
- Graph type.
- Graph source.

The Monitoring application allows you to open a graph in the Modeler, view graph configurations, or stop process runs.

You can also view the execution and configuration of replication flows and their associated tasks.

## Monitoring Policy

Developer member users assigned the `sap.dh.monitoring` policy can view analytics and instances of graphs for all tenant users. However, the policy doesn't provide member user access to schedules.

Without the policy, member users can monitor only their own graphs.

## Related Information

[Log in to SAP Data Intelligence Monitoring \[page 205\]](#)

[Using the Monitoring Application \[page 205\]](#)

[Pre-Delivered Policies](#)

## 12.1 Log in to SAP Data Intelligence Monitoring

Access the SAP Data Intelligence Monitoring application from the SAP Data Intelligence Launchpad or directly launch the application with a stable URL.

### Prerequisites

You must have administrator permission to perform this task.

### Procedure

1. Open the SAP Data Intelligence Launchpad URL in a browser.  
The welcome screen opens.
2. Log into the SAP Data Intelligence Launchpad using the following information:
  - Tenant name
  - Username
  - Password

For new instance, the tenant name is "default". For user name and password, use the credentials that you used when you created a service.

#### i Note

If you enter an incorrect password five consecutive times within a minute, your account is temporarily locked. Wait 10 seconds until your next attempt.

The SAP Data Intelligence Launchpad opens and displays the initial home page. To view user details, such as the tenant ID, select the profile icon in the upper right of the screen.

The home page displays the application tiles available in the tenant based on your assigned policies.

3. Select the *Monitoring* tile.

The Monitoring application opens to the *Analysis* tab.

## 12.2 Using the Monitoring Application

The SAP Data Intelligence Monitoring application offers the following capabilities.

Tenant administrator users, and developer member users with the sap.dh.monitoring policy can view all tenant users statistics. Member users without the monitoring policy can view information only for their graphs.

## i Note

Developer member users with the sap.dh.monitoring policy can filter the information based on specific users.




Actions	Description
Visualize various aspects of graph execution in the <i>Analytics</i> tab.	The <i>Analytics</i> tab is a dashboard that contains various tiles with targeted information about graphs. The following table describes each tile of the <i>Analytics</i> tab.

Tile	Description
<i>Status</i>	<p>A pie chart with the following information:</p> <ul style="list-style-type: none"> <li>The number of graph instances executed in the Modeler.</li> <li>The status of graph instances executed. Each sector in the pie chart represents a graph state.</li> </ul>
<i>Runtime Analysis</i>	<p>A scatter chart that includes a time period axis and duration of graph execution (in seconds) axis. This chart provides information on each graph execution and plots them in the scatter chart against the time and duration of execution.</p> <p>Each point in the chart represents a graph instance. Place your cursor on one instance point for more information about that particular graph instance.</p> <p>Configure the chart to view results for a selected time period. Use the filter in the chart header to select the time period (hour, day, and week).</p>
<i>Recently Executed</i>	Displays the top five instances and execution status. To view all the instances, select <i>Show All</i> .
<i>Memory Usage</i>	A line chart for the memory consumption of graphs. Use the filter to display by graph, status, and submission time. You can also see the resource usage in the last hour, day, 2 days, or set the custom time range for which to view the resource consumption.
<i>CPU Usage</i>	A line chart for the CPU consumption of graphs. Filter to display by graph, status, user, or submission time. Also see the resource usage in the last hour, day, 2 days, or set the custom time range for which you want to view the resource consumption.

Actions	Description
View execution details of individual graph instances in the <i>Instances</i> tab.	<div data-bbox="624 365 708 392" data-label="Section-Header"><b>i Note</b></div> <div data-bbox="624 414 1345 470" data-label="Text"> <p>The tiles in the <i>Analytics</i> tab don't include the information on the archived graph instances.</p> </div> <hr/> <div data-bbox="603 533 1396 589" data-label="Text"> <p>Use the <i>Instances</i> tab of the SAP Data Intelligence Monitoring application to view execution details of individual graph instances.</p> </div> <div data-bbox="624 622 708 649" data-label="Section-Header"><b>i Note</b></div> <div data-bbox="624 678 1366 804" data-label="Text"> <p>If you're a tenant administrator, you can see graph instances of all users. By default, there's a filter that shows only the administrator's graph instances. Modify the filter to view graph instances of select tenant users on which you can perform limited actions.</p> </div> <div data-bbox="624 860 708 887" data-label="Section-Header"><b>i Note</b></div> <div data-bbox="624 916 1377 972" data-label="Text"> <p>If you're a developer member user with the sap.dh.monitoring policy, you can see graph instances of all users just like a tenant administrator.</p> </div> <div data-bbox="603 1021 1396 1115" data-label="Text"> <p>For each graph instance, the Monitoring application provides information, such as the status of the graph execution, the graph execution name, the source of the graph in the repository, and the time of execution.</p> </div> <div data-bbox="603 1140 1396 1196" data-label="Text"> <p>To open the execution details pane, select a graph instance. The execution details pane displays more execution details that help you monitor the graph execution.</p> </div> <div data-bbox="624 1234 708 1261" data-label="Section-Header"><b>i Note</b></div> <div data-bbox="624 1290 1355 1415" data-label="Text"> <p>By default, a filter is applied to exclude the subgraphs and the archived instances from the <i>Instances</i> list. You can remove the filter by selecting the <b>×</b> icon that corresponds to a filter in the <i>Filters</i> bar. Alternately, select the <b>Filter</b> icon to set or remove filters.</p> </div>











Actions	Description
View subgraph execution details in the <i>Instances</i> tab.	<p>When a graph execution triggers or spawns the execution of another graph, the spawned execution is called a subgraph.</p> <p>You can filter the view of the Monitoring application to view execution details of all subgraph instances.</p> <p>To view subgraphs, remove the filter <i>Exclude Subgraphs</i> by selecting the <b>X Delete</b> icon that corresponds to a filter in the <i>Filters</i> bar. Or use the <b>Filter</b> icon to set or remove filters. The application refreshes the list view and displays all subgraph instances along with all other graph instances.</p> <div data-bbox="603 667 1394 786" style="background-color: #f0f0f0; padding: 5px;"> <p><b>i Note</b></p> <p>All subgraph instances are named as <i>subgraph</i>.</p> </div> <p>To open the execution details pane, select a subgraph instance. The subgraph instance pane displays additional execution details.</p>
View subgraph hierarchy details in the <i>Instances</i> tab.	<p>View the parent graph of a selected subgraph, or view the complete hierarchy of graph instances associated with the subgraph.</p> <ol style="list-style-type: none"> <li>1. Open the <i>Instances</i> tab.</li> <li>2. Select the <b>••• Select an Action</b> icon next to the applicable subgraph instance.</li> <li>3. Select <i>Show Hierarchy</i>. The Hierarchy dialog box opens displaying all graph instances associated with the selected subgraph in hierarchical order.</li> <li>4. Select an applicable graph.</li> <li>5. Select <i>View details</i>.</li> </ol>
Filter graph instances in the <i>Instances</i> tab.	<p>Use the filter tool to control the graphs to view:</p> <ol style="list-style-type: none"> <li>1. Open the <i>Instances</i> tab.</li> <li>2. Select the <b>Filter</b> <i>Add Filter</i> icon.</li> <li>3. Define the required filter conditions. Filter the listed graphs based on attributes, such as the source name, execution status, time of execution and more.</li> <li>4. To filter and view graph instances executed on the same day, hour, or week, select <i>Hour</i>, <i>Day</i>, or <i>Week</i> in the menu bar and the view changes to records for that time frame.</li> </ol> <p>If you're a tenant administrator, you can modify the filter to view graph instances of other users.</p> <p>Developer member users can modify the filter to view graph instances of other users when they are assigned the sap.dh.monitoring policy.</p>

Actions	Description
Open source graph in the <i>Instances</i> tab.	<p>For any selected graph instance, launch the source graph in the Modeler application from the Monitoring application.</p> <ol style="list-style-type: none"> <li>1. Open the <i>Instances</i> tab.</li> <li>2. Select an instance. The application opens a graph overview pane at right that displays the source graph of the selected graph instance.</li> <li>3. Select <i>Open Source Graph</i> in the menu bar of the overview pane. The Monitoring application opens the Modeler application in a new browser tab with the source graph of the selected instance opened in the graph editor.</li> <li>4. View or edit the graph configuration.</li> </ol>
View graph configurations in the <i>Instances</i> tab.	<p>For any selected graph instance, you can view the configurations defined for its source graph.</p> <ol style="list-style-type: none"> <li>1. Open the <i>Instances</i> tab.</li> <li>2. Select an instance. The application opens a graph overview pane at right that displays the source graph of the selected graph instance.</li> <li>3. Select the  <i>Show Configuration</i> icon. The application opens the <i>Configuration</i> pane at right where you can view read-only graph configuration information.</li> <li>4. <b>Optional:</b> Right-click an operator in the graph overview pane and choose <i>Open Configuration</i>. The <i>Configuration</i> pane shows read-only operator configuration information.</li> </ol>
Stop a graph instance execution in the <i>Instances</i> tab.	<p>Stop the execution of a graph instance when it's the running or pending state.</p> <ol style="list-style-type: none"> <li>1. Open the <i>Instances</i> tab.</li> <li>2. Select a graph instance that has a running status. The graph overview pane opens at right.</li> <li>3. Select the  <i>Stop Execution</i> icon in the menu bar.</li> </ol>
Search graph instances in the <i>Instances</i> tab.	<p>To search for any graph instance, use the search bar in the <i>Instances</i> tab. Search for any graph instance based on the instance name or its source graph name.</p>
Archive graph instance in the <i>Instances</i> tab.	<p>Archive a completed or dead graph instance from the Pipeline Engine only.</p> <ol style="list-style-type: none"> <li>1. Open the <i>Instances</i> tab.</li> <li>2. Select the  <i>Select an Action</i> icon in the <i>Actions</i> column of the applicable instance.</li> <li>3. Select <i>Archive</i>.</li> </ol>

### **i** Note

The archived instances remain in the system for a default period of 90 days. The tenant administrator can configure the retention time via the System Management application.

Actions	Description
Download diagnostics information and view logs in the <a href="#">Instances</a> tab.	<p>Use the Monitoring application to download graph diagnostics information and view logs to help diagnose and solve errors with graphs.</p> <p>View logs generated for certain operators, such as the data workflow operator, when you execute the graph.</p> <ol style="list-style-type: none"> <li>1. Open the <a href="#">Instances</a> tab.</li> <li>2. Select the <b>...</b> <a href="#">Select an Action</a> icon in the <a href="#">Actions</a> column of the applicable instance.</li> <li>3. Select <a href="#">Download Diagnostic Info</a>.</li> </ol> <p>The application downloads a zipped archive of graph information automatically.</p> <p>For certain operators, such as the data workflow operators, view logs generated for the operator execution. To view these logs:</p> <ol style="list-style-type: none"> <li>1. Open the <a href="#">Instances</a> tab.</li> <li>2. Select the graph. The graph overview pane opens at right.</li> <li>3. Select the <a href="#">Process Logs</a> tab in the lower pane. The <a href="#">Logs</a> tab contains a list of logs for the selected graph.</li> <li>4. Use the filter lists at the top of the tab to filter for specific groups and processes, or search for the specific log. Scroll through the processes to read the log texts.</li> </ol>
Manage schedules in the <a href="#">Schedules</a> tab.	<p>View graphs that are scheduled for execution and create, edit, or delete schedules.</p> <p>For more information about scheduling graphs for execution, see “Schedule Graph Executions” in the <i>Modeling Guide</i>.</p> <p>A tenant administrator can view their schedules and the schedules of other users in the Monitoring application. If you're a tenant administrator, you can perform the following tasks in the <a href="#">Schedules</a> tab on other users' schedules:</p> <ul style="list-style-type: none"> <li>• View schedules by applying filters.</li> <li>• Edit or stop schedules.</li> <li>• Suspend or resume schedules.</li> <li>• View executed instances of schedules.</li> </ul>

Actions	Description
Monitor the execution of replication flows in the <i>Replications</i> tab.	<p>View information about replication data flows, including connection details, load progress, and user information.</p> <p>For tasks, the <i>Replications</i> tab displays information, such as the number of tasks in different states, priority settings, number of operations and partitions, and execution timestamps. (The number of operations displayed may be higher than the actual amount of records transferred as data may be transferred twice in case of error situations.) Use the <i>Search</i> box in the menu bar to filter the list of replications on any metadata value.</p> <p>Select a replication flow to perform the following tasks:</p> <ul style="list-style-type: none"> <li>• Display the replication in the Modeler application by selecting <i>Go to Monitoring</i> in the menu bar.</li> <li>• Select the  <i>Settings</i> icon to configure the following settings: <ul style="list-style-type: none"> <li>• <i>Replication Priority</i>: Select <i>High</i>, <i>Medium</i>, or <i>Low</i>. Selecting <i>High</i> runs this replication flow before others. The default is <i>Medium</i>.</li> <li>• <i>Source Maximum Connections</i>: Increase or limit the number of source connections permitted to the source. The default is 10.</li> <li>• <i>Target Maximum Connections</i>: Increase or limit the number of target connections permitted to the target. The default is 10.</li> </ul> </li> <li>• To start the replication flow, select the  <i>Start Execution</i> icon.</li> <li>• To suspend the execution of the replication flow, select the  <i>Suspend Execution</i> icon.</li> <li>• To refresh the list of replication flows, select the  <i>Refresh</i> icon.</li> <li>• Select a replication to display the tasks in the lower pane. Perform the following tasks in the <i>Tasks</i> area, after selecting a task: <ul style="list-style-type: none"> <li>• Select the  <i>Settings</i> icon to configure <i>Task Priority</i> for the task. For example, select <i>High</i> to run this task before others. The default is <i>Medium</i>.</li> <li>• Select the  <i>Start Execution</i> icon to run the task.</li> <li>• Select the  <i>Suspend Execution</i> icon to suspend execution of the task.</li> <li>• Select the  <i>Refresh</i> icon to refresh the list of tasks.</li> </ul> </li> </ul>

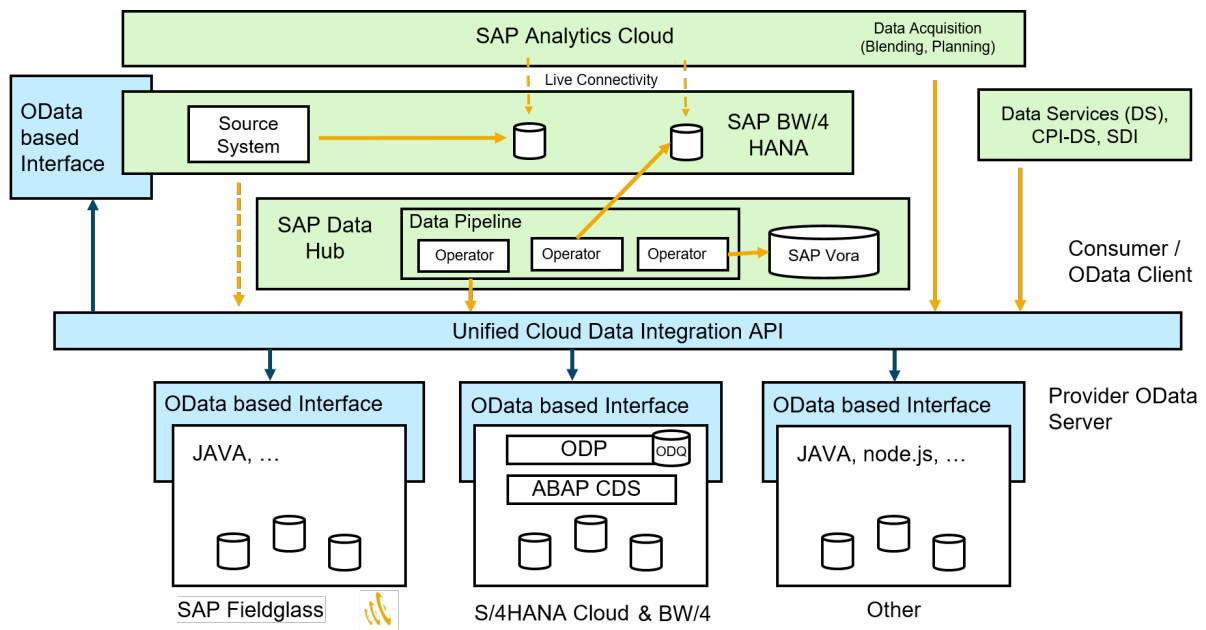
## Related Information

[Schedule Graph Executions \[page 84\]](#)

# 13 Integrating SAP Cloud Applications with SAP Data Intelligence

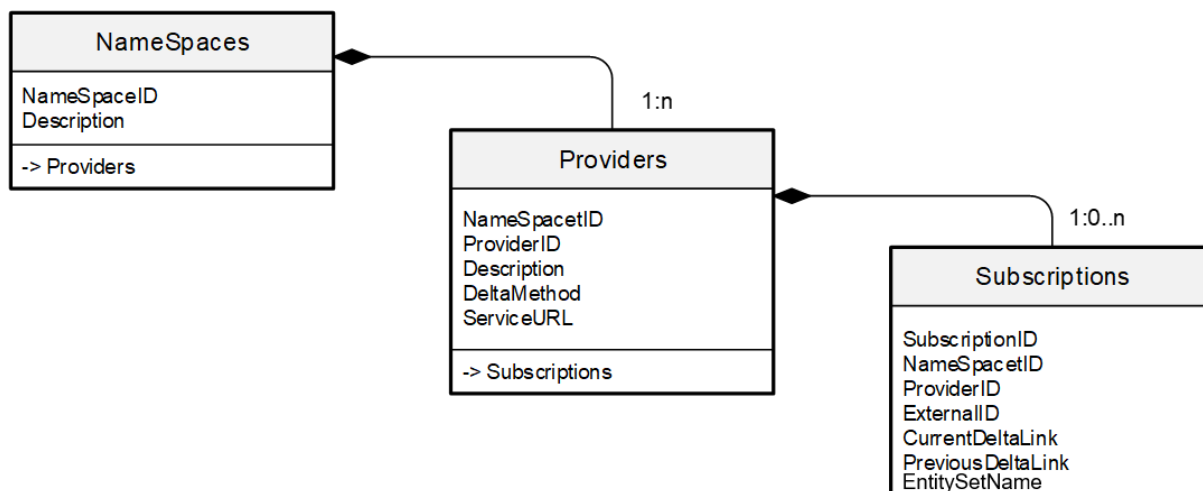
SAP Data Intelligence enables you to integrate SAP cloud applications with SAP Data Intelligence for holistic data management. The integration is supported using the Cloud Data Integration API, which is based on the OData V4 specifications.

## SAP Cloud Application Data Integration – Providers and Consumers



## Cloud Data Integration API

SAP cloud applications implement the Cloud Data Integration API, which consists of two types of OData services. The first type is the Administrative service that exists once per system. The structure of the administrative service is as below.



The administrative service provides a catalog of providers that are organized along namespaces. Each provider represents a business object and consists of one more entity sets that semantically belong together.

### i Note

Every implementation must produce the same \$metadata.

The second type of service exists once per data provider, and the structure of these providers depends on the entity sets of the provider. This service provides access to the metadata and the data of the entity sets.

In the current version, you can integrate the following SAP Cloud applications with SAP Data Intelligence.

- SAP Fieldglass
- SAP Sales Cloud
- SAP Service Cloud

## Creating a Connection

In the SAP Data Intelligence Connection Management application, you can use the connection type, `CLOUD_DATA_INTEGRATION` to create connections to SAP Cloud applications. In the connection definition, configure the service endpoints. For more information see, [CLOUD\\_DATA\\_INTEGRATION](#)

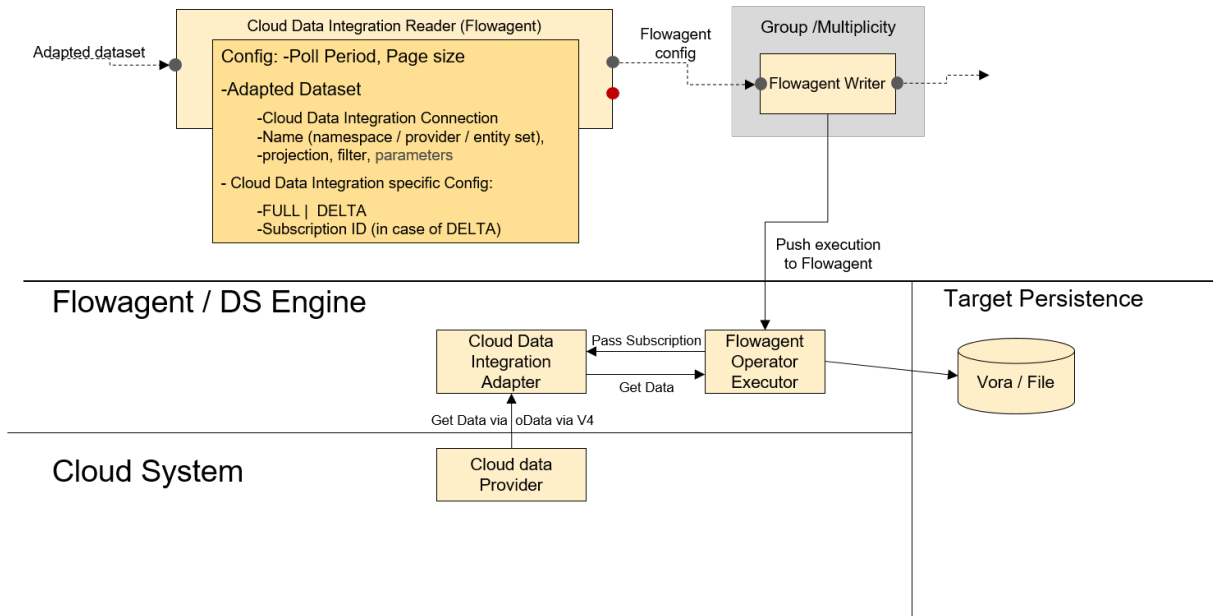
## Capabilities

Creating a connection to an SAP Cloud application enables you to use entity sets of the service providers as datasets in the Metadata explorer. Additionally, the connection also maps the namespaces and providers to folders in the Metadata Explorer. As a result, you can view the metadata of entity sets and preview them in the Metadata Explorer application.

You can use the Cloud Data Integration Operator in the SAP Data Intelligence Modeler for data ingestion. For more information see, [Cloud Data Integration Consumer](#)

The following graphical representation shows a snapshot of streaming replications using the Flowagent.

## Streaming Pipeline



# 14 Service-Specific Information

## Related Information

[Alibaba Cloud Object Storage Service \(OSS\) \[page 216\]](#)

[Amazon Simple Storage Service \(AWS S3\) \[page 220\]](#)

[Google Cloud Storage \(GCS\) \[page 223\]](#)

[Hadoop Distributed File System \(HDFS\) \[page 225\]](#)

[Microsoft Azure Data Lake \(ADL\) \[page 226\]](#)

[Microsoft Azure Blob Storage \(WASB\) \[page 228\]](#)

[Local File System \(/file\) \[page 230\]](#)

[WebHDFS \[page 230\]](#)

## 14.1 Alibaba Cloud Object Storage Service (OSS)

Many of the SAP Data Intelligence storage operators support the Alibaba Cloud OSS, and there are common characteristics that this service has across the operators.

### i Note

This topic sometimes refers to an “object” as a “file”, and to an object's “prefix” as a “directory” when the term fits the context of the operator.

## Connection

To use any operator that connects to Alibaba OSS, either use a connection ID (Connection Management application) or set a manual connection using the values in the following table.

Value	Description	Parameters
Endpoint	<b>Required.</b> Allows using an endpoint to access Alibaba Cloud OSS.	<ul style="list-style-type: none"><li>• <b>ID:</b> endpoint</li><li>• <b>Type:</b> string</li><li>• <b>Default:</b> “oss-cn-hangzhou.aliyuncs.com”</li></ul>



Value	Description	Parameters
Protocol	<b>Required.</b> Sets the protocol to use. This value overwrites the protocol prefixed in the endpoint configuration, if any.	<ul style="list-style-type: none"> <li>• <b>ID:</b> Protocol</li> <li>• <b>Type:</b> string</li> <li>• <b>Default:</b> "HTTPS" Possible values include "HTTP" or "HTTPS".</li> </ul>
Region	The Alibaba Cloud region to which the configured bucket belongs. The bucket name is in the root path.	<ul style="list-style-type: none"> <li>• <b>ID:</b> region</li> <li>• <b>Type:</b> string</li> <li>• <b>Default:</b> "oss-cn-hangzhou"</li> </ul>
Access Key	<b>Required.</b> The Access Key ID to authenticate to the service. To authenticate, the Access Key ID pairs with the Secret Key.	<ul style="list-style-type: none"> <li>• <b>ID:</b> accessKey</li> <li>• <b>Type:</b> string</li> <li>• <b>Default:</b> "OSSAccessKeyID"</li> </ul>
Secret Key	<b>Required.</b> The Secret Key to authenticate to the service. To authenticate, the Secret Key pairs with the Access Key.	<ul style="list-style-type: none"> <li>• <b>ID:</b> secretKey</li> <li>• <b>Type:</b> string</li> <li>• <b>Default:</b> ""</li> </ul>
Root Path	The bucket name and an optional root path name for browsing. The path starts with a forward slash (/) and then the bucket name. Optionally add another forward slash and the root path.	<ul style="list-style-type: none"> <li>• <b>ID:</b> rootPath</li> <li>• <b>Type:</b> string</li> <li>• <b>Default:</b> "/ &lt;MyBucket&gt; / &lt;My Folder&gt;"</li> </ul>

**❖ Example**

`/ <MyBucket> / <My Folder>`

Dataset names for this connection don't contain segments of the rootPath; instead, the first segment of the dataset is a subdirectory of the root path.

## Permissions

Permissions in Alibaba Cloud are required to operate over Alibaba Cloud OSS objects. For more information, see [Access and Control](#) in the Alibaba Cloud OSS documentation.

Alibaba OSS provides an Access Control List (ACL) for bucket-level access control. One of the following permissions are required:

- public-read-write
- public-read
- private

For descriptions of the permissions, see [ACL types](#) in the Alibaba Cloud OSS documentation.

If you don't set an ACL for a bucket when you create it, the bucket's ACL is set to private automatically. If the ACL rule of the bucket is set to private, only authorized users can access and operate on objects in the bucket. To authorize other users to access your Alibaba Cloud OSS resources, refer to [Bucket Policy](#) in the Alibaba Cloud OSS documentation.

## Read File Permissions

To read a single object ("file"), you need the permission `oss:GetObject` for the given object. For more information, see [GetObject](#) in the Alibaba Cloud OSS documentation.

To read multiple objects in a prefix ("directory"), you need the permission `oss:GetBucket` for the bucket where the prefix is to be listed. The permission can be narrowed to a directory inside the bucket, and the prefix is subject to this restriction. For more information, see [GetBucket \(ListObjects\)](#) in the Alibaba Cloud OSS documentation.

## Write File Permissions

To write an object ("file"), you need the permission `oss:PutObject` for the bucket to receive the object.

If using mode "Append", you also need `oss:GetObject` for the given object.

## Remove File Permissions

To remove an object ("file"), you need the permission `oss:DeleteObject` for the given object.

## Move File Permissions

Because moving consists of copying and removing in Alibaba Cloud OSS, you need the permissions documented in [Remove File Permissions](#) and [Copy File Permissions](#).

## Copy File Permissions

To copy an object ("file"), you need the following permissions:

- `oss:GetObject` for the source object.
- `oss:PutObject` for the bucket to receive the copied object.

For more information, see [Alibaba Cloud OSS Multipart Upload Operations documentation](#).

If copying by prefix ("directory"), the operation is bound to the same permissions documented in [Read File Permissions](#).

## Restrictions

### Directory Restrictions

- Directories: For a path to be interpreted as a directory, end the path with a forward slash (/). For example: `/tmp/` is a directory, while `/tmp` is a file named **tmp**.
- Working directory: Because there's no concept of a "working directory", any relative directory given to or by this service has the root directory (/) as working directory.

## Write File Restrictions

Alibaba Cloud OSS API doesn't support the "Append" mode. If you use the "Append" mode, the operation retrieves the whole file from the service and then writes the data back to OSS, which compromises the operation's efficiency.

## Move File Restrictions

Alibaba Cloud OSS API doesn't support the "move" operation. If you use the move operation, the operation copies the file and then removes the source file. However, if there's a failure, the operation may copy the file but not remove the source file.

## Copy File Restrictions

Because the "copy" operation has a source and a destination path, the following restrictions apply:

- If the destination is a file, the source must also be a file.
- If the destination is a directory, the directory must be empty.

### ❁ Example

In the given file structure:

```
.
|
+-- a
|   +-- file1.txt
|   +-- file2.txt
+-- b
    +-- f1.txt
    +-- f2.txt
```

The copy operation has the following results:

- Copying source `a/file1.txt` to destination `newfile.txt` succeeds because the destination doesn't exist.
- Copying source `a/file1.txt` to destination `b/f1.txt` succeeds and overwrites `b/f1.txt`, because the destination is an existing file.
- Copying source `a/file1.txt` to destination: `b/` fails because `b/` already exists and is empty.
- Copying source `a/` to destination `b/` fails because `b/` already exists and isn't empty.
- Copying source `a/` to destination `b/dir/` succeeds because `b/dir/` doesn't exist.

## Related Information

[Alibaba Cloud's Official Website](#) ➔

## 14.2 Amazon Simple Storage Service (AWS S3)

Many of the SAP Data Intelligence connectors support AWS S3, and there are common characteristics that this service has across the operators.

### i Note

This topic sometimes refers to an “object” as a “file”, and to an object's “prefix” as a “directory” when the term fits the context of the operator.

AWS S3 is an object store service, which is documented in the [AWS S3](#) Website.

### ! Restriction

SAP has tested the following services to ensure that they support AWS S3 API: Rook, Minio, and Swift. SAP doesn't guarantee any other service that supports the AWS S3 API to be compatible.

## Connection

To use any operator that connects to AWS S3, use a connection ID from the Connection Management application. Set further connection configurations, which aren't in the Connection Management application, described in the following table.

Value	Description	Parameters
Bucket	Optional bucket name to access. The Bucket works as a “fallback” of the connection root path configuration.  <b>❖ Example</b> If there's no bucket name, in the root path, the value from the Bucket is used.	<ul style="list-style-type: none"><li>• <b>ID:</b> awsBucket</li><li>• <b>Type:</b> string</li><li>• <b>Default:</b> “com.sap.datahub.test”</li></ul>
Proxy	Optional proxy to use in the connection to the service.	<ul style="list-style-type: none"><li>• <b>ID:</b> awsProxy</li><li>• <b>Type:</b> string</li><li>• <b>Default:</b> “”</li></ul>
Use SSL	Specifies whether to use SSL/TLS when connecting to the service.	<ul style="list-style-type: none"><li>• <b>ID:</b> useSSL</li><li>• <b>Type:</b> Boolean</li><li>• <b>Default:</b> true</li></ul>

## Permissions

Permissions in AWS are required to operate over AWS S3 objects. Each operator can require a determined set of permissions to operate successfully.

### Read File Permissions

To read an object ("file") or objects, you need the following permissions:

- `s3:GetObject` for the given object. See also [AWS S3 GET Object](#) in the AWS documentation.
- `s3:GetObjectVersion` for the given object.
- `s3:ListBucket` to read multiple objects in a prefix ("directory"). AWS requires the permission for the bucket where the prefix is to be listed.

#### Note

You can narrow the permission to a directory inside the bucket, and the prefix is subject to this restriction. For more information, see [ListBuckets](#) in the AWS documentation.

- `s3:DeleteObject` for the given object when you use *Delete After Send*. For more information, see [DELETE Object](#) in the AWS documentation.

### Write File Permissions

To write an object ("file"), you need the permission `s3:PutObject` for the bucket to receive the object. For more information, see [Policies and Permissions in Amazon S3](#) in the AWS documentation.

If you use the mode "Append", you also need the permission `s3:GetObject` for the given object. This permission is required because of the write file permission restriction documented in [Restrictions \[page 218\]](#). For more information, see [GetObject](#) in the AWS documentation.

### Remove File Permissions

To remove an object ("file"), you need the permission `s3:DeleteObject` for the given object. For more information, see [DeleteObject](#) in the AWS documentation.

### Move File Permissions

Because moving files consists of copying and removing in AWS S3, use the permissions documented in [Remove File Permissions](#) and [Copy File Permissions](#) sections.

### Copy File Permissions

To copy an object ("file"), you need the following permissions:

- `s3:GetObject` for the source object.
- `s3:PutObject` for the bucket to receive the copied object.  
If copying by prefix ("directory"), the operation is bound to the same permissions documented in [Read File Permissions](#).

## Restrictions

### Directory Restrictions

- Directories: For a path to be interpreted as a directory, end the path with a forward slash (/). For example: /tmp/ is a directory, while /tmp is a file named **tmp**.
- Working directory: Because there's no concept of a "working directory", any relative directory given to or by this service has the root directory (/) as working directory.

### Write File Restrictions

AWS S3 API doesn't support the "Append" mode. If you use the "Append" mode, the operation retrieves the whole file from the service, and then writes the data back to AWS S3, which compromises the operation's efficiency.

### Move File Restrictions

AWS S3 API doesn't support the "move" operation. If you use the "Move" operation, the operation copies the file and then removes the source file. However, if there's a failure, the operation may copy the file but not remove the source file.

### Copy File Restrictions

Because the "copy" operation has a source and a destination path, the following restrictions apply:

- If the destination is a file, the source must also be a file.
- If the destination is a directory, the directory must be empty.

#### ❖ Example

In the give file structure:

```
.
|
+-- a
|   +-- file1.txt
|   +-- file2.txt
+-- b
    +-- f1.txt
    +-- f2.txt
```

The copy operation has the following results:

- Copying source a/file1.txt to destination newfile.txt succeeds because the destination doesn't exist.
- Copying source a/file1.txt to destination b/f1.txt succeeds and overwrites b/f1.txt because the destination is an existing file.
- Copying source a/file1.txt to destination b/ fails because b/ already exists and isn't empty.
- Copying source a/ to destination b/ fails because b/ already exists and is empty.
- Copying source a/ to destination b/dir/ succeeds because b/dir/ doesn't exist.

## Related Information

[Amazon S3 Owner's Page](#) ➔

## 14.3 Google Cloud Storage (GCS)

GCS is Google's Object Storage cloud service. Additional information, including the documentation, can be found at the official GCS Homepage.

Many of the Storage operators offer support for GCS. This documentation regards the common characteristics that this service has across operators.

This document may refer to an object as a "file", and to an object's prefix as a "directory", if it fits the context of the operator.

## Connection

### i Note

In order to use any operator that connects to GCS, you may use a Connection ID from the Connection Management.

Further connection configurations may be set, which are not in the Connection Management. Such as:

- **Bucket**  
Optional bucket name to be accessed. It works as a "fallback" of the Connection's Root Path configuration. For instance, if no bucket is given in the Root Path, the value from Bucket is used.
  - ID: `gcsBucket`
  - Type: `string`
  - Default: `"bucket"`
- **Content Type**  
Informs the type of data being sent, allowing the correct rendering of objects. For instance, if you send a JSON file, this should be set to `application/json`. Additional information here: [Working With Object Metadata](#) ➔ .
  - ID: `gcsContentType`
  - Type: `string`
  - Default: `""`

## Permissions

[GCS Permissions](#) ➔ for manipulating objects are described in the [Access Control List](#) ➔ documentation as WRITER, READER and OWNER. Each operator may require a determined set to successfully operate.

## Read File Permissions

To read a single object ("file"), you need `READER` and `WRITER` permissions on the bucket.

## Write File Permissions

To write an object ("file"), you need `WRITER` permission on the bucket.

## Remove File Permissions

To remove an object ("file"), you need the `READER` and `WRITER` permissions on the bucket.

## Move File Permissions

To move an object ("file"), you need the `READER` and `WRITER` permissions on the origin bucket plus the `READER` and `WRITER` permissions on the destination bucket.

## Copy File Permissions

To copy an object ("file"), you need the `READER` permission on the origin bucket, plus the `READER` and `WRITER` permission on the destination bucket.

## Restrictions

- Directories:  
In order for a path to be interpreted as a directory, it should end with `/`. For example: `/tmp/` is a directory, while `/tmp` is a file named `tmp`.
- Working directory:  
Since there is no concept of a "working directory", any relative directory given to/by this service will have the root directory (`/`) as working directory.

## Move File Restrictions

As the GCS API does not support the move operation, the operation consists of a copy followed by removing the source file. Thus, in cases of failure, the file may be copied and not removed.

## Copy File Restrictions

Taking that the operation has a "source" and a "destination" path:

- If the destination is a file, source must also be a file.
- If the destination is a directory, it must be empty.

For instance, in the given file structure:

```
.
|
+-- a
|   +-- file1.txt
|   +-- file2.txt
+-- b
    +-- f1.txt
    +-- f2.txt
```

- Copying source: `a/file1.txt` to destination: `newfile.txt`, would succeed, since the destination does not exist.



- Copying source: `a/file1.txt` to destination: `b/f1.txt`, would succeed and overwrite `b/f1.txt`, since the destination is an existing file.
- Copying source: `a/file1.txt` to destination: `b/`, would fail, since `b/` already exists and is not empty.
- Copying source: `a/` to destination: `b/` would fail, since `b/` already exists and is not empty.
- Copying source: `a/` to destination: `b/dir/` would succeed, since `b/dir/` does not exist.

## Related Information

[Official GCS Documentation](#) ➔

## 14.4 Hadoop Distributed File System (HDFS)

Hadoop Distributed File System is Apache's distributed storage solution. For more information, see the official HDFS documentation.

### i Note

Some configurations are only supported with connections defined via Connection Management.

Many of the SAP Data Intelligence storage operators offer support for HDFS. This documentation covers the common characteristics that this service has across operators.

## Connection

In order to use any operator that connects to HDFS, you may use a Connection ID from the Connection Management.

## Permissions

The [HDFS Permissions](#) ➔ for files and directories are based on the POSIX model, that is, for each file or directory there are W, R and X permissions that may be attributed separately to the owner, the group associated with the file/directory and the group of remaining users.

For a finer control, it is possible to define an [Access Control List](#) ➔, which allows the definition of specific rules for each user or each group of users.

### Read File Permissions

To read a file, you need `w` and `r` permissions on the file.

## Write File Permissions

To write a new file, you need `w` permission on the directory where the file will be created.

To append or overwrite an existing file, you need `w` permission on the file.

## Remove File Permissions

To remove a file or directory, you need `w` and `x` permissions on the corresponding file/directory.

## Move File Permissions

- **Moving a File:**  
To move a file you need `w` permission on the original file and `x` permission on the original directory.  
If the destination file already exists and is being overwritten, you need `w` permission on the destination file.  
On the other hand, if the file does not exist on the destination, you need `w` permission on the destination directory.
- **Moving a Directory:**  
To move a directory, you need `w` and `x` permissions on the original directory and `w` permission on every file within it.  
On the destination folder, you need `w` permission.  
If any of the files already exist at the destination and needs to be overwritten, you need the `w` permission on the destination file as well.

## Restrictions

- **Working directory:**  
Since there is no concept of a "working directory", any relative directory given to/by this service will have the root directory (`/`) as working directory.

## Copy File Restrictions

Since the HDFS API does not support the copy operation, this behavior can be achieved through `Read + Write`.

## Related Information

[Official HDFS Documentation](#) 

## 14.5 Microsoft Azure Data Lake (ADL)

Azure Data Lake (ADL) is Microsoft's Data Lake cloud storage service. Additional information, including the documentation, can be found at the official ADL Homepage.

Many of the SAP Data Intelligence storage operators offer support for ADL. This documentation covers the common characteristics that this service has across operators.

## Connection

In order to use any operator that connects to ADL, you may use a Connection ID from the Connection Management.

## Permissions

The ADL interface is based on the WebHDFS service, thus the [ADL Permissions](#) for files and directories are based on the POSIX model, that is, for each file or directory there are W, R and X permissions that may be attributed separately to the owner, the group associated with the file/directory and the group of remaining users.

For a finer control, it is possible to define an [Access Control List](#), which allows the definition of specific rules for each user or each group of users.

### Read File Permissions

To read a file, you need `w` and `r` permissions on the file.

### Write File Permissions

To write a new file, you need `w` permission on the directory where the file will be created.

To append or overwrite an existing file, you need `w` permission on the file.

### Remove File Permissions

To remove a file or directory, you need `w` and `r` permissions on the corresponding file/directory.

### Move File Permissions

- Moving a File:  
To move a file you need `w` permission on the original file and `r` permission on the original directory.  
If the destination file already exists and is being overwritten, you need `w` permission on the destination file.  
On the other hand, if the file does not exist on the destination, you need `w` permission on the destination directory.
- Moving a Directory:  
To move a directory, you need `w` and `r` permissions on the original directory and `w` permission on every file within it.  
On the destination folder, you need `w` permission.  
If any of the files already exist at the destination and needs to be overwritten, you need the `w` permission on the destination file as well.

## Restrictions

- Working directory: Because there is no concept of a "working directory," any relative directory given to or used by this service will have the root directory (`/`) as its working directory.

- Copying: Because the ADL API does not support the copy operation, this behavior can be achieved through Read + Write.

## Related Information

[Official ADL Homepage](#) ↗

## 14.6 Microsoft Azure Blob Storage (WASB)

Azure Storage Blob (WASB) is one of Microsoft's cloud storage services. Additional information, including the documentation, can be found at the official WASB homepage.

Many of the SAP Data Intelligence storage operators offer support for WASB. This documentation covers the common characteristics that this service has across operators.

This document may refer to an object as a "file" and to an object's prefix as a "directory" if it fits the context of the operator.

## Connection

To use any operator that connects to WASB, you may use a Connection ID from the Connection Management.

Further connection configurations may be set, which are not in the Connection Management:

- Container. Optional container name to be accessed. It works as a "fallback" of the Connection's Root Path configuration. For example, if no bucket is given in the Root Path, the value from Container is used.
  - ID: `containerName`
  - Type: `string`
  - Default: `"mycontainer"`
- Blob Type. Only used in Write File operator. It sets the blob type of the destination blob ("file").
  - ID: `wasbBlobType`
  - Type: `string`
  - Default: `"BlockBlob"`
  - Values:
    - `"BlockBlob"`
    - `"PageBlob"`
    - `"AppendBlob"`

## Permissions

Permissions in Azure Blob Storage are required to operate over blobs. WASB currently restricts access to blobs through the container's policy:

- Full public read access
- Public read access for blobs only
- No public read access

Learn more at [Set Container ACL](#) and [Authorize requests to Azure Storage](#).

Operators will need full access to the data, thus the container should have "Full public read access" if the given credentials are not from the owner of the container; otherwise, any permission should be enough.

## Restrictions

- Directories. For a path to be interpreted as a directory, it should end with /. For example: /tmp/ is a directory, while /tmp is a file named tmp.
- Working directory. Because there is no concept of a "working directory", any relative directory given to or by this service will have the root directory (/) as working directory.

### Move File Restrictions

As the WASB API does not support the move operation, the operation consists of a copy followed by removing the source file. Thus, in cases of failure, the file may be copied and not removed.

### Copy File Restrictions

Given that the operation has a "source" and a "destination" path:

- If the destination is a file, source must also be a file.
- If the destination is a directory, it must be empty.

For instance, consider this file structure:

```
.
|
+-- a
|   +-- file1.txt
|   +-- file2.txt
+-- b
    +-- f1.txt
    +-- f2.txt
```

- Copying the source a/file1.txt to the destination newfile.txt would succeed because the destination does not exist.
- Copying the source a/file1.txt to the destination b/f1.txt would succeed and overwrite b/f1.txt because the destination is an existing file.
- Copying the source a/file1.txt to the destination: b/ would fail because b/ already exists and is not empty.
- Copying the source a/ to the destination b/ would fail because b/ already exists and is not empty.

- Copying the source `a/` to the destination `b/dir/` would succeed because `b/dir/` does not exist.

## Related Information

[Official WASB Homepage](#) 

## 14.7 Local File System (/file)

Many of the SAP Data Intelligence storage operators offer support for the local file system.

The local file system is subject to the cluster's file system. The `/files/` folder seen in the System Management application can be accessed through the `/vrep/` path. For example: `/files/myfile.txt` becomes `/vrep/myfile.txt`.

## Restrictions

- **/vrep is deprecated:** The usage of the `/vrep` folder in graphs is deprecated. The `/vrep` path is not meant to be used for writing and reading files. If your graphs and operators need temporary storage, consider using the `/vrep` connection type only to access a folder local to the graph container. If you need to share files between subengines or groups, consider using blob storage such as S3.
- **Temporary files:** Operators can write files to the `/tmp` folder, which is a folder local to the graph container. However, `/tmp` is not a shared folder, so when working with groups and subengines it can't be used for file sharing.
- **Working directory:** For any path that is given to or by this service, the current working directory (`.`) will be the HOME directory of the user running the local graph container. The HOME directory is `/home/vflow/` in the base docker image. You can't access other directories using absolute paths (`..`) to access the parent directory, because the user `/vflow` doesn't have the necessary permission.
- **Copy file:** Because the local file system API doesn't support the `copy` operation, you must copy files using `Read + Write`.

## 14.8 WebHDFS

WebHDFS supports Hadoop Distributed File System through the REST API. It is one of the protocols of Apache's distributed storage solution. For more information, see the official WebHDFS home page.

### i Note

Some configurations are only supported with connections defined via Connection Management.

Many of the SAP Data Intelligence storage operators offer support for WebHDFS. This documentation covers the common characteristics that this service has across operators.

## Connection

In order to use any operator that connects to WebHDFS, you may use a Connection ID from the Connection Management.

Further connection configurations may be set, which are not in the Connection Management. Such as:

- Token  
The Token to authenticate to WebHDFS with.
  - ID: `webhdfsToken`
  - Type: `string`
  - Default: `""`
- OAuth Token  
The OAuth Token to authenticate to WebHDFS with.
  - ID: `webhdfsOAuthToken`
  - Type: `string`
  - Default: `""`
- Do As  
The user to impersonate. Has to be used together with `user`.
  - ID: `webhdfsDoAs`
  - Type: `string`
  - Default: `""`

## Permissions

The [WebHDFS Permissions](#) for files and directories are based on the POSIX model, that is, for each file or directory there are W, R and X permissions that may be attributed separately to the owner, the group associated with the file/directory and the group of remaining users.

For a finer control, it is possible to define an [Access Control List](#), which allows the definition of specific rules for each user or each group of users.

### Read File Permissions

To read a file, you need `w` and `r` permissions on the file.

### Write File Permissions

To write a new file, you need `w` permission on the directory where the file will be created.

To append or overwrite an existing file, you need `w` permission on the file.

### Remove File Permissions

To remove a file or directory, you need `w` and `r` permissions on the corresponding file/directory.

## Move File Permissions

- Moving a File:  
To move a file you need `w` permission on the original file and `x` permission on the original directory.  
If the destination file already exists and is being overwritten, you need `w` permission on the destination file.  
On the other hand, if the file does not exist on the destination, you need `w` permission on the destination directory.
- Moving a Directory:  
To move a directory, you need `w` and `x` permissions on the original directory and `w` permission on every file within it.  
On the destination folder, you need `w` permission.  
If any of the files already exist at the destination and needs to be overwritten, you need the `w` permission on the destination file as well.

## Restrictions

- Working directory:  
Since there is no concept of a "working directory", any relative directory given to/by this service will have the root directory (`/`) as working directory.

## Copy File Restrictions

Since the WebHDFS API does not support the copy operation, this behavior can be achieved through `Read + Write`.

## Related Information

[Official WebHDFS Homepage](#) ➔



# 15 Changing Data Capture (CDC)

SAP Data Intelligence supports CDC technology for some connection types.

## Databases

The following databases support CDC capability by using a trigger-based approach for capturing changes:

- DB2
- HANA
- MSSQL
- MySQL
- Oracle

Because creating the graph can become complex, SAP Data Intelligence provides a [Table Replicator V3 \(Deprecated\)](#) operator that helps you create the appropriate graphs.

## Cloud Data Integration

Cloud Data Integration (CDI) supports polling-based CDC technology.

## Related Information

[Improving CDC Graph Generator Operator Performance](#)  
[CDC Graph Generator \(Deprecated\)](#)

# 16 Subengines

Subengines in the SAP Data Intelligence Modeler allow you to use operators for different runtimes apart from the main engine. The main engine coordinates graph and native operator executions.

Subengines allow a graph to contain operators that run as follows:

- One operator runs in the main engine.
- Other operators are implemented in Python and run in a Python subengine.
- Other operators are implemented in C++ and run in the C++ subengine.

Operators can have implementations for more than one engine, including the main engine and subengines. With multiple engines, you can select a subset of the available engines in the operator's configuration panel from which the optimizer chooses. The optimizer assigns an engine for each operator to minimize the number of edges crossing different engines. When you schedule a cluster of connected operators to run in the same engine, all operators run in the same operating system process.

The available subengines in SAP Data Intelligence are:

- ABAP
- C++
- Node.js
- Python 3.9

## i Note

When you build a graph with subengines, communication between the engines incurs different communication costs.

### ❁ Example

A *File Consumer* operator runs only in the main engine. If a *File Consumer* operator sends data to the *Python3 Operator*, the Modeler must first serialize the data, send the data through a pipe to another operating system process, and finally deserialize the data.

Advantages of using subengines include the following:

- Run connected operators that belong to the same subengine in a single process. Running in a single process is better than using the *Process Executor* operator that executes an external script to launch a new process for each operator.
- Use scriptable operators in different languages, such as *Python3 Operator* and *Node Base Operator* on Node.js. Edit the scripts for these operators in the Modeler user interface without the requirement to handle serializing and deserializing outgoing and incoming data.
- SAP can develop and deliver operators in programming languages other than the language used in the main engine.
- Create and add new operators to the SAP Data Intelligence Modeler in different programming languages. For the Python subengine, you can extend the script operator *Python3 Operator* with new behavior. However, you can also develop new operators in your own machine, and then upload the new operators to

the cluster through SAP Data Intelligence System Management. For more information about the Python subengine, see [Create Operators with the Python Subengine \[page 257\]](#).

## Related Information

[Working with the C++ Subengine to Create Operators \[page 235\]](#)

[Create Operators with the Python Subengine \[page 257\]](#)

[Working with the Node.js Subengine to Create Operators \[page 274\]](#)

[Working with Flowagent Subengine to Connect to Databases \[page 286\]](#)

[SAP Data Intelligence Operators](#)

## 16.1 Working with the C++ Subengine to Create Operators

The SAP Data Intelligence Modeler C++ subengine lets you code your own operators in C++ and make them available for use from the Modeler application.

### Introduction

#### i Note

The C++ Subengine is deprecated. It will be removed in a future SAP Data Intelligence release.

The C++ subengine detects and runs operators that are compiled into shared objects ( \*.so ). When you run a graph in the modeler application, the main engine (which also serves the user interface over HTTP) breaks the graph into large subgraphs. With large subgraphs, the same subengine runs every operator in a subgraph. The main engine then runs a subengine process for each subgraph.

When launched, the C++ subengine performs the following tasks:

1. Looks for and registers operators.
2. Initializes the mechanisms for communicating with the main engine.
3. Receives its subgraph from the main engine.
4. Instantiates the processes in the subgraph and initializes them.
5. Sets up the connections among processes.
6. Starts the graph, handles its input, invokes user-supplied handlers, and writes the output.
7. When instructed by the main engine to stop, or when a fatal error occurs, it cleans up and terminates.

## Quick Start

```
#include <v2/subengine.h>
// Port handler
const char* echo_input( v2_process_t proc, v2_datum_t datum )
{
    // Write the input unchanged
    v2_process_output( proc, "output", datum );
    return NULL; // No error
}
// Shutdown handler (optional)
const char* echo_shutdown( v2_process_t proc )
{
    // Clean up echo's resources here.
    return NULL;
}
// Init handler
const char* echo_init( v2_process_t proc )
{
    // Call echo_input whenever port "input" receives data
    v2_process_set_port_handler( proc, "input", echo_input );
    // Call echo_shutdown when the process is stopped
    v2_process_set_shutdown_handler( proc, echo_shutdown );
    return NULL;
}
// Init function
// Remove `extern "C"` when compiling as pure C
extern "C" V2_EXPORT void init( v2_context_t ctx )
{
    // Create an operator called "Echo" with ID "demo.echo" and call
    // echo_init when initializing its processes.
    auto op = v2_operator_create( ctx, "demo.echo", "Echo", echo_init );
    // Add an input port called "input" of type string.
    v2_operator_add_input( op, "input", "string" );
    // Add an output port called "output" of type string.
    v2_operator_add_output( op, "output", "string" );
}
```

## Related Information

[Getting Started with the C++ Subengine \[page 237\]](#)

[Creating an Operator \[page 237\]](#)

[Logging and Error Handling \[page 239\]](#)

[Port Data \[page 240\]](#)

[Setting Values for Configuration Properties \[page 243\]](#)

[Process Handlers \[page 243\]](#)

[API Reference \[page 245\]](#)

## 16.1.1 Getting Started with the C++ Subengine

At initialization, the C++ subengine iterates recursively over the contents of its `lib/` directory (in the root folder of the subengine) and loads each shared object (`.so`) file.

Use the recursive iteration to organize your libraries into subdirectories, as necessary. The subengine processes only shared object (`.so`) files and ignores other file types.

The C++ subengine also expects to find a symbol called `init` in each of the shared objects. This function in pure C is a function with signature as follows:

```
V2_EXPORT void init( v2_context_t ctx ) { ... }
```

### i Note

If you compile your library from C++, you must prepend the function declaration with `extern "C"` to prevent name mangling.

The signatures of the `init` function use the `V2_EXPORT` macro to ensure that the symbol is visible from the engine executable. The main role of the `init` function is to tell the C++ subengine about the operators implemented by this library. Therefore, before its subgraph is received, the application calls the `init` function once when the engine starts running.

## 16.1.2 Creating an Operator

To register an operator, call `v2_operator_create` from `init`:

```
extern "C" V2_EXPORT void init( v2_context_t ctx )
{
    v2_operator_t op = v2_operator_create(
        ctx,           // the context passed by the engine to init
        "demo.strlen", // operator ID; must be globally unique
        "Strlen Demo", // user-friendly operator name
        init_strlen   // initialization handler (see below)
    );
    // Add an input port so the operator can receive and process data:
    v2_operator_add_input(
        op,           // pointer returned by v2_operator_create
        "inString",  // port name; must be unique within this operator
        "string"     // port type
    );
    // Add an output port so the operator can send data:
    v2_operator_add_output( op, "outLength", "int64" );
}
```

The subengine does not impose any special format for port names, but we recommended prefixing them with `in` or `out`. The prefixing makes it easier to identify them apart when their names appear in the logs.

### i Note

We recommend that you namespace your operator IDs with a meaningful and unique prefix to avoid name clashes. For example, you can prefix with the operator IDs with the organization name.

op has an input port of type string and an output port of type int64 . Now, define its initialization handler (init\_strlen) that will be called every time a process is created to instantiate the "demo\_strlen" operator:

```
const char* init_strlen( v2_process_t proc )
{
    v2_process_set_port_handler(
        proc,          // the process, created by the engine
        "inString",    // which port this handler is associated with
        strlen_on_input // the actual handler (below)
    );
    return NULL; // no error
}
const char* strlen_on_input( v2_process_t proc, v2_datum_t datum )
{
    // Extract the string contained in `datum`
    const char* str = v2_datum_get_string( datum );
    size_t len = strlen( str );
    // Create the output datum containing the length
    // of the input string
    v2_datum_t out = v2_datum_from_int64( (int64_t)len );
    // Write it to the output port
    v2_process_output( proc, "outLength", out );
    // Release the `out` handle (see explanation below)
    v2_datum_release( out );
    return NULL; // no error
}
```

## Result

You have created a basic operator. The initialization and termination, configuration properties, and timed handlers (timers) are described in the subsequent chapters of this guide.

## Compiling an Operator Library

Compile the code to create an operator into a dynamic library (shared object), so that it can be consumed by the C++ subengine executable.

The minimal commands to compile are:

```
# Compile C/C++ sources into position-independent object files (.o)
$ gcc -c -fPIC <sources>
# Link the object files together into a shared object
$ gcc -shared -o <output-name>.so <object-files>
```

After compiling, you can place the <output-name>.so file in the lib/ directory of the subengine and call run.sh -j. This call helps generate the JSON descriptions for your new operator.

## Standard and External Libraries

The C++ subengine runs in a Docker container with Debian 9.2. This operating system provides libstdc++.so.6 that the C++ subengine executable requires to consume the operators.

If your dynamic libraries use a different version or vendor, then you must either:

- Compile with `-static-libstdc++`; or
- Write a custom Dockerfile based on Debian 9.2 in which your desired Standard Library version or vendor is installed and available.

### ⚠ Caution

Do not replace `libstdc++.so.6` provided by the operating system.

Next, associate your operator with the docker image by adding a tag using the `v2_operator_add_tag` function in your operator library.

The same concept applies to all external libraries on which the operators may depend. You can either link to them statically or include them in a custom Dockerfile.

## 16.1.3 Logging and Error Handling

The C++ subengine defines the following logging levels:

- INFO
- DEBUG
- WARNING
- ERROR
- FATAL

The engine is in the debug mode when the debug tracing is enabled for the main engine. The debug messages are printed only when the engine is in the debug mode.

### → Remember

Fatal messages stop the graph execution. Error messages do not cause the graph execution to stop.

To log a single string, use the `v2_log_<level>_string` set of function that are declared in `v2/subengine.h`.

For convenience, the variadic functions (such as, printf-like) are declared and implemented in the optional `v2/log.h` header. If you want to include their implementation in your operator library, then before including `v2/log.h` in a source file, define the `V2_LOG_IMPLEMENTATION` macro.

### → Tip

Include in only one of your source files to avoid multiple definitions of those symbols.

For example:

```
#define V2_LOG_IMPLEMENTATION
#include <v2/log.h>
#undef V2_LOG_IMPLEMENTATION
```

This helps to include the header from multiple files at wherever those function declarations are required, and still continue to keep their implementation in a single place.

## Error Handling

All types of handlers, which users can provide to the subengine have the return type, `const char*`. This return type is a null-terminated string containing the error message or a `NULL` value, if no error occurred. Errors returned by handlers are always fatal (otherwise, use `v2_log_error_string` or `v2_log_error` instead).

### 16.1.4 Port Data

Port types are not known at compile time. The subengine uses the handle, `v2_datum_t` to carry the generic inputs and outputs between the operators. The `v2_datum_t` handle is a reference-counted handle to an underlying piece of data.

Also, because this is a pure C API, you cannot use constructors and destructors to acquire and release these handles. Therefore, we recommend that you follow these guidelines:

- If you get a datum as a parameter in the port handler, do not release it. The engine does that.
- If you create a datum using the `v2_datum_create` set of functions, release it before it gets out of scope. Not releasing it may result in memory leaks. It is also important to release it even if you output the datum because this datum is going to be given to the next operator downstream, and the call to `v2_process_output` increases its reference count by one. This is similar to making a copy of a `std::shared_ptr`.

#### → Tip

You can explicitly increase reference count of datum by calling `v2_datum_acquire`. This increase can be useful if you want to store a datum given to a port handler. But, do not forget to release it later.

## Data Types

A datum may contain any one of the following types:

Modeler type	C type	ID ( <code>v2_type_t</code> )	Size (bytes)
<code>string</code>	<code>char*</code>	<code>V2_TYPE_STRING</code>	Length of the string
<code>int64</code>	<code>int64_t</code>	<code>V2_TYPE_INT64</code>	8
<code>float64</code>	<code>double</code>	<code>V2_TYPE_DOUBLE</code>	8
<code>blob</code>	<code>blob</code>	<code>V2_TYPE_BLOB</code>	Size of the blob
<code>uint64</code>	<code>uint64_t</code>	<code>V2_TYPE_UINT64</code>	8



Modeler type	C type	ID ( v2_type_t )	Size (bytes)
message	v2_message_t	V2_TYPE_MESSAGE	0 (Size of a message is only known when it is serialized, so it cannot be queried).
byte	char	V2_TYPE_BYTE	1
User-defined data types	void*	V2_TYPE_CUSTOM	0 (Engine does not know the size of user-defined data types.)

For each data type, the ID column in the table shows the return value of `v2_datum_get_type`. This allows you to have generic operators (for example, ports with type `any`) that can check their input type at runtime.

The Size column in the table shows the return value of `v2_datum_get_size`.

### ! Restriction

Array types are not supported in the current version.

## Ownership

Whenever the reference count of datum reaches zero, the engine deallocates the memory that belongs to the datum. For `string`, `blob`, `message`, and custom types this behavior can have one further consequence: whether or not to free the memory associated with the data itself. So, when creating a datum from one of those types, you can choose whether or not you want it to own the data:

- A nonowning datum is created using the `v2_datum_from_<type>` functions, which do not deallocate the given data pointer. Thus, ensure that the functions remain available throughout the lifetime of the datum. This behavior is typical of static lifetime variables. For example:

```
const char* YES = "yes";
const char* NO  = "no";
// A port handler
const char* is_even_input( v2_process_t proc, v2_datum_t datum )
{
    int64_t i = v2_datum_get_int64( datum );
    v2_datum_t out;
    if ( i % 2 ) // not even
        out = v2_datum_from_string( NO, 2 );
    else
        out = v2_datum_from_string( YES, 3 );
    v2_process_output( proc, "outEven", out );
    // Release local handle
    v2_datum_release( out );
    return NULL;
}
```

- An owning datum is created using the `v2_datum_own_<type>` functions. These functions take not only the pointer to the data, but also a "destructor" function pointer, which will be invoked with the data pointer as the argument. For example, an operator that concatenates the strings it receives two by two:

```
// Port handler
```

```

const char* concat_on_input( v2_process_t proc, v2_datum_t datum )
{
    v2_datum_t previous = (v2_datum_t)v2_process_get_user_data( proc );
    if ( previous == NULL ) {
        // Store this datum for later use
        v2_process_set_user_data( proc, datum );
        // Let the engine know we've just stored an additional
        // reference to this datum
        v2_datum_acquire( datum );
    } else {
        char*    prev_str  = v2_datum_get_string( previous );
        uint64_t prev_size = v2_datum_get_size( previous );
        char*    curr_str  = v2_datum_get_string( datum );
        uint64_t curr_size = v2_datum_get_size( datum );
        // Allocate enough memory for both strings together. Null-terminating it
        // is optional, since we provide its length to v2_datum_own_string.
        uint64_t out_size = prev_size + curr_size;
        char* out_string = (char*)malloc( out_size );
        // Copy them to out_string
        strncpy( out_string, prev_str, prev_size );
        strncpy( out_string + prev_size, curr_str, curr_size );
        // Since out_string is on the heap, we need an owning datum
        // so it will call `free` on the string once its done.
        v2_datum_t out = v2_datum_own_string( out_string, out_size, free );
        v2_process_output( proc, "output", out );
        // Cleanup
        v2_datum_release( out );
        v2_datum_release( previous );
        v2_process_set_user_data( proc, NULL );
    }
    return NULL;
}

```

### i Note

The datum never makes a copy of the data you provide regardless of the ownership. It only stores the given pointer.

### i Note

Both datum constructors for type string require the length of the string as a parameter. This requirement not only ensures safety (if the string is not null-terminated, buffer overruns might occur), but also enables different datum objects to point to different substrings of the same string without making any copies.

## User-defined Types

Declare the Modeler name for a user-defined types in the `meta.json` file that is under the root directory of the subengine. Typically, it is `<repo-root>/subengines/<sub-engine-id>/meta.json`.

For example, if you are creating image-processing operators, you may want to define your own "image" type. This user-defined type helps the data to flow among such operators without having to conform to the standard types in the modeler. In this example, your `meta.json` might look like this:

```

{
  "versions": ["1.0"],
  "types": ["image"]
}

```

Adding a type to `meta.json` enables the main engine to recognize its existence and accept graphs that use these types as valid.

#### → Remember

The underlying structure of this type is completely unknown to both the main engine and the subengine. Thus, you cannot convey user-defined types between two operators pertaining to different subengines.

## 16.1.5 Setting Values for Configuration Properties

You set the configuration properties at the operator level and each property is associated with a default value.

If you have provided new values to the properties through the user interface of the Modeler application, then the main engine sends the value to the subengine, when you execute the graph. To set an operator configuration property along with its default value, use the `v2_operator_config_add_<type>` set of functions.

Using the `v2_process_t` handle, which is first obtained in the `init` handler of that operator, you can query the actual values for each property by calling the `v2_process_config_get_<type>` functions.

If the defined value for a property does not correspond to its declared value, the engine throws an error for the type mismatch. The only exception is for `double` configurations that can receive an integer value by implicitly converting it to `double`.

## 16.1.6 Process Handlers

You can use the handler types to have more control over the behavior of the operators.

The following are the handler types that you can use.

### Initialization Handler

The initialization handler is set for an operator when it is created (`v2_operator_create`) and is invoked for every process that is instantiated for that operator.

The role of this handler is to initialize per-process data, resources, and connections, as required. Typically, the configuration values are also queried here.

## Shutdown Handlers

In some cases, you use the `init` handler to only set a port handler. But, if it was a more complex case, you may have acquired some resources or allocated memory for that process individually. The place to clean up such resources is the shutdown handler:

```
// shutdown handler
const char* proc_shutdown( v2_process_t proc )
{
    // free proc's memory and resources here
}
// init handler
const char* proc_init( v2_process_t proc )
{
    ...
    v2_process_set_shutdown_handler( proc, proc_shutdown );
    ...
}
```

Shutdown handlers are called when the graph stops if (and only if) the `init` handler for that process was called (regardless of its returned status). This means that, the only case in which the shutdown handler will not be called is when the `init` handler is not even called because a previous process failed to initialize.

## Input Handlers

Operator input is given to the handler bound to the port that received the data. Port handlers are set per process instead of per operator. The handlers allow you to change the behavior of a process depending on its configuration properties. This means that, your operator can have different "modes" without having to check which one was set at every input.

An input port will be blocked while its handler is still running on the last piece of data it received. Therefore, the handler of a given port will never be called multiple times simultaneously, but rather sequentially for every consecutive piece of data. On the other hand, handlers for different ports may be called simultaneously depending on when each port received the input.

One port may not have multiple handlers. Calling `v2_process_set_port_handler` on an already bound port will replace the existing handler. This call can be useful to change the behavior of the process at runtime.

## Timers

Timed handlers are a way of executing logic repeatedly without having to wait for the input on a port. They are completely independent from input ports and may, if desired, produce output. Use `v2_process_add_timed_handler` to register a timer.

Timed handlers are the only ones that may be registered multiple times, as they behave independently from one another, calling the provided callback function at every period.

For example, you can use a timer in an operator whose purpose is to generate random numbers at a specific interval:

```
// Timed handler
const char* gen_tick( v2_process_t proc )
{
    v2_datum_t datum = v2_datum_from_int64( rand() );
    v2_process_output( proc, "output", datum );
    v2_datum_release( datum );
    return NULL;
}
// Init handler
const char* gen_init( v2_process_t proc )
{
    ...
    v2_process_add_timed_handler(
        proc,
        "gen", // Name to be used in the logs to tell handlers apart
        gen_tick, // Handler function
        0, // Repeat count (zero for unlimited)
        1000 // Interval (ms)
    );
    ...
}
```

## 16.1.7 API Reference

Use the C++ API to create operators and to work with the SAP Data Intelligence subengine.

Required header files to work with the C++ subengine:

- log.h
- subengine.h

### Related Information

[log.h \[page 245\]](#)

[subengine.h \[page 246\]](#)

#### 16.1.7.1 log.h

Declares and implements convenience wrappers for logging.

In one of your source files, add

Source Code

```
#define V2_LOG_IMPLEMENTATION
#include "v2/log.h"
```

```
#undef V2_LOG_IMPLEMENTATION
```

### → Remember

Always add the code block in only one of your source files.

This helps to compile the code with not only their declarations, but with also the definition of these functions. Everywhere else in your code, when `log.h` is included without the macro, it brings over the declarations only.

We recommend adding the code in only one of your source file, because it includes variadic functions that cannot be implemented within the engine for consumption from another binary. If you add them in more than one file, the executable would make assumptions about the implementation (for plugin) of variadic arguments that may not hold for different compiler vendors and versions.

### → Tip

If you do not prefer to use the convenience functions, and if you want to call only the `_string-` suffixed ones declared in the `subengine.h` header file, then it is not required to define the `V2_LOG_IMPLEMENTATION` macro or to include `log.h`.

## Formatted Logging

The following are convenience functions to send `sprintf`-formatted strings to the engine log.

- `void v2_log_info (const char *fmt, ...)`
- `void v2_log_debug (const char *fmt, ...)`
- `void v2_log_warning (const char *fmt, ...)`
- `void v2_log_error (const char *fmt, ...)`
- `void v2_log_fatal (const char *fmt, ...)`

## 16.1.7.2 subengine.h

Establishes the core set of functions for the subengine interface.

### Macros

```
#define V2_TYPE_ARRAY 0x0B  
#define V2_TYPE_BLOB 0x03  
#define V2_TYPE_BOOL 0x08  
#define V2_TYPE_CUSTOM 0x07  
#define V2_TYPE_DOUBLE 0x02
```

```

#define V2_TYPE_INT64 0x01
#define V2_TYPE_MAP 0x0A
#define V2_TYPE_MESSAGE 0x05
#define V2_TYPE_NULL 0x09
#define V2_TYPE_STRING 0x00
#define V2_TYPE_UINT64 0x04

```

## Typedef

Typedef	Description
<code>typedef void* v2_context_t</code>	Handle to the internal data that the engine may need to relay from the <code>init()</code> function to some other with the help from the plugin .
<code>typedef struct v2_datum* v2_datum_t</code>	Handle to a piece of data exchanged between two ports.
	<div style="background-color: #f0f0f0; padding: 10px; border-left: 2px solid #007bff;"> <p><b>i Note</b></p> <p>A datum can hold any of the types known to the engine or a pointer to a user-defined type. In the latter, the datum should not cross the subengine boundaries, as there is no guarantee that other subengines or the main engine will recognize this type.</p> </div>
<code>typedef void* v2_map_t</code>	Handle to a JSON-like object, which is a mapping of <code>const char*</code> keys to <code>v2_value_t</code> .
<code>typedef void* v2_message_t</code>	Handle to a message, when the data type transferred between ports of type "message".
<code>typedef void* v2_process_t</code>	Handle to a process (a running instance of an operator), which is created internally by the engine and supplied to the plugin in a call to the <code>init</code> handler of the operator.
<code>typedef void* v2_value_t</code>	<p>Handle to a JSON-like value.</p> <p>This can contain one of the following:</p> <ul style="list-style-type: none"> <li>• <code>string(const char*)</code></li> <li>• <code>integer(int64_t)</code></li> <li>• <code>decimal(double)</code></li> <li>• <code>byte(char)</code></li> <li>• <code>null</code> (see <code>v2_value_is_null()</code>)</li> <li>• <code>object(v2_map_t)</code></li> <li>• <code>array(v2_array_t)</code></li> </ul>

## Functions

- `V2_EXPORT v2_datum_t v2_datum_acquire (v2_datum_t datum)`

*Use:*

Notifies that a new handle to datum is being held, incrementing its (handle) internal reference counter.

*Description:*

If datum were an `std::shared_ptr`, this function would be analogous to the copy constructor.

*Returns*

datum

### i Note

Every call to acquire must be paired with a call to release. If it is not paired, the memory for the datum will be leaked.

- `V2_EXPORT v2_datum_t v2_datum_copy (v2_datum_t datum)`

*Use:*

Creates a new datum that contains a copy of the value of the datum.

*Description:*

If datum contains a string, blob, or message, a deep copy will be made and the resulting datum will own this copy.

If datum contains custom data, this function will raise an error. To make a copy of a custom type:

```
void* ptr = v2_datum_get_custom( my_datum );
void* ptr_copy = copy_my_type( (my_type*)ptr );
v2_datum_t my_datum_copy = v2_datum_own_custom( ptr_copy, free_my_type );
```

Here, it is assumed that `copy_my_type` allocates a new `my_type` value that can later be deallocated with `free_my_type`.

### → Remember

Release the returned datum (even if it was outputted) once you are done with it. See, `v2_datum_release()`.

- `V2_EXPORT v2_datum_t v2_datum_from_blob (void * buf, uint64_t size )`

*Use:*

Creates a nonowning datum that refers to an existing blob.

*Description:*

The blob will not be copied and will not be deallocated when the datum is destroyed.

### → Remember

Release the returned datum (even if it was outputted) once you are done with it. See, `v2_datum_release()`.

- `V2_EXPORT v2_datum_t v2_datum_from_byte (char b)`

*Use:*

Creates a datum containing a byte.

### → Remember

Release the returned datum (even if it was outputted) once you are done with it. See, `v2_datum_release()`.



- `V2_EXPORT v2_datum_t v2_datum_from_custom (void * ptr)`

*Use:*

Creates a nonowning datum that refers to a user-defined area in the memory.

*Description:*

The contents of this area will not be copied and the area will not be deallocated when the datum is destroyed.

→ Remember

Release the returned datum (even if it was outputted) once you are done with it. See, `v2_datum_release()`.

- `V2_EXPORT v2_datum_t v2_datum_from_double (double d)`

*Use:*

Creates a datum containing a `double`.

→ Remember

Release the returned datum (even if it was outputted) once you are done with it. See, `v2_datum_release()`.

- `V2_EXPORT v2_datum_t v2_datum_from_int64 (int64_t i)`

*Use:*

Creates a datum containing an `int64_t`.

→ Remember

Release the returned datum (even if it was outputted) once you are done with it. See, `v2_datum_release()`.

- `V2_EXPORT v2_datum_t v2_datum_from_message (v2_message_t message)`

*Use:*

Creates a nonowning datum that refers to an existing message.

*Description:*

The message will not be copied and will not be deallocated when the datum is destroyed.

→ Remember

Release the returned datum (even if it was outputted) once you are done with it. See, `v2_datum_release()`.

- `V2_EXPORT v2_datum_t v2_datum_from_string (char * s, uint64_t size)`

*Use:*

Creates a nonowning datum that refers to an existing string.

*Description:*

The string will not be copied and it will not be deallocated when the datum is destroyed.

*Parameters*

[ in ] `s`: pointer to the start of the string

[ in ] `size`: the size of the string in bytes, excluding the terminating null byte

→ Remember

Release the returned datum (even if it was outputted) once you are done with it. See, `v2_datum_release()`.

- `V2_EXPORT v2_datum_t v2_datum_from_uint64 (uint64_t u)`  
*Use:*  
 Creates a datum containing a `uint64_t`.

### → Remember

Release the returned datum (even if it was outputted) once you are done with it. See, `v2_datum_release()`.

- `V2_EXPORT void* v2_datum_get_blob (v2_datum_t datum)`  
*Use:*  
 Returns datum as a blob.  
*Description:*  
 This function will not check if the contained data is a blob (see, `v2_datum_get_type()`). Also, you can use `v2_datum_get_size()` to get the size of the blob.
- `V2_EXPORT char* v2_datum_get_byte (v2_datum_t datum)`  
*Use:*  
 Returns datum as a byte.  
*Description:*  
 This function will not check if the contained data is a byte (see, `v2_datum_get_type()`).
- `V2_EXPORT void* v2_datum_get_custom (v2_datum_t datum)`  
*Use:*  
 Returns datum as custom data.  
*Description:*  
 This function will not check if the contained data is a custom pointer (see, `v2_datum_get_type()`).
- `V2_EXPORT double* v2_datum_get_double (v2_datum_t datum)`  
*Use:*  
 Returns datum as a double.  
*Description:*  
 This function will not check if the contained data is a `double` (see, `v2_datum_get_type()`).
- `V2_EXPORT int64_t* v2_datum_get_int64 (v2_datum_t datum)`  
*Use:*  
 Returns datum as an `int64_t`.  
*Description:*  
 This function will not check if the contained data is an `int64_t` (see, `v2_datum_get_type()`).
- `V2_EXPORT v2_message_t v2_datum_get_message (v2_datum_t datum)`  
*Use:*  
 Returns datum as a message.  
*Description:*  
 This function will not check if the contained data is a message (see, `v2_datum_get_type()`).
- `V2_EXPORT uint64_t v2_datum_get_size (v2_datum_t datum)`  
*Use:*  
 Returns the size in bytes of the data held by the datum.  
*Description:*  
 The return value for each contained type is:
  - `V2_TYPE_STRING`: the length of the string in bytes, excluding the terminating null byte.
  - `V2_TYPE_INT64`: 8
  - `V2_TYPE_DOUBLE`: 8
  - `V2_TYPE_BLOB`: the size of the blob in bytes.

- `V2_TYPE_UINT64`: 8
  - `V2_TYPE_MESSAGE`: 0 (only known when the message is serialized)
  - `V2_TYPE_BYTE`: 1
  - `V2_TYPE_CUSTOM`: 0 (not known by the engine)
- `V2_EXPORT char* v2_datum_get_string (v2_datum_t datum)`  
*Use:*  
Returns datum as a null-terminated string.  
*Description:*  
This function will not check if the contained data is a string (see, `v2_datum_get_type()`). Also, you can use `v2_datum_get_size()` to get the size of the string (not including the terminating null byte).
  - `V2_EXPORT v2_type_t v2_datum_get_type (v2_datum_t datum)`  
*Use:*  
Returns a `v2_type_t` constant that describes the type held by datum.  
*Description:*  
The possible values are:
    - `V2_TYPE_STRING`
    - `V2_TYPE_INT64`
    - `V2_TYPE_DOUBLE`
    - `V2_TYPE_BLOB`
    - `V2_TYPE_UINT64`
    - `V2_TYPE_MESSAGE`
    - `V2_TYPE_BYTE`
    - `V2_TYPE_CUSTOM`
  - `V2_EXPORT uint64_t* v2_datum_get_uint64 (v2_datum_t datum)`  
*Use:*  
Returns datum as a `uint64_t`.  
*Description:*  
This function will not check if the contained data is a `uint64_t` (see, `v2_datum_get_type()`).
  - `V2_EXPORT v2_datum_t v2_datum_own_blob (void * buf, uint64_t size, v2_destructor_t destructor)`  
*Use:*  
Creates a datum that takes ownership of an existing blob.  
*Description:*  
The blob is not copied, and it will be deallocated when the data is destroyed by calling `destructor(s)`.

### → Remember

Release the returned datum (even if it was outputted) once you are done with it. See, `v2_datum_release()`.

- `V2_EXPORT v2_datum_t v2_datum_own_custom (void * ptr, v2_destructor_t destructor)`  
*Use:*  
Creates a datum that takes ownership of a user-defined area in the memory.  
*Description:*  
The contents of this area will not be copied, and the area will be deallocated when the data is destroyed by calling `destructor(s)`.

### → Remember

Release the returned datum (even if it was outputted) once you are done with it. See, `v2_datum_release()`.

- `V2_EXPORT v2_datum_t v2_datum_own_message (v2_message_t message)`

#### *Use:*

Creates a datum that takes ownership of an existing message.

#### *Description:*

The message will not be copied, and the message will be deallocated when the data is destroyed by calling `destructor(s)`.

### → Remember

Release the returned datum (even if it was outputted) once you are done with it. See, `v2_datum_release()`.

- `V2_EXPORT v2_datum_t v2_datum_own_string (char * s, uint64_t size, v2_destructor_t destructor)`

#### *Use:*

Creates a datum that takes ownership of an existing string.

#### *Description:*

The string will not be copied, and the string will be deallocated when the data is destroyed by calling `destructor(s)`.

### → Remember

Release the returned datum (even if it was outputted) once you are done with it. See, `v2_datum_release()`.

- `V2_EXPORT void v2_datum_release (v2_datum_t datum)`

#### *Use:*

Notifies that the handle `datum` is being released.

#### *Description:*

After this call, it cannot be guaranteed that the actual data underlying the handle will be available. Therefore, you should no longer use `datum`. If `datum` were an `std::shared_ptr`, this function would be analogous to the destructor.

### i Note

Every call to `v2_datum_acquire`, `v2_datum_from_*`, `v2_datum_own_*`, or `v2_datum_copy` must be paired with a call to `release`, if not the memory for the datum will be leaked. If you output such a datum before releasing, the engine will ensure that it survives the call to `release` and that the downstream operators have access to it.

- `V2_EXPORT void v2_log_debug_string (const char * str)`

#### *Use:*

Logs the debug information.

#### *Description:*

If either the main engine was run in debug mode, or if the subengine was forced into debug mode by the `run.sh` script, then this debug information will show up in the output of the main engine,

- `V2_EXPORT void v2_log_fatal_string (const char * str)`

*Use:*

Logs a string as a fatal error that will cause the main engine to terminate the subengine, its graph, and processes.

- `V2_EXPORT void v2_message_destroy (v2_message_t msg)`

*Use:*

Frees the memory associated with a message.

*Description:*

Call this function only if you obtained the message using `v2_message_create()`.

- `V2_EXPORT v2_value_t v2_message_get_attribute (v2_message_t msg, const char * key)`

*Use:*

Returns the value associated to a message attribute.

*Description:*

This is a convenience for `v2_map_find( v2_message_get_attributes( msg ), key )`.

- `V2_EXPORT void v2_message_set_body (v2_message_t msg, v2_datum_t body)`

*Use:*

Sets the message body by releasing the previous one (if any) and by calling `v2_datum_acquire( body )`.

- `V2_EXPORT void v2_operator_add_input (v2_operator_t op, const char * port_id, const char * port_type)`

*Use:*

Adds an input port called `port_id` to `op`.

*Description:*

`port_type` is a null-terminated string describing the type accepted by the port. The possible port types are:

- "string"
- "blob"
- "int64"
- "uint64"
- "float64"
- "message"
- "byte"
- anything else will be regarded as a custom (user-defined) type, which can be conveyed between any two operators, provided they are both implemented by the same subengine.

- `V2_EXPORT void v2_operator_add_output (v2_operator_t op, const char * port_id, const char * port_type)`

*Use:*

Adds an output port called `port_id`.

*Description:*

`port_type` will be treated as described in `v2_operator_add_input()`.

- `V2_EXPORT v2_array_t v2_operator_config_add_array (v2_operator_t op, const char * key)`

*Use:*

Creates and returns a JSON array configuration value named `key` to `op`.

*Description:*

You can modify the returned array to customize the initial value for this property.

### ⚠ Caution

The array may be relocated in memory in a subsequent call to `v2_operator_config_add_*`. Thus, ensure you are modifying it immediately after obtaining it. Do not store the array for later use.

- `V2_EXPORT v2_map_t v2_operator_config_add_map (v2_operator_t op, const char * key)`

#### *Use:*

Creates and returns a JSON object configuration value named `key` to `op`.

#### *Description:*

You can modify the returned map to customize the initial value for this property.

### i Note

The map may be relocated in memory in a subsequent call to `v2_operator_config_add_*`. Thus, ensure you are modifying it immediately after obtaining it. Do not store the array for later use.

- `V2_EXPORT void v2_operator_config_add_string (v2_operator_t op, const char * key, const char * default_value)`

#### *Use:*

Adds a string configuration value named `key` to `op`.

#### *Description:*

`default_value` must be a null-terminated string containing the initial value for this property.

- `V2_EXPORT v2_operator_t v2_operator_create (v2_context_t ctx, const char * id, const char * name, v2_init_handler_t init)`

#### *Use:*

Creates an operator.

#### *Parameters*

[in] `ctx`: the context handle supplied to the `init()` function

[in] `id`: the operator ID

[in] `name`: the operator name, which will be visible on the GUI

[in] `init`: the callback function that the engine will call when a process is instantiated from this operator.

#### *Description:*

The init handler can (not necessarily in this order):

- Access configuration values: use the `v2_process_get_config_*` set of functions to get the actual configuration values for your process. These values remain available throughout the lifetime of the processes, but the init handler is usually the first (rarely) place where they are read.
- Set input handlers: if your operator has input ports, the process is expected to set a handler for each of them (see, `v2_process_set_port_handler()`). It is not an error to leave a port unhandled, provided it is never connected. It is also possible to conditionally set input handlers depending on the configuration values.

### i Note

Connected ports that are unhandled results in an error and cause the engine to stop.

- Set the shutdown handler: if your operator needs to clean up after running. This callback will be invoked by the engine when the graph is stopped. This is also usually the place to release resources acquired in the init handler.
- `V2_EXPORT void v2_operator_set_visibility (v2_operator_t op, int visible)`

*Use:*

Sets whether or not `op` should be visible.

*Description:*

Visible components have their JSON descriptions automatically generated by the subengine so that they show up on the user interface.

- `V2_EXPORT void v2_process_add_timed_handler (v2_process_t proc, const char * name, v2_timed_handler_t handler, uint64_t repeat, uint64_t interval_ms)`

*Use:*

Adds a timed handler.

*Parameters*

[in] `proc`

[in] `name`: an optional name for the handler (for clearer logging). It can be NULL or empty too. It does not have to be unique.

[in] `handler`: the callback function

[in] `repeat`: the number of times the handler is to be called, or zero for unlimited (as long as the graph is running).

[in] `interval_ms` the period in milliseconds for the timer

- `V2_EXPORT double v2_process_config_get_double (v2_process_t proc, const char * key)`

*Use:*

Returns the final value for a `double` configuration property.

### **i** Note

Properties originally declared with `v2_operator_config_add_int64()` can also be retrieved using this function. It automatically converts the integer to double.

- `V2_EXPORT v2_bool_t v2_process_config_is_null (v2_process_t proc, const char * key)`

*Use:*

Returns whether the final value for a configuration property is the JSON keyword `null`.

- `V2_EXPORT const char* v2_process_get_id (v2_process_t proc)`

*Use:*

Returns the process ID, which uniquely identifies it in the graph.

### **→** Remember

This is not the same as the operator ID.

- `V2_EXPORT void v2_process_output (v2_process_t proc, const char * port_id, v2_datum_t data)`

*Use:*

Sends output to a port.

*Description:*

This call will block until the destination (the process downstream) is ready to consume.

- `V2_EXPORT void v2_process_set_port_handler (v2_process_t proc, const char * port_id, v2_port_handler_t handler)`

*Use:*

Sets a callback function to be invoked when the input port named `port_id` receives data.

*Description:*

Calls to `handler` will be made sequentially, never concurrently. Successive calls to this function replace the currently set handler.

- `V2_EXPORT void v2_process_set_user_data (v2_process_t proc, void * data)`

*Use:*

Stores a user-supplied pointer.

*Description:*

This function is useful if you need each instance of an operator to carry some individual information (for example, a file descriptor, a connection object, and so on). The initial value is `NULL`.

- `V2_EXPORT v2_array_t v2_value_get_array (v2_value_t v)`

*Use:*

Returns the array contained in `v`.

*Description:*

If `v` does not contain an array, it will raise an error.

- `V2_EXPORT v2_bool_t v2_value_get_bool (v2_value_t v)`

*Use:*

Returns the boolean contained in `v`.

*Description:*

If `v` does not contain a boolean, it will raise an error.

- `V2_EXPORT double v2_value_get_double (v2_value_t v)`

*Use:*

Returns the decimal contained in `v`.

*Description:*

If `v` does not contain a decimal, it will raise an error.

- `V2_EXPORT int64_t v2_value_get_int64 (v2_value_t v)`

*Use:*

Returns the integer contained in `v`.

*Description:*

If `v` does not contain an integer, it will raise an error.

- `V2_EXPORT v2_map_t v2_value_get_map (v2_value_t v)`

*Use:*

Returns the object contained in `v`.

*Description:*

If `v` does not contain an object, it will raise an error.

- `V2_EXPORT const char* v2_value_get_string (v2_value_t v)`

*Use:*

Returns the string contained in `v`.

*Description:*

If `v` does not contain a string, it will raise an error.

- `V2_EXPORT v2_type_t v2_value_get_type (v2_value_t v)`

*Use:*

Returns the type contained in a JSON value.

*Description:*

The possible values are:

- `V2_TYPE_STRING`
- `V2_TYPE_INT64`
- `V2_TYPE_DOUBLE`
- `V2_TYPE_BOOL`



- V2\_TYPE\_NULL
- V2\_TYPE\_MAP
- V2\_TYPE\_ARRAY

## 16.2 Create Operators with the Python Subengine

Create operators in Python using the Python subengine, and use them to build graphs in the SAP Data Intelligence Modeler.

When you run a graph in the Modeler, the main engine that coordinates all subengines breaks the graph into large subgraphs. The advantage is that every operator in the same subgraph is run by the same subengine. The main engine then starts a subengine process for each subgraph.

### i Note

Python 2.7 is deprecated as of SAP Data Intelligence 1911. Currently, SAP Data Intelligence supports only Python 3.9. You can no longer create new operators based on Python 2.7. Port any existing custom operators based on Python 2.7 to Python 3.9.

### i Note

When you create a new dockerfile to use in graphs that have Python operators, at minimum you must associate the following tags with the dockerfile: `'tornado': '6.1.0'`, `'sles': ''` and `'python': '3.9'`. Make sure that the resources are installed on the dockerfile.

## Data Type Mapping

The following table shows the data type mapping between the Modeler and the Python subengine.

Modeler Data Type	Python Subengine Data Type
string	str
blob	bytes
int64	int
uint64	int
float64	float
byte	int
message	Message

## Modeler Data Type

## Python Subengine Data Type

[ ]x

list

Replace the letter *x* with any of the allowed data types, excluding message. For example, [ ]string, [ ]uint64, [ ]blob, and so on.

If you create operators that receive or send data types that have no direct mapping to Modeler data types, you must create ports with type `python 3.9`.

### ❖ Example

You can connect many Python 3 operators in a graph, all of which have in ports and out ports of type `python 3.9`.

Operators created with ports of type `python 3.9` can communicate with any Python object, including objects with no corresponding Modeler type, such as set, numpy arrays, and so on. There's one exception: Subengine-specific types can't cross the boundary between two Modeler groups. However, if you place your Python-specific object inside the body of a message, then the body is correctly serialized and deserialized with pickle (a Python module) when crossing the boundary between two groups.

### i Note

The behavior of serialization of the body with pickle is specific to the Python subengine; don't expect it to work with other subengines.

Create the message type using the format `Message(body, attributes)`, where:

- `body` can be any object.
- `attributes` is a dictionary-mapping string to any object.

If `m` is an object of type message, then you can use the commands `m.body` and `m.attributes` to access the fields initialized in the class constructor. The `attributes` argument is optional in the message constructor. If you want an empty `attributes` string, you can construct your message using the format `Message(body)`.

SAP recommends that you don't use a message inside the body of another message. Instead, use a dictionary inside the body of a message. The dictionary must have the keys "Body" and "Attributes". Using a message object as the body of another message can result in unexpected behavior when you transfer the message across the boundary of different subengines.

### ❖ Example

The inner message can be converted automatically to a dictionary when crossing the boundaries of different subengines, but it isn't converted back to a message when coming back to your operator's subengine.

Therefore, SAP recommends to always prefer dictionaries over messages inside the body of a message, because it won't change type during the communication.

```
# DO NOT DO THE FOLLOWING:  
inner_msg = Message("body_of_inner_msg")  
outer_msg = Message(body=inner_msg, attributes={})
```

```
# If you want to have a message as the body of another message, do this instead:
inner_msg = {"body": "body_of_inner_msg"}
outer_msg = Message(inner_msg)
```

## Methods to Create Python Operators

Use one of the following two methods to create your own Python operators:

- **Normal usage:** To create normal Python subengines, use only the Modeler user interface.
- **Advanced usage:** To create new Python operators, use the advanced method, which shows you how to use your own machine and then upload the new operators to SAP Data Intelligence. The advanced usage method of creating operators is useful when you want to code in your own IDE and create unit tests for your operators.

## Related Information

[Normal Usage \[page 259\]](#)

[Advanced Usage \[page 261\]](#)

### 16.2.1 Normal Usage

If you want to create simple scripts quickly, we recommend that you customize the behavior of operators using the SAP Data Intelligence Modeler user interface.

There are two ways to customize the behavior of a *Python3 Operator*:

- Drag the *Python3 Operator* to the graph canvas and edit the script of the operator by right-clicking on it and clicking the open script option.
- Create a new operator in the *Repository* tab and select *Python3 Operator* as the base operator to be extended.

The advantage of the second approach is that you can reuse your new operator across many graphs, while in the first approach, the edited script is specific to the given graph and operator instance.

To create a new operator in the *Repository* tab using the *Python3 Operator*:

1. Open the *Repository* tab and expand the *Operators* section.
2. To create an operator, right-click the appropriate *Operators* subfolder, and select *Create Operator*.
3. In the dialog, enter the operator name, display name, and base operator and choose *Python3 Operator* as the base operator.
4. In the operator editor view, you can create input ports, output ports, tags, and configuration parameters, and write your script.

## Related Information

[Using Python Libraries \[page 260\]](#)

### 16.2.1.1 Using Python Libraries

To make Python libraries available to your operator, you can add tags to it so that upon graph execution, the appropriate docker image can be chosen.

You can create or extend dockerfiles through the user interface and add tags to associate it with an operator. The following example details the necessary steps.

Suppose that you want to use the numpy library (version 1.16.1) on your custom Python operator.

First, create a new dockerfile:

1. Open the [Repository](#) tab and select a subfolder.
2. Right-click the subfolder and select [Create Docker File](#).
3. Provide a name and choose [OK](#).

There are many ways to create a dockerfile that contains Python and the numpy library. For example, you can enter `FROM <public_numpy_docker_image>` in the first line. If you use a public image, make sure that it also contains the requirements to run the Python subengine, which are: `tornado==6.1.0` and `python3.9`. The tag key-value pairs for the requirements are: `'tornado': '6.1.0'` and `'python39': ''`.

Alternatively, you can inherit from the default Modeler python3 dockerfile and add numpy to it. In this way, all of the requirements for the standard [Python3 Operator](#) are already satisfied. The following dockerfile shows one way that you can accomplish this:

```
FROM $com.sap.sles.base
RUN python3.9 -m pip install --user numpy="1.16.1"
```

4. After you write the dockerfile content, add tags on the configuration panel for every relevant resource that this dockerfile offers, in addition to the tags from its parent dockerfile. That is, you don't need to repeat the tags that the parent dockerfile already has. For example, the numpy for python3 dockerfile needs only the following tag `numpy39: 1.16.1`.

#### **i** Note

Make sure you are installing the library for the right version of Python, which is 3.9. Notice that we added the suffix ``36`` to the numpy tag to reflect the Python version where they were installed. This can be useful if, in the future, we create a new subengine that uses a different python version.

5. The final step is to add the new tag to your operator in the operator editor view. Add the tag `"numpy39": "1.16.1"` to your Python operator. Alternatively, you can also add tags to a group defined on the graph.

For more information, see [Python3 Operator](#) documentation.

## Related Information

[Creating Dockerfiles \[page 291\]](#)

### 16.2.2 Advanced Usage

You can create operators without using the Modeler user interface.

#### i Note

This method of creating operators requires that you restart the Modeler instance for the changes to take effect.

The following example assumes that you use a UNIX-like system to develop your operators. To run tests, you must download the `pysix_subengine` package on your local machine.

To download the `pysix_subengine` package, follow these steps:

1. Log in to SAP Data Intelligence System Management as an administrator.
2. Click the *File* button.
3. Click the *Union View* button.
4. Navigate to `files/vflow/subdevkits` on the file explorer.
5. Right-click the `pysix_subengine` folder and select *Export File*.

To create Python 3.9 operators, you must structure your solution (a package that can be imported by SAP Data Intelligence System Management) as follows:

```
my_solution/  
  vrep/vflow/  
    subengines/com/sap/python3.9/operators/  
      myOperator1/  
        operator.py  
        operator.json  
      com/  
        mydomain/  
          myOperator2/  
            operator.py  
            operator.json  
  vsolution.json
```

The `vsolution.json` file should look like the following:

```
{  
  "name": "vsolution_vflow_my_solution",  
  "description": "My Solution",  
  "license": "my license"  
}
```

#### i Note

Python 2.7 is deprecated as of SAP Data Intelligence 1911; only Python 3.9 is now supported. The creation of new operators based on Python 2.7 is now disabled. Any existing custom operators based on Python 2.7 should be ported to Python 3.9.

## Define the Operator Attributes and Behavior

The operator .py script defines both the operator attributes and the operator behavior.

### Note

The script must be called operator .py.

The following is example code in the operator .py file:

```
from pysix_subengine.base_operator import PortInfo, OperatorInfo
from pysix_subengine.base_operator import BaseOperator
# The operator class name (which inherits BaseOperator) should have the same
name as
# the operator's folder, except that its first letter should be uppercase.
class AppendString(BaseOperator):
    def __init__(self, inst_id, op_id):
        super(AppendString, self).__init__(inst_id=inst_id, op_id=op_id)
        # Default configuration fields. They will be shown in the UI.
        self.config.stringToAppend = ""
        self.config.method = "NORMAL"
        # Adds a callback '_data_in' that is called every time the
        # operator receives data in the inport 'inString'.
        self._set_port_callback('inString', self._data_in)
        self.__transform_data = None
    # This method is mandatory.
    # The operator.json will be generated based mostly on the OperatorInfo
returned by this method.
    def _get_operator_info(self):
        inports = [PortInfo("inString", required=True, type_="string")]
        outports = [PortInfo("outString", required=True, type_="string")]
        return OperatorInfo("Append String",
            inports=inports,
            outports=outports,
            icon="puzzle-piece",
            tags={"numpy36": "1.16.1"}
            dollar_type="http://sap.com/vflow/
com.mydomain.appendString.schema.json#")
    def _set_websocket(self, handler):
        self.web_handler = handler
        self.web_handler.set_message_callback(self._on_message)
    def _on_message(self, message):
        self.web_handler.write_message(str(message))
        self._send_message("outString", message)
    def _data_in(self, data):
        self.metric.include_value(1)
        self._send_message('outString', self.__transform_data(data +
self.config.stringToAppend))
        # Configs set in the UI are already available when this method is called.
        # _init is called before any operator main loop has started.
    def _init(self):
        metric_infos = []
        metric_infos.append(self._registry.create_metric_info(u'count', 1,
u'name', u'display_name', u'unit'))
        self.metric = self.register_metric(u"int", u"TEST", metric_infos)
        self.register_websocket_route('/socket', 'test', self._set_websocket)
        self.register_static_route('/ui', 'static')
        self.register_rproxy_route('/rproxy/*path', 'http://localhost:' +
str(self.config.port) + '/')

        if self.config.method == "NORMAL":
            self.__transform_data = lambda x: x
        elif self.config.method == "UPPERCASE":
            self.__transform_data = lambda x: x.upper()
        else:
```

```

        raise ValueError("Unknown config set in configuration: '%s'." %
self.config.method)
    # Called before the operator's main loop execution.
    # Other operators may already have started execution.
    def _prestart(self):
        pass
    # Called when the graph is being terminated.
    def shutdown(self):
        pass

```

Let's examine the script step-by-step. First, look at the class definition:

```
class AppendString(BaseOperator):
```

As you can see, the `AppendString` class extends the built-in `BaseOperator` class.

### Note

The class name must have the same name as the folder, except the first letter must be uppercase, while the folder's first letter is lowercase. For example, if the folder is named `appendString`, the class should be named `AppendString`.

Now let's examine each of the methods:

```

def __init__(self, inst_id, op_id):
    super(AppendString, self).__init__(inst_id=inst_id, op_id=op_id)
    # Default configuration fields. They will be shown in the UI.
    self.config.stringToAppend = ""
    self.config.method = "NORMAL"
    # Adds a callback '_data_in' that is called every time the
    # operator receives data in the inport 'inString'.
    self._set_port_callback('inString', self._data_in)
    self.__transform_data = None

```

This method is the class constructor. First it calls `super(AppendString, self).__init__(inst_id=inst_id, op_id=op_id)`.

In the next line, the method creates two new configuration fields called `stringToAppend` and `method`, and the default values `" "` and `"NORMAL"` are assigned to it, respectively. All configuration fields that you create appear in the user interface and are configurable by the user. You can always access these values in the script, as we will do soon.

Next, we set a callback called `_data_in`, which is called every time the operator receives data in the inport `inString`. To set the callback, use the method `_set_port_callback` defined by the `BaseOperator` class.

```

def _get_operator_info(self):
    inports = [PortInfo("inString", required=True, type_="string")]
    outports = [PortInfo("outString", required=True, type_="string")]
    return OperatorInfo("Append String",
                        inports=inports,
                        outports=outports,
                        icon="puzzle-piece",
                        tags={"numpy36": "1.16.1"},
                        dollar_type="http://sap.com/vflow/
com.mydomain.appendString.schema.json#")

```

All of the operators that you create in this way must always implement the `_get_operator_info` method. The `_get_operator_info` method is used to generate the operator json automatically, so you must specify

the operator attributes here. To specify the operator attributes, the method must return an `OperatorInfo` object. The `OperatorInfo` object has the following attributes:

```
class OperatorInfo(object):
    """
    Attributes:
        description (str): Human readable name of the operator.
        icon (str): Icon name in font-awesome.
        iconsrc (str): Path to a icon image. Alternative to the option to a icon
from font-awesome.
        inports (list[PortInfo]): List of input ports.
        outports (list[PortInfo]): List of output ports.
        tags (dict[str,str]): Tags for dependencies. dict[library_name,
lib_version].
        component (str): This field will be set automatically.
        config (dict[string,any]): This field will be set automatically.
        dollar_type (str): Url to $type schema.json for this operator.
    """
```

You can specify all of the attributes using the `get_operator_info` method. In the example, we specify an input port called *inString* that receives a string, an output port called *outString* that also receives a string, the built-in icon *puzzle-piece* as the operator's icon, and *Append String* as operator description. The *inString* port name is used in the constructor to set the correct callback to the port.

```
def _init(self):
    metric_infos = []
    metric_infos.append(self._registry.create_metric_info(u'count', 1, u'name',
u'display_name', u'unit'))
    self.metric = self.register_metric(u"int", u"TEST", metric_infos)
    self.register_websocket_route('/socket', 'test', self._set_websocket)
    self.register_static_route('/ui', 'static')
    self.register_rproxy_route('/rproxy/*path', 'http://localhost:' +
str(self.config.port) + '/')

    if self.config.method == "NORMAL":
        self.__transform_data = lambda x: x
    elif self.config.method == "UPPERCASE":
        self.__transform_data = lambda x: x.upper()
    else:
        raise ValueError("Unknown config set in configuration: '%s'." %
self.config.method)
```

The method is overridden by the `BaseOperator` superclass. The method is called after the user-defined configurations have already been set. The method is called for all graph operators before any operator has been started. Here, we are deciding which `transform_data` function to use based on the `self.config.method` parameter set by the user in the interface. In this method, the user can register metrics (`register_metric`) and routes (`register_websocket_route`, `register_static_route` and `register_rproxy_route`). It is only possible to register these functionalities using the following function:

```
def _data_in(self, data):
    self.metric.include_value(1)
    self._send_message('outString', self.__transform_data(data +
self.config.stringToAppend))
```

The function is the callback that we create to handle inputs to the *inString* input port. Here, we append the input data with our `stringToAppend` configuration field, apply the function to it, and use the `BaseOperator_send_message` method to send the result to our output port *outString*. In addition, we set a new value in



the `self.metric` value, because the only `MetricInfo` is of type `count`, we update the counter of processed strings.

```
def _set_websocket(self, handler):
    self.web_handler = handler
    self.web_handler.set_message_callback(self._on_message)
def _on_message(self, message):
    self.web_handler.write_message(str(message))
    self._send_message("outString", message)
```

The method is overridden by the `BaseOperator` superclass. It contains code that is executed before the operator main loop is started:

```
def _prestart(self):
    pass
```

The method is overridden by the `BaseOperator` superclass. It contains code that is executed after the operator main loop is finished:

```
def shutdown(self):
    pass
```

A list of all methods is available at [List of BaseOperator Methods \[page 268\]](#).

## Create an Operator

After you implement the `operator.py` file to define the operator, run a bash script to create the operator.

To create an operator, create a folder (or a series of folders) inside the `<ROOT>/subengines/com/sap/python36/operators/` directory, where `<ROOT>` is `my_solution/vrep/vflow` in the example structure. For example, to create an operator with the ID `com.mydomain.util.appendString`, create the folders for the path `<ROOT>/subengines/com/sap/python36/operators/com/mydomain/util/appendString` and place two files inside the last directory: `operator.py` and `operator.json`. To automatically generate the `operator.json` file, run the script `gen_operator_jsons.py`, which is located inside the `pysix_subengine` package.

The following bash script shows how you can use `gen_operator_jsons.py` in your solution:

```
SCRIPT_PATH=<PYSIX_SUBENGINE_PATH>/scripts
SUBENGINE_ROOT=<ROOT>/subengines/com/sap/python36
START_DIR=operators
python $SCRIPT_PATH/gen_operator_jsons.py --subengine-root $SUBENGINE_ROOT --
start-dir $START_DIR "$@"
```

After you create an operator, restart the Modeler instance for the changes to take effect.

## Related Information

[Using Python Library \[page 266\]](#)

[Adding Documentation \[page 266\]](#)

- [Creating Tests \[page 267\]](#)
- [List of BaseOperator Methods \[page 268\]](#)
- [List of BaseOperator Attributes \[page 270\]](#)
- [List of Metric Methods \[page 271\]](#)
- [Logging \[page 271\]](#)
- [Uploading to SAP Data Intelligence System Management \[page 272\]](#)

## 16.2.2.1 Using Python Library

Create a dockerfile as specified in the *Using Python Library* sub-section in the *Normal Usage* section and add the relevant tags to the `_get_operator_info` method of `BaseOperator`.

### Related Information

- [Using Python Libraries \[page 260\]](#)

## 16.2.2.2 Adding Documentation

Create the operator documentation in a file named `README.md` in the operator's folder. For example, to create the documentation for the dummy operator `AppendString`, create a new `README.md` file in the following path:

```
{ROOT}/subengines/com/sap/python36/operators/com/mydomain/util/appendString/  
README.md
```

The documentation is written using Markdown syntax (more information [here](#)). You can check the result by right-clicking your operator and choosing *Open Documentation*, in the SAP Data Intelligence Modeler UI.

To add a custom icon for your operator, copy the icon to the operator's folder; for example:

```
{ROOT}/subengines/com/sap/python36/operators/com/mydomain/util/appendString/  
icon.png
```

Then, make sure that your `_get_operator_info` method sets the `icon.png` file (or another file) as the `iconsrc` field, as shown in the following example:

```
def _get_operator_info(self):  
    ...  
    return OperatorInfo(...,  
                        iconsrc="icon.png",  
                        ...)
```

Last, you must regenerate the operator's json.

## 16.2.2.3 Creating Tests

When you create a new operator extending *BaseOperator*, you can also create unit tests using the Python unit testing framework.

We recommend that you create a folder called `test` in your project root (parallel with the folder "my\_solution" in our sample hierarchy) and in it, use the same folder structure that you used when creating the operator. For example, for the operator `appendString`, we create the following folders:

```
test/operators/com/mydomain/util/appendString/
```

Create a file named `test_append_string.py` inside the `appendString` folder:

```
import unittest
from Queue import Queue
from operators.com.mydomain.appendString.operator import AppendString
class TestAppendString(unittest.TestCase):
    def testUpper(self):
        op = AppendString.new(inst_id="1", op_id="anything")
        qin1 = Queue(maxsize=1)
        op.inqs["inString"] = qin1
        qout1 = Queue(maxsize=1)
        op.outqs["outString"] = qout1
        input_config = "config_test"
        op.config.stringToAppend = input_config
        op.config.method = "UPPERCASE"
        op.base_init() # This will execute the _init method and also other
things.
        op.start() # Start the operator's thread.
        try:
            input_str = "input_test|"
            qin1.put(input_str) # Send data in the 'inString' input port.
            ret = qout1.get() # Gets data from the 'outString' output port.
            self.assertEqual((input_str + input_config).upper(), ret) # Verify
that the output is equal to the expected result.
        finally:
            op.stop() # stop the operator no matter what happens. Otherwise the
test may hang forever.
```

To run unit tests for all Python files that start with the prefix "test", create a script such as the following:

```
PYSIX_PATH=<PYSIX_SUBENGINE_PATH> # replace <PYSIX_SUBENGINE_PATH> by the path
to your pysix_subengine folder.
SCRIPT_DIR=$(dirname "$0")
cd $SCRIPT_DIR # changes working dir to the folder where the script is located
(which should be the 'test' folder).
SUBENGINE_ROOT=$(realpath ../my_pysolution/vrep/vflow/subengines/com/sap/
python36) # Gets absolute path of python36 folder.
export PYTHONPATH=$PYTHONPATH:$PYSIX_PATH:$SUBENGINE_ROOT # Append paths to
pythonpath.
python3.6 -m unittest discover -s . -p "test*.py" -v
cd -
```

Place the script in the `test` directory that you created.

To check the official documentation of the Python unit test module, see <https://docs.python.org/2/library/unittest.html> .

## 16.2.2.4 List of BaseOperator Methods

When subclassing *BaseOperator*, you will need to use a set of methods to set callbacks, send messages, etc. The list below summarizes the *BaseOperator* methods you might need.

### i Note

Each callback registered in the script runs in a different thread.

As the script developer, you should handle potential concurrency issues such as race conditions. You can, for instance, use primitives from the Python threading module to get protected against those issues.

For more information, see the [official documentation at python.org](https://docs.python.org/3/). 🐘

### **`_get_operator_info(self)`**

This method should be overridden by a subclass. It should return the `OperatorInfo` of this operator. `OperatorInfo` defines some key features of the operator such as port names and other things.

Returns:

`OperatorInfo`

### **`_init(self)`**

This method can be overridden by a subclass. This method will be called after the user defined configurations have already been set. This method will be called for all graph's operators before any operator has been started.

### **`_prestart(self)`**

This method can be overridden by a subclass. It should contain code to be executed before the operator's callbacks start being called. The code here should be non-blocking. If it is blocking, the callbacks you have registered will never be called. You are allowed to use the method `self._send_message` in `self._prestart`. However, bare in mind that other operators will only receive this data sent when their `_prestart` have already finished. This is because their callback will only be active after their `_prestart` have finished

### **`shutdown(self)`**

This method can be overridden by a subclass. It should contain code to be executed after the operator's main loop is finished.

### **`_send_message(self, port, message)`**

Puts an item into an output queue.

Args:

- `port (str)`: name of output port
- `message (...)`: item to be put

### **`_set_port_callback(self, ports, callback)`**

This method associate input 'ports' to the 'callback'. The 'callback' will only be called when there are messages available in all ports in 'ports'. If this method is called multiple times for the same group of ports then the old 'callback' will be replaced by a new one. Different ports group cannot overlap.

Args:

- ports (str|list[str]): input ports to be associated with the callback. 'ports' can be a list of strings with the name of each port to be associated or a string if you want to associate the callback just to a single port.
- callback (func[...]): a callback function with the same number of arguments as elements in 'ports'. Also the arguments will be passed to 'callback' in the same order of their corresponding ports in the 'ports' list.

#### **`_remove_port_callback(self, callback)`**

Remove the 'callback' function. If it doesn't exist, the method will exit quietly.

Args:

- callback: Callback function to be removed.

#### **`_add_periodic_callback(self, callback, milliseconds=0)`**

Multiple distinct periodic callbacks can be added. If an already added callback is added again, the period 'milliseconds' will be replaced by the new one. If you want two callbacks with identical behavior to be run simultaneously then you will need to create two different functions with identical body.

Args:

- callback (func): Callback function to be called every 'milliseconds'.
- milliseconds (int|float): Period in milliseconds between calls of 'callback'. When not specified it is assumed that the period is zero.

#### **`_change_period(self, callback, milliseconds)`**

Args:

- callback (func): Callback for which the period will be changed. If callback is not present on the registry, nothing will happen.
- milliseconds (int|float): New period in milliseconds.

#### **`_remove_periodic_callback(self, callback)`**

Args:

- callback (func): Callback function to be removed.

#### **`register_metric(self, metric_type, name, metric_infos=[])`**

Register a new metric and its respective 'MetricInfo' (to create it check next section).

Args:

- name (str): It is the metric name defined in the operator (this should be a string without spaces).
- metric\_type (str): It defines the type of the updated value. They can be 'consumer', 'producer', 'float' or 'int'.  
Consumer and producer aggregators are 'int' and already have the 'metric\_infos' field pre-configured, so these aggregators ignore it.
- metric\_infos (list[MetricInfo]): It is the list of information to be presented about a specific metric.

Returns:

- Metric: a metric instance which allows the operator to update its value.

#### **`create_metric_info(self, metric_type, scale, name, display_name, unit)`**

This method is a helper function to create 'MetricInfo' in order to register metrics for the operator.

Args:

- `metric_type` (str): It is the metric type and it can be: 'value', 'sum', 'max', 'min', 'count', 'avg', 'rate'.
- `scale` (int): It is an integer factor that multiplies all 'MetricInfo' output values.
- `name` (str): It is a metric name following the previous 'Name' pattern.
- `display_name` (str): The name displayed to the user (it should be friendlier).
- `unit` (str): It is a string with the metric unit. The following units are pre-configured to scale up when necessary: 'Bytes', 'KB', 'MB', 'GB', 'TB', 'PB', 'Bytes/s', 'KB/s', 'MB/s', 'GB/s', 'TB/s' and 'PB/s'. For example, when a value of unit 'Bytes' reaches '1024' it automatically becomes 'KB'.

Returns:

- MetricInfo

#### **register\_websocket\_route(self, path, target, handler)**

Args:

- `path` (str): operator complete url.
- `target` (str): target path to possible websocket connection to be used.
- `handler` (func[websocket\_handler]): function that will be called with the websocket handler as argument.

#### **register\_static\_route(self, path, target)**

Registers a new static route for the operator. This method only works as expected when called during the operator initialization.

Args:

- `path` (str): path for the operator complete url.
- Registers a new websocket route for the operator. This method only workstarget (str): target path to directory whose elements will be served.

#### **register\_rproxy\_route(self, path, target)**

Registers a new rproxy route for the operator. This method only works as expected when called during the operator initialization.

Args:

- `path` (str): path from the operator complete url.
- `target` (str): target url.

## **16.2.2.5 List of BaseOperator Attributes**

`self_inst_id`

Operator instance identifier in the graph.

`self_op_id`

Operator identifier. For example: com.sap.dataGenerator.

**self.\_repo\_root**

Path to the modeler root directory.

**self.\_subengine\_root**

Path to the dub-engine root directory.

**self.\_graph\_name**

Graph identifier. For example: com.sap.dataGenerator.

**self.\_graph\_handle**

Unique graph instance identifier.

**self.\_group\_id**

Group at which the subgraph is being executed. If no graph is specified the default is: default.

## 16.2.2.6 List of Metric Methods

**include\_value(self, value)**

This method allows the user to update a metric value. Depending on the 'metric\_info' of that metric the update will be different.

Args:

- value (int\float): New metric value, the value type should follow 'metric\_type'.

## 16.2.2.7 Logging

There are a set of built-in methods to enable logging in the Python subengine. The table below summarizes the set of methods you can use to log information.

Subclassing BaseOperator	Description
self.logger.info(str)	Log with level <i>INFO</i>
self.logger.debug(str)	Log with level <i>DEBUG</i>
self.logger.error(str)	Log with level <i>ERROR</i>
self.logger.warning(str)	Log with level <i>WARNING</i>
self.logger.fatal(str)	Log with level <i>FATAL</i> (stops the graph)
self.logger.critical(str)	Log with level <i>CRITICAL</i> (stops the graph)

### i Note

If you want to report an error and stop the graph from inside the operator code, raise an exception instead of logging with the `logger.fatal( " " )` or `logger.critical( " " )` command. Using those will have unintended consequences.

### i Note

If you start a new thread inside your operator then you should capture your exceptions inside this thread and use `self._propagate_exception(e)` function so that the main thread can handle it.

You can check all logs using SAP Data Intelligence Modeler's UI. To do that, you must click the [Trace](#) tab. You can filter the logging messages by level or you can even search for a specific log. You can also download the logging history as a CSV file.

You can also send the logging messages to SAP Data Intelligence Modeler's external log. To achieve that, open [Trace Publisher Settings](#), next to the search bar. Turn on [Trace Streaming](#) and configure the set of logging messages you want to publish. For example, if you wanted to publish all logging messages which have level *fatal* or *error*, you would have to change [Trace Level](#) to *ERROR*.

## 16.2.2.8 Uploading to SAP Data Intelligence System Management

You can either upload your solution to the SAP Data Intelligence cluster as a new solution (need to be logged as an admin) or you can upload your solution into your tenant or user workspace.

### Related Information

[Uploading Solution in System Management \[page 272\]](#)

[Uploading Solution in System Management to Tenant or User Workspace \[page 273\]](#)

### 16.2.2.8.1 Uploading Solution in System Management

#### Procedure

1. First you need to compress your solution to a .tar.gz format: 

```
tar -czf my_pysolution.tar.gz -C my_pysolution/ .
```
2. Log into SAP Data Intelligence System Management as Tenant Administrator.



3. Choose the *Tenant* tab on the horizontal bar at the top of the screen.
4. Click *Solutions* tab under Cluster Management and then the **+** (Add Solution) to create a new solution. Give a name in the *File name* field and find the `my_solution.tar.gz` file in your system and click *Open*.
5. Add this new layer to an existing strategy or create a new one by clicking the *Strategies* tab.  
If you create a new strategy you will need to associate it with a tenant in the *Tenants* tab.

## Results

If you log into a user from the tenant that you just associated the strategy containing the new layer, you will be able to see the operators from your solution when you launch a new SAP Data Intelligence Modeler instance or when you restart an existing one.

## 16.2.2.8.2 Uploading Solution in System Management to Tenant or User Workspace

### Context

In this option you will upload your solution to the workspace of a given tenant or user in the SAP Data Intelligence cluster. The downside of this option is that it cannot be reversed easily. In the first option you can always remove a solution from a given strategy. On the other hand, in this option if some of the files in your solution overwrite some existing files in the tenant or user workspace, this will not be able to be reversed.

### Procedure

1. First you need to compress your solution to a `.tar.gz` format: `tar -czf my_pysolution.tar.gz -C my_pysolution/ .`
2. Log into your user in SAP Data Intelligence System Management and click the *File* tab at the top of the screen.
3. Make sure no folder is selected in the *File* window.
  - a. For the *current user* - Click on *Import File* > *Import Solution File* button on the *My Workspace* section.
  - b. For *all users* in your current tenant (only possible for the tenant admin) - Click on *Import File* > *Import Solution File* button on the *Tenant Workspace* section
4. In the popup window, select your `tar.gz` file and your solution will then be uploaded to the selected workspace.

## Results

You will be able to see the operators from your solution when you launch a new SAP Data Intelligence Modeler instance or restart the application.

## 16.3 Working with the Node.js Subengine to Create Operators

Use the `Node.js` subengine to code custom operators in a specified programming language to use in the SAP Data Intelligence Modeler application.

The `Node.js` subengine runs `Node.js` operators with other operators designed for different platforms, like Go, Python, and C++. The `Node.js` subengine runs individual operators or entire subgraphs. The more you directly interconnect `Node.js` operators, the bigger the pure `Node.js` subgraph.

The `Node.js` subengine provides the following features:

- Use modern JavaScript, such as ECMAScript V6, to develop operators.
- Use the Node JavaScript runtime environment built on Google Chrome V8.
- Use your preferred language that can be compiled into JavaScript, such as TypeScript.
- Use your own required set of third-party libraries.

### Related Information

[Node.js Operators and Operating System Processes \[page 274\]](#)

[Use Cases for the Node.js Subengine \[page 275\]](#)

[The Node.js Subengine SDK \[page 276\]](#)

[Node.js Data Types \[page 279\]](#)

[Node.js Safe and Unsafe Integer Data Types \[page 279\]](#)

[Create a Node.js Operator \[page 280\]](#)

[Node.js Project Structure \[page 282\]](#)

[Node.js Project Files and Resources \[page 282\]](#)

[Node.js Subengine Logging \[page 285\]](#)

### 16.3.1 Node.js Operators and Operating System Processes

Every `Node.js` operator runs in its own operating system process, which allows the `Node.js` subengine to use multicore CPUs and parallel system architectures.

Running `Node.js` operators in their own operating system provides isolation between `Node.js` operators for enhanced security.

Node.js processes communicate by TCP socket-based IPC (interprocess communication), which is coordinated by the Node.js subengine. In turn, the Node.js subengine runs in its own process. There's one coordinating Node.js subengine process per subgraph that consists of only Node.js operators.

### i Note

All multiplexers run directly in the Node.js subengine process, which reduces communication overhead. Therefore, ensure that you set the *Subengine* to `com.sap.node` in the Node.js subgraph configuration pane. Otherwise, the subgraph divides into two Node.js subgraphs with a multiplexer in between running in Go or another subengine.

### ❖ Example

A subgraph consists of 10 Node.js operators. Three of the nodes are multiplexer. The multiplexer nodes are configured to run in subengine 'com.sap.node'. The number of operating system processes computes to eight operating system processes as follows:

```
1 (Node.js Subengine) + 10 (Node.js operators) - 3 (Node.js Multiplexer) = 8
operating system processes.
```

### i Note

A graph can result in multiple Node.js subgraphs, depending on which operators are interconnected.

## 16.3.2 Use Cases for the Node.js Subengine

There are two use cases for the Node.js subengine: Use the Node.js subengine in the SAP Data Intelligence Modeler application, or use a dedicated local Node.js project to develop a Node.js operator.

The Node.js subengine consists of two major building blocks:

- Node.js subengine core
- Node.js Subengine SDK

While you probably never work directly with the core engine, you'll work with the Node.js Subengine SDK to develop your operators. For more information about the SDK, see [The Node.js Subengine SDK \[page 276\]](#).

### Use Node.js in the Modeler

Using the SAP Data Intelligence Modeler is the easiest way to change or add graph functionality quickly. Just add a Node.js Script operator to existing graphs, or modify the JavaScript code of existing Node.js script operators.

### i Note

The only external node-module that can be required inside the SAP Data Intelligence Modeler is '@sap/vflow-sub-node-sdk'.

## Use Node.js in a Dedicated Project

To develop a Node.js operator, use a dedicated local Node.js project. Using a dedicated project allows you to use your own tools and frameworks, and code Node.js operators like any other Node.js program.

Some of the advantages of using a dedicated project include the following:

- Use JavaScript libraries that you've already created.
- Use your own required set of third-party node modules.
- Use test driven development.
- Use other languages that compile to JavaScript, such as TypeScript.

## Related Information

[Availability of the Node.js Subengine SDK \[page 277\]](#)

### 16.3.3 The Node.js Subengine SDK

The Node.js subengine SDK module simplifies the development of Node.js operators.

The Node.js subengine SDK is a Node.js module named `'@sap/vflow-sub-node-sdk'`. The Node.js module allows easy access to the following elements:

- In and out ports defined in the `operator.json` file.
- Static operator configuration defined in the `operator.json` file.
- Additional ports that you add directly in the SAP Data Intelligence Modeler.
- Dynamic configuration that you add directly in the SAP Data Intelligence Modeler.
- System-logging API with which you send status information about the operator.

The Node.js SDK also supports a shutdown hook for the operator. The shutdown hook cleans resources, if necessary, such as closing files or logging out from remote systems before terminating the operator.

## Related Information

[Availability of the Node.js Subengine SDK \[page 277\]](#)

[Node.js SDK API Reference \[page 277\]](#)

### 16.3.3.1 Availability of the Node.js Subengine SDK

The Node.js SDK is included into the Node.js subengine, but you can also download and install it.

Configure any Node.js operator to require the Node.js SDK at runtime. The Node.js operator uses the Node.js subengine SDK module named '@sap/vflow-sub-node-sdk' as follows:

```
const { operator } = require("@sap/vflow-sub-node-sdk");
const operator = Operator.getInstance();
```

However, if you use a dedicated local Node.js development project, download and install the Node.js subengine SDK so that you can program against its API.

### 16.3.3.2 Node.js SDK API Reference

To use the Node.js SDK, use the API commands as described in this reference.

#### getInstance(basePath)

- `basePath` <string> Optional. The path to the operator descriptor (operator.json) directory. Default `basePath` is the directory where the operator's script file is located.
- Returns: <Operator>

Creates an operator instance (singleton).

#### config

- Returns: <object>

`operator.config` returns the configuration of this operator. If you change this operator in the Modeler application, the command returns the runtime configuration. Otherwise it returns the design time configuration, which is specified in the operator.json.

#### addShutdownHandler(handler)

- `handler` <Function> The handler to call before terminating this operator.
- Returns: <void>

`operator.addShutdownHandler()` adds a shutdown handler when terminating the process. The method accepts a callback function that takes only one optional error parameter, for example, taking an `(err) => ... callback`.

The operator fails when the shutdown handler is not a function or it is undefined.

The following code snippet shows how to use a callback function:

```
const handler = (cb: (err?: Error) => void): void => {
  // do something ...
  cb(...); // callback with void or an error
};
const operator: Operator = Operator.getInstance();
operator.addShutdownHandler(handler);
```

#### done()

- Returns: `<void>`

Terminates the operator process with exit code 0.

### **fail(message)**

- `message` `<string>` Error message to write when the operator fails.
- Returns: `<void>`

Terminates this operator process with exit code 1, then writes an error message to `process.stderr`.

### **getInPort(name)**

- `name` `<string>` The name of the input port.
- Returns: `<InPort>`

Finds a single input port by its name and returns the name.

The operator fails if it doesn't find the port.

### **getOutPort(name)**

- `name` `<string>` The name of the output port.
- Returns: `<OutPort>`

Finds a single output port by its name and returns the name.

The operator fails if it doesn't find the port.

### **getInPorts()**

- Returns: `<Map<string, InPort>>`

Returns all input ports of this operator. Returns an empty map when it doesn't find any input ports.

### **getOutPorts()**

- Returns: `<Map<string, OutPort>>`

Returns all output ports of this operator. Returns an empty map when it doesn't find any output ports.

### **logger**

- Returns: `<Logger>`

Returns a logger instance to use for logging. For more information, see [Node.js Subengine Logging \[page 285\]](#).

## 16.3.4 Node.js Data Types

SAP Data Intelligence maps Modeler data types to the Node.js JavaScript data types.

Inside a Node.js operator, you use JavaScript types and data structures. The following table shows the SAP Data Intelligence Modeler data types and the corresponding JavaScript data types:

Modeler Data Type	JavaScript Data Type
string	String
blob	Buffer
int64	Number
uint64	Number
float64	Number
byte	Number
message	Object
[ ]x	Array

Replace the letter x in [ ]x (the last row of the table) with any of the allowed data types (except for message). For example, [ ]string, [ ]uint64, [ ][ ]blob, and so on. The Node.js subengine always does the conversion. If data reaches the Node.js operator script, it's converted already based on the conversions shown in the table.

### i Note

Always use the SAP Data Intelligence Modeler data types to specify the data types of the in and out ports in the `operator.json` file. This mapping is necessary, for example, if you develop a vsolution using an external project.

## 16.3.5 Node.js Safe and Unsafe Integer Data Types

Because JavaScript doesn't have an integer data type, there are safe and unsafe integer data types when you use the Node.js subengine.

SAP Data Intelligence Modeler converts JavaScript number data type to uint64 or int64. The Modeler represents a number data type internally as an IEEE-754 double precision float. Therefore, for Node.js, SAP Data Intelligence Modeler represents only integer values up to 53 bits. Any integer value greater than 53 bits is an unsafe integer data type.

To prevent silent and undetected problems with the JavaScript number conversion, the Node.js subengine stops with a corresponding error message when it receives or sends an unsafe integer value. If a Node.js operator has an unsafe integer value, the corresponding graph also stops.

An unsafe integer value is defined as follows:

### Sample Code

```
Number.isSafeInteger(data) === false
```

The safe integers consist of all integers from  $-(2^{53} - 1)$  to  $2^{53} - 1$  (inclusive). Therefore, the full range of SAP Data Intelligence Modeler types `uint64` and `int64` can't be represented in Node.js JavaScript.

```
type uint64 - range: 0 ... 2**64 -1 (0 through 18446744073709551615)
type int64: -(2**63 -1) ... 2**63 -1 (-9223372036854775808 through
9223372036854775807)
```

If converted into number, the mathematical rules of equality can be violated in JavaScript as shown in the following example code snippet:

```
( 1 === 2 ) === false // <- this always evaluates to false and is mathematically
exact

// Now, add MAX_SAFE_INTEGER to both sides:
( 1 + Number.MAX_SAFE_INTEGER === 2 + Number.MAX_SAFE_INTEGER ) === true // <-
results to true
// Also a legal expression, it is mathematically incorrect
```

For more information about safe integers, see the external link [developer.mozilla.org, datatype \(IEEE-754\)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/MAX_SAFE_INTEGER) .

## 16.3.6 Create a Node.js Operator

Create Node.js operators using a dedicated Node.js project.

### Prerequisites

You must be a cluster administrator to perform this task.

Before performing the following steps, download the Node.js subengine SDK from an SAP Data Intelligence System. For more information, see [The Node.js Subengine SDK \[page 276\]](#).

### Procedure

1. Log in to SAP Data Intelligence System Management as a cluster administrator.
2. Open *Files* and then open *Union View*.
3. Expand the following nodes: *files/vflow/subengines/com/sap/node*.
4. Select *vflow-sub-node-sdk.tar.gz* and choose the *Export files or solutions* icon in the toolbar.
5. Export the file to your local Node.js project. Save it to an appropriate location inside your project.
6. Install the SDK file *vflow-sub-node-sdk.tar.gz* using the Node.js package manager so that it's available to your JavaScript code.

```
npm install --save [path_to]/vflow-sub-node-sdk.tar.gz
```

7. Check that the export added the `(runtime-)` dependency to your `package.json`.

```
{
```



```

    "name": ...,
    "version": ...,
    "dependencies": {
      "@sap/vflow-sub-node-sdk": "file:vflow-sub-node-sdk.tar.gz",
      ...
    }
  }
}

```

8. **Optional:** Personalize the `Node.js` operator behavior in the Modeler in one of the following ways:

Method	Directions
<b>Use Node.js Script operator</b>	<p>The edited script using this method is specific to the given graph and operator instance.</p> <ol style="list-style-type: none"> <li>1. Drag and drop the <code>Node.js Script</code> operator to the graph canvas in the Modeler.</li> <li>2. Open the script editor.</li> </ol>
<b>Create a new operator</b>	<p>The advantage of the following method is that you can reuse the new operator across many graphs.</p> <ol style="list-style-type: none"> <li>1. Create a new operator in a select subfolder in the Operators node of the <i>Repository</i> tab.</li> <li>2. Choose <b>Node Base Operator</b> for <i>Base Operator</i> in the <i>Create Operator</i> dialog box.</li> <li>3. Create input and output ports, tags, configuration parameters, and custom script as necessary.</li> </ol>

## Related Information

[Requiring Node Modules \[page 281\]](#)

### 16.3.6.1 Requiring Node Modules

A Node.js operator runs in its own process, therefore, it can also require its own individual set and version of node modules.

A Node.js operator with its own set of node modules, provides you with greater flexibility, for example:

- Reuse your JavaScript libraries.
- Reuse your third-party Node.js modules.

## 16.3.7 Node.js Project Structure

To deploy a Node.js operator to an SAP Data Intelligence system, create a project structure. To deploy your operator later to a different SAP Data Intelligence system, create a project structure.

Structure your Node.js project as in the following script:

```
| - my_solution/  
  | - vsolution.json  
  | - vrep/vflow/  
    | - subengines/com/sap/node/operators/  
      |-- example  
        |-- operator.json  
        |-- script.js  
        |-- icon.svg
```

Before you deploy this project structure to a different SAP Data Intelligence system, you must first archive it. To create an archive, use the tar archiver (on macOS and Linux) as in the following script:

```
tar -C <path to my_solution> -czf my_solution.tar.gz
```

### Note

To verify the structure of your archive, ensure that the folder `vrep` is at the top level of the archive, beside the file `'vsolution.json'`.

## 16.3.8 Node.js Project Files and Resources

Creating a Node.js operator requires a Node.js project file and other resources.

The resources in a Node.js project includes the following files:

- `vsolution.json`
- `operator.json`
- `script.js`
- `README.md`
- operator icon (`*.svg`, `*.png`, `*.jpeg`)

### `vsolution.json`

The `vsolution.json` file describes the solution. The following script is an example `vsolution.json` file:

#### Example

```
{  
  "name": "vsolution_vflow_my_solution",  
  "version": "n.n.n",  
  "description": "my Solution",  
  "license": "my license"
```

```
}
```

**Increment the version number after every change.** If you don't increment the version number after every change, the SAP Data Intelligence system doesn't overwrite a deployed version.

## operator.json

The `operator.json` specifies the base operator (component), the interface (in ports, out ports), configuration data, and other details of the operator. The following script is an example `operator.json` file:

### ❖ Example

```
{
  "component": "com.sap.node.operator", // extend the basic node operator
  "description": "Operator description", // visible title of the operator in
  operators explorer in SAP Data Intelligence Modeler
  "iconsrc": "icon.svg",
  "inports": [
    {
      "name": "in1",
      "type": "string"
    }
  ],
  "outports": [
    {
      "name": "out1",
      "type": "string"
    }
  ],
  "config": {
    "codelanguage": "javascript",
    "script": "file://script.js"
  }
}
```

If the Node.js operator doesn't have any ports, you can omit the in ports and out ports, or leave them empty. If there is more than one port, the port names must be unique. In the unique port names, exclude special characters, such as white spaces.

You must specify the script configuration under the following circumstances:

- You don't save the script file to the same root path as the `operator.json` file.
- You name the script file differently than the operator name.

If you don't specify the script configuration under these circumstances, SAP Data Intelligence can leave the `operator.json` file out.

## Operator Script

The `script.js` file is the entry point of your operator. The following sample code of a `script.js` file is from the operator 'Node.js Counter', which is also available in the SAP Data Intelligence Modeler:

### ❁ Example

```
const SDK = require("@sap/vflow-sub-node-sdk");
const operator = SDK.Operator.getInstance();
let counter = 0;
/**
 * This operator receives messages on port "in1",
 * increases a counter and forwards the counter value
 * to port "out1".
 */
operator.getInPort("in1").onMessage((msg) => {
  // The content of the actual message is ignored.
  // We will only count the number of messages here.
  counter++;
  operator.getOutPort("out1").send(counter.toString());
});
/**
 * A keep alive hook for the node process.
 * @param tick length of a heart beat of the operator
 */
function keepAlive(tick) {
  setTimeout(() => {
    keepAlive(tick);
  }, tick);
}
// keep the operator alive in 1sec ticks
keepAlive(1000);
```

Whenever the Node.js Counter operator receives a message at input port 'in1', it increments a counter and sends the current counter value to output port 'out1'. To prevent the Node.js process from terminating immediately after the start, a 'keepAlive' timer has been added. It resets a time-out every 1000 milliseconds.

## Operator Documentation

Create the operator's documentation in a file named `README.md` in the operator's folder.

### ❁ Example

To create the documentation for the example operator named Counter, create a new `README.md` file in the following path:

```
my_solution/subengines/com/sap/node/operators/com/mydomain/util/counter/
README.md
```

Use Markdown syntax to write the documentation. For more information about Markdown syntax, see [Basic Writing and Formatting Syntax](#) in the GitHub documentation.

## Operator Icon

You can add a custom icon (file format: svg, .jpg or .png) for your operator by copying the icon to the operator's folder:

### ❖ Example

```
my_solution/subengines/com/sap/node/operators/com/mydomain/util/counter/  
icon.svg
```

## 16.3.9 Node.js Subengine Logging

The logging library provides different logging APIs that propagate logging information to the Node.js subengine. In local development that isn't running with Node.js subengine core, the console is used for logging.

## Logging Levels

Logging levels use the npm severity ordering, which ascends from the most important to the least important. In the following example, errors are the most important and debug level is the least important:

### ❖ Example

```
{  
  ERROR: 0,  
  WARN: 1,  
  INFO: 2,  
  DEBUG: 3,  
}
```

## Setting Logging Levels

By default, the logging level is set to *INFO*. You can change the log level as shown in the following code snippet:

```
const operator: Operator = Operator.getInstance();  
operator.logger.logLevel = "ERROR"; // case insensitive
```

## Use the Logging API

The logging APIs are directly accessible from the operator instance. The following code snippet shows the logging APIs:

```
const operator: Operator = Operator.getInstance();
// if not running in subengine, log is outputted to console
operator.logger.info("message", ...); // log level INFO
// DEBUG level
operator.logger.debug("message", ...);
// WARNING level
operator.logger.warn("message", ...);
// ERROR level
operator.logger.error("message", ...);
```

You can also create a logger instance in your operators as shown in the following code snippet:

```
import { Logger } from "@sap/vflow-sub-node-sdk";
const logger: Logger = new Logger("LOG_LEVEL");
logger.info("message");
```

## Log Message Format

The log message accepts zero or more placeholder tokens. The system replaces each placeholder with the converted value from the corresponding argument. The log message is then a `printf`-like format string. For supported placeholders, see [nodejs.org](https://nodejs.org) on the Nodejs.org website.

```
operator.logger.debug("debug message %j", { foo: "bar" });
// 2018-11-22T13:03:01.088Z - debug: Operator "sampleiotdevicegeneratingdata1"
(pid: 11912)|debug message { "foo": "bar" }
```

## 16.4 Working with Flowagent Subengine to Connect to Databases

Partitioning the source data allows us to load data in chunks there by overcome memory pressure and by doing it in parallel we can load data faster and improve overall performance.

### Partitioning

Below are supported partitioning methods with the [Connectivity \(via Flowagent\)](#) operators.

#### Logical Partition

Logical Partitions are user-defined partitions on how to read data from source. For Table Consumer, the user can choose the partition type then add the partition specification as described below.

- `defaultPartition`: The partition which fetches all rows that are not filtered by the other partitions (you can disable it by setting `defaultPartition` to `false`);
- `conditions`: Each condition represents a set of filters that are performed on a certain column;
  - `columnExpression`: The column where the filters are applied. It can be either the column name or a function on it (for example: `TRUNC(EMPLOYEE_NAME)`);
  - `type`: The filter type. It can be either "LIST" (to filter exact values) or "RANGE" (to filter a range of values);
  - `dataType`: The data type of the elements. It can be either "STRING", "NUMBER" or "EXPRESSION";
  - `elements`: Each element represents a different filter, and its semantics depends on the "type" (see above).
- When more than one condition is presented, a cartesian product performed among the elements of each condition. For example:
  - Condition A: [C1, C2]
  - Condition B: [C3, C4]
  - Resulting filters: [(C1, C3), (C1, C4), (C2, C3), (C2, C4)]

Ex: Let's assume we have a source table called EMPLOYEE that has the following schema:

```
CREATE TABLE EMPLOYEE(
  EMPLOYEE_ID NUMBER(5) PRIMARY KEY,
  EMPLOYEE_NAME VARCHAR(32),
  EMPLOYEE_ADMISSION_DATE DATE
);
```

## RANGE

A JSON representing a RANGE partition is shown below, where the elements represent ranges. Thus, they need to be sorted in ascending order.

### Numeric Partitions

```
{
  "defaultPartition": true,
  "conditions": [
    {
      "columnExpression": "EMPLOYEE_ID",
      "type": "RANGE",
      "dataType": "NUMBER",
      "elements": [
        "10",
        "20",
        "30"
      ]
    }
  ]
}
```

### String Partitions

```
{
  "defaultPartition": true,
  "conditions": [
    {
      "columnExpression": "EMPLOYEE_NAME",
      "type": "RANGE",
      "dataType": "STRING",
      "elements": [
        "M",

```

```

    "T"
  ]
}
]
}

```

## Expression Partitions

```

{
  "defaultPartition": true,
  "conditions": [
    {
      "columnExpression": "EMPLOYEE_ADMISSION_DATE",
      "type": "RANGE",
      "dataType": "EXPRESSION",
      "elements": [
        "TO_DATE('2012-01-01', 'YYYY-MM-DD')",
        "TO_DATE('2015-01-01', 'YYYY-MM-DD')"
      ]
    }
  ]
}

```

## LIST

List partitions can be used to filter exact values. Each element represents a different filter.

### Numeric partitions

```

{
  "defaultPartition": true,
  "conditions": [
    {
      "columnExpression": "EMPLOYEE_ID",
      "type": "LIST",
      "dataType": "NUMBER",
      "elements": [
        "10",
        "20",
        "50"
      ]
    }
  ]
}

```

### String partitions

```

{
  "defaultPartition": false,
  "conditions": [
    {
      "columnExpression": "EMPLOYEE_NAME",
      "type": "LIST",
      "dataType": "STRING",
      "elements": [
        "Jhon",
        "Ana",
        "Beatrice"
      ]
    }
  ]
}

```



## Expression partitions

```
{
  "defaultPartition": false,
  "conditions": [
    {
      "columnExpression": "TRUNC(EMPLOYEE_ADMISSION_DATE)",
      "type": "LIST",
      "dataType": "EXPRESSION",
      "elements": [
        "TO_DATE('2012-07-17', 'YYYY-MM-DD')"
      ]
    }
  ]
}
```

## COMBINED

```
{
  "defaultPartition": true,
  "conditions": [
    {
      "columnExpression": "EMPLOYEE_NAME",
      "type": "LIST",
      "dataType": "NUMBER",
      "elements": [
        "Beatrice",
        "Ana"
      ]
    },
    {
      "columnExpression": "EMPLOYEE_ADMISSION_DATE",
      "type": "RANGE",
      "dataType": "EXPRESSION",
      "elements": [
        "TO_DATE('2015-01-01', 'YYYY-MM-DD')",
        "TO_DATE('2016-01-01', 'YYYY-MM-DD')"
      ]
    }
  ]
}
```

### i Note

- Operator auto-generates the default partition, so you don't have to define it. It can be disabled by setting the property "defaultPartition": false.
- For range partition the default partition is greater than last element. So ensure that the elements are ordered in ascending order.

## Physical Partition

### i Note

Applicable to [Oracle Table Consumer \(Deprecated\)](#) only.

Physical partitions are partitions defined directly on the source, using Oracle partitioning concept.

Limitations:

- Hash partitioning is not supported.
- Sub-partitioning is not supported.

## Row ID Partition

### i Note

Applicable to [Oracle Table Consumer \(Deprecated\)](#) only.

Row ID partitions are partitions generated automatically based on the row id of the columns. You must supply the number of partitions, and the range of row id partitions is generated automatically based on it.

### Parallel Load With Partitioning

When an operator has partitions, it sends SQL statements in the output port equal to the number of partitions (including the default partition), each SQL with a WHERE condition equivalent to the partition. In order to enable parallel load of partitions, it is necessary to put the next Flowagent producer in the pipeline inside a group with multiplicity higher than one, preferentially with multiplicity equal to the number of partitions to enable parallel load of all partitions at the same time.

Note that each producer inside the group generates an output. If you want to prevent the next operator after the producer to be triggered before all partitions are finished (example: Graph Terminator), connect the producer to a Constant Generator with `counter` equal to the number of partitions.

### Partition Recovery

Currently there are none automatic recovery mechanism in place. In case any partition fail during the load, the graph stops (unless the error port of the producer is mapped).

Consider the following for partition recovery:

- Retrigger the graph again with table producer mode set to truncate or overwrite;
- Use a Javascript operator to retrigger only the failed partitions (see the sample graph provided here).

# 17 Creating Dockerfiles

Dockerfiles contain all commands that you call on a command line to assemble a docker image.

## Context

In the Modeler application, you can create a library of Dockerfiles. Dockerfiles provide a predefined runtime environment to run the operators in a graph (pipeline). The Modeler stores Dockerfiles in the repository in a Dockerfile folder.

## Procedure

1. Open the *Repository* tab in the navigation pane of the Modeler application.
2. Create a folder.
  - a. Right-click the Dockerfiles category folder and choose *Create Folder*.
  - b. Enter a name for the root folder in *Name your folder* text box and select *Create*.

### i Note

To create a folder structure (subdirectories) in the root folder, create a new folder in a subdirectory under the dockerfiles folder.

The Modeler creates a new folder in which you create a Dockerfile.



3. Create a Dockerfile.
  - a. Right-click the folder in which to create the Docker file and choose *Create Docker File*.
  - b. Enter a name in *Name*.
  - c. Choose *OK*.
  - d. In the editor, write the script that defines your Dockerfile.




### i Note

For security reasons, SAP recommends that you start the image with the non-root user.

4. Create tags.

Tags describe the runtime requirements of operators and are the annotations of Dockerfiles that you create.

  - a. In the editor toolbar, choose  (*Show Configuration*).
  - b. In the Configuration pane, choose  (*Add Tag*).
  - c. Choose a tag from the first text box list.
  - d. Enter a version in the second text box.

5. In the editor toolbar, choose  (*Save*).
6. Build a docker image.
  - a. In the editor toolbar, choose  (*Build*) to build a docker image for the Dockerfile.
  - b. **Optional:** Select  (*Force Build*) in the editor toolbar.

*Force Build* rebuilds the existing Docker image. The Modeler finds changes in the Docker image and rebuilds based on the changes.

## Next Steps

For security reasons, the Modeler application has stopped supporting images that run with a root user.

```
RUN groupadd -g 1972 vflow && useradd -g 1972 -u 1972 -m vflow
USER 1972:1972
WORKDIR /home/vflow
ENV HOME=/home/vflow
```

### ! Restriction

Use numeric IDs in the USER directive instead of using group names. Otherwise the graph run fails.

Make sure that the commands you run after the USER directive operate with the new environment. Therefore, consider the following situations:

- If you install new Python packages with `pip`, ensure that you have the `--user` flag for local installations.
- If you install software without using the package manager, or you install software that doesn't support local installation, then you must install the software manually inside the user home directory, such as `/home/vflow`. Do the same for any other files added to the image.

## Related Information

[Dockerfile Inheritance \[page 292\]](#)

[Referencing Parent Docker Images \[page 294\]](#)

## 17.1 Dockerfile Inheritance

The Dockerfile inheritance feature in SAP Data Intelligence allows inheriting attributes, such as tags, from other Dockerfiles defined in the Modeler.

It's possible to inherit attributes from other Dockerfiles defined in the Modeler by using the following syntax:

```
docker
FROM $<dockerfile_id>
...
```

Dockerfiles in the Modeler repository each have a corresponding file named `Tags.json`, which is always located in the same directory as the Dockerfile. Along with the software modules used by the Dockerfile, the `Tags.json` file contains the following information about the Dockerfile:

- Runtimes to use, such as Python.
- Tools to use, such as Zypper, tar, and gzip.
- Compilers to use, such as g-cc or gcc+.

The information in the `Tags.json` file allows the Modeler to select the correct Docker image at runtime for the operators and groups in a graph. For more information about Dockerfiles, see [Groups, Tags, and Dockerfiles \[page 94\]](#).

The Modeler determines whether a child Dockerfile inherits tags from a parent Dockerfile using the criteria in the following table.

Tag Origin	Inherited	Not Inherited
Child Dockerfile (not in the parent Dockerfile)	X	
Parent Dockerfile (not in the child Dockerfile)	X	
Child and parent Dockerfile (uses the tag value from the child Dockerfile)	X	
Parent Dockerfile, marked as special "default"		X

### Note

The Modeler never transfers the special tag "default" from the parent to the child Dockerfile, but it still transfers the tag "deprecated".

### Example

To create a new Dockerfile with instructions to install the NumPy library to use in a Python3 operator, use the prebuilt `com.sap.sles.base` Docker image. This Docker image already satisfies the requirements of the Python3 operator. The following code snippet shows the entry in the new Dockerfile:

```
docker
FROM $com.sap.sles.base
RUN pip3.9 install numpy
```

For the new Dockerfile, add the tag "numpy39" to the `Tags.json` file located in the same directory as the new Dockerfile:

```
{
  "numpy39": ""
}
```

The Modeler expands the new Dockerfile tags automatically to include the tags from the `com/sap/sles/base/Tags.json` file at runtime because of the automatic tags inheritance feature.

## i Note

For more control over the Docker image, such as installing additional compilers and tools with Zypper, create a Dockerfile that only you control. To create this Dockerfile, SAP includes a Dockerfile template named `org.opensuse` in the repository at `dockerfile/org/opensuse`. Use the template to create a customized copy. For more information about the `org.opensuse` template, see the `README.md` file in the Modeler's navigation pane, under the *Repository* tab: `dockerfiles/org/opensuse/`.

## Added Security

For security reasons, SAP stopped supporting images that run with a root user in SAP Data Intelligence on-premise version 3.0 and cloud version 2003. Therefore, SAP modified the Docker images to create and use nonroot users. If you own a custom Dockerfile that doesn't inherit from a Dockerfile delivered by SAP, add code to your Dockerfile similar to the following example:

## ❖ Example

```
RUN groupadd -g 1972 vflow && useradd -g 1972 -u 1972 -m vflow
USER 1972:1972
WORKDIR /home/vflow
ENV HOME=/home/vflow
```

SAP requires that you use numeric IDs in the `USER` directive instead of the user and group names. If you don't use numeric IDs, the graph doesn't run. Ensure that the commands you execute after the `USER` directive work with the new environment as follows:

- When you install new Python packages with “pip”, the install must have the “user” flag for local installation.
- When you install other software without the package managers that support user local installation, you must install the software manually inside your home directory, such as `/home/vflow`. Install any other files added to the image in the same way.

## Related Information

[Python3 Operator V2](#)

## 17.2 Referencing Parent Docker Images

To create a new Docker image, SAP Data Intelligence Modeler allows you to reference a parent Docker image, which can be an existing Dockerfile from the repository or a prebuilt Docker image.

In a Dockerfile, use the `FROM` instruction to specify the parent Docker image from which to build a new image. The following sections contain information about each type of parent Docker image to use, and includes the method with additional considerations.

## Use an Existing Dockerfile with the '\$' Symbol.

### Parent Docker Image

Existing Dockerfile

### Method

```
FROM $
```

### Details

Use the '\$' symbol to name an existing Dockerfile from the repository. The '\$' symbol provides inheritance of the tags and the resource limits of the referenced Dockerfile. The inheritance is transitive; the Docker image can inherit from a Dockerfile that already inherits from another Dockerfile. The inheritance ends when the system encounters either a reference to a custom Dockerfile or a '\$' reference.

When you build a Dockerfile that uses inheritance, the system builds all necessary images in the inheritance chain as needed. However, you can't reference a specific version of the parent Docker image directly using the '\$' symbol because the topmost Dockerfile in the inheritance chain has already defined the version of the parent Docker image to use. For more information about inheritance, see [Dockerfile Inheritance \[page 292\]](#).

#### i Note

Compared to the '§' symbol, the '\$' symbol references another Dockerfile and not a prebuilt Docker image.

In the following Dockerfile example, the Dockerfile inherits all properties from the existing Dockerfile `com.sap.sles.base`. The definition of the referenced Dockerfile is in the repository at `com/sap/sles/base`. The new parent Docker image inherits all tags and resource limits of the referenced Dockerfile.

#### ❁ Example

```
FROM $com.sap.sles.base
...
```

#### i Note

If you want to specify a specific version of the parent Docker image, use the '§' symbol method, which bases your Docker image on a prebuilt Docker image.

## Considerations for Using an Existing Dockerfile

### Advantages

- Ready to use
- Inherited tags from parent Docker image
- Automatic updates
- Few customizations needed

### Disadvantages

- No control over updates

## Use a Prebuilt Docker Image with the '\$' Symbol

### Parent Docker Image

Prebuilt Docker image

### Method

```
FROM $
```

### Details

SAP Data Intelligence includes prebuilt Docker images with the software package and stores the images in the Docker registry. The '\$' symbol is a placeholder for the SAP Data Intelligence Docker registry address. In the Dockerfile, enter the name and version of the parent Docker image after the location.

#### i Note

Compared to the '\$' symbol, the '§' symbol references a prebuilt Docker image and not another Dockerfile.

The following example Dockerfile references the address for the Docker registry that contains the prebuilt Docker image and version `vflow-customer-sles:<version>`:

#### ❁ Example

```
FROM §/com.sap.datahub.linuxx86_64/vflow-customer-sles:<version>
...
```

#### i Note

When you reference a specific version, there's a risk that the image isn't available over time. However, SAP guarantees certain versions and duration of availability of the Docker image `vflow-customer-sles`. For details, see [3320629](#). SAP recommends that you check this SAP Note before every system upgrade.

## Considerations for Using a Prebuilt Docker Image

### Advantages

- Explicit image URL
- Controllable updates

### Disadvantages

- SAP controls the image updates
- Tags are provided on inheriting image
- More customizations needed

## Use a Prebuilt Image from a Custom Docker Registry

### Parent Docker Image

Prebuilt Docker image from custom Docker registry

### Method

```
FROM
```



## Details

Use this method to create a new Docker image based on a prebuilt image that you store in a custom Docker registry.

In the following example, the Dockerfile references a prebuilt Docker image, `my-image:1.0.0`, from a custom Docker registry, `<custom-registry>`:

### ❖ Example

```
FROM <custom-registry>/my-image:1.0.0  
...
```

For details on how to configure a custom Docker registry, see [Manage Modeler Custom Registry Secret](#).

## Considerations for Using a Prebuilt Image From a Custom Docker Registry

### Advantages

- Full control over image content (except for SAP requirements)
- Full control over image lifecycle (except for SAP requirement changes)

### Disadvantages

- Greater initial effort
- Required to provide operator and subengine requirements to use SAP standard
- Manual updates
- No SAP testing

# 18 Creating Configuration Types

Configuration types are JSON files that allow you to define properties and bind them with data types. You can also associate the properties with certain validations, define its UI behavior, and more.

## Context

Configuration types are based on JSON schema, but you can create them in the SAP Data Intelligence Modeler application. Reuse your configuration types, for example, in the type definition to define the operator configurations or of specific parameters within the operator configuration definition.

The Modeler also provides global configuration types, which are associated with the configuration parameters of certain default operators (base operators) available within the application. To create a new configuration type, perform the following steps in the Modeler application:

## Procedure

1. Open the *Configuration Types* tab in the navigation pane.
2. Choose **+** (*Create Configuration Type*) in the navigation pane toolbar.
3. **Optional:** Type a helpful description for the new configuration type in the *Description* text field.
4. Choose **+** (*Add property*) in the *Properties* pane.  
Configuration types can have more than one property.
5. Type a name and display name for the property, and optionally enter a description.  
The application uses the value in the *Title* as the display name for the property in the UI.



### i Note

If you don't provide a *Title*, then the application uses the value in the *Name* text field as the display name.

6. Select the required data type value from the *Type* list based on the descriptions in the following table.

Value	Description
String	<p>Define helpers for properties of data type string. Helpers allow you to identify the property type and define values accordingly. Complete the following properties as helpers:</p> <ul style="list-style-type: none"> <li><i>Format</i>: Select the applicable format.</li> </ul> <div style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> <p><b>i Note</b></p> <p>Applicable only when you select <i>Auto</i> for <i>UI Control</i>. The Modeler supports the formats, Datetime, Email, URL, Password, or com.sap.dh.connection.id for properties of data type string.</p> </div> <ul style="list-style-type: none"> <li><i>Value Help</i>: Select one of the following values: <ul style="list-style-type: none"> <li>Pre-defined Values: Preconfigure the property with a list of values that your users can choose from the user interface. The Modeler displays the property as a dropdown list of values. When you select Pre-defined Values, you must also provide a list of values in the <i>Values</i> text field.</li> <li>Value from Service: Specify a URL to obtain the property values from the REST API. The application displays the response from the service call as auto-suggestions to users. In the <i>Url</i> text field, specify the service URL. The response from the REST API can be an array of strings or an array of objects. If the response is an array of objects, in the <i>Value path</i> field, provide the name of an object property. The application renders the values of this object property in a dropdown list in the UI to define the operator configuration. If the response is object, in the <i>Data path</i> field, provide the name of an object property.</li> </ul> </li> </ul> <div style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> <p><b>! Restriction</b></p> <p>The URL should be of the same origin. Cross-origin requests aren't supported.</p> </div>
Object	For properties of data type object, the application lets specify the schema of the object by drilling down into the object definition. In the <i>Properties</i> section, double-click the property to drilldown further and define the object schema.
Array	<p>For properties of data type array, you can specify the data types of items in the array. In the <i>Item Type</i> dropdown list, select a value. The application supports string, number, object, and custom as data types for array items.</p> <p>If the <i>Item Type</i> is <i>Object</i>, in the <i>Properties</i> section, double-click the property to drilldown further and define the object schema.</p> <p>If the <i>Item Type</i> is <i>Custom</i>, select and reuse any of the predefined types for the property definition.</p>
Number	For properties of data type number, you can provide numbered values to the property.
Boolean	For properties of data type Boolean, you can provide Boolean values to the property.
Integer	For properties of data type integer, you can provide integer values to the property.

Value	Description
Custom Type	Custom data types enable you to set the data type of a property to another user-defined type. In the <i>Type</i> dropdown list, select a value. The application populates the dropdown list with the global schema types.

7. **Optional:** Set the *Required* toggle to true (on).  
When on, users must provide values to the property in the user interface. The property value can't be empty.
8. **Optional:** Set *ReadOnly* to true (on).  
Applicable for certain types. When on, the Modeler doesn't allow any edits to the property from the user interface.
9. **Optional:** Control property visibility by selecting one of the following options:
  - *Visible:* All properties are visible in the user interface.
  - *Hidden:* Some properties are visible in the user interface.
  - *Conditional:* Properties are visible in the user interface based on set conditions. The application performs AND operation of all conditions to determine whether the property is visible or hidden.
10. Create additional properties as necessary.
11. Choose  (*Save*) and enter a name that includes the fully qualified path to the new configuration type.  
For example, `com.sap.others.<typename>`.  
The Modeler stores the configuration type in a folder structure in the repository.
12. **Optional:** To save another instance of the configuration type, choose  (*Save As*) and enter a different name with the fully qualified path.

## Related Information

[Creating Operators \[page 36\]](#)

# 19 Security and Data Protection

When you develop a threat model, you must consider how to protect your data and keep personal identifiable information (PII) private.

## Data Protection and Privacy (DPP)

You, as a user of SAP Data Intelligence, are responsible for keeping the PII in your data private and secure. While SAP Data Intelligence acts as the data processor, you control the input and output of data processed through SAP Data Intelligence.

### ❖ Example

The Modeler application runs graphs. However, you instruct the Modeler to process data in the graph in a specific way, and you initiate the graph to run.

SAP Data Intelligence doesn't audit log the input of personal or sensitive data from source systems, nor does it audit log transformations or ingestions into target systems. You, as the data owner (the owner of the source and target systems) are responsible to ensure traceability. You must instruct source and target systems to properly generate relevant audit logs so that you're compliant with local data protection and privacy laws.

## Sensitive Information in Logs

The Modeler creates trace information logs and includes design time objects, such as solution files, custom operators, and pipeline descriptions. Therefore, ensure that you don't include sensitive information statically in those types of design time objects. Instead, use alternative methods for securing storage of connectivity information, such as the Connection Manager.

## Audit Logging Recommendations

SAP Data Intelligence uses the Modeler to create and run data pipelines (graphs) that access source and target systems. Based on your DPP requirements, consider SAP Data Intelligence audit logging behavior. Then revise audit logging configuration in your source and target systems to comply with established DPP regulations, such as GDPR, to log security, configuration, personal data read, and change events for relevant data.

For more information about DPP, see "Data Protection and Privacy in SAP Data Intelligence" in the *Administration Guide*.

# 20 Using Data Types

SAP Data Intelligence uses data types for compatibility, structure, and ease of converting data types between operators.

Data types provide a strong type system for graphs (pipelines). Data types provide the following advantages:

- Compatibility checks for connections between operators.
- A structured way of specifying messages and carrying metadata through the graph.
- Converts data types without the need for specific type converter operators.

### i Note

The Modeler stores all default data types in the *Data Types* tab in the navigation pane.

## Data Type Categories

In SAP Data Intelligence, data types have the following data type categories:

Category	Description
Scalars	Primitive data types used as base for structures and tables.  <b>Example</b> Integers, floats, strings, and dates.  Create new scalar data types only in the local scope. You can't create new scalars in the Global Scope.
Structures	Collection of fields where each field has a name and a value. Field names are strings, but values can be any existing scalar data type.
Tables	Complex data types composed of fields (as columns). Each field has a list of values. The fields are meant to model tabular entities that are typical of SQL databases. Each table can be interpreted as a list of similar structures. Therefore, each table field (column) is composed of scalars.

You can create structure or table data types. When creating the structure data type, you need to select the scalars to compose the structure. When creating table data types, you can also define the primary key columns.

## Data Type Scopes

Data Types can have the scopes described in the following table.

Scope	Description
Global	Data types that are global are accessible to any graph (pipeline).
Local	Local data types are visible only in the graph in which the data type is created.
Dynamic	Dynamic data types are created during runtime and exist only in memory.

### i Note

Dynamic data types are created by the operator's internal API. They're only available for the Generation 2 Python operators. In the port specification, the dynamic scope is indicated by an asterisk (\*) as the Data Type ID. Keep in mind the different data types are still valid. Therefore, a table can only be connected to another table port.

### i Note

All pre-existing data types are global.

## Related Information

[Creating Global Data Types \[page 303\]](#)

[Creating Local Data Types \[page 305\]](#)

## 20.1 Creating Global Data Types

Extend the data type system provided by SAP Data Intelligence Modeler by creating new data types.

### Prerequisites

Before you create data types, ensure that you understand the concepts of data type category, scope, and compatibility. For complete information about data types, see [Using Data Types \[page 302\]](#) and [Data Types in Operator Ports \[page 32\]](#).

## Context

You can create global data types in addition to the data types provided by SAP Data Intelligence. The system also creates dynamic data types during runtime that are applicable only for the Generation 2 Python operators.

The following steps are for creating global data types.

There are three data types: Scalar, structure, and table:

- The scalar data types are predefined.
- You can define structure and table data types.

To create structure or table data types, perform the following steps:

## Procedure

1. Start the SAP Data Intelligence Modeler.
2. In the navigation pane, choose the *Data Types* tab.

If you don't see the *Data Types* tab, select the ▼ *(Additional Tabs)* icon at the bottom of the tab list and select *Data Types*.

3. In the navigation pane toolbar, choose + *(Create Data Type)*.

The *Create Data Type* dialog box opens.

4. Enter a name for the data type in the *ID* text box.

The name must be two or more identifiers separated by a period (.).

5. Choose either *Structure* or *Table* for the *Type*.
6. Select *OK*.

The data type editor opens in the workspace area.

7. Select + *(Add property)* in the *Properties* box.

Data types can have more than one property.

Property parameters appear to the right of the *Properties* box.

8. If you select *Structure*, complete the properties by performing the following substeps:
  - a. Enter a name for the new property in the *Name* text box.
  - b. Choose a global scalar data type from the *Scalar Type ID* list.

Scalars compose the structure.

9. If you select *Table*, complete the properties by performing the following substeps:
  - a. Enter a name for the new property in the *Name* text box.
  - b. Choose a global scalar data type from the *Scalar Type ID* list.
  - c. Switch the *Key Property* toggle to on to make this data type the primary key column.

10. Select  *(Save)*.

You can save an instance of the data type after you save it by selecting *Save As* from the *Save* list.

The data types are stored in a tree structure in the *Data Types* tab.



11. **Optional:** To edit structure or table data type, double-click the data type in the tree view of the navigation pane.
12. **Optional:** To delete a data type, right-click the data type in the tree view of the navigation pane and choose [Delete](#).

## 20.2 Creating Local Data Types

Local data types are bound to the graph in which you created them, and are visible only in the graph.

### Prerequisites


Before you create data types, ensure that you understand the concepts of data type category, scope, and compatibility. For complete information about data types, see [Using Data Types \[page 302\]](#) and [Data Types in Operator Ports \[page 32\]](#).

### Context


You can create local data types in addition to global and the data types provided by SAP Data Intelligence. The system also creates dynamic data types during runtime that are applicable only for the Generation 2 Python operators.

To create a local data type, perform the following steps in the Modeler:

### Procedure

1. Open the applicable pipeline.
2. Select  ([Show configuration](#)).

Make sure that none of the operators in the graph are selected when you select Show configuration.

3. Expand the Data Types section in the [Configuration](#) panel.
4. Select  ([Create](#)).

The [Create Data Type](#) dialog box opens.

5. Enter a name for the data type in the *ID* text box.

The name must start with an alphabet and can consist of a string with alphabets, digits, or underscores.

6. Choose either [Structure](#), [Table](#), or [Scalar](#) for the *Type*.

You can choose [Scalar](#) only for local data types.



7. If you chose *Structure* or *Table*, perform the following substeps:
  - a. Enter a description for the new data type.
  - b. Add one or more properties to the new data type.
  - c. Select *Save*.
8. If you chose *Scalar*, perform the following substeps:
  - a. Enter a description for the new data type.
  - b. Select a template from the Template list.
  - c. Select *Save*.

# Important Disclaimers and Legal Information

## Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.

About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering an SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

## Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

## Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.

The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

## Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

## Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

© 2023 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see <https://www.sap.com/about/legal/trademark.html> for additional trademark information and notices.