JavaScript (KLayJS)

KLayJS is deprecated and has been replaced by elkjs. The documentation below may be **outdated**.

The KLay JS project provides our Java-based layout algorithms to the JavaScript community. We leverage the Java to JavaScript compiler of the Google Web Toolkit (GWT) to convert our Java code into a JavaScript library. This allows you to use the full power of our layout algorithms in pure JavaScript.

GitHub

The library is now available via OpenKieler on GitHub. Please refer to those sides for further information. The documentation below might be incomplete and outdated.

Downloads

Error rendering macro 'excerpt-include'

No link could be created for 'Downloads - KLayJS'.

- **Standard Linker**
  I.e. GWT's IFrameLinker that "*loads the GWT module in a separate iframe*".
- **Custom Linker**
  The linker extends GWT's DirectInstallLinker and enables the library to be used with, for instance, Chrome Packaged Apps. According to the javadoc the linker "*adds a script tag to the iframe rather than downloading the code as a string and then installing it into the iframe*". However, when using this linker a lot of GWT's variables will be added to the global namespace.
- **Web Worker Linker**
  The linker allows to use our library with a Web Worker. It removes GWT generations that are not required for our use case, e.g. loading of browser specific permutations. A bower component is available on GitHub (Thanks to automata), a specific example further down on this page.

API

This documentation targets the *Default Linker* and *Custom Linker*. The *Web Worker Linker* has a slightly different API, please refer to the GitHub page for more information.

```
$klay.layout({ graph, options, success, error });
```

- **graph** - the graph to be layouted in our JSON Format.
- **options** - a JSON object containing layout options that should be used for every hierarchy level of the graph. The same effect can be achieved by specifying the *properties* for every compound node, however, using the *options* object offers a more convenient way. Further information on available layout options can be found here.
- **success(*layouted*)** - a function to be called upon success, the layouted graph is passed as argument.
- **error(*obj*)** - a function to be called if an error occurs, an object is passed as argument which contains a *text* field with further information about the error.

Field	Description
type	The type of the error, e.g. an invalid graph format.

text	Further description of the problem that occurred.
stacktrace	The strack trace of the Java exception, if any.
Additional Information Invalid Graph Format	
value	The JSON object that caused the problem.
context	The context of the `value`, e.g. if the value is an edge, the context is the containing node.

Dedicated KlayJS Options

We offer some options that have influence of the behavior of the JavaScript interface. These options are listed below.

- **intCoordinates** - when set to `true` all calculated layout coordinates (by default doubles) are cast to an integer value.

Example

Below is small example that executes layout on a small graph. Upon success the returned JSON is printed to the console and added to the body of the document.

This example shows the usage of the Default and Custom Linker. For a Web Worker example see next example.

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <title>KIELER Klay JS Layout Test</title>
    <script type="text/javascript" src="klay.nocache.js"></script>
  </head>
  <body>
    <h1>KIELER Klay JS Layout Test</h1>
  </body>

  <script>
      // the 'klayinit' method is called as soon as GWT finished inititalizing
    function klayinit() {

        // assemble a graph
        var graph = {
            "id": "root",
            "properties": {
                "direction": "DOWN",
                "spacing": 40
            },
            "children": [{
                "id": "n1",
                "width": 40,
                "height": 40
            }, {
                "id": "n2",
                "width": 40,
                "height": 40
            }, {
                "id": "n3",
                "width": 40,
                "height": 40
            }],
            "edges": [{
                "id": "e1",
                "source": "n1",
                "target": "n2"
            },
            {
                "id": "e2",
                "source": "n1",
                "target": "n3"
```

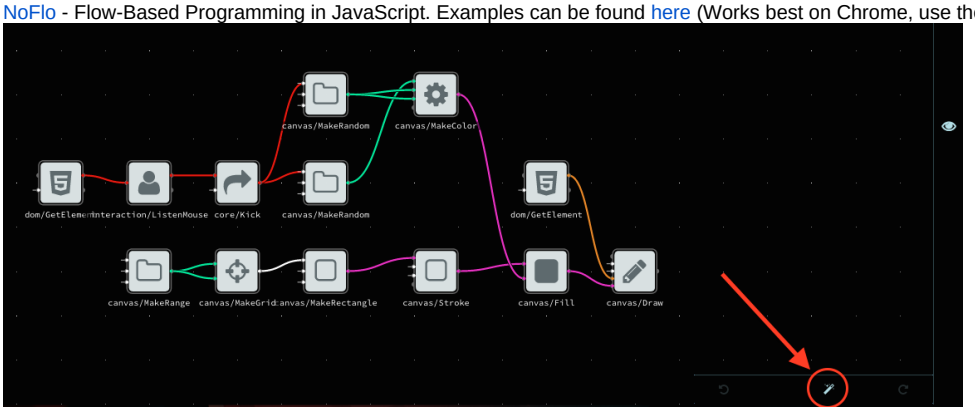
```
            },
            {
                "id": "e3",
                "source": "n2",
                "target": "n3"
            }
            ]
        };

        // execute the layout
        $klay.layout({
            graph: graph,
            options: {
                spacing: 50
            },
            success: function(layouted) {
                console.log(layouted);
                document.body.innerHTML = "<pre>" + JSON.stringify(layouted, null, "  ") + "</pre>";
            },
            error: function(error) {
                console.log(error);
                document.body.innerHTML = "<pre>" + JSON.stringify(error, null, "  ") + "</pre>";
            }
        });
      }
    </script>

</html>
```

Example (Web Worker)

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <title>KIELER Klay JS Layout Test</title>
  </head>
  <body>
    <h1>KIELER Klay JS Layout Test</h1>
  </body>
  <script>
    (function () {
      // assemble a graph
      var graph = {
        "id": "root",
        "properties": {
          "direction": "DOWN",
          "spacing": 40
        },
        "children": [{
          "id": "n1",
          "width": 40,
          "height": 40
        }, {
          "id": "n2",
          "width": 40,
          "height": 40
        }, {
          "id": "n3",
          "width": 40,
          "height": 40
        }],
        "edges": [{
          "id": "e1",
          "source": "n1",
          "target": "n2"
        }, {
          "id": "e2",
          "source": "n1",
          "target": "n3"
        }, {
          "id": "e3",
          "source": "n2",
          "target": "n3"
        }]
      };

      // Creates a KlayJS Web Worker
      var worker = new Worker('klayjs_worker.js');

      // Receives the layouted graph from the Web Worker
      worker.addEventListener('message', function (e) {
        console.log('Layouted graph:', e.data);
      }, false);

      // Sends the original graph to the Web Worker
      worker.postMessage({
        "graph": graph,
        "options": {
          "spacing": 50
        }
      });
    })();
  </script>
</html>
```

See it in Action

- Proofscape – Visualizing mathematical proofs with graphs.
- NoFlo - Flow-Based Programming in JavaScript. Examples can be found here (Works best on Chrome, use the Magic Stick).



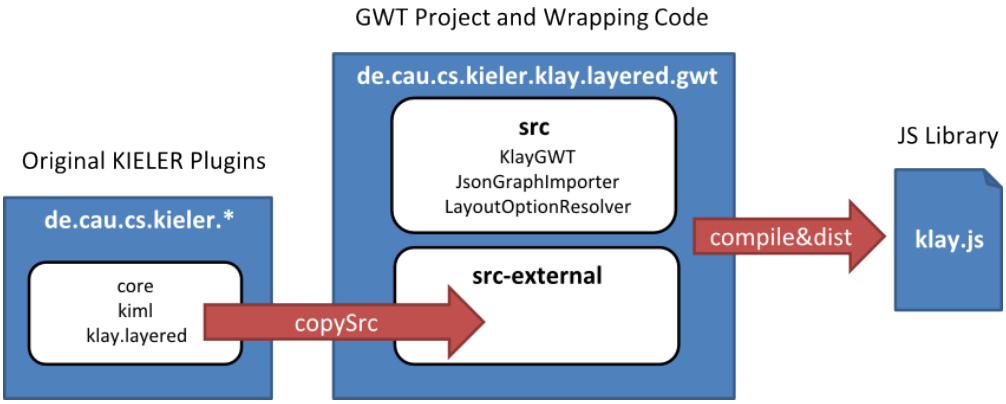
Development

Git branch **gwt-export**

The GWT plugin that is used for the building proccess resides in the build folder and is called `de.cau.cs.kieler.klay.layered.gwt`. If you want to develop on the classes of the project you want to use `ant copySrc` to copy all required klay layered classes from the main repository.

Build Anatomie

The following figure illustrates the build process. First the ant script copies the original KIELER source code files into a dedicated GWT project (`de.cau.cs.kieler.klay.layered.gwt`). This project contains further classes that define the JavaScript interface and the conversion from the JSON Graph format into our internal graph representation. Second, the GWT compiler is used to generate JavaScript code from the Java sources. Finally, we remove superfluous GWT elements and pack a zip archive containing the generated JavaScript library.



(Thanks to automata)