

Tables

Tables in DGLux5 have named columns and a list of rows. The first column is always row; it holds the row index (0, 1, 2, ...) and is renumbered automatically every time the table changes.

Script can work with tables in two ways:

- call the [table functions](#) from any script, to read and write whole tables;
- write a short script in a table block — [Filter](#), [Column Mapping](#), [Edit Rows](#), or [Remove Rows](#) — that runs once for each row.

Table functions

The table argument of these functions is usually a table value from the page model, for example `@parent.table` (the table property of a Table block) or `@parent.filter.output`.

Writing a table into the Script block

`tableSet` can also create a table and store it in a property of the Script block itself. Pass the property name as a string, or as `@.name`. You can then read it back with `@.name` and bind other blocks to it.

```
tableSet("t", [[1, "a"], [2, "b"]], ["id", "name"]);  
return tableGetRows(@.t);
```

Output: `[[0, 1, a], [1, 2, b]]`

`tableAddRows`, `tableRemoveRows`, and `tableClear` can also change such a property; pass it as `@.name` written directly in the call (not through a variable).

The examples below use this technique so that they run in a Script block on their own. In a real page you would normally pass a Table block's table property instead of `@.t`.

tableGetRows

`tableGetRows(table, [mode], [excludeRow])`

Returns the rows of `table`.

- With no mode: a copy of the rows, as a list of lists. Each row starts with the row value. You can

change the copy freely.

- `mode = false`: the table's own row lists, without copying. This is faster, but do not change them.
- `mode = "object"`: a list of objects that map column names to values. If `excludeRow` is `true`, the row column is left out.

Returns `null` if table is not a table.

```
tableSet("t", [[1, "a"], [2, "b"]], ["id", "name"]);  
return tableGetRows(@.t, "object", true);
```

Output: [{id: 1, name: a}, {id: 2, name: b}]

tableGetColumns

`tableGetColumns(table)`

Returns a list with one object per column. Each object has `name`, `type` (such as `string`, `number`, `bool`, `time`), `width`, `formula`, and `meta`. Returns `null` if table is not a table.

```
tableSet("t", [[1, "a"]], ["id", "name"]);  
cols = tableGetColumns(@.t);  
names = [];  
for (i = 0; i < cols.length; i++) {  
  names.push(cols[i].name + ":" + cols[i].type);  
}  
return names;
```

Output: [row:number, id:string, name:string]

tableSet

`tableSet(table, rows, [columns], [serialize])`

Replaces all data in table.

- `rows` — a list of rows. A row is either a list of values in column order, or an object that maps column names to values. Do **not** include a value for the row column; it is added for you.
- `columns` (optional) — a list of column names, or column objects such as `{name: "v", type: "number"}`. If you leave it out, the table keeps its current columns. In that case the current columns include row, so each list row must start with a placeholder value for it.
- `serialize` (optional) — `true` saves the new data with the page (for a table that is stored in the page). The default is `false`.

Throws an error if table is not a table or rows is not a list.

```
tableSet("t", [{id: 1, name: "a"}, {name: "b", id: 2}], [{name: "id", type: "number"}, "name"]);
return tableGetRows(@.t);
```

Output: [[0, 1, a], [1, 2, b]]

To fill a Table block on the page:

```
tableSet(@parent.table, [{"a", "b"}, {"c", "d"}], ["col1", "col2"]);
```

tableAddRows

`tableAddRows(table, rows)`

Adds rows at the end of table. Each row can be a list in the table's column order, **including** the row column (so start each row with a placeholder value, which is replaced by the row index), or a map from column names to values.

Throws an error if table is not a table or rows is not a list.

```
tableSet("t", [[1, "a"]], ["id", "name"]);
tableAddRows(@.t, [{"", 2, "b"}, [{"", 3, "c"}]);
return tableGetRows(@.t);
```

Output: [[0, 1, a], [1, 2, b], [2, 3, c]]

Rows as maps:

```
tableSet("t", [[1, "a"]], ["id", "name"]);
tableAddRows(@.t, [{id: 2, name: "b"}]);
return tableGetRows(@.t);
```

Output: [[0, 1, a], [1, 2, b]]

tableRemoveRows

`tableRemoveRows(table, start, [count])`

Removes count rows (default 1) starting at row index start. Does nothing if start is outside the table. If start + count is past the end, the rows up to the end are removed.

```
tableSet("t", [[1], [2], [3], [4]], ["v"]);
tableRemoveRows(@.t, 1, 2);
```

```
return tableGetRows(@.t);
```

Output: [[0, 1], [1, 4]]

tableClear

```
tableClear(table)
```

Removes all rows. The columns stay.

```
tableSet("t", [[1], [2]], ["v"]);  
tableClear(@.t);  
return [tableGetRows(@.t).length, tableGetColumns(@.t).length];
```

Output: [0, 2]

Scripts in table blocks

The Filter, Column Mapping, Edit Rows, and Remove Rows blocks run a script once for each row of the input table. In that script:

- each column is a variable with the column's name, for example v1 or row;
- `$thisRow["column name"]` reads any column, including names with spaces or other characters that are not valid variable names;
- `$` is an empty object at the start of each run of the block. It is shared by all rows, so you can use it to keep a running total or a counter;
- `@` is the block itself, so `@parent` is the block's parent;
- `print(...)` writes to the block's **print** property.

The value of the last statement (or of a return statement) is the result for that row.

If the script contains an error, the block's **print** property shows it.

Filter

In the [Filter](#) block, the **condition** script decides which rows to keep. A row is kept when the result is true.

```
v1 > 10 && $thisRow["Energy Usage"].indexOf("History") > -1
```

Keep only the first 5 rows that match:

```

if ($.count == null) $.count = 0;
if (v1 > 10 && $.count < 5) {
  $.count++;
  return true;
}
return false;

```

Column Mapping

In the [Column Mapping](#) block, each output column has a **name** and a **from** value.

- If **from** is a column name, the column is copied.
- If **from** starts with =, the rest is a script expression, and its value becomes the cell value.
- + instead of = does the same and marks the column as a number column.
- & instead of = does the same and marks the column as a bool column.

Columns are calculated in order. A later column can use an earlier result with `$rowResult["name"]`.

name	from
total	+v1 * v2
label	="Total: " + numberFormat(\$rowResult["total"], "#,##0.00")
big	&\$rowResult["total"] > 100
status	=\$thisRow["Energy Usage"].indexOf("History") > -1

Edit Rows

The [Edit Rows](#) block changes cells of an editable table, such as the **table** property of a Table block. When the block is invoked, the **condition** script runs for each row. In each row where the result is true, each **column *n*** is set to **value *n***. The values are fixed values, not scripts.

```
status == "open" && priority > 3
```

Remove Rows

The [Remove Rows](#) block removes rows from an editable table. When the block is invoked, the **condition** script runs for each row, and rows where the result is true are removed.

```
$thisRow["Device Name"] == "" || value == null
```

From:
<http://wiki.dglogik.com/> - **DGLogik**

Permanent link:
http://wiki.dglogik.com/dglux5_wiki:dgscript:tables

Last update: **2026/10/22 16:32**

