

JS Widget Components

A **JS Widget** is a component whose content is drawn by your own JavaScript code. Use it to turn a third-party JavaScript library, such as [D3](#) or [three.js](#), into a DGLux5 component. The widget's properties appear in the Property Inspector like the parameters of a [symbol](#), so you can set them and bind them to data like any other component property.

The **dglux5_js_lib** project on GitHub contains a ready-to-use widget template, the three.js and D3 libraries, and examples:

github.com/dglux/dglux5_js_lib

A JS Widget runs your code directly in the DGLux5 page. To show an external web page or code that must stay isolated from DGLux5, use an [IFrame](#) instead (see [Custom JS Components](#)). For calculations without a visual component, use the [JavaScript](#) dataflow block or a [Script](#) block.

Installing the Library

1. Download or clone [dglux5_js_lib](#).
2. On the DGLux5 server, create a project named `js` in the `files/proj` directory, and copy all files from the repository into it.
3. Open the `js` project in DGLux5. Open any `.dg5` file in the `examples` folder to see an example.

The repository contains:

Folder	Contents
palette	<code>js.dg5</code> , a blank JS Widget that appears in the widget palette when the project is used as a library
examples/parameters	Input and output parameters of different types
examples/table_input	Draws an HTML table from a table parameter
examples/table_output	Creates a table and writes it to an output parameter
examples/size_auto	A widget that sizes itself to its content
examples/size_sensor	A widget that is notified when its size changes
examples/threejs	Loads three.js and shows a rotating 3D box
three, d3	three.js and D3, prepared for the DGLux5 module loader

Using JS Widgets in Your Project

1. In your project, right-click the **lib** folder in the [Project panel](#) and import the `js` project as a library.
2. A blank JS Widget template now appears in the [widget palette](#). Drag it to the stage.
3. In the Property Inspector, set the **Symbol** field to the JavaScript file that defines your widget, for example `lib/js/examples/parameters/parameters.js`. The field also accepts an `http://`

or `https://` URL.

4. The widget's parameters appear in the Property Inspector. Set or bind them as usual.

The template has a light gray background so that you can see it on the stage. You can remove it in the Fill and Stroke section.

Writing a Widget

A widget is a JavaScript file that calls `dgdefine` and returns an object with a `dgNewWidget` function. `dgNewWidget` receives the `div` that the widget draws into and returns a new widget object. The widget class extends `dgluxjs.Widget`.

This minimal widget has one input property, `text`, and one output property, `length`:

```
dgdefine([], function () {

    var MyWidget = (function (_super) {
        __extends(MyWidget, _super);

        function MyWidget(div, model) {
            _super.call(this, div, model);
            this.contentDiv = document.createElement('div');
            div.appendChild(this.contentDiv);
        }

        // The properties shown in the Property Inspector
        MyWidget.prototype.getDefinition = function () {
            return {
                "name": "",
                "variables": [
                    { "t": "string", "n": "text" },
                    { "t": "number", "n": "length" }
                ],
                "layout": {
                    "type": "vbox",
                    "children": ["text", "length"]
                }
            };
        };

        // Called when a property value changes
        MyWidget.myPropMap = {
            "text": function (widget, value) {
                widget.contentDiv.textContent = value;
                widget.updateModelValue("length", value ? value.length : 0);
            }
        };
    })(_super);
});
```

```

    };
    MyWidget.prototype.getPropMap = function () {
        return MyWidget.myPropMap;
    };

    // Called when the widget is removed
    MyWidget.prototype.destroy = function () {
        this.contentDiv.remove();
    };

    return MyWidget;
})(dgluxjs.Widget));

function dgNewWidget(div, model) {
    return new MyWidget(div, model);
}
return { "dgNewWidget": dgNewWidget };
});

```

Property Definition

`getDefinition()` returns an object with these fields:

Field	Description
name	Name of the widget. Can be empty.
variables	List of properties. Each property has n (name) and t (type), such as "string", "number", "bool", "enum", "path", "fill", "trigger" or "tabledata". These are the same types as symbol parameters . Optional fields: label, nullValue (default value), minimum, maximum, precision, enums and enumLabels (for enum), and helpTitle with helpContent (help text in the Property Inspector).
layout	How the properties are arranged in the Property Inspector: {"type": "vbox", "children": [...]}. type is "vbox", "hbox" or "section". children contains property names or nested layouts. A layout can have a label.
size	Optional. "auto": the widget sizes itself to its content. "sensor": DGLux5 calls <code>onResize()</code> whenever the widget's size changes.

dgluxjs.Widget Reference

Methods that you can override:

Method	Description
<code>getDefinition()</code>	Returns the property definition .
<code>getPropMap()</code>	Returns a map from property names to functions <code>function(widget, value)</code> . Each function is called when that property changes, and once for every property when the widget is created.

Method	Description
<code>commitProperties(changes)</code>	Called with the list of changed property names. The default implementation calls the functions from <code>getPropMap()</code> .
<code>onResize()</code>	Called when the size changes, if size is "sensor". Read the size from <code>this.parentDiv</code> .
<code>destroy()</code>	Called when the widget is removed, or when its JavaScript file changes. Remove your elements and stop timers here.

Methods and fields that you can use:

Name	Description
<code>this.parentDiv</code>	The div element the widget draws into.
<code>this.destroyed</code>	<code>true</code> after the widget is destroyed. Check it in animation loops and timers.
<code>getModelValue(name)</code>	Returns the current value of a property.
<code>updateModelValue(name, value)</code>	Sets a property value, for example an output property. The value is not saved in the page file.
<code>setModelValue(name, value)</code>	Sets a property value and saves it in the page file.
<code>updateModelTable(name, columns, rows)</code>	Sets a table property. <code>columns</code> is a list of column names; <code>rows</code> is a list of row lists.
<code>getResourceUrl(path)</code>	Converts a project file path, such as <code>lib/js/examples/threejs/crate.gif</code> , to a URL that the browser can load.
<code>dgluxjs.getTableRows(table)</code>	Returns the rows of a table value as a list of lists. The first value of each row is the row index column.
<code>dgluxjs.getTableColumns(table)</code>	Returns the column names of a table value. The first column is row.

Using a Third-Party Library

DGLux5 loads widget files with its own AMD-style module loader, `dgdefine`. List the files your widget needs as the first argument of `dgdefine`. The modules they return are passed to your function in the same order:

```
dgdefine(["lib/js/three/three.r82.min.js"], function (THREE) {
    // THREE is available here
});
```

- Paths that start with `lib/` refer to the project's **lib** folder. Other relative paths are relative to the widget file, and `.js` is added if it is missing.
- A library file that is not written for `dgdefine` must tell DGLux5 what it exports. Add a `dgdefine` call to the end of the library file. For example, the copy of `three.js` in the repository ends with:

```
dgdefine([], function () {
    return THREE;
});
```

See [examples/threejs](#) in the repository for a complete example.

From:

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

Permanent link:

http://wiki.dglogik.com/dglux5_wiki:dgscrip:js_widget

Last update: **2026/10/22 17:45**

