JS JavaScript

Description

This node creates a new dataset with the specified structure and parameters of fields and populates it with data through running JavaScript code.

To populate the dataset, your code can use data from the node's input ports.

To configure the fields and parameters for the new dataset, you can use one of the following options:

- Use the node setup wizard.
- Specify them directly in the JS code.

Node input

On the left side of a JS node, you'll find the following inputs:

- **Input data source**: An optional port that accepts a dataset (table).
- **Add another port**: Click to add another input port for a data set. New ports are automatically named **Input dataset [N]**, where N is an index number starting from 2.
- **Input variables**: An optional port for passing variables into the node.

Output

On the right side of the node, there are the following outputs:

- **Output dataset**: The resulting data table.
- **Add another port**: Click to add another output port. New ports are automatically named **Output dataset [N]**, where N is an index number starting from 2.

Configuration wizard

The configuration wizard offers you to go through the following pages to set up various node properties:

- **Configure input columns**: This page appears for each data table port which didn't receive any input data. It allows you to configure the input table manually using the toolbar.

- **Configure JavaScript output table columns**
- **JavaScript**

Configure JavaScript output table columns

This wizard page allows you to create columns to the node's output ports. If the node has multiple output ports, you can add columns for each of them on the corresponding tabs: OutputTable[0], OutputTable[1], and so on. If the node has only one output port, no tabs appear; you set this port's columns directly on the page.

Apart for the wizard page, you can set the output table columns dynamically through executing JS code. To do so, mark the **Allow creating output columns in script** checkbox: This will allow you to create, edit, and remove output table fields through coding.

> **Note:** Dynamic creation, editing, and removing of output fields in code is possible only until the first record is added to the output set, i.e., until the first call to the `Append` method of the `OutputTable` and `OutputTables[N]` objects.

JavaScript

This wizard page provides a code editor. The Preview button at the bottom allows you to see a part of the resulting dataset in a separate window and open the debug console if necessary.

> **Note:** Clicking **Preview** activates all input ports of the node. The **Preview** window opens after their succesful activation.

In **Preview**, the count of displayed data is limited:

- **If the node has one output port:** Up to the first 100 rows of the output table are displayed.
- **If the node has multiple output ports:** You can preview data from each port on the corresponding tabs: **OutputTable[0]**, **OutputTable[1]**, and so on. Each tab shows no less than 100 rows if they were added through code.

> **Note:** The number of rows displayed in **Preview** depends on the order of appending output tables in code. Their result rows are output to corresponding OutputTable[N] tabs until their count in each tab reaches 100 (or the end of the table is reached). For example: The code first appends the first output port's dataset of 200 rows, and then, the second port's dataset of 150 rows. In this case, the **Preview** will show 200 rows on the **OutputTable[0]** tab and 100 rows on the **OutputTable[1]** tab.

The code editor supports import of external JS modules.

> **Important:** JavaScript for Linux doesn't support `Atomics.xx`. For implementing wait preiods, use `setTimeout`.

To read about the hotkeys supported in the editor, see Hotkeys.

Accessing port data and other built-in objects from JS script

To access the ports' data and other built-in objects through the JavaScript code, you can employ the following objects:

- Input datasets: (`InputTables` and `InputTable`)
- Input variables: (`InputVariables`)
- Output datasets: (`OutputTables` and `OutputTable`)
- Required enumerations: (`DataType` , `DataKind` , and `UsageType`)

These objects are imported from the built-in module `"builtIn/Data"`. A string for importing them is added to the code by default.

You can also implement **Calculator** functions in your code. Import these from the `"builtIn/Calc"` module.

Fetch API

In the context of JS code execution, you can employ Fetch API interface objects for working with HTTP requests and responses. Import them from the built-in `"builtIn/Fetch"` module before use.

File Storage API

File Storage API interface objects used for working with the file system are also available. Import them from the `"builtIn/FS"` module.

Error processing

When you open the **Preview** window or run the node, you may get a message about syntax or execution errors. The message will indicate the position of the errors in code.

> **Important:** Uncaught errors in Promise are recorded to the server log. In such cases, the node is executed without error messages.

Articles in Section:

- External Modules
- Input Sets
- Input Variables
- Output Sets
- Global Functions
- Enumerations
- Calculator Functions
- Fetch API
- File Storage API
- Debug Console
- Full API Description
- Hotkeys