



web: pr-dc.com, email: info@pr-dc.com, github: github.com/PR-DC



JSLAB v1.0.2 SOURCE CODE

github.com/PR-DC/JSLAB

Contents

1	root	2
2	config	6
3	cpp	32
4	css	63
5	html	127
6	js	150
6.1	code	174
6.2	dev	187
6.3	editor	207
6.4	main	227
6.5	sandbox	280
6.6	windows	730

1 root

```

1  {
2    "name": "JSLAB",
3    "version": "1.0.2",
4    "description": "JavaScript LABoratory environment",
5    "main": "js/init.js",
6    "repository": "https://github.com/PR-DC/JSLAB",
7    "license": "GPL-3.0-or-later",
8    "author": "Milos Petrasinovic <mpetrasinovic@prdc.rs>",
9    "homepage": "https://pr-dc.com",
10   "scripts": {
11     "preinstall": "npm install rimraf & npm install node-7z & npm install 7zip
12       -bin & node js/dev/prepare-libs.js & npm install -g node-gyp",
13     "postinstall": "node js/dev/build-configure.js & node-gyp rebuild & node
14       js/dev/make-doc.js & electron js/init.js",
15     "start": "electron js/init.js",
16     "debug": "electron js/init.js --debug-app",
17     "test": "electron js/init.js --test-app",
18     "build": "node js/dev/build-configure.js & node-gyp rebuild & electron js/
19       init.js",
20     "pack": "node js/dev/build-configure.js --action pack & node-gyp rebuild &
21       electron-builder --dir",
22     "dist": "node js/dev/build-configure.js --action dist & node-gyp rebuild &
23       electron-builder --win",
24     "dist-portable": "node js/dev/build-configure.js --action dist-portable &
25       node-gyp rebuild & electron-builder --win portable",
26     "dist-signed": "node js/dev/build-configure.js --action dist --sign-build
27       & node-gyp rebuild & electron-builder --win",
28     "clear-app-data": "node js/dev/clear-app-data.js --confirm",
29     "make-doc": "node js/dev/make-doc.js",
30     "make-source-code-book": "node js/dev/make-source-code-book.js",
31     "update-libs": "node js/dev/download-libs.js --force --confirm",
32     "upload-source": "node js/dev/upload-source-code.js"
33   },
34   "build": {
35     "asar": false,
36     "appId": "com.pr-dc.jslab",
37     "productName": "PR-DC JSLAB",
38     "copyright": "Copyright 2024 @ PR-DC",
39     "nodeGypRebuild": true,
40     "directories": {
41       "buildResources": "build",
42       "output": "dist"
43     },
44     "files": [
45       "**",
46       "!cpp",
47       "!dev",
48       "!dist",
49       "!build",
50       "!bin",
51       "!binding.gyp",
52       "!js/dev",
53       "!lib/boost-1.86.0",
54     ]
55   }
56 }
```



```
47      " ! lib/cgal-6.0.1",
48      " ! lib/eigen-3.4.0"
49  ],
50  "fileAssociations": [
51    {
52      "ext": "jsl",
53      "name": "JSL",
54      "description": "JavaScript LABoratory script",
55      "role": "Editor",
56      "icon": "icons/icon.ico"
57    }
58  ],
59  "extraResources": [
60    {
61      "from": "./build/Release/",
62      "to": "./app/build/Release/",
63      "filter": [
64        "*.*node",
65        "*.*dll"
66      ]
67    },
68    {
69      "from": "./node_modules/npm/",
70      "to": "./app/node_modules/npm/"
71    },
72    {
73      "from": "./node_modules/node-gyp/",
74      "to": "./app/node_modules/node-gyp/"
75    }
76  ],
77  "compression": "maximum",
78  "mac": {
79    "category": "public.app-category.utilities",
80    "icon": "icons/icon.icns",
81    "target": "dmg"
82  },
83  "win": {
84    "icon": "icons/icon.ico",
85    "target": "nsis",
86    "signingHashAlgorithms": [
87      "sha256"
88    ],
89    "publisherName": "PR-DC"
90  },
91  "linux": {
92    "icon": "icons/",
93    "target": "AppImage"
94  },
95  "portable": {
96    "splashImage": "icons/splash.bmp"
97  },
98  "nsis": {
99    "oneClick": false,
100   "perMachine": true,
101   "allowToChangeInstallationDirectory": true,
```

```
102     "installerIcon": "icons/nsis_in_ico.ico",
103     "uninstallerIcon": "icons/nsis_un_ico.ico",
104     "installerHeader": "img/nsis_in_header.bmp",
105     "installerHeaderIcon": "icons/icon.ico",
106     "installerSidebar": "img/nsis_in_welcom.bmp",
107     "uninstallerSidebar": "img/nsis_un_welcom.bmp",
108     "uninstallDisplayName": "JSLAB ${version}",
109     "shortcutName": "JSLAB",
110     "language": "1033",
111     "displayLanguageSelector": true,
112     "installerLanguages": [
113       "en_US",
114       "sr_RS"
115     ],
116     "multiLanguageInstaller": true,
117     "warningsAsErrors": false
118   },
119   "artifactName": "JSLAB_1.0.2.${ext}"
120 },
121 "devDependencies": {
122   "electron": "29.1.2",
123   "electron-builder": "24.13.3"
124 },
125 "dependencies": {
126   "@babel/parser": "7.26.2",
127   "@babel/plugin-syntax-top-level-await": "7.14.5",
128   "7zip-bin": "^5.2.0",
129   "big-json-viewer": "0.1.7",
130   "bytenode": "1.5.6",
131   "dir-compare": "4.2.0",
132   "electron-context-menu": "3.6.1",
133   "electron-store": "8.2.0",
134   "eslint": "9.14.0",
135   "fast-xml-parser": "4.4.1",
136   "fmin": "0.0.4",
137   "glob": "10.3.10",
138   "jsdoc-api": "9.3.4",
139   "ml-regression-polynomial": "3.0.1",
140   "node-7z": "^3.0.0",
141   "node-addon-api": "8.0.0",
142   "node-gyp": "10.2.0",
143   "node-mavlink": "2.0.7",
144   "npm": "10.9.0",
145   "path-equal": "1.2.5",
146   "pdfkit": "0.14.0",
147   "recast": "0.23.9",
148   "rimraf": "^5.0.10",
149   "seedrandom": "3.0.5",
150   "serialport": "12.0.0",
151   "source-map": "0.7.4",
152   "svg-to-pdfkit": "0.1.8",
153   "tcp-port-used": "1.0.2",
154   "usb": "2.9.0",
155   "zeromq": "6.0.0-beta.20"
156 }
```

157 }

Listing 1 - package.json

```

1  {
2    "targets": [
3      {
4        "target_name": "native_module",
5        "sources": [
6          "cpp/native-module.cpp"
7        ],
8        "include_dirs": [
9          "<!(node -p \"require('node-addon-api').include\")",
10         "<(module_root_dir)/lib/eigen-3.4.0/"
11       ],
12      "cflags!": [
13        "-fno-exceptions"
14      ],
15      "cflags_cc!": [
16        "-fno-exceptions"
17      ],
18      "defines": [
19        "NAPI_DISABLE_CPP_EXCEPTIONS"
20      ],
21      "msvs_settings": {
22        "VCCLCompilerTool": {
23          "AdditionalOptions": [
24            "-std:c++17"
25          ]
26        }
27      }
28    },
29    {
30      "target_name": "alpha_shape_3d",
31      "sources": [
32        "cpp/alpha-shape-3d.cpp"
33      ],
34      "include_dirs": [
35        "<!(node -p \"require('node-addon-api').include\")",
36        "<(module_root_dir)/lib/cgal-6.0.1/include/",
37        "<(module_root_dir)/lib/cgal-6.0.1/auxiliary/gmp/include",
38        "<(module_root_dir)/lib/boost-1.86.0/"
39      ],
40      "libraries": [
41        "<(module_root_dir)/lib/cgal-6.0.1/auxiliary/gmp/lib/gmp.lib",
42        "<(module_root_dir)/lib/cgal-6.0.1/auxiliary/gmp/lib/mpfr.lib"
43      ],
44      "cflags!": [
45        "-fno-exceptions"
46      ],
47      "cflags_cc!": [
48        "-fno-exceptions",
49        "-O3",
50        "-DNDEBUG"
51      ],
52      "defines": [

```

```

53         "NAPI_DISABLE_CPP_EXCEPTIONS"
54     ],
55     "copies": [
56     {
57       "destination": "<(module_root_dir)/build/Release",
58       "files": [
59         "<(module_root_dir)/lib/cgal-6.0.1/auxiliary/gmp/bin/gmp-10.dll",
60         "<(module_root_dir)/lib/cgal-6.0.1/auxiliary/gmp/bin/mpfr-6.dll"
61       ]
62     }
63   ],
64   "msvs_settings": {
65     "VCCLCompilerTool": {
66       "AdditionalOptions": [
67         "-std:c++17",
68         "/GR",
69         "/EHsc"
70       ]
71     }
72   }
73 }
74 ]
75 }
```

Listing 2 - binding.gyp

2 config

```

1 /**
2  * @file JSLAB global configuration
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 /**
9  * Class for application configuration.
10 */
11 class PRDC_APP_CONFIG {
12
13 /**
14  * Create JSLAB configuration object.
15 */
16 constructor() {
17   this.PRODUCTION = false;
18   this.DEBUG = false;
19   this.TEST = false;
20   this.SIGN_BUILD = false;
21
22   this.GROUP_RAF = true;
23   this.OUTPUT_COMPLETE_JSDOC = false;
24
25   this.DEBUG_FUN_SHADOW = false;
26   this.DEBUG_NEW_FUN = false;
```

```

27   this.DEBUG_RENDER_GONE_ERROR = true;
28   this.DEBUG_SYM PYTHON_EVAL_CODE = false;
29   this.DEBUG_PRE_TRANSFORMED_CODE = false;
30   this.DEBUG_TRANSFORMED_CODE = false;
31   this.DEBUG_PARALLEL_WORKER_SETUP_FUN = false;
32   this.DEBUG_PARALLEL_WORKER_WORK_FUN = false;
33
34   this.LOG_RENDER_GONE_ERROR = true;
35
36   // Log codes
37   this.LOG_CODES = {
38     'other': 0,
39     'render-gone-error': 1,
40   };
41
42   // JSLAB settings
43   this.FORBIDDEN_NAMES = [ 'jsl', 'config', 'language', 'app_path', 'packed' ];
44   this.MATHJS_PREVENT_OVERRIDE = [ 'config', 'print', 'Infinity', 'NaN', 'isNaN', 'Node' ];
45   this.SUBMODULES = {
46     'builtin': [
47       {name: 'basic', file: 'basic', class_name: 'PRDC_JSLAB_LIB_BASIC'},
48       {name: 'math', file: 'math', class_name: 'PRDC_JSLAB_LIB_MATH'},
49       {name: 'non_blocking', file: 'non-blocking', class_name: 'PRDC_JSLAB_LIB_NON_BLOCKING'},
50       {name: 'path', file: 'path', class_name: 'PRDC_JSLAB_LIB_PATH'},
51       {name: 'windows', file: 'windows', class_name: 'PRDC_JSLAB_LIB_WINDOWS'},
52       {name: 'figures', file: 'figures', class_name: 'PRDC_JSLAB_LIB FIGURES'},
53       {name: 'time', file: 'time', class_name: 'PRDC_JSLAB_LIB_TIME'},
54       {name: 'array', file: 'array', class_name: 'PRDC_JSLAB_LIB_ARRAY'},
55       {name: 'color', file: 'color', class_name: 'PRDC_JSLAB_LIB_COLOR'},
56       {name: 'conversion', file: 'conversion', class_name: 'PRDC_JSLAB_LIB_CONVERSION'},
57       {name: 'device', file: 'device', class_name: 'PRDC_JSLAB_LIB_DEVICE'},
58       {name: 'file_system', file: 'file-system', class_name: 'PRDC_JSLAB_LIB_FILE_SYSTEM'},
59       {name: 'system', file: 'system', class_name: 'PRDC_JSLAB_LIB_SYSTEM'},
60       {name: 'geography', file: 'geography', class_name: 'PRDC_JSLAB_LIB_GEOGRAPHY'},
61       {name: 'networking', file: 'networking', class_name: 'PRDC_JSLAB_LIB_NETWORKING'},
62       {name: 'format', file: 'format', class_name: 'PRDC_JSLAB_LIB_FORMAT'},
63       {name: 'render', file: 'render', class_name: 'PRDC_JSLAB_LIB_RENDER'},
64       {name: 'geometry', file: 'geometry', class_name: 'PRDC_JSLAB_LIB_GEOMETRY'},
65       {name: 'control', file: 'control', class_name: 'PRDC_JSLAB_LIB_CONTROL'},
66       {name: 'optim', file: 'optim', class_name: 'PRDC_JSLAB_LIB_OPTIM'},
67     ],
68     'lib': [
69       {name: 'parallel', file: 'parallel', class_name: 'PRDC_JSLAB_PARALLEL'}
    ],
  }

```

```

70     {name: 'mat', file: 'matrix-math', class_name: 'PRDC_JSLAB_MATRIX_MATH',
71      },
72     {name: 'vec', file: 'vector-math', class_name: 'PRDC_JSLAB_VECTOR_MATH',
73      },
74     {name: 'sym', file: 'sym-math', class_name: 'PRDC_JSLAB_SYMBOLIC_MATH'},
75   ],
76 };
77
78 this.DOC_SUBMODULES_ADDITIONAL = [
79   {name: 'Matrix', file: 'matrix-math', class_name: 'PRDC_JSLAB_MATRIX'},
80   {name: 'Vector', file: 'vector-math', class_name: 'PRDC_JSLAB_VECTOR'},
81   {name: 'Symbolic', file: 'sym-math', class_name: 'PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL'},
82   {name: 'Window', file: 'windows', class_name: 'PRDC_JSLAB_WINDOW'},
83   {name: 'Figure', file: 'figures', class_name: 'PRDC_JSLAB_PICTURE'},
84   {name: 'Plot', file: 'figures', class_name: 'PRDC_JSLAB_PLOT'},
85   {name: 'freecad_link', file: 'freecad-link', class_name: 'PRDC_JSLAB_FREECAD_LINK'},
86   {name: 'om_link', file: 'om-link', class_name: 'PRDC_JSLAB_OPENMODELICA_LINK'},
87   {name: 'tcp_client', file: 'networking-tcp', class_name: 'PRDC_JSLAB_TCP_CLIENT'},
88   {name: 'tcp_server', file: 'networking-tcp', class_name: 'PRDC_JSLAB_TCP_SERVER'},
89   {name: 'udp_client', file: 'networking-udp', class_name: 'PRDC_JSLAB_UDP'},
90   {name: 'udp_server', file: 'networking-udp', class_name: 'PRDC_JSLAB_UDP_SERVER'},
91   {name: 'video_call', file: 'networking-video-call', class_name: 'PRDC_JSLAB_VIDEOCALL'},
92   {name: 'mathjs', file: 'mathjs-doc', class_name: 'PRDC_JSLAB_MATHJS_DOC'},
93   {name: 'rcmiga', file: 'optim-rcmiga', class_name: 'PRDC_JSLAB_OPTIM_RCMIGA'},
94   {name: 'space_search', file: 'geometry-spacesearch', class_name: 'PRDC_JSLAB_GEOMETRY_SPACE_SEARCH'},
95   {name: 'map', file: 'geography-map', class_name: 'PRDC_JSLAB_GEOGRAPHY_MAP'},
96   {name: 'map_3d', file: 'geography-map-3d', class_name: 'PRDC_JSLAB_GEOGRAPHY_MAP_3D'},
97   {name: 'Gamepad', file: 'device-gamepad', class_name: 'PRDC_JSLAB_DEVICE_GAMEPAD'},
98 ];
99
100 this.SOURCE_CODE_BOOK_FILES = [
101   'package.json',
102   'binding.gyp',
103   'config',
104   'cpp',
105   'css',
106   'html',
107   'js'
108 ];

```

```

108     this.LINT_OPTIONS = {
109         overrideConfigFile: true,
110         overrideConfig: {
111             languageOptions: {
112                 ecmaVersion: "latest",
113                 sourceType: "module"
114             },
115             rules: {
116                 "no-unused-vars": "warn",
117                 "semi": ["warn", "always"],
118                 "no-extra-semi": "warn",
119                 "no-unreachable": "warn",
120                 "consistent-return": "warn",
121                 "no-shadow": "warn",
122                 "no-use-before-define": "warn"
123             }
124         }
125     };
126
127     this.COMPRESSSED_LIBS = [
128         'sympy-0.26.2',
129         'cgal-6.0.1',
130         'boost-1.86.0',
131         'codemirror-5.49.2',
132         'eigen-3.4.0',
133         'three.js-r162',
134         'Cesium-1.124',
135     ];
136     this.COMPILE_LIBS = [];
137
138     // Language
139     this.langs = ["en", "rs", "rsc"];
140
141     // Windows
142     this.WIN_SAVE_DEBOUNCE_TIME = 50; // [ms]
143
144     // Other
145     this.PLOTER = ['plotly', 'echarts'][0];
146     this.DOC_LATEX_RERUNS_NUMBER = 3;
147     this.SOURCE_CODE_BOOK_LATEX_RERUNS_NUMBER = 3;
148     this.MAX_ACTIVE_WEBGL_CONTEXTS = '128';
149
150     // Build sign
151     this.COMPANY_NAME = process.env.COMPANY_NAME;
152     this.TIMESTAMP_SERVER = process.env.TIMESTAMP_SERVER;
153     this.SIGN_TOOL_PATH = process.env.SIGN_TOOL_PATH;
154
155     // Upload and download libs from server
156     this.SERVER_SOURCE_PATH = process.env.SERVER_PATH + "JSLAB/";
157     this.SOURCE_UPLOAD_EXCLUDE = ['/bin', '/build', '/dist', '/node_modules',
158         '/package-lock.json', '/binding.gyp', '/lib'];
159     this.SERVER_LIBS_PATH = process.env.SERVER_LIBS_PATH;
160
161     this.USED_LIBS = [
162         'sprintf-1.1.3',

```

```

162      'sympy-0.26.2',
163      'cgal-6.0.1',
164      'boost-1.86.0',
165      'codemirror-5.49.2',
166      'complete.ly.1.0.1',
167      'd3-7.8.5',
168      'draggably-2.3.0',
169      'eigen-3.4.0',
170      'highlight-11.0.1',
171      'jquery-3.7.0',
172      'jshint-2.13.0',
173      'math-11.8.2',
174      'tex-mml-chtml-3.2.0',
175      'luxon-3.4.4',
176      'plotly-2.24.2',
177      'three.js-r162',
178      'inflate-0.3.1',
179      'hammer-2.0.8',
180      'anime-3.2.1',
181      'tween.js-23.1.1',
182      'leaflet-1.9.4',
183      'leaflet.rotatedMarker-0.2.0',
184      'Cesium-1.124',
185      'PRDC_APP_LOGGER',
186      'PRDC_PANEL',
187      'PRDC_TABS',
188      'PRDC_POPUP',
189    ];
190
191  this.UPLOAD_COMPARE_SIZE = false;
192  this.UPLOAD_COMPARE_CONTENT = true;
193  this.UPLOAD_COMPARE_DATE = false;
194  this.UPLOAD_COMPARE_SIZE_ON_DISTINCT = false;
195
196  // Conditional variables
197  if(typeof process_arguments != 'undefined') {
198    var args = process_arguments.map(function(e) { return e.toLowerCase(); });
199    if(args.includes("--debug-app")) {
200      this.DEBUG = true;
201    }
202    if(args.includes("--test-app")) {
203      this.TEST = true;
204    }
205    if(args.includes("--sign-build")) {
206      this.SIGN_BUILD = true;
207    }
208  }
209
210  this.PANEL_RESIZER_WIDTH = 10;
211  this.PANEL_MIN_SIZE = 10;
212  this.PANEL_DEFAULT_COLUMNS = [20, 80];
213  this.PANEL_DEFAULT_LEFT_ROWS = [100/3, 100/3, 100/3];
214  this.PANEL_DEFAULT_WORKSPACE_COLUMNS = [50, 25, 25];
215}

```

```

216 }
217
218 exports.PRDC_APP_CONFIG = PRDC_APP_CONFIG;

```

Listing 3 - config.js

```

1  {
2      "1": {
3          "en": "Editor",
4          "rs": "Uredivač",
5          "rsc": "Уређивач"
6      },
7      "2": {
8          "en": "Help",
9          "rs": "Pomoć",
10         "rsc": "Помоћ"
11     },
12     "3": {
13         "en": "Info",
14         "rs": "Info",
15         "rsc": "Инфо"
16     },
17     "4": {
18         "en": "File Browser",
19         "rs": "Pretraživač fajlova",
20         "rsc": "Претраживач фајлова"
21     },
22     "5": {
23         "en": "Workspace",
24         "rs": "Radni prostor",
25         "rsc": "Радни простор"
26     },
27     "6": {
28         "en": "Command History",
29         "rs": "Istorija Komandi",
30         "rsc": "Историја Команди"
31     },
32     "7": {
33         "en": "Command Window",
34         "rs": "Komandni Prozor",
35         "rsc": "Командни Прозор"
36     },
37     "8": {
38         "en": "version",
39         "rs": "verzija",
40         "rsc": "верзија"
41     },
42     "9": {
43         "en": "Settings",
44         "rs": "Podešavanja",
45         "rsc": "Подешавања"
46     },
47     "10": {
48         "en": "New",
49         "rs": "Nova",
50         "rsc": "Нова"
51     }
52 }

```

```
51 },
52 "11": {
53     "en": "Open",
54     "rs": "Otvořit",
55     "rsc": "Отвори"
56 },
57 "12": {
58     "en": "Save",
59     "rs": "Sačuvaj",
60     "rsc": "Сачувай"
61 },
62 "13": {
63     "en": "Save As",
64     "rs": "Sačuvaj Kao",
65     "rsc": "Сачувай КАО"
66 },
67 "14": {
68     "en": "Run",
69     "rs": "Pokreni",
70     "rsc": "Покрени"
71 },
72 "15": {
73     "en": "Sandbox paused",
74     "rs": "Radni prostor pauziran",
75     "rsc": "Радни простор паузиран"
76 },
77 "16": {
78     "en": "Language",
79     "rs": "Jezik",
80     "rsc": "Језик"
81 },
82 "17": {
83     "en": "Number of latest messages",
84     "rs": "Broj najnovijih poruka",
85     "rsc": "Број најновијих порука"
86 },
87 "18": {
88     "en": "Close log save dialog",
89     "rs": "Zavřete dialog pro uložení dnevníku",
90     "rsc": "Затворите диалог за чување дневника"
91 },
92 "19": {
93     "en": "Close command history",
94     "rs": "Zavřete historii komand",
95     "rsc": "Затвори историју команди"
96 },
97 "20": {
98     "en": "Close script directory dialog",
99     "rs": "Zavřete dialog adresáře skriptů",
100    "rsc": "Затворите дијалог директоријума скрипте"
101 },
102 "21": {
103     "en": "Close paths menu",
104     "rs": "Zavřete meni cesta",
105     "rsc": "Затвори мени путања"
```

```
106 },
107 "22": {
108     "en": "Close help",
109     "rs": "Zatvori pomoc",
110     "rsc": "Затвори помоћ"
111 },
112 "23": {
113     "en": "Close info",
114     "rs": "Zatvori informacije",
115     "rsc": "Затворите информације"
116 },
117 "24": {
118     "en": "Close settings",
119     "rs": "Zatvorite podešavanja",
120     "rsc": "Затворите подешавања"
121 },
122 "25": {
123     "en": "Select language",
124     "rs": "Izaberite jezik",
125     "rsc": "Изаберите језик"
126 },
127 "26": {
128     "en": "Open new script",
129     "rs": "Otvorite novu skriptu",
130     "rsc": "Отворите нову скрипту"
131 },
132 "27": {
133     "en": "Open script from file",
134     "rs": "Otvorite skriptu iz datoteke",
135     "rsc": "Отворите скрипту из датотеке"
136 },
137 "28": {
138     "en": "Save script",
139     "rs": "Sačuvaj skriptu",
140     "rsc": "Сачувай скрипту"
141 },
142 "29": {
143     "en": "Script save as",
144     "rs": "Skript sačuvaj kao",
145     "rsc": "Скрипт сачувай као"
146 },
147 "30": {
148     "en": "Save script and run",
149     "rs": "Sačuvajte skriptu i pokreni",
150     "rsc": "Сачувайте скрипту и покрени"
151 },
152 "31": {
153     "en": "Click to go back",
154     "rs": "Kliknite da biste se vratili",
155     "rsc": "Кликните да бисте се вратили"
156 },
157 "32": {
158     "en": "Click to go forward",
159     "rs": "Kliknite da idete napred",
160     "rsc": "Кликните да идете напред"
```

```
161 },
162 "33": {
163     "en": "Click to go up",
164     "rs": "Kliknite da idete gore",
165     "rsc": "Кликните да идете горе"
166 },
167 "34": {
168     "en": "Select folder",
169     "rs": "Izaberi direktorijum",
170     "rsc": "Изабери директоријум"
171 },
172 "35": {
173     "en": "Save this path",
174     "rs": "Sačuvajte ovu putanju",
175     "rsc": "Сачувајте ову путању"
176 },
177 "36": {
178     "en": "More folder options",
179     "rs": "Više opcija direktorijuma",
180     "rsc": "Више опција директоријума"
181 },
182 "37": {
183     "en": "Refresh file browser",
184     "rs": "Osvežite pregledač datoteka",
185     "rsc": "Освежите прегледач датотека"
186 },
187 "38": {
188     "en": "Clear workspace",
189     "rs": "Očistite radni prostor",
190     "rsc": "Очистите радни простор"
191 },
192 "39": {
193     "en": "Clear command history",
194     "rs": "Obrišite istoriju komandi",
195     "rsc": "Обришите историју команди"
196 },
197 "40": {
198     "en": "Open settings",
199     "rs": "Otvorite podešavanja",
200     "rsc": "Отворите подешавања"
201 },
202 "41": {
203     "en": "Hide timestamp",
204     "rs": "Sakrij vremensku oznaku",
205     "rsc": "Сакриј временску ознаку"
206 },
207 "42": {
208     "en": "Turn off auto scroll",
209     "rs": "Isključite automatsko pomeranje",
210     "rsc": "Искључите автоматско померање"
211 },
212 "43": {
213     "en": "Clear command window",
214     "rs": "Očistite komandni prozor",
215     "rsc": "Очистите командни прозор"
```

```
216 },
217 "44": {
218     "en": "Save log to file",
219     "rs": "Sačuvajte dnevnik u datoteku",
220     "rsc": "Сачувайте дневник у датотеку"
221 },
222 "45": {
223     "en": "Scroll to bottom",
224     "rs": "Pomerite se do dna",
225     "rsc": "Померите се до дна"
226 },
227 "46": {
228     "en": "Close settings dialog",
229     "rs": "Zatvorite dijalog podešavanja",
230     "rsc": "Затворите дијалог подешавања"
231 },
232 "47": {
233     "en": "File modified",
234     "rs": "Fajl je izmenjen",
235     "rsc": "Фајл је изменјен"
236 },
237 "48": {
238     "en": "The file",
239     "rs": "Fajl",
240     "rsc": "Фајл"
241 },
242 "49": {
243     "en": "is about to be closed but has been modified. Do you want to save or
244         discard the changes?",
245     "rs": "će biti zatvoren ali je izmenjen. Da li želite da sačuvate ili
246         odbacite izmene?",
247     "rsc": "ће бити затворен али је изменјен. Да ли желите да сачувате или одба
248         ците измене?"
249 },
250 "50": {
251     "en": "Save",
252     "rs": "Sačuvaj",
253     "rsc": "Сачувай"
254 },
255 "51": {
256     "en": "Discard",
257     "rs": "Odbaci",
258     "rsc": "Одбаци"
259 },
260 "52": {
261     "en": "Cancel",
262     "rs": "Otkaži",
263     "rsc": "Откажи"
264 },
265 "53": {
266     "en": "Variable",
267     "rs": "Promenljiva",
268     "rsc": "Променљива"
269 },
270 "54": {
```

```
268      "en": "Type",
269      "rs": "Tip",
270      "rsc": "Тип"
271  },
272  "55": {
273      "en": "Class",
274      "rs": "Klasa",
275      "rsc": "Класа"
276  },
277  "56": {
278      "en": "Command window settings",
279      "rs": "Podešavanja komandnog prozora",
280      "rsc": "Подешавања командног прозора"
281  },
282  "57": {
283      "en": "Change settings",
284      "rs": "Promenite podešavanja",
285      "rsc": "Промените подешавања"
286  },
287  "58": {
288      "en": "Save log",
289      "rs": "Sačuvaj dnevnik",
290      "rsc": "Сачувај дневник"
291  },
292  "59": {
293      "en": "Write timestamps",
294      "rs": "Upisuj vremenske oznake",
295      "rsc": "Уписуј временске ознаке"
296  },
297  "60": {
298      "en": "Save log",
299      "rs": "Sačuvaj dnevnik",
300      "rsc": "Сачувај дневник"
301  },
302  "61": {
303      "en": "Session command history",
304      "rs": "Istorijska komandi",
305      "rsc": "Историја команди"
306  },
307  "62": {
308      "en": "Script directory",
309      "rs": "Direktorijum skripte",
310      "rsc": "Директоријум скрипте"
311  },
312  "63": {
313      "en": "The directory",
314      "rs": "Direktorijum",
315      "rsc": "Директоријум"
316  },
317  "64": {
318      "en": "is not a working directory nor a saved directory, would you like to
319          change a working directory or save this directory?",
320      "rs": "nije radni direktorijum niti sačuvani direktorijum, da li želite da
321          promenite radni direktorijum ili sačuvate ovaj direktorijum?",
322      "rsc": "није радни директоријум нити сачувани директоријум, да ли желите да
```

а промените радни директоријум или сачувате овај директоријум?"

```
321 },
322 "65": {
323     "en": "Change working directory",
324     "rs": "Promeni radni direktorijum",
325     "rsc": "Промени радни директоријум"
326 },
327 "66": {
328     "en": "Save directory",
329     "rs": "Sačuvaj direktorijum",
330     "rsc": "Сачувај директоријум"
331 },
332 "67": {
333     "en": "Run",
334     "rs": "Pokreni",
335     "rsc": "Покрени"
336 },
337 "68": {
338     "en": "Saved paths",
339     "rs": "Sačuvane putanje",
340     "rsc": "Сачуване путање"
341 },
342 "69": {
343     "en": "Command window keyboard shortcuts",
344     "rs": "Prečice tastature komandnog prozora",
345     "rsc": "Пречице тастатуре командног прозора"
346 },
347 "70": {
348     "en": "Shortcut",
349     "rs": "Prečica",
350     "rsc": "Пречица"
351 },
352 "71": {
353     "en": "Action",
354     "rs": "Akcija",
355     "rsc": "Акција"
356 },
357 "72": {
358     "en": "Clear input / Close dialog",
359     "rs": "Obriši unos / Zatvori dijalog",
360     "rsc": "Обриши унос / Затвори дијалог"
361 },
362 "73": {
363     "en": "Go backward through the command history",
364     "rs": "Idi unazad kroz istoriju komandi",
365     "rsc": "Иди уназад кроз историју команди"
366 },
367 "74": {
368     "en": "Go forward through the command history",
369     "rs": "Idi unapred kroz istoriju komandi",
370     "rsc": "Иди унапред кроз историју команди"
371 },
372 "75": {
373     "en": "First command in the command history",
374     "rs": "Prva komanda u istoriji komandi",
```

```
375     "rsc": "Прва команда у историји команди"
376 },
377 "76": {
378     "en": "Last command in the command history",
379     "rs": "Poslednja komanda u istoriji komandi",
380     "rsc": "Последња команда у историји команди"
381 },
382 "77": {
383     "en": "Break the line",
384     "rs": "Prekini liniju",
385     "rsc": "Прекини линију"
386 },
387 "78": {
388     "en": "Repeat the last command",
389     "rs": "Ponovi poslednju komandu",
390     "rsc": "Понови последњу команду"
391 },
392 "79": {
393     "en": "Show the command history",
394     "rs": "Prikaži istoriju komandi",
395     "rsc": "Прикажи историју команди"
396 },
397 "80": {
398     "en": "Clear the command history",
399     "rs": "Obriši istoriju komandi",
400     "rsc": "Обриши историју команди"
401 },
402 "81": {
403     "en": "Complete current command from the command history",
404     "rs": "Završi trenutnu komandu iz istorije komandi",
405     "rsc": "Заврши тренутну команду из историје команди"
406 },
407 "82": {
408     "en": "Focus command input",
409     "rs": "Fokusiraj na unos komandi",
410     "rsc": "Фокусирај на унос команди"
411 },
412 "83": {
413     "en": "Open help",
414     "rs": "Otvorite pomoć",
415     "rsc": "Отворите помоћ"
416 },
417 "84": {
418     "en": "Open settings dialog",
419     "rs": "Otvorite dijalog podešavanja",
420     "rsc": "Отворите дијалог подешавања"
421 },
422 "85": {
423     "en": "Open log save dialog",
424     "rs": "Otvorite dijalog za čuvanje dnevnika",
425     "rsc": "Отворите дијалог за чување дневника"
426 },
427 "86": {
428     "en": "This software uses open-source components stated below, copyright  
and other proprietary rights of these components belong to their
```

```
429     respective owners." ,  
429     "rs": "Ovaj softver koristi open-source komponente navedene ispod ,  
430         autorska prava i druga prava intelektualne svojine ovih komponenti  
430         pripadaju njihovim vlasnicima." ,  
430     "rsc": "Овај софтвер користи open-source компоненте наведене испод, ауторс  
431         ка права и друга права интелектуалне својине ових компоненти припадају  
431         њиховим власницима ."  
431 },  
432     "87": {  
433         "en": "Ready... " ,  
434         "rs": "Spremno... " ,  
435         "rsc": "Спремно..."  
436 },  
437     "88": {  
438         "en": "Evaluating... " ,  
439         "rs": "Evaluacija... " ,  
440         "rsc": "Евалуација..."  
441 },  
442     "89": {  
443         "en": "Stop request sent by user... " ,  
444         "rs": "Korisnik je posao zahtev za zaustavljanje... " ,  
445         "rsc": "Корисник је послао захтев за заустављање..."  
446 },  
447     "90": {  
448         "en": "Stop loop triggered... " ,  
449         "rs": "Petlja izvršavanja je zaustavljena... " ,  
450         "rsc": "Петља извршавања је заустављена..."  
451 },  
452     "91": {  
453         "en": "Sandbox activity" ,  
454         "rs": "Aktivnost radnog prostora" ,  
455         "rsc": "Активност радног простора"  
456 },  
457     "92": {  
458         "en": "Unable to scan directory" ,  
459         "rs": "Nije moguće skenirati direktorijum" ,  
460         "rsc": "Није могуће скенирати директоријум"  
461 },  
462     "93": {  
463         "en": "Running total of" ,  
464         "rs": "Ukupno pokrenuto" ,  
465         "rsc": "Укупно покренуто"  
466 },  
467     "94": {  
468         "en": "tests" ,  
469         "rs": "testova" ,  
470         "rsc": "тестова"  
471 },  
472     "95": {  
473         "en": "Test" ,  
474         "rs": "Test" ,  
475         "rsc": "Тест"  
476 },  
477     "96": {  
478         "en": "failed with error" ,
```

```
479      "rs": "nije uspeo zbog greške",
480      "rsc": "није успео због грешке"
481  },
482  "97": {
483    "en": "failed",
484    "rs": "nije uspeo",
485    "rsc": "није успео"
486  },
487  "98": {
488    "en": "passed",
489    "rs": "uspešan",
490    "rsc": "успешан"
491  },
492  "99": {
493    "en": "Final results",
494    "rs": "Konačni rezultati",
495    "rsc": "Коначни резултати"
496  },
497  "100": {
498    "en": "Tests passed",
499    "rs": "Testovi uspešni",
500    "rsc": "Тестови успешни"
501  },
502  "101": {
503    "en": "Tests failed",
504    "rs": "Testovi nisu uspešni",
505    "rsc": "Тестови нису успешни"
506  },
507  "102": {
508    "en": "There is no implemented tests.",
509    "rs": "Nema implementiranih testova.",
510    "rsc": "Нема имплементираних тестова."
511  },
512  "103": {
513    "en": "Script not found at",
514    "rs": "Skripta nije pronađena na",
515    "rsc": "Скрипта није пронађена на"
516  },
517  "104": {
518    "en": "Not enough lines in file",
519    "rs": "Nema dovoljno linija u fajlu",
520    "rsc": "Нема довољно линија у фајлу"
521  },
522  "105": {
523    "en": "script",
524    "rs": "skripta",
525    "rsc": "скрипта"
526  },
527  "106": {
528    "en": "File",
529    "rs": "Fajl",
530    "rsc": "Фајл"
531  },
532  "107": {
533    "en": "is selected. There are also",
```

```
534     "rs": "je odabran. Takođe postoje",
535     "rsc": "је одабран. Такође постоје"
536 },
537 "108": {
538     "en": "is selected. There is also",
539     "rs": "je odabran. Takođe postoji",
540     "rsc": "је одабран. Такође постоји"
541 },
542 "109": {
543     "en": "Failed to find",
544     "rs": "Nije pronađeno",
545     "rsc": "Није пронађено"
546 },
547 "110": {
548     "en": "module",
549     "rs": "modul",
550     "rsc": "модул"
551 },
552 "111": {
553     "en": "Provided path is not string",
554     "rs": "Data putanja nije niz karaktera",
555     "rsc": "Дата путања није низ карактера"
556 },
557 "112": {
558     "en": "line",
559     "rs": "linija",
560     "rsc": "линија"
561 },
562 "113": {
563     "en": "column",
564     "rs": "kolona",
565     "rsc": "колона"
566 },
567 "114": {
568     "en": "at",
569     "rs": "na",
570     "rsc": "на"
571 },
572 "115": {
573     "en": "Not implemented!",
574     "rs": "Nije implementirano!",
575     "rsc": "Није имплементирано!"
576 },
577 "116": {
578     "en": "Code arrived to user defined end point at line",
579     "rs": "Kod je došao do korisnički definisane tačke zaustavaljanja na
580         liniji",
581     "rsc": "Код је дошао до кориснички дефинисане тачке заустављања на линији"
582 },
583 "117": {
584     "en": "Unable to write to file",
585     "rs": "Nije moguće pisati u fajl",
586     "rsc": "Није могуће писати у фајл"
587 },
588 "118": {
```

```
588     "en": "File content is not valid JSON",
589     "rs": "Sadržaj fajla nije ispravan JSON",
590     "rsc": "Садржај фајла није исправан JSON"
591 },
592 "119": {
593     "en": "Folder selection canceled",
594     "rs": "Odabir direktorijuma je otkazan",
595     "rsc": "Одабир директоријума је отказан"
596 },
597 "120": {
598     "en": "Content of binding.gyp is not valid JSON",
599     "rs": "Sadržaj binding.gyp nije ispravan JSON",
600     "rsc": "Садржај binding.gyp није исправан JSON"
601 },
602 "121": {
603     "en": "Unknown response",
604     "rs": "Nepoznat odgovor",
605     "rsc": "Непознат одговор"
606 },
607 "122": {
608     "en": "No targets defined",
609     "rs": "Nisu definisani ciljevi",
610     "rsc": "Нису дефинисани циљеви"
611 },
612 "123": {
613     "en": "File binding.gyp not found",
614     "rs": "Fajl binding.gyp nije pronađen",
615     "rsc": "Фајл binding.gyp није пронађен"
616 },
617 "124": {
618     "en": "Format not supported.",
619     "rs": "Format nije podržan.",
620     "rsc": "Формат није подржан."
621 },
622 "125": {
623     "en": "User requested loop stop!",
624     "rs": "Korisnik je zatražio zaustavljanje petlje!",
625     "rsc": "Корисник је захтевао заустављање петље!"
626 },
627 "126": {
628     "en": "Open canceled",
629     "rs": "Otvaranje je otkazano",
630     "rsc": "Отварање је отказано"
631 },
632 "127": {
633     "en": "Callback is missing",
634     "rs": "Callback nedostaje",
635     "rsc": "Callback недостаје"
636 },
637 "128": {
638     "en": "Unable to find files in folder",
639     "rs": "Nije moguće pronaći fajlove u direktorijumu",
640     "rsc": "Није могуће пронаћи фајлове у директоријуму"
641 },
642 "129": {
```

```

643      "en": "Save canceled",
644      "rs": "Čuvanje je otkazano",
645      "rsc": "Чување је отказано"
646    },
647    "130": {
648      "en": "Path override",
649      "rs": "Pregaćena putanja",
650      "rsc": "Прегажена путања"
651    },
652    "131": {
653      "en": "Script run aborted, file not saved.",
654      "rs": "Izvršavanje skripte prekinuto, fajl nije sačuvan.",
655      "rsc": "Извршавање скрипте прекинуто, фајл није сачуван."
656    },
657    "132": {
658      "en": "File open canceled",
659      "rs": "Otvaranje fajla je otkazano",
660      "rsc": "Отварање фајла је отказано"
661    },
662    "133": {
663      "en": "Script",
664      "rs": "Skripta",
665      "rsc": "Скрипта"
666    },
667    "134": {
668      "en": "already open",
669      "rs": "je već otvoren",
670      "rsc": "је већ отворен"
671    },
672    "135": {
673      "en": "For more information, visit",
674      "rs": "Za više informacija, posetite",
675      "rsc": "За више информација, посетите"
676    },
677    "136": {
678      "en": "Copyright",
679      "rs": "Autorsko pravo",
680      "rsc": "Ауторско право"
681    },
682    "137": {
683      "en": "This program is free software: you can redistribute it and/or
684      modify it under the terms of the GNU Lesser General Public License as
685      published by the Free Software Foundation, either version 3 of the
686      License, or (at your option) any later version.",
687      "rs": "Ovaj program je besplatan softver: možete ga redistribuirati i/ili
688      modifikovati pod uslovima GNU manje opšte javne licence koju je
689      objavila Fondacija za slobodni softver, bilo verzije 3 licence, ili (
690      po vašem izboru) bilo koje kasnije verzije.",
691      "rsc": "Овај програм је бесплатан софтвер: можете га редистрибуирати и/или
692      модификовати под условима ГНУ мање опште јавне лиценце коју је објави
693      ла Фондација за слободни софтвер, било верзије 3 лиценце, или (по ваше
694      м избору) било које касније верзије."
695    },
696    "138": {
697      "en": "This program is distributed in the hope that it will be useful, but
698

```

WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.",
689 "rs": "Ovaj program se distribuira u nadi da c e biti koristan , ali BEZ
ИКАКВЕ ГАРАНЦИЈЕ; чак и без implicitne garancije o PRODAJNOSTI ili
PRIKLADNOSTI ZA ODREĐENU NAMENU. Pogledajte GNU manje opštu javnu
licencu za više detalja.",
690 "rsc": "Овај програм се дистрибуира у нади да ће бити користан, али БЕЗ ИК
АКВЕ ГАРАНЦИЈЕ; чак и без имплицитне гаранције о ПРОДАЈНОСТИ или ПРИКЛ
АДНОСТИ ЗА ОДРЕЂЕНУ НАМЕНУ. Погледајте ГНУ мање општу јавну лиценцу за
више детаља."
691 },
692 "139": {
693 "en": "For help type",
694 "rs": "Za pomoć unesite",
695 "rsc": "За помоћ унесите"
696 },
697 "140": {
698 "en": "No saved paths.",
699 "rs": "Nema sačuvanih putanja.",
700 "rsc": "Нема сачуваних путања."
701 },
702 "141": {
703 "en": "Choose N-API module folder",
704 "rs": "Izaberite direktorijum N-API modula",
705 "rsc": "Изаберите директоријум N-API модула"
706 },
707 "142": {
708 "en": "Choose folder",
709 "rs": "Izaberite direktorijum",
710 "rsc": "Изаберите директоријум"
711 },
712 "143": {
713 "en": "Save figure",
714 "rs": "Sačuvajte grafik",
715 "rsc": "Сачувайте график"
716 },
717 "144": {
718 "en": "Save script to file",
719 "rs": "Sačuvajte skriptu u datoteku",
720 "rsc": "Сачувайте скрипту у датотеку"
721 },
722 "145": {
723 "en": "Save script",
724 "rs": "Sačuvajte skriptu",
725 "rsc": "Сачувайте скрипту"
726 },
727 "146": {
728 "en": "Load script from file",
729 "rs": "Učitajte skriptu iz datoteke",
730 "rsc": "Учитајте скрипту из датотеке"
731 },
732 "147": {
733 "en": "Load script",
734 "rs": "Učitajte skriptu",

```
735     "rsc": "Учитајте скрипту"
736 },
737 "148": {
738     "en": "Choose working directory",
739     "rs": "Izaberite radni direktorijum",
740     "rsc": "Изаберите радни директоријум"
741 },
742 "149": {
743     "en": "Set",
744     "rs": "Postavi",
745     "rsc": "Постави"
746 },
747 "150": {
748     "en": "Save log file",
749     "rs": "Sačuvajte datoteku dnevnika",
750     "rsc": "Сачувајте датотеку дневника"
751 },
752 "151": {
753     "en": "Save log",
754     "rs": "Sačuvajte dnevnik",
755     "rsc": "Сачувајте дневник"
756 },
757 "152": {
758     "en": "Plot save as",
759     "rs": "Sačuvaj grafikon kao",
760     "rsc": "Сачувај графикон као"
761 },
762 "153": {
763     "en": "Zoom part of the plot",
764     "rs": "Zumiraj deo grafikona",
765     "rsc": "Зумирај део графикона"
766 },
767 "154": {
768     "en": "Pan plot",
769     "rs": "Pomeranje grafikona",
770     "rsc": "Померање графикона"
771 },
772 "155": {
773     "en": "Rotate plot",
774     "rs": "Rotiraj grafikon",
775     "rsc": "Ротирај графикон"
776 },
777 "156": {
778     "en": "Zoom in",
779     "rs": "Uvećaj",
780     "rsc": "Увећај"
781 },
782 "157": {
783     "en": "Zoom out",
784     "rs": "Umanji",
785     "rsc": "Умањи"
786 },
787 "158": {
788     "en": "Fit",
789     "rs": "Prilagodi",
```

```
790      "rsc": "Прилагоди"
791  },
792  "159": {
793    "en": "Save As",
794    "rs": "Sačuvaj kao",
795    "rsc": "Сачувај као"
796  },
797  "160": {
798    "en": "Zoom",
799    "rs": "Zumiraj",
800    "rsc": "Зумирај"
801  },
802  "161": {
803    "en": "Pan",
804    "rs": "Pomeranje",
805    "rsc": "Померање"
806  },
807  "162": {
808    "en": "Rotate",
809    "rs": "Rotiraj",
810    "rsc": "Ротирај"
811  },
812  "163": {
813    "en": "Zoom in",
814    "rs": "Uvećaj",
815    "rsc": "Увећај"
816  },
817  "164": {
818    "en": "Zoom out",
819    "rs": "Umanji",
820    "rsc": "Умањи"
821  },
822  "165": {
823    "en": "Fit",
824    "rs": "Prilagodi",
825    "rsc": "Прилагоди"
826  },
827  "166": {
828    "en": "Show timestamp",
829    "rs": "Prikaži vremensku oznaku",
830    "rsc": "Прикажи временску ознаку"
831  },
832  "167": {
833    "en": "Turn on auto scroll",
834    "rs": "Uključite automatsko pomeranje",
835    "rsc": "Укључите аутоматско померање"
836  },
837  "168": {
838    "en": "Search",
839    "rs": "Pretraga",
840    "rsc": "Претрага"
841  },
842  "169": {
843    "en": "Show search dialog",
844    "rs": "Prikaži dijalog za pretragu",
```

```
845     "rsc": "Прикажи дијалог за претрагу"
846 },
847 "170": {
848     "en": "Failed to compile module!",
849     "rs": "Neuspešno kompajliranje modula!",
850     "rsc": "Неуспешно компајлирање модула!"
851 },
852 "171": {
853     "en": "Unable to recompile module, try reseting sandbox with <span class='eval-code'> resetSandbox()</span> function.",
854     "rs": "Nije moguće rekompajlirati modul, pokušajte reset sandbox-a sa <span class='eval-code'>resetSandbox()</span> funkcijom.",
855     "rsc": "Није могуће рекомпајлирати модул, покушајте ресет sandbox-a са <span class='eval-code'>resetSandbox()</span> функцијом."
856 },
857 "172": {
858     "en": "Figure is not opened.",
859     "rs": "Grafik nije otvoren.",
860     "rsc": "График није отворен."
861 },
862 "173": {
863     "en": "Source directory does not exist.",
864     "rs": "Izvorni direktorijum ne postoji.",
865     "rsc": "Изворни директоријум не постоји."
866 },
867 "174": {
868     "en": "Failed to open specified file in new window!",
869     "rs": "Neuspešno otvaranje specificirane datoteke u novom prozoru!",
870     "rsc": "Неуспешно отварање специфициране датотеке у новом прозору!"
871 },
872 "175": {
873     "en": "Wait for lib to be loaded!",
874     "rs": "Sačekajte da se biblioteka učita!",
875     "rsc": "Сачекајте да се библиотека чита!"
876 },
877 "176": {
878     "en": "Arrays must be of the same length!",
879     "rs": "Nizovi moraju biti iste dužine!",
880     "rsc": "Низови морају бити исте дужине!"
881 },
882 "177": {
883     "en": "Invalid input types!",
884     "rs": "Nevažeći tipovi ulaza!",
885     "rsc": "Неважећи типови улаза!"
886 },
887 "178": {
888     "en": "Array size does not match the dimensions provided.",
889     "rs": "Veličina niza ne odgovara datim dimenzijama.",
890     "rsc": "Величина низа не одговара датим димензијама."
891 },
892 "179": {
893     "en": "FreeCAD instance already active!",
894     "rs": "FreeCAD instanca je već aktivna!",
895     "rsc": "FreeCAD инстанца је већ активна!"
896 },
```

```
897 "180": {
898     "en": "Could not start FreeCAD!",
899     "rs": "Nije moguće pokrenuti FreeCAD!",
900     "rsc": "Није могуће покренути FreeCAD!"
901 },
902 "181": {
903     "en": "Could not find FreeCAD TCP Server!",
904     "rs": "Nije moguće pronaći FreeCAD TCP server!",
905     "rsc": "Није могуће пронаћи FreeCAD TCP сервер!"
906 },
907 "182": {
908     "en": "No FreeCAD instance!",
909     "rs": "Nema FreeCAD instance!",
910     "rsc": "Нема FreeCAD инстанце!"
911 },
912 "183": {
913     "en": "File does not exist!",
914     "rs": "Datoteka ne postoji!",
915     "rsc": "Датотека не постоји!"
916 },
917 "184": {
918     "en": "is a forbidden name.",
919     "rs": "je zabranjeno ime.",
920     "rsc": "је забрањено име."
921 },
922 "185": {
923     "en": "Invalid variable declaration",
924     "rs": "Nevažeća deklaracija promenljive",
925     "rsc": "Неважећа декларација променљиве"
926 },
927 "186": {
928     "en": "Invalid function declaration",
929     "rs": "Nevažeća deklaracija funkcije",
930     "rsc": "Неважећа декларација функције"
931 },
932 "187": {
933     "en": "Invalid class declaration",
934     "rs": "Nevažeća deklaracija klase",
935     "rsc": "Неважећа декларација класе"
936 },
937 "188": {
938     "en": "Invalid import",
939     "rs": "Nevažeći import",
940     "rsc": "Неважећи импорт"
941 },
942 "189": {
943     "en": "Increment dx cannot be zero.",
944     "rs": "Povećanje dx ne može biti nula.",
945     "rsc": "Повећање dx не може бити нула."
946 },
947 "190": {
948     "en": "Input must be an array.",
949     "rs": "Ulaz mora biti niz.",
950     "rsc": "Улаз мора бити низ."
951 },
```

```

952  "191": {
953    "en": "Array is empty or contains only NaN values.",
954    "rs": "Niz je prazan ili sadrži samo NaN vrednosti.",
955    "rsc": "Низ је празан или садржи само NaN вредности."
956  },
957  "192": {
958    "en": "Input must be a number, a complex number object, or an array
959    thereof.",
960    "rs": "Ulaz mora biti broj, objekat kompleksnog broja ili niz takvih
961    objekata.",
962    "rsc": "Улаз мора бити број, објекат комплексног броја или низ таквих обје
963    ката."
964  },
965  "193": {
966    "en": "The file is not a valid OFF one.",
967    "rs": "Datoteka nije validna OFF datoteka.",
968    "rsc": "Датотека није валидна OFF датотека."
969  },
970  "194": {
971    "en": "Incomplete header information in OFF file.",
972    "rs": "Nepotpune informacije u zagлављу OFF datoteke.",
973    "rsc": "Непотпуне информације у заглављу OFF датотеке."
974  },
975  "195": {
976    "en": "Problem in reading vertices.",
977    "rs": "Problem pri čitanju tačaka.",
978    "rsc": "Проблем при читању тачака."
979  },
980  "196": {
981    "en": "Problem in reading faces.",
982    "rs": "Problem pri čitanju stranica.",
983    "rsc": "Проблем при читању страница."
984  },
985  "197": {
986    "en": "Step size cannot be zero.",
987    "rs": "Veličina koraka ne može biti nula.",
988    "rsc": "Величина корака не може бити нула."
989  },
990  "198": {
991    "en": "Step size does not align with start and end values.",
992    "rs": "Veličina koraka se ne slaže sa početnim i krajnjim vrednostima.",
993    "rsc": "Величина корака се не слаже са почетним и крајњим вредностима."
994  },
995  "199": {
996    "en": "No file for window!",
997    "rs": "Nema datoteke za prozor!",
998    "rsc": "Нема датотеке за прозор!"
999  },
1000  "200": {
1001    "en": "Alias 'as' is required when import is '*' in module",
1002    "rs": "Alias 'as' je potreban kada je import '*' u modulu",
1003    "rsc": "Alias 'as' је потребан када је import '*' у модулу"
1004  },
1005  "201": {
1006    "en": "No connection with OMC. Create a new instance of OpenModelicaLink"
1007  }

```

```
session.",
1004 "rs": "Nema veze sa OMG-om. Kreirajte novu instancu OpenModelicaLink
      sesije.",
1005 "rsc": "Нема везе са OMG-ом. Креирајте нову инстанцу OpenModelicaLink сесије."
1006 },
1007 "202": {
1008   "en": "Model is not Linearized",
1009   "rs": "Model nije linearizovan",
1010   "rsc": "Модел није линеаризован"
1011 },
1012 "203": {
1013   "en": "Filename and modelname are required.",
1014   "rs": "Ime datoteke i ime modela su obavezni.",
1015   "rsc": "Име датотеке и име модела су обавезни."
1016 },
1017 "204": {
1018   "en": "XML file is not generated.",
1019   "rs": "XML datoteka nije generisana.",
1020   "rsc": "XML датотека није генерисана."
1021 },
1022 "205": {
1023   "en": "Model cannot be Simulated: executable not found.",
1024   "rs": "Model se ne može simulirati: izvršna datoteka nije pronađena.",
1025   "rsc": "Модел се не може симулирати: извршна датотека није пронађена."
1026 },
1027 "206": {
1028   "en": "Model cannot be Simulated: xmlfile not found.",
1029   "rs": "Model se ne može simulirati: XML datoteka nije pronađena.",
1030   "rsc": "Модел се не може симулирати: XML датотека није пронађена."
1031 },
1032 "207": {
1033   "en": "Linearization cannot be performed: ",
1034   "rs": "Linearizacija se ne može izvršiti: ",
1035   "rsc": "Линеаризација се не може извршити: "
1036 },
1037 "208": {
1038   "en": "Result File does not exist!",
1039   "rs": "Datoteka sa rezultatima ne postoji!",
1040   "rsc": "Датотека са резултатима не постоји!"
1041 },
1042 "209": {
1043   "en": " is not a parameter",
1044   "rs": " nije parametar",
1045   "rsc": " није параметар"
1046 },
1047 "210": {
1048   "en": " is not a Simulation Option",
1049   "rs": " nije opcija za simulaciju",
1050   "rsc": " није опција за симулацију"
1051 },
1052 "211": {
1053   "en": " is not a Linearization Option",
1054   "rs": " nije opcija za linearizaciju",
1055   "rsc": " није опција за линеаризацију"
```

```
1056 },
1057 "212": {
1058     "en": " is not an Input",
1059     "rs": " nije ulaz",
1060     "rsc": " није улаз"
1061 },
1062 "213": {
1063     "en": "You can see current workspace in main window. Would you like to
1064         continue?",
1065     "rs": "Možete videti trenutni radni prostor u glavnom prozoru. Da li ž
1066         elite da nastavite?",
1067     "rsc": "Можете видети тренутни радни простор у главном прозору. Да ли жели
1068         те да наставите?"
1069 },
1070 "214": {
1071     "en": "yes",
1072     "rs": "da",
1073     "rsc": "да"
1074 },
1075 "215": {
1076     "en": "no",
1077     "rs": "ne",
1078     "rsc": "не"
1079 },
1080 "216": {
1081     "en": "Code arrived to breakpoint at line",
1082     "rs": "Kod je dostigao breakpoint na liniji",
1083     "rsc": "Код је досегао breakpoint на линији"
1084 },
1085 "217": {
1086     "en": "Code paused at line",
1087     "rs": "Kod je pauziran na liniji",
1088     "rsc": "Код је паузиран на линији"
1089 },
1090 "218": {
1091     "en": "No help found for ",
1092     "rs": "Nije pronadjena dokumentacija za ",
1093     "rsc": "Није пронађена документација за "
1094 },
1095 "219": {
1096     "en": "Open documentation",
1097     "rs": "Otvorite dokumentaciju",
1098     "rsc": "Отворите документацију"
1099 },
1100 "220": {
1101     "en": "No source code found for ",
1102     "rs": "Nije pronadjen kod za ",
1103     "rsc": "Није пронађен код за "
1104 },
1105 "221": {
1106     "en": "Reset",
1107     "rs": "Reset",
1108     "rsc": "Pecet"
```

```

1108      "en": "Unable to find audio/video device!",
1109      "rs": "Nije moguce pronaci audio/video uređaj!",
1110      "rsc": "Није могуће пронаћи аудио/видео уређај!"
1111  },
1112  "223": {
1113      "en": "Number of commands in history",
1114      "rs": "Broj komandi u istoriji",
1115      "rsc": "Број команди у историји"
1116  },
1117  "224": {
1118      "en": "Program arduino-cli not found.",
1119      "rs": "Program arduino-cli nije pronadjen.",
1120      "rsc": "Програм arduino-cli није пронађен."
1121  },
1122  "225": {
1123      "en": "Configuration file for arduino-cli not found.",
1124      "rs": "Konfiguracioni fajl za arduino-cli nije pronadjen.",
1125      "rsc": "Конфигурациони фајл за arduino-cli није пронађен."
1126  },
1127  "226": {
1128      "en": "Submit command to workspace.",
1129      "rs": "Pošalji komandu radnom prostoru.",
1130      "rsc": "Пошаљи команду радном простору."
1131  }
1132 }
```

Listing 4 - lang.json

3 cpp

```

1 // AlphaShape3D - alpha-shape-3d.cpp
2 // Author: Milos Petrasinovic <mpetrasinovic@prdc.rs>
3 // PR-DC, Republic of Serbia
4 // info@prdc.rs
5 // -----
6
7 #include "alpha-shape-3d.h"
8
9 namespace alpha_shape_3d_ns {
10
11 #ifdef PROFILE_SOLVER_MP_6RSS
12 // Function to start the timer and return the start time
13 time_point<steady_clock> tic() {
14     return steady_clock::now();
15 }
16
17 // Function to stop the timer and return the elapsed time
18 long toc(const time_point<steady_clock>& startTime) {
19     return duration_cast<milliseconds>(steady_clock::now() - startTime).count();
20 }
21 #endif
22
23 // Function to get current time
24 std::string getCurrentTime() {
```

```

25 // get current time
26 auto now = system_clock::now();
27
28 // get number of milliseconds for the current second
29 // (remainder after division into seconds)
30 auto ms = duration_cast<milliseconds>(now.time_since_epoch()) % 1000;
31
32 // convert to std::time_t in order to convert to std::tm (broken time)
33 auto timer = system_clock::to_time_t(now);
34
35 // convert to broken time
36 std::tm bt = *std::localtime(&timer);
37
38 std::ostringstream oss;
39
40 oss << std::put_time(&bt, "%H:%M:%S"); // HH:MM:SS
41 oss << '.' << std::setfill('0') << std::setw(3) << ms.count();
42
43 return oss.str();
44 }
45
46 // Function to console log data
47 int consoleLog(uint8_t level, const char* format, ...) {
48 #ifdef DEBUG_SOLVER_MP_6RSS_LEVEL
49   if(level <= DEBUG_SOLVER_MP_6RSS_LEVEL) {
50     printf("\033[0;33m%s SolverMP6RSS]\033[0m ", getCurrentTime().c_str());
51     va_list vl;
52     va_start(vl, format);
53     auto ret = vprintf(format, vl);
54     va_end(vl);
55     printf("\n");
56     return ret;
57   }
58 #endif
59   return 0;
60 }
61
62 // AlphaShape3D()
63 // Object constructor
64 // -----
65 AlphaShape3D::AlphaShape3D(const Napi::CallbackInfo& info) : Napi::ObjectWrap<
66   AlphaShape3D>(info) {
67 #ifdef DEBUG_ALPHA_SHAPE_3D
68   consoleLog(0, "Called constructor");
69 #endif
70   this->alphaShape = nullptr;
71   this->delaunayTriangulation = nullptr;
72 }
73
74 // ~AlphaShape3D()
75 // Object destructor
76 // -----
77 AlphaShape3D::~AlphaShape3D(void) {
78   if(this->delaunayTriangulation){
79     delete this->delaunayTriangulation;

```

```

79     this->delaunayTriangulation = nullptr ;
80   }
81   if(this->alphaShape){
82     delete this->alphaShape;
83     this->alphaShape = nullptr ;
84   }
85 }
86
87 // Init() function
88 // -----
89 Napi::Object AlphaShape3D::Init(Napi::Env env, Napi::Object exports) {
90   Napi::Function func = DefineClass(env, "AlphaShape3D", {
91     InstanceMethod("newShape", &AlphaShape3D::NewShapeJS),
92     InstanceMethod("getAlpha", &AlphaShape3D::GetAlphaJS),
93     InstanceMethod("setAlpha", &AlphaShape3D::SetAlphaJS),
94     InstanceMethod("getNumRegions", &AlphaShape3D::GetNumRegionsJS),
95     InstanceMethod("getAlphaSpectrum", &AlphaShape3D::GetAlphaSpectrumJS),
96     InstanceMethod("getCriticalAlpha", &AlphaShape3D::GetCriticalAlphaJS),
97     InstanceMethod("getSurfaceArea", &AlphaShape3D::GetSurfaceAreaJS),
98     InstanceMethod("getVolume", &AlphaShape3D::GetVolumeJS),
99     InstanceMethod("getBoundaryFacets", &AlphaShape3D::GetBoundaryFacetsJS),
100    InstanceMethod("writeBoundaryFacets", &AlphaShape3D::WriteBoundaryFacetsJS
101      ),
102    InstanceMethod("checkInShape", &AlphaShape3D::CheckInShapeJS),
103    InstanceMethod("writeOff", &AlphaShape3D::WriteOffJS),
104    InstanceMethod("getTriangulation", &AlphaShape3D::GetTriangulationJS),
105    InstanceMethod("getNearestNeighbor", &AlphaShape3D::GetNearestNeighborJS),
106    InstanceMethod("getSimplifiedShape", &AlphaShape3D::GetSimplifiedShapeJS),
107    InstanceMethod("removeUnusedPoints", &AlphaShape3D::RemoveUnusedPointsJS)
108  });
109  *constructor = Napi::Persistent(func);
110  env.SetInstanceData(constructor);
111
112  exports.Set("AlphaShape3D", func);
113  return exports;
114 }
115
116 // NewShapeJS() function
117 // -----
118 void AlphaShape3D::NewShapeJS(const Napi::CallbackInfo& info) {
119 #ifdef DEBUG_ALPHA_SHAPE_3D
120   consoleLog(0, "Called NewShapeJS()");
121 #endif
122
123  Napi::Env env = info.Env();
124  if(info.Length() < 1 || !info[0].IsArray()){
125    Napi::TypeError::New(env, "Array of points expected").
126      ThrowAsJavaScriptException();
127  }
128
129  Napi::Array jsPoints = info[0].As<Napi::Array>();
130  uint32_t numPoints = jsPoints.Length();
131

```

```

132 // Clear existing data to prevent accumulation
133 this->inputPoints.resize(0, 0);
134 this->Points.clear();
135 this->Vertices.clear();

136
137 // Initialize inputPoints matrix
138 this->inputPoints.resize(numPoints, 3);

139
140 for(uint32_t i = 0; i < numPoints; i++){
141   Napi::Value point = jsPoints[i];
142   if(!point.isArray()){
143     Napi::TypeError::New(env, "Each point should be an array of 3 numbers").
144       ThrowAsJavaScriptException();
145     return;
146   }
147   Napi::Array jsPoint = point.As<Napi::Array>();
148   if(jsPoint.Length() != 3){
149     Napi::TypeError::New(env, "Each point should have exactly 3 coordinates").
150       ThrowAsJavaScriptException();
151     return;
152   }
153   for(uint32_t j = 0; j < 3; j++){
154     this->inputPoints(i, j) = jsPoint.Get(j).As<Napi::Number>().DoubleValue();
155   }
156
157 #ifdef DEBUG_ALPHA_SHAPE_3D
158   std::chrono::steady_clock::time_point begin =
159     std::chrono::steady_clock::now();
160 #endif
161
162   uint32_t n = this->inputPoints.numRows();
163
164 #ifdef DEBUG_ALPHA_SHAPE_3D
165   std::cout << "Reading " << n << " points " << std::endl;
166 #endif
167
168   for(std::size_t i = 0; i < n; i++){
169     this->Points.emplace_back(
170       Point(this->inputPoints(i, 0), this->inputPoints(i, 1), this->
171         inputPoints(i, 2)));
172     this->Vertices.emplace_back(std::make_pair(this->Points.back(), i));
173   }
174
175 #ifdef DEBUG_ALPHA_SHAPE_3D
176   std::cout << "Computing delaunay triangulation." << std::endl;
177 #endif
178
179   // Delete existing triangulation and alphaShape to prevent memory leaks
180   if(this->delaunayTriangulation){
181     delete this->delaunayTriangulation;
182     this->delaunayTriangulation = nullptr;
183   }

```

```

183   if (this->alphaShape){
184     delete this->alphaShape;
185     this->alphaShape = nullptr;
186   }
187
188   this->delaunayTriangulation = new Dt(this->Vertices.begin(), this->Vertices.
189   end());
190
191 #ifdef DEBUG_ALPHA_SHAPE_3D
192   std::cout << "Number of triangulation cells is "
193   << this->delaunayTriangulation->number_of_finite_cells() << std::endl;
194
195 #endif
196
197   this->triangulationMatrix.resize(this->delaunayTriangulation->
198   number_of_finite_cells()*4, 3);
199   uint64_t i_idx = 0;
200   for(Dt::Finite_cells_iterator cit = this->delaunayTriangulation->
201   finite_cells_begin();
202     cit != this->delaunayTriangulation->finite_cells_end(); cit++){
203     this->triangulationMatrix(i_idx, 0) = cit->vertex(0)->info();
204     this->triangulationMatrix(i_idx, 1) = cit->vertex(1)->info();
205     this->triangulationMatrix(i_idx, 2) = cit->vertex(2)->info();
206     i_idx++;
207     this->triangulationMatrix(i_idx, 0) = cit->vertex(0)->info();
208     this->triangulationMatrix(i_idx, 1) = cit->vertex(2)->info();
209     this->triangulationMatrix(i_idx, 2) = cit->vertex(3)->info();
210     i_idx++;
211     this->triangulationMatrix(i_idx, 0) = cit->vertex(1)->info();
212     this->triangulationMatrix(i_idx, 1) = cit->vertex(2)->info();
213     this->triangulationMatrix(i_idx, 2) = cit->vertex(3)->info();
214     i_idx++;
215   }
216
217 #ifdef DEBUG_ALPHA_SHAPE_3D
218   std::cout << "Computing alpha shapes." << std::endl;
219 #endif
220   this->alphaShape = new As3(*this->delaunayTriangulation, As3::GENERAL);
221
222   this->numAlphaValues = this->alphaShape->number_of_alphas();
223
224 #ifdef DEBUG_ALPHA_SHAPE_3D
225   std::cout << "Number of alpha values is "
226   << this->numAlphaValues << std::endl;
227   std::cout << "Max alpha value is "
228   << this->alphaShape->get_nth_alpha(this->numAlphaValues) << std::endl;
229   std::cout << "Min of alpha value is "
230   << this->alphaShape->get_nth_alpha(1) << std::endl;
231 #ifdef PROFILE_SOLVER_MP_6RSS
232   std::chrono::steady_clock::time_point end =

```

```

233     std::chrono::steady_clock::now() ;
234     std::cout << "Time elapsed = "
235         << std::chrono::duration_cast<std::chrono::milliseconds>
236             (end - begin).count()
237         << " ms" << std::endl;
238 #endif
239 #endif
240 }
241
242 // GetAlphaJS() function
243 // -----
244 Napi::Value AlphaShape3D::GetAlphaJS(const Napi::CallbackInfo& info) {
245 #ifdef DEBUG_ALPHA_SHAPE_3D
246     consoleLog(0, "Called GetAlphaJS()");
247 #endif
248
249     Napi::Env env = info.Env();
250     double result = this->getAlpha();
251     return Napi::Number::New(env, result);
252 }
253
254 // SetAlphaJS() function
255 // -----
256 void AlphaShape3D::SetAlphaJS(const Napi::CallbackInfo& info) {
257 #ifdef DEBUG_ALPHA_SHAPE_3D
258     consoleLog(0, "Called SetAlphaJS()");
259 #endif
260
261     Napi::Env env = info.Env();
262     if(info.Length() < 1 || !info[0].IsNumber()){
263         Napi::TypeError::New(env, "Number expected").ThrowAsJavaScriptException();
264     }
265     double alpha = info[0].As<Napi::Number>().DoubleValue();
266     this->setAlpha(alpha);
267 }
268
269 // GetNumRegionsJS() function
270 // -----
271 Napi::Value AlphaShape3D::GetNumRegionsJS(const Napi::CallbackInfo& info) {
272 #ifdef DEBUG_ALPHA_SHAPE_3D
273     consoleLog(0, "Called GetNumRegionsJS()");
274 #endif
275
276     Napi::Env env = info.Env();
277     double result = this->numRegions();
278     return Napi::Number::New(env, result);
279 }
280
281 // GetAlphaSpectrumJS() function
282 // -----
283 Napi::Value AlphaShape3D::GetAlphaSpectrumJS(const Napi::CallbackInfo& info) {
284 #ifdef DEBUG_ALPHA_SHAPE_3D
285     consoleLog(0, "Called GetAlphaSpectrumJS()");
286 #endif
287 }
```

```

288 Napi::Env env = info.Env();
289 Matrix spectrum = this->getAlphaSpectrum();
290 Napi::Array result = Napi::Array::New(env, spectrum.numCols());
291 for(size_t i = 0; i < spectrum.numCols(); i++){
292     result.Set(i, Napi::Number::New(env, spectrum(0, i)));
293 }
294 return result;
295 }

296 // GetCriticalAlphaJS() function
297 // -----
298 Napi::Value AlphaShape3D::GetCriticalAlphaJS(const Napi::CallbackInfo& info) {
299 #ifdef DEBUG_ALPHA_SHAPE_3D
300     consoleLog(0, "Called GetCriticalAlphaJS()");
301 #endif

302 Napi::Env env = info.Env();
303 if(info.Length() < 1 || !info[0].IsString()){
304     Napi::TypeError::New(env, "String expected").ThrowAsJavaScriptException();
305     return env.Null();
306 }
307 std::string type = info[0].As<Napi::String>().Utf8Value();
308 double result = this->getCriticalAlpha(type);
309 return Napi::Number::New(env, result);
310 }

311 // GetSurfaceAreaJS() function
312 // -----
313 Napi::Value AlphaShape3D::GetSurfaceAreaJS(const Napi::CallbackInfo& info) {
314 #ifdef DEBUG_ALPHA_SHAPE_3D
315     consoleLog(0, "Called GetSurfaceAreaJS()");
316 #endif

317 Napi::Env env = info.Env();
318 double result = this->getSurfaceArea();
319 return Napi::Number::New(env, result);
320 }

321 // GetVolumeJS() function
322 // -----
323 Napi::Value AlphaShape3D::GetVolumeJS(const Napi::CallbackInfo& info) {
324 #ifdef DEBUG_ALPHA_SHAPE_3D
325     consoleLog(0, "Called GetVolumeJS()");
326 #endif

327 Napi::Env env = info.Env();
328 double result = this->getVolume();
329 return Napi::Number::New(env, result);
330 }

331 // GetBoundaryFacetsJS() function
332 // -----
333 Napi::Value AlphaShape3D::GetBoundaryFacetsJS(const Napi::CallbackInfo& info)
334 {
335 #ifdef DEBUG_ALPHA_SHAPE_3D
336     consoleLog(0, "Called GetBoundaryFacetsJS()");
337 #endif
338 }
```

```

342     consoleLog(0, "Called GetBoundaryFacetsJS()");
343 #endif
344
345     Napi::Env env = info.Env();
346     Matrix facets = this->getBoundaryFacets();
347     Napi::Array result = Napi::Array::New(env, facets.numRows());
348     for(size_t i = 0; i < facets.numRows(); i++){
349         Napi::Array facet = Napi::Array::New(env, 3);
350         for(size_t j = 0; j < 3; j++){
351             facet.Set(j, Napi::Number::New(env, facets(i, j)));
352         }
353         result.Set(i, facet);
354     }
355     return result;
356 }
357
358 // WriteBoundaryFacetsJS() function
359 // -----
360 void AlphaShape3D::WriteBoundaryFacetsJS(const Napi::CallbackInfo& info) {
361 #ifdef DEBUG_ALPHA_SHAPE_3D
362     consoleLog(0, "Called WriteBoundaryFacetsJS()");
363 #endif
364
365     Napi::Env env = info.Env();
366     if(info.Length() < 1 || !info[0].IsString()){
367         Napi::TypeError::New(env, "String expected").ThrowAsJavaScriptException();
368     }
369     std::string filename = info[0].As<Napi::String>().Utf8Value();
370     this->writeBoundaryFacets(filename);
371 }
372
373 // CheckInShapeJS() function
374 // -----
375 Napi::Value AlphaShape3D::CheckInShapeJS(const Napi::CallbackInfo& info) {
376 #ifdef DEBUG_ALPHA_SHAPE_3D
377     consoleLog(0, "Called CheckInShapeJS()");
378 #endif
379
380     Napi::Env env = info.Env();
381     if(info.Length() < 1 || !info[0].IsArray()){
382         Napi::TypeError::New(env, "Array expected").ThrowAsJavaScriptException();
383         return env.Null();
384     }
385     Napi::Array jsArray = info[0].As<Napi::Array>();
386     Matrix QP(jsArray.Length(), 3);
387     for(size_t i = 0; i < jsArray.Length(); i++){
388         Napi::Array point = jsArray.Get(i).As<Napi::Array>();
389         for(size_t j = 0; j < 3; j++){
390             QP(i, j) = point.Get(j).As<Napi::Number>().DoubleValue();
391         }
392     }
393     Matrix result = this->checkInShape(QP);
394     Napi::Array jsResult = Napi::Array::New(env, result.numRows());
395     for(size_t i = 0; i < result.numRows(); i++){
396         jsResult.Set(i, Napi::Boolean::New(env, result(i, 0) != 0));

```

```

397     }
398     return jsResult;
399 }
400
401 // WriteOffJS() function
402 // -----
403 void AlphaShape3D::WriteOffJS(const Napi::CallbackInfo& info) {
404 #ifdef DEBUG_ALPHA_SHAPE_3D
405   consoleLog(0, "Called WriteOffJS()");
406 #endif
407
408   Napi::Env env = info.Env();
409   if(info.Length() < 3 || !info[0].IsString() || !info[1].IsArray() || !info[2].IsArray()){
410     Napi::TypeError::New(env, "Expected arguments: filename (string), points (array), facets (array)").ThrowAsJavaScriptException();
411   }
412
413   std::string filename = info[0].As<Napi::String>().Utf8Value();
414   Napi::Array jsPoints = info[1].As<Napi::Array>();
415   Napi::Array jsFacets = info[2].As<Napi::Array>();
416
417   Matrix Points(jsPoints.Length(), 3);
418   Matrix bf(jsFacets.Length(), 3);
419
420   for(size_t i = 0; i < jsPoints.Length(); i++){
421     Napi::Array point = jsPoints.Get(i).As<Napi::Array>();
422     for(size_t j = 0; j < 3; j++){
423       Points(i, j) = point.Get(j).As<Napi::Number>().DoubleValue();
424     }
425   }
426
427   for(size_t i = 0; i < jsFacets.Length(); i++){
428     Napi::Array facet = jsFacets.Get(i).As<Napi::Array>();
429     for(size_t j = 0; j < 3; j++){
430       bf(i, j) = facet.Get(j).As<Napi::Number>().DoubleValue();
431     }
432   }
433
434   this->writeOff(filename, Points, bf);
435 }
436
437 // GetTriangulationJS() function
438 // -----
439 Napi::Value AlphaShape3D::GetTriangulationJS(const Napi::CallbackInfo& info) {
440 #ifdef DEBUG_ALPHA_SHAPE_3D
441   consoleLog(0, "Called GetTriangulationJS()");
442 #endif
443
444   Napi::Env env = info.Env();
445   Matrix triangulation = this->getTriangulation();
446   Napi::Array result = Napi::Array::New(env, triangulation.numRows());
447   for(size_t i = 0; i < triangulation.numRows(); i++){
448     Napi::Array row = Napi::Array::New(env, 3);
449     for(size_t j = 0; j < 3; j++){

```

```

450         row . Set (j , Napi::Number::New(env , triangulation(i , j))) ;
451     }
452     result . Set (i , row) ;
453 }
454 return result ;
455 }

456 // GetNearestNeighborJS () function
457 // -----
458 Napi::Value AlphaShape3D::GetNearestNeighborJS (const Napi::CallbackInfo& info)
459 {
460 #ifdef DEBUG_ALPHA_SHAPE_3D
461     consoleLog(0 , "Called GetNearestNeighborJS ()");
462 #endif
463
464 Napi::Env env = info.Env();
465 if (info.Length() < 1 || !info[0].IsArray()){
466     Napi::TypeError::New(env , "Array expected").ThrowAsJavaScriptException();
467     return env.Null();
468 }
469 Napi::Array jsArray = info[0].As<Napi::Array>();
470 Matrix QP(jsArray.Length() , 3);
471 for (size_t i = 0; i < jsArray.Length(); i++){
472     Napi::Array point = jsArray.Get(i).As<Napi::Array>();
473     for (size_t j = 0; j < 3; j++){
474         QP(i , j) = point.Get(j).As<Napi::Number>().DoubleValue();
475     }
476 }
477 std::pair<Matrix , Matrix> result = this->getNearestNeighbor(QP);

478
479 Napi::Object jsResult = Napi::Object::New(env);
480
481 Napi::Array indices = Napi::Array::New(env , result.first.numRows());
482 Napi::Array distances = Napi::Array::New(env , result.second.numRows());
483
484 for (size_t i = 0; i < result.first.numRows(); i++){
485     indices.Set(i , Napi::Number::New(env , result.first(i , 0)));
486     distances.Set(i , Napi::Number::New(env , result.second(i , 0)));
487 }
488
489 jsResult.Set("indices" , indices);
490 jsResult.Set("distances" , distances);
491 return jsResult;
492 }

493 //
494 // GetSimplifiedShapeJS () function
495 // -----
496 Napi::Value AlphaShape3D::GetSimplifiedShapeJS (const Napi::CallbackInfo& info)
497 {
498 #ifdef DEBUG_ALPHA_SHAPE_3D
499     consoleLog(0 , "Called GetSimplifiedShapeJS ()");
500 #endif
501
502 Napi::Env env = info.Env();

```

```

503     std::pair<Matrix, Matrix> result;
504
505     if (info.Length() == 0){
506         result = this->getSimplifiedShape();
507     }
508     else if (info.Length() == 1){
509         if (info[0].IsNumber()){
510             double stop_ratio = info[0].As<Napi::Number>().DoubleValue();
511             result = this->getSimplifiedShape(stop_ratio);
512         }
513         else if (info[0].IsString()){
514             std::string filename = info[0].As<Napi::String>().Utf8Value();
515             result = this->getSimplifiedShape(filename);
516         }
517         else{
518             Napi::TypeError::New(env, "Invalid argument type").
519                 ThrowAsJavaScriptException();
520             return env.Null();
521         }
522     }
523     else if (info.Length() == 2 && info[0].IsNumber() && info[1].IsString()){
524         double stop_ratio = info[0].As<Napi::Number>().DoubleValue();
525         std::string filename = info[1].As<Napi::String>().Utf8Value();
526         result = this->getSimplifiedShape(stop_ratio, filename);
527     }
528     else{
529         Napi::TypeError::New(env, "Invalid arguments").ThrowAsJavaScriptException
530             ();
531         return env.Null();
532     }
533
534     Napi::Object jsResult = Napi::Object::New(env);
535     Napi::Array points = Napi::Array::New(env, result.first.numRows());
536     Napi::Array facets = Napi::Array::New(env, result.second.numRows());
537
538     for (size_t i = 0; i < result.first.numRows(); i++){
539         Napi::Array point = Napi::Array::New(env, 3);
540         for (size_t j = 0; j < 3; j++){
541             point.Set(j, Napi::Number::New(env, result.first(i, j)));
542         }
543         points.Set(i, point);
544     }
545
546     for (size_t i = 0; i < result.second.numRows(); i++){
547         Napi::Array facet = Napi::Array::New(env, 3);
548         for (size_t j = 0; j < 3; j++){
549             facet.Set(j, Napi::Number::New(env, result.second(i, j)));
550         }
551         facets.Set(i, facet);
552     }
553
554     jsResult.Set("points", points);
555     jsResult.Set("facets", facets);
556     return jsResult;
557 }
```

```

556
557 // RemoveUnusedPointsJS() function
558 // -----
559 Napi::Value AlphaShape3D::RemoveUnusedPointsJS(const Napi::CallbackInfo& info)
560 {
561 #ifdef DEBUG_ALPHA_SHAPE_3D
562   consoleLog(0, "Called RemoveUnusedPointsJS()");
563 #endif
564
565   Napi::Env env = info.Env();
566   if(info.Length() < 2 || !info[0].IsArray() || !info[1].IsArray()){
567     Napi::TypeError::New(env, "Two arrays expected").
568       ThrowAsJavaScriptException();
569     return env.Null();
570   }
571
572   Napi::Array jsPoints = info[0].As<Napi::Array>();
573   Napi::Array jsFacets = info[1].As<Napi::Array>();
574
575   Matrix Pi(jsPoints.Length(), 3);
576   Matrix bfi(jsFacets.Length(), 3);
577
578   for(size_t i = 0; i < jsPoints.Length(); i++){
579     Napi::Array point = jsPoints.Get(i).As<Napi::Array>();
580     for(size_t j = 0; j < 3; j++){
581       Pi(i, j) = point.Get(j).As<Napi::Number>().DoubleValue();
582     }
583
584   for(size_t i = 0; i < jsFacets.Length(); i++){
585     Napi::Array facet = jsFacets.Get(i).As<Napi::Array>();
586     for(size_t j = 0; j < 3; j++){
587       bfi(i, j) = facet.Get(j).As<Napi::Number>().DoubleValue();
588     }
589
590   std::pair<Matrix, Matrix> result = this->removeUnusedPoints(Pi, bfi);
591
592   Napi::Object jsResult = Napi::Object::New(env);
593   Napi::Array points = Napi::Array::New(env, result.first.numRows());
594   Napi::Array facets = Napi::Array::New(env, result.second.numRows());
595
596   for(size_t i = 0; i < result.first.numRows(); i++){
597     Napi::Array point = Napi::Array::New(env, 3);
598     for(size_t j = 0; j < 3; j++){
599       point.Set(j, Napi::Number::New(env, result.first(i, j)));
600     }
601     points.Set(i, point);
602   }
603
604   for(size_t i = 0; i < result.second.numRows(); i++){
605     Napi::Array facet = Napi::Array::New(env, 3);
606     for(size_t j = 0; j < 3; j++){
607       facet.Set(j, Napi::Number::New(env, result.second(i, j)));
608     }
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2497
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2629
2630
2631
2632
2633
```

```

609     facets.Set(i, facet);
610 }
611
612 jsResult.Set("points", points);
613 jsResult.Set("facets", facets);
614 return jsResult;
615 }
616
617 // getAlpha() function
618 // -----
619 double AlphaShape3D::getAlpha(void) {
620 #ifdef DEBUG_ALPHA_SHAPE_3D
621     consoleLog(0, "Called getAlpha()");
622 #endif
623
624     return this->alphaShape->get_alpha();
625 }
626
627 // setAlpha() function
628 // -----
629 void AlphaShape3D::setAlpha(double alpha) {
630 #ifdef DEBUG_ALPHA_SHAPE_3D
631     consoleLog(0, "Called setAlpha()");
632 #endif
633
634     this->surface_mesh.clear();
635     this->alphaShape->set_alpha(alpha);
636
637 #ifdef DEBUG_ALPHA_SHAPE_3D
638     std::cout << "Number of solid components for alpha " << alpha
639             << " is " << this->numRegions() << std::endl;
640     std::list<As3::Cell_handle> cells;
641     std::list<As3::Facet> facets;
642     std::list<As3::Edge> edges;
643     std::list<As3::Vertex_handle> vertices;
644     this->alphaShape->get_alpha_shape_cells(std::back_inserter(cells),
645             As3::INTERIOR);
646     this->alphaShape->get_alpha_shape_facets(std::back_inserter(facets),
647             As3::REGULAR);
648     this->alphaShape->get_alpha_shape_facets(std::back_inserter(facets),
649             As3::SINGULAR);
650     this->alphaShape->get_alpha_shape_edges(std::back_inserter(edges),
651             As3::SINGULAR);
652     this->alphaShape->get_alpha_shape_vertices(std::back_inserter(vertices),
653             As3::REGULAR);
654     std::cout << "Number of interior tetrahedra is "
655             << cells.size() << std::endl;
656     std::cout << "Number of boundary facets is "
657             << facets.size() << std::endl;
658     std::cout << "Number of singular edges is "
659             << edges.size() << std::endl;
660     std::cout << "Number of singular vertices is "
661             << vertices.size() << std::endl;
662     std::cout << "Boundary surface construction." << std::endl;
663 #endif

```

```

664
665     std::vector<As3::Facet> bfacets;
666     this->alphaShape->get_alpha_shape_facets( std::back_inserter(bfacets) ,
667         As3::REGULAR);
668
669     std::size_t nbf = bfacets.size();
670     std::vector<CGAL_Polygon> polygons;
671     CGAL_Polygon p;
672
673     for(std::size_t i = 0; i < nbf; i++){
674         if(this->alphaShape->classify(bfacets[i].first) != As3::EXTERIOR)
675             bfacets[i] = this->alphaShape->mirror_facet(bfacets[i]);
676
677         int32_t indices[3] = {
678             (bfacets[i].second + 1) % 4,
679             (bfacets[i].second + 2) % 4,
680             (bfacets[i].second + 3) % 4,
681         };
682
683         // Consistent orientation
684         if(bfacets[i].second % 2 == 0) std::swap(indices[0], indices[1]);
685
686         p.clear();
687         for(uint8_t j = 0; j < 3; j++){
688             p.push_back(bfacets[i].first->vertex(indices[j])->info());
689         }
690         polygons.push_back(p);
691     }
692
693     PMP::polygon_soup_to_polygon_mesh(this->Points,
694         polygons, this->surface_mesh);
695 }
696
697 // numRegions() function
698 // -----
699 double AlphaShape3D::numRegions(void) {
700 #ifdef DEBUG_ALPHA_SHAPE_3D
701     consoleLog(0, "Called numRegions()");
702 #endif
703
704     return this->alphaShape->number_of_solid_components();
705 }
706
707 // getAlphaSpectrum() function
708 // -----
709 Matrix AlphaShape3D::getAlphaSpectrum() {
710 #ifdef DEBUG_ALPHA_SHAPE_3D
711     consoleLog(0, "Called getAlphaSpectrum()");
712 #endif
713
714     Matrix a(1, this->numAlphaValues);
715     for(uint32_t i = 0; i < this->numAlphaValues; i++){
716         a(0, i) = this->alphaShape->get_nth_alpha(i + 1);
717     }
718     return a;

```

```
719 }
720
721 // getCriticalAlpha() function
722 // -----
723 double AlphaShape3D::getCriticalAlpha(std::string type) {
724 #ifdef DEBUG_ALPHA_SHAPE_3D
725     consoleLog(0, "Called getCriticalAlpha()");
726 #endif
727
728     if(type == "all-points"){
729         return this->alphaShape->find_alpha_solid();
730     }
731     else if(type == "one-region"){
732         return *this->alphaShape->find_optimal_alpha(1);
733     }
734     else{
735         return nan("");
736     }
737 }
738
739 // getSurfaceArea() function
740 // -----
741 double AlphaShape3D::getSurfaceArea(void) {
742 #ifdef DEBUG_ALPHA_SHAPE_3D
743     consoleLog(0, "Called getSurfaceArea()");
744 #endif
745
746     return PMP::area(this->surface_mesh);
747 }
748
749 // getVolume() function
750 // -----
751 double AlphaShape3D::getVolume(void) {
752 #ifdef DEBUG_ALPHA_SHAPE_3D
753     consoleLog(0, "Called getVolume()");
754 #endif
755
756     return PMP::volume(this->surface_mesh);
757 }
758
759 // getBoundaryFacets() function
760 // -----
761 Matrix AlphaShape3D::getBoundaryFacets(void) {
762 #ifdef DEBUG_ALPHA_SHAPE_3D
763     consoleLog(0, "Called getBoundaryFacets()");
764 #endif
765
766     Matrix bf(this->surface_mesh.number_of_faces(), 3);
767     for(Mesh::Face_index face_index : this->surface_mesh.faces()){
768         CGAL::Vertex_around_face_circulator<Mesh>
769             vcirc(this->surface_mesh.halfedge(face_index), this->surface_mesh);
770         bf(face_index.idx(), 0) = *vcirc++;
771         bf(face_index.idx(), 1) = *vcirc++;
772         bf(face_index.idx(), 2) = *vcirc++;
773     }
```

```

774     return bf ;
775 }
776
777 // getBoundaryFacets() function with filename
778 // -----
779 Matrix AlphaShape3D :: getBoundaryFacets(std :: string filename) {
780     Matrix bf = this->getBoundaryFacets();
781     this->writeOff(filename, this->inputPoints, bf);
782     return bf;
783 }
784
785 // writeBoundaryFacets() function
786 // -----
787 void AlphaShape3D :: writeBoundaryFacets(std :: string filename) {
788 #ifdef DEBUG_ALPHA_SHAPE_3D
789     consoleLog(0, "Called writeBoundaryFacets()");
790 #endif
791
792     this->writeOff(filename, this->inputPoints, this->getBoundaryFacets());
793 }
794
795 // checkInShape() function
796 // -----
797 Matrix AlphaShape3D :: checkInShape(Matrix QP) {
798 #ifdef DEBUG_ALPHA_SHAPE_3D
799     consoleLog(0, "Called checkInShape()");
800 #endif
801
802     Matrix tf(QP numRows(), 1);
803     for(uint32_t i = 0; i < QP numRows(); i++){
804         tf(i, 0) = this->alphaShape->classify(Point(QP(i, 0), QP(i, 1), QP(i, 2)));
805         ;
806     }
807     return tf;
808 }
809
810 // getTriangulation() function
811 // -----
812 Matrix AlphaShape3D :: getTriangulation(void) {
813 #ifdef DEBUG_ALPHA_SHAPE_3D
814     consoleLog(0, "Called getTriangulation()");
815 #endif
816
817     return this->triangulationMatrix;
818 }
819
820 // getNearestNeighbor() function
821 // -----
822 std :: pair<Matrix, Matrix> AlphaShape3D :: getNearestNeighbor(Matrix QP) {
823 #ifdef DEBUG_ALPHA_SHAPE_3D
824     consoleLog(0, "Called getNearestNeighbor()");
825 #endif
826
827     Matrix I(QP numRows(), 1);
828     Matrix D(QP numRows(), 1);

```

```

828
829 Mesh surface_mesh_s(this->surface_mesh);
830 PMP::remove_isolated_vertices(surface_mesh_s);
831
832 std::vector<Point> points;
833 std::vector<uint32_t> vs;
834 for(Mesh::Vertex_index i : vertices(surface_mesh_s)){
835   if(!surface_mesh_s.is_removed(i)){
836     points.push_back(surface_mesh_s.point(i));
837     vs.push_back(i);
838   }
839 }
840
841 search_map map(points);
842 Tree tree(boost::counting_iterator<std::size_t>(0),
843            boost::counting_iterator<std::size_t>(
844              vertices(surface_mesh_s).size()),
845            Tree::Splitter(), Traits(map));
846 K_neighbor_search::Distance tr_dist(map);
847
848 for(uint32_t i = 0; i < QP.numRows(); i++){
849   K_neighbor_search search(tree, Point(QP(i, 0), QP(i, 1), QP(i, 2)),
850                           1, 0, true, tr_dist);
851   I(i, 0) = vs[search.begin()>first];
852   D(i, 0) =
853     tr_dist.inverse_of_transformed_distance(search.begin()>second);
854 }
855 return std::make_pair(I, D);
856 }
857
858 // getSimplifiedShape() function with stop_ratio
859 // -----
860 std::pair<Matrix, Matrix>
861 AlphaShape3D::getSimplifiedShape(double stop_ratio) {
862 #ifdef DEBUG_ALPHA_SHAPE_3D
863   consoleLog(0, "Called getSimplifiedShape()");
864 #endif
865
866 Mesh surface_mesh_s(this->surface_mesh);
867 SMS::Edge_count_ratio_stop_predicate<Mesh> stop(stop_ratio);
868 uint32_t r = SMS::edgeCollapse(surface_mesh_s, stop);
869 PMP::remove_isolated_vertices(surface_mesh_s);
870 surface_mesh_s.collect_garbage();
871
872 #ifdef DEBUG_ALPHA_SHAPE_3D
873   std::cout << "Number of edges removed is " << r << std::endl
874             << "Number of final edges is "
875             << surface_mesh_s.number_of_edges() << std::endl;
876 #endif
877
878 Matrix Points(surface_mesh_s.number_of_vertices(), 3);
879 Matrix bf(surface_mesh_s.number_of_faces(), 3);
880
881 for(Mesh::Vertex_index vertex_index : surface_mesh_s.vertices()){
882   Point p = surface_mesh_s.point(vertex_index);

```

```

883     Points(vertex_index.idx(), 0) = p[0];
884     Points(vertex_index.idx(), 1) = p[1];
885     Points(vertex_index.idx(), 2) = p[2];
886 }
887
888 for(Mesh::Face_index face_index : surface_mesh_s.faces()){
889     CGAL::Vertex_around_face_circulator<Mesh>
890         vcirc(surface_mesh_s.halfedge(face_index), this->surface_mesh);
891     bf(face_index.idx(), 0) = *vcirc++;
892     bf(face_index.idx(), 1) = *vcirc++;
893     bf(face_index.idx(), 2) = *vcirc++;
894 }
895 return std::make_pair(Points, bf);
896 }
897
898 // getSimplifiedShape() function without parameters
899 // -----
900 std::pair<Matrix, Matrix>
901     AlphaShape3D::getSimplifiedShape() {
902     double stop_ratio = 0.05;
903     return this->getSimplifiedShape(stop_ratio);
904 }
905
906 // getSimplifiedShape() function with filename
907 // -----
908 std::pair<Matrix, Matrix>
909     AlphaShape3D::getSimplifiedShape(std::string filename) {
910     double stop_ratio = 0.05;
911     std::pair<Matrix, Matrix> ret = this->getSimplifiedShape(stop_ratio);
912     this->writeOff(filename, ret.first, ret.second);
913     return ret;
914 }
915
916 // getSimplifiedShape() function with stop_ratio and filename
917 // -----
918 std::pair<Matrix, Matrix>
919     AlphaShape3D::getSimplifiedShape(double stop_ratio,
920         std::string filename) {
921     std::pair<Matrix, Matrix> ret = this->getSimplifiedShape(stop_ratio);
922     this->writeOff(filename, ret.first, ret.second);
923     return ret;
924 }
925
926 // removeUnusedPoints() function
927 // -----
928 std::pair<Matrix, Matrix>
929     AlphaShape3D::removeUnusedPoints(Matrix Pi, Matrix bfi) {
930 #ifdef DEBUG_ALPHA_SHAPE_3D
931     consoleLog(0, "Called removeUnusedPoints()");
932 #endif
933
934     Mesh surface_mesh_s;
935     std::vector<CGAL_Polygon> polygons;
936     CGAL_Polygon p;
937     std::vector<Point> points;

```

```

938
939 #ifdef DEBUG_ALPHA_SHAPE_3D
940     std::cout << "Boundary surface reconstruction. " << std::endl;
941 #endif
942
943     uint32_t n = Pi.numRows();
944     uint32_t nbf = bfi.numRows();
945
946     for (std::size_t i = 0; i < n; i++) {
947         points.push_back(Point(Pi(i, 0), Pi(i, 1), Pi(i, 2)));
948     }
949
950     for (std::size_t i = 0; i < nbf; i++) {
951         p.clear();
952         for (uint8_t j = 0; j < 3; j++) {
953             p.push_back(bfi(i, j));
954         }
955         polygons.push_back(p);
956     }
957
958     PMP::orient_polygon_soup(points, polygons);
959     PMP::repair_polygon_soup(points, polygons);
960     PMP::polygon_soup_to_polygon_mesh(points,
961                                         polygons, surface_mesh_s);
962     surface_mesh_s.collect_garbage();
963
964     Matrix Points(surface_mesh_s.number_of_vertices(), 3);
965     Matrix bf(surface_mesh_s.number_of_faces(), 3);
966
967     for (Mesh::Vertex_index vertex_index : surface_mesh_s.vertices()) {
968         Point p = surface_mesh_s.point(vertex_index);
969         Points(vertex_index.idx(), 0) = p[0];
970         Points(vertex_index.idx(), 1) = p[1];
971         Points(vertex_index.idx(), 2) = p[2];
972     }
973
974     for (Mesh::Face_index face_index : surface_mesh_s.faces()) {
975         CGAL::Vertex_around_face_circulator<Mesh>
976             vcirc(surface_mesh_s.halfedge(face_index), this->surface_mesh);
977         bf(face_index.idx(), 0) = *vcirc++;
978         bf(face_index.idx(), 1) = *vcirc++;
979         bf(face_index.idx(), 2) = *vcirc++;
980     }
981     return std::make_pair(Points, bf);
982 }
983
984 // writeOff() function
985 // -----
986 void AlphaShape3D::writeOff(std::string filename, Matrix Points, Matrix bf) {
987 #ifdef DEBUG_ALPHA_SHAPE_3D
988     consoleLog(0, "Called writeOff()");
989 #endif
990
991     uint32_t n = Points.numRows();
992     uint32_t nbf = bf.numRows();

```

```

993
994     std::stringstream pts;
995     std::stringstream ind;
996
997     for (std::size_t i = 0; i < n; i++){
998         pts << Points(i, 0) << " " << Points(i, 1) << " " << Points(i, 2) << std::endl;
999     }
1000
1001    for (std::size_t i = 0; i < nbf; i++){
1002        ind << "3 " << (uint64_t)bf(i, 0) << " " << (uint64_t)bf(i, 1)
1003            << " " << (uint64_t)bf(i, 2) << std::endl;
1004    }
1005
1006    std::ofstream of(filename);
1007    CGAL::set_ascii_mode(of);
1008    of << "OFF" << std::endl << n << " " << nbf << " 0" << std::endl;
1009    of << pts.str();
1010    of << ind.str();
1011    of.close();
1012 }
1013
1014 Napi::Object InitAll(Napi::Env env, Napi::Object exports) {
1015     return AlphaShape3D::Init(env, exports);
1016 }
1017
1018 NODE_API_MODULE(NODE_GYP_MODULE_NAME, InitAll)
1019
1020 } // namespace alpha_shape_3d_ns

```

Listing 5 - alpha-shape-3d.cpp

```

1 // AlphaShape3D - alpha-shape-3d.h
2 // Author: Milos Petrasinovic <mpetrasinovic@prdc.rs>
3 // PR-DC, Republic of Serbia
4 // info@prdc.rs
5 // -----
6
7 #ifndef ALPHA_SHAPE_3D_H
8 #define ALPHA_SHAPE_3D_H
9
10 //#define DEBUG_ALPHA_SHAPE_3D
11 //#define DEBUG_ALPHA_SHAPE_3D_LEVEL 0
12 //#define PROFILE_ALPHA_SHAPE_3D
13
14 #include <napi.h>
15 #include <chrono>
16 #include <thread>
17 #include <Windows.h>
18 #include <ctime>
19 #include <iomanip>
20 #include <sstream>
21 #include <string>
22 #include <fstream>
23 #include <iostream>
24 #include <filesystem>

```

```

25 #include <cassert>
26 #include <list>
27 #include <vector>
28
29 #include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
30
31 #include <CGAL/Delaunay_triangulation_3.h>
32 #include <CGAL/Triangulation_vertex_base_with_info_3.h>
33 #include <CGAL/Alpha_shape_3.h>
34 #include <CGAL/Alpha_shape_cell_base_3.h>
35 #include <CGAL/Alpha_shape_vertex_base_3.h>
36
37 #include <CGAL/Surface_mesh.h>
38 #include <CGAL/Surface_mesh_simplification/edgeCollapse.h>
39 #include <CGAL/Surface_mesh_simplification/Policies/EdgeCollapse/
  Edge_count_ratio_stop_predicate.h>
40
41 #include <CGAL/Polygon_mesh_processing/repair_polygon_soup.h>
42 #include <CGAL/Polygon_mesh_processing/orient_polygon_soup.h>
43 #include <CGAL/Polygon_mesh_processing/polygon_soup_to_polygon_mesh.h>
44 #include <CGAL/Polygon_mesh_processing/measure.h>
45 #include <CGAL/Polygon_mesh_processing/repair.h>
46
47 #include <CGAL/Search_traits_3.h>
48 #include <CGAL/Search_traits_adapter.h>
49 #include <CGAL/Orthogonal_k_neighbor_search.h>
50 #include <CGAL/boost/iterator/counting_iterator.hpp>
51
52 namespace alpha_shape_3d_ns {
53
54 using namespace std;
55 using namespace std::chrono;
56
57 namespace SMS = CGAL::Surface_mesh_simplification;
58 namespace PMP = CGAL::Polygon_mesh_processing;
59
60 typedef CGAL::Exact_predicates_inexact_constructions_kernel Gt;
61
62 typedef CGAL::Triangulation_vertex_base_with_info_3<unsigned, Gt> Tvb;
63 typedef CGAL::Alpha_shape_vertex_base_3<Gt, Tvb> Vb;
64 typedef CGAL::Alpha_shape_cell_base_3<Gt> Fb;
65 typedef CGAL::Triangulation_data_structure_3<Vb, Fb> Tds;
66 typedef CGAL::Delaunay_triangulation_3<Gt, Tds, CGAL::Fast_location> Dt;
67 typedef Dt::Point Point;
68
69 class search_map {
70   const std::vector<Point>& points;
71 public:
72   typedef Point value_type;
73   typedef const value_type& reference;
74   typedef std::size_t key_type;
75   typedef boost::lvalue_property_map_tag category;
76   search_map(const std::vector<Point>& pts) : points(pts){}
77   reference operator[](key_type k) const {return points[k];}
78   friend reference get(const search_map& ppmap, key_type i)

```

```

79     {return ppmap[ i ];}
80 };
81
82 typedef CGAL::Alpha_shape_3<Dt> As3;
83 typedef CGAL::Surface_mesh<Point> Mesh;
84 typedef std::vector<std::size_t> CGAL_Polygon;
85
86 typedef CGAL::Search_traits_3<Gt> Trb;
87 typedef CGAL::Search_traits_adapter<std::size_t, search_map, Trb> Traits;
88 typedef CGAL::Orthogonal_k_neighbor_search<Traits> K_neighbor_search;
89 typedef K_neighbor_search::Tree Tree;
90
91 template <typename T>
92 class CustomMatrix {
93 private:
94     std::vector<T> data;
95     uint32_t rows;
96     uint32_t cols;
97
98 public:
99     CustomMatrix() : rows(0), cols(0) {}
100    CustomMatrix(uint32_t r, uint32_t c) : rows(r), cols(c), data(r * c) {}
101
102    T& operator()(uint32_t i, uint32_t j) {
103        return data[i * cols + j];
104    }
105
106    const T& operator()(uint32_t i, uint32_t j) const {
107        return data[i * cols + j];
108    }
109
110    uint32_t numRows() const { return rows; }
111    uint32_t numCols() const { return cols; }
112
113    void resize(uint32_t r, uint32_t c) {
114        rows = r;
115        cols = c;
116        data.resize(r * c);
117    }
118};
119 typedef CustomMatrix<double> Matrix;
120
121 class AlphaShape3D : public Napi::ObjectWrap<AlphaShape3D> {
122 public:
123     static Napi::Object Init(Napi::Env env, Napi::Object exports);
124     AlphaShape3D(const Napi::CallbackInfo& info);
125     ~AlphaShape3D();
126
127     Matrix inputPoints;
128     std::vector<Point> Points;
129     std::vector<std::pair<Point, unsigned>> Vertices;
130     double getAlpha(void);
131     void setAlpha(double);
132     double numRegions(void);
133     Matrix getAlphaSpectrum(void);

```

```

134 double getCriticalAlpha( std :: string );
135 double getSurfaceArea( void );
136 double getVolume( void );
137 Matrix getBoundaryFacets( void );
138 Matrix getBoundaryFacets( std :: string );
139 void writeBoundaryFacets( std :: string );
140 Matrix checkInShape( Matrix );
141 Matrix getTriangulation( void );
142 std :: pair<Matrix , Matrix> getNearestNeighbor( Matrix );
143 std :: pair<Matrix , Matrix> getSimplifiedShape( double );
144 std :: pair<Matrix , Matrix> getSimplifiedShape();
145 std :: pair<Matrix , Matrix> getSimplifiedShape( std :: string );
146 std :: pair<Matrix , Matrix> getSimplifiedShape( double , std :: string );
147 std :: pair<Matrix , Matrix> removeUnusedPoints( Matrix , Matrix );
148 void writeOff( std :: string , Matrix , Matrix );
149
150 // New JavaScript wrapper methods
151 void NewShapeJS( const Napi::CallbackInfo& info );
152 Napi::Value GetAlphaJS( const Napi::CallbackInfo& info );
153 void SetAlphaJS( const Napi::CallbackInfo& info );
154 Napi::Value GetNumRegionsJS( const Napi::CallbackInfo& info );
155 Napi::Value GetAlphaSpectrumJS( const Napi::CallbackInfo& info );
156 Napi::Value GetCriticalAlphaJS( const Napi::CallbackInfo& info );
157 Napi::Value GetSurfaceAreaJS( const Napi::CallbackInfo& info );
158 Napi::Value GetVolumeJS( const Napi::CallbackInfo& info );
159 Napi::Value GetBoundaryFacetsJS( const Napi::CallbackInfo& info );
160 void WriteBoundaryFacetsJS( const Napi::CallbackInfo& info );
161 Napi::Value CheckInShapeJS( const Napi::CallbackInfo& info );
162 void WriteOffJS( const Napi::CallbackInfo& info );
163 Napi::Value GetTriangulationJS( const Napi::CallbackInfo& info );
164 Napi::Value GetNearestNeighborJS( const Napi::CallbackInfo& info );
165 Napi::Value GetSimplifiedShapeJS( const Napi::CallbackInfo& info );
166 Napi::Value RemoveUnusedPointsJS( const Napi::CallbackInfo& info );
167
168 private:
169   As3 *alphaShape;
170   Dt *delaunayTriangulation;
171   Matrix triangulationMatrix;
172   std :: size_t numAlphaValues;
173   Mesh surface_mesh;
174 };
175 }
176 } // namespace alpha_shape_3d_ns
177
178 #endif // ALPHA_SHAPE_3D_H

```

Listing 6 - alpha-shape-3d.h

```

1 [
2 {
3   "target_name": "native_module",
4   "sources": [
5     "cpp/native-module.cpp"
6   ],
7   "include_dirs": [
8     "<!@(node -p \\\"require('node-addon-api').include\\\")",

```

```

9      "<(module_root_dir)/lib/eigen-3.4.0/"  

10     ],  

11     "cflags!": [  

12       "-fno-exceptions"  

13     ],  

14     "cflags_cc!": [  

15       "-fno-exceptions"  

16     ],  

17     "defines": [  

18       "NAPI_DISABLE_CPP_EXCEPTIONS"  

19     ],  

20     "msvs_settings": {  

21       "VCCLCompilerTool": {  

22         "AdditionalOptions": [  

23           "-std:c++17"  

24         ]  

25       }  

26     },  

27   },  

28   {  

29     "target_name": "alpha_shape_3d",  

30     "sources": [  

31       "cpp/alpha-shape-3d.cpp"  

32     ],  

33     "include_dirs": [  

34       "<!(node -p \\\"require('node-addon-api').include\\\")",  

35       "<(module_root_dir)/lib/cgal-6.0.1/include/",  

36       "<(module_root_dir)/lib/cgal-6.0.1/auxiliary/gmp/include",  

37       "<(module_root_dir)/lib/boost-1.86.0/"  

38     ],  

39     "libraries": [  

40       "<(module_root_dir)/lib/cgal-6.0.1/auxiliary/gmp/lib/gmp.lib",  

41       "<(module_root_dir)/lib/cgal-6.0.1/auxiliary/gmp/lib/mpfr.lib"  

42     ],  

43     "cflags!": [  

44       "-fno-exceptions"  

45     ],  

46     "cflags_cc!": [  

47       "-fno-exceptions",  

48       "-O3",  

49       "-DNDEBUG"  

50     ],  

51     "defines": [  

52       "NAPI_DISABLE_CPP_EXCEPTIONS"  

53     ],  

54     "copies": [  

55       {  

56         "destination": "<(module_root_dir)/build/Release",  

57         "files": [  

58           "<(module_root_dir)/lib/cgal-6.0.1/auxiliary/gmp/bin/gmp-10.dll",  

59           "<(module_root_dir)/lib/cgal-6.0.1/auxiliary/gmp/bin/mpfr-6.dll"  

60         ]  

61       }  

62     ],  

63     "msvs_settings": {

```

```

64   "VCCLCompilerTool": {
65     "AdditionalOptions": [
66       "-std:c++17",
67       "/GR",
68       "/EHsc"
69     ]
70   }
71 }
72 ]
73 ]

```

Listing 7 - binding.gyp

```

1 // JSLAB - native-module.cpp
2 // Author: Milos Petrasinovic <mpetrasinovic@prdc.rs>
3 // PR-DC, Republic of Serbia
4 // info@prdc.rs
5 // -----
6
7 #include "native-module.h"
8
9 namespace native_module_ns {
10
11 #ifdef PROFILE_NATIVE_MODULE
12 // Function to start the timer and return the start time
13 time_point<steady_clock> tic() {
14   return steady_clock::now();
15 }
16
17 // Function to stop the timer and return the elapsed time
18 long toc(const time_point<steady_clock>& startTime) {
19   return duration_cast<milliseconds>(steady_clock::now() - startTime).count
20   ();
21 }
22 #endif
23
24 // Function to get current time
25 std::string getCurrentTime() {
26   // get current time
27   auto now = system_clock::now();
28
29   // get number of milliseconds for the current second
30   // (remainder after division into seconds)
31   auto ms = duration_cast<milliseconds>(now.time_since_epoch()) % 1000;
32
33   // convert to std::time_t in order to convert to std::tm (broken time)
34   auto timer = system_clock::to_time_t(now);
35
36   // convert to broken time
37   std::tm bt = *std::localtime(&timer);
38
39   std::ostringstream oss;
40
41   oss << std::put_time(&bt, "%H:%M:%S"); // HH:MM:SS
42   oss << '.' << std::setfill('0') << std::setw(3) << ms.count();

```

```

43   return oss.str();
44 }
45
46 // Function to console log data
47 int consoleLog(uint8_t level, const char* format, ...) {
48 #ifdef DEBUG_NATIVE_MODULE_LEVEL
49   if(level <= DEBUG_NATIVE_MODULE_LEVEL) {
50     printf("\033[0;33m[%s NativeModule]\033[0m ", getCurrentTime().c_str());
51     va_list vl;
52     va_start(vl, format);
53     auto ret = vprintf(format, vl);
54     va_end(vl);
55     printf("\n");
56     return ret;
57   }
58 #endif
59   return 0;
60 }
61
62 // NativeModule()
63 // Object constructor
64 // -----
65 NativeModule::NativeModule(const Napi::CallbackInfo& info)
66   : Napi::ObjectWrap<NativeModule>(info) {
67 #ifdef DEBUG_NATIVE_MODULE
68   consoleLog(0, "Called constructor");
69 #endif
70 }
71
72 // ~NativeModule()
73 // Object destructor
74 // -----
75 NativeModule::~NativeModule() {
76 }
77
78 // Init() function
79 // -----
80 Napi::Object NativeModule::Init(Napi::Env env, Napi::Object exports) {
81   Napi::Function func =
82     DefineClass(env,
83       "NativeModule", {
84         InstanceMethod("roots", &NativeModule::roots),
85         InstanceMethod("cumtrapz", &NativeModule::cumtrapz),
86         InstanceMethod("trapz", &NativeModule::trapz),
87       });
88
89   Napi::FunctionReference* constructor = new Napi::FunctionReference();
90   *constructor = Napi::Persistent(func);
91   env.SetInstanceData(constructor);
92
93   exports.Set("NativeModule", func);
94   return exports;
95 }
96
97

```

```

98 // roots() function
99 // -----
100 Napi::Value NativeModule::roots(const Napi::CallbackInfo& info) {
101   Napi::Env env = info.Env();
102   const double TOLERANCE = 1e-10;
103
104   // Ensure the input is an array
105   if (!info[0].IsArray()) {
106     Napi::TypeError::New(env, "Expected an array of coefficients").
107       ThrowAsJavaScriptException();
108     return Napi::Array::New(env);
109   }
110
111   // Extract the polynomial coefficients from the input
112   Napi::Array coefficientsArray = info[0].As<Napi::Array>();
113   int degree = coefficientsArray.Length() - 1;
114
115   // Create an Eigen vector for the coefficients
116   VectorXd coefficients(degree + 1);
117   for (int i = 0; i <= degree; ++i) {
118     coefficients(i) = coefficientsArray.Get(i).As<Napi::Number>().DoubleValue();
119   }
120
121   // Create the companion matrix
122   MatrixXd companionMatrix = MatrixXd::Zero(degree, degree);
123   for (int i = 1; i < degree; ++i) {
124     companionMatrix(i, i - 1) = 1.0;
125   }
126   for (int i = 0; i < degree; ++i) {
127     companionMatrix(i, degree - 1) = -coefficients(degree - i) / coefficients(0);
128   }
129
130   // Use Eigen's eigenvalue solver to find the roots
131   EigenSolver<MatrixXd> solver(companionMatrix);
132   VectorXcd roots = solver.eigenvalues();
133
134   // Convert the result to a JavaScript array
135   Napi::Array result = Napi::Array::New(env, degree);
136   for (int i = 0; i < degree; ++i) {
137     double realPart = roots(i).real();
138     double imagPart = roots(i).imag();
139
140     if (abs(imagPart) < TOLERANCE || isnan(imagPart)) {
141       // If the imaginary part is close to zero, return as a real number
142       result[i] = Napi::Number::New(env, realPart);
143     } else {
144       // Otherwise, return as a complex number object
145       Napi::Object complexRoot = Napi::Object::New(env);
146       complexRoot.Set("real", Napi::Number::New(env, realPart));
147       complexRoot.Set("imag", Napi::Number::New(env, imagPart));
148       result[i] = complexRoot;
149     }
150   }

```

```

150     return result;
151 }
152 }
153
154 // cumtrapz() function
155 // -----
156 Napi::Value NativeModule::cumtrapz(const Napi::CallbackInfo& info) {
157   Napi::Env env = info.Env();
158
159   // Ensure at least one argument is provided
160   if(info.Length() < 1) {
161     Napi::TypeError::New(env, "cumtrapz expects at least one argument").
162       ThrowAsJavaScriptException();
163     return env.Null();
164   }
165
166   // Ensure the first argument is an array
167   if(!info[0].IsArray()) {
168     Napi::TypeError::New(env, "First argument must be an array").
169       ThrowAsJavaScriptException();
170     return env.Null();
171   }
172
173   Napi::Array yInput = info[0].As<Napi::Array>();
174   Napi::Array xInput;
175   bool hasX = info.Length() > 1;
176
177   // If x is provided, ensure it's an array
178   if(hasX) {
179     if(!info[1].IsArray()) {
180       Napi::TypeError::New(env, "Second argument must be an array").
181         ThrowAsJavaScriptException();
182     }
183     xInput = info[1].As<Napi::Array>();
184
185   // Get the length of yInput
186   uint32_t n = yInput.Length();
187
188   // If x is provided, its length must match yInput
189   if(hasX && xInput.Length() != n) {
190     Napi::RangeError::New(env, "x and y arrays must have the same length").
191       ThrowAsJavaScriptException();
192     return env.Null();
193   }
194
195   // Handle empty array
196   if(n == 0) {
197     return Napi::Array::New(env, 0);
198
199   // Initialize Eigen vectors
200   Eigen::VectorXd y(n);
201   Eigen::VectorXd x(n);

```

```

201 Eigen::VectorXd result(n);
202
203 // Load y values from JavaScript array
204 for(uint32_t i = 0; i < n; ++i) {
205   Napi::Value val = yInput.Get(i);
206   if(!val.IsNumber()) {
207     Napi::TypeError::New(env, "y array must contain only numbers").
208     ThrowAsJavaScriptException();
209   }
210   y[i] = val.As<Napi::Number>().DoubleValue();
211 }
212
213 // If x is provided, load x values; otherwise, assume uniform spacing
214 if(hasX) {
215   for(uint32_t i = 0; i < n; ++i) {
216     Napi::Value val = xInput.Get(i);
217     if (!val.IsNumber()) {
218       Napi::TypeError::New(env, "x array must contain only numbers").
219       ThrowAsJavaScriptException();
220     }
221     x[i] = val.As<Napi::Number>().DoubleValue();
222   }
223 } else {
224   // Uniform spacing: x = [0, 1, 2, ..., n-1]
225   for(uint32_t i = 0; i < n; ++i) {
226     x[i] = static_cast<double>(i);
227   }
228 }
229
230 // Initialize result: first value is always 0
231 result[0] = 0.0;
232
233 // Cumulative trapezoidal integration
234 for(uint32_t i = 1; i < n; ++i) {
235   double dx = x[i] - x[i - 1]; // Difference in x
236   double dy = 0.5 * (y[i] + y[i - 1]); // Average height (trapezoid rule)
237   result[i] = result[i - 1] + dx * dy;
238 }
239
240 // Convert Eigen vector back to JavaScript array
241 Napi::Array jsResult = Napi::Array::New(env, n);
242 for(uint32_t i = 0; i < n; ++i) {
243   jsResult.Set(i, Napi::Number::New(env, result[i]));
244 }
245
246 return jsResult;
247 }
248
249 // trapz() function
250 // -----
251 Napi::Value NativeModule::trapz(const Napi::CallbackInfo& info) {
252   Napi::Env env = info.Env();
253 }
```

```
254 // Ensure at least one argument is provided
255 if(info.Length() < 1) {
256     Napi::TypeError::New(env, "trapz expects at least one argument").
257         ThrowAsJavaScriptException();
258     return env.Null();
259 }
260 // Ensure the first argument is an array
261 if(!info[0].IsArray()) {
262     Napi::TypeError::New(env, "First argument must be an array").
263         ThrowAsJavaScriptException();
264     return env.Null();
265 }
266 Napi::Array yInput = info[0].As<Napi::Array>();
267 Napi::Array xInput;
268 bool hasX = info.Length() > 1;
269 // If x is provided, ensure it's an array
270 if(hasX) {
271     if(!info[1].IsArray()) {
272         Napi::TypeError::New(env, "Second argument must be an array").
273             ThrowAsJavaScriptException();
274         return env.Null();
275     }
276     xInput = info[1].As<Napi::Array>();
277 }
278 // Get the length of yInput
279 uint32_t n = yInput.Length();
280 // If x is provided, its length must match yInput
281 if(hasX && xInput.Length() != n) {
282     Napi::RangeError::New(env, "x and y arrays must have the same length").
283         ThrowAsJavaScriptException();
284     return env.Null();
285 }
286 // Handle cases with fewer than 2 points
287 if(n < 2) {
288     Napi::RangeError::New(env, "trapz requires at least two data points").
289         ThrowAsJavaScriptException();
290     return env.Null();
291 }
292 // Initialize Eigen vectors
293 Eigen::VectorXd y(n);
294 Eigen::VectorXd x(n);
295 // Load y values from JavaScript array
296 for(uint32_t i = 0; i < n; ++i) {
297     Napi::Value val = yInput.Get(i);
298     if(!val.IsNumber()) {
299         Napi::TypeError::New(env, "y array must contain only numbers").
300             ThrowAsJavaScriptException();
301     }
302 }
```

```

303         return env.Null();
304     }
305     y[i] = val.As<Napi::Number>().DoubleValue();
306 }
307
308 // If x is provided, load x values; otherwise, assume uniform spacing
309 if(hasX) {
310     for(uint32_t i = 0; i < n; ++i) {
311         Napi::Value val = xInput.Get(i);
312         if (!val.IsNumber()) {
313             Napi::TypeError::New(env, "x array must contain only numbers").
314                 ThrowAsJavaScriptException();
315             return env.Null();
316         }
317         x[i] = val.As<Napi::Number>().DoubleValue();
318     }
319 } else {
320     // Uniform spacing: x = [0, 1, 2, ..., n-1]
321     for(uint32_t i = 0; i < n; ++i) {
322         x[i] = static_cast<double>(i);
323     }
324
325 // Perform trapezoidal integration
326 double total = 0.0;
327 for(uint32_t i = 1; i < n; ++i) {
328     double dx = x[i] - x[i - 1];
329     double dy = 0.5 * (y[i] + y[i - 1]);
330     total += dx * dy;
331 }
332
333 return Napi::Number::New(env, total);
334 }
335
336 Napi::Object InitAll(Napi::Env env, Napi::Object exports) {
337     return NativeModule::Init(env, exports);
338 }
339
340 NODE_API_MODULE(NODE_GYP_MODULE_NAME, InitAll)
341
342 } // namespace native_module_ns

```

Listing 8 - native-module.cpp

```

1 // JSLAB - native-module.h
2 // Author: Milos Petrasinovic <mpetrasinovic@prdc.rs>
3 // PR-DC, Republic of Serbia
4 // info@prdc.rs
5 // -----
6
7 #ifndef NATIVE_MODULE_H
8 #define NATIVE_MODULE_H
9
10 // #define DEBUG_NATIVE_MODULE
11 // #define DEBUG_NATIVE_MODULE_LEVEL 0
12 // #define PROFILE_NATIVE_MODULE

```

```

13
14 #include <napi.h>
15 #include <chrono>
16 #include <thread>
17 #include <Windows.h>
18 #include <ctime>
19 #include <iomanip>
20 #include <sstream>
21 #include <string>
22 #include <fstream>
23 #include <iostream>
24 #include <filesystem>
25 #include <vector>
26 #include <complex>
27 #include <Eigen/Dense>
28
29 namespace native_module_ns {
30
31 using namespace std;
32 using namespace std::chrono;
33 using namespace Eigen;
34
35 class NativeModule : public Napi::ObjectWrap<NativeModule> {
36 public:
37     static Napi::Object Init(Napi::Env env, Napi::Object exports);
38     NativeModule(const Napi::CallbackInfo& info);
39     ~NativeModule();
40
41     Napi::Value roots(const Napi::CallbackInfo& info);
42     Napi::Value cumtrapz(const Napi::CallbackInfo& info);
43     Napi::Value trapz(const Napi::CallbackInfo& info);
44 };
45
46 } // namespace native_module_ns
47
48 #endif // NATIVE_MODULE_H

```

Listing 9 - native-module.h

4 CSS

```

1 html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre, a,
  abbr, acronym, address, big, cite, code, del, dfn, em, img, ins, kbd, q, s, samp, small,
  strike, strong, sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol, ul, li, fieldset, form,
  label, legend, table, caption, tbody, tfoot, thead, tr, th, td, article, aside, canvas
  , details, embed, figure, figcaption, footer, header, hgroup, menu, nav, output, ruby
  , section, summary, time, mark, audio, video { border: 0; font-size: 100% ;
  font: inherit; vertical-align: baseline; margin: 0; padding: 0; }
  article, aside, details, figcaption, figure, footer, header, hgroup, menu, nav, section {
  display: block; }
  body { line-height: 1; }
  ol, ul { list-style: none; }
  blockquote, q { quotes: none; }
  blockquote:before, blockquote:after, q:before, q:after { content: none; }
  table { border-collapse: collapse; border-spacing: 0; }

```

```

4     overflow: hidden;
5 }
6
7 :focus {
8     outline: 0;
9 }
10
11 div {
12     z-index: 2;
13     box-sizing: border-box;
14 }
15
16 section {
17     position: relative;
18     z-index: 2;
19 }
20
21 body {
22     font-family: 'Roboto', Arial;
23     width: 100%;
24     background: #fff;
25 }
26
27 .clear {
28     clear: both;
29 }
30
31 .float-left {
32     float: left;
33 }
34
35 .float-right {
36     float: right;
37 }
```

Listing 10 - basic.css

```

1 .json-node-root a {
2     font-weight: 400! important;
3 }
4
5 .json-node-type {
6     color: #7f7f7f;
7 }
8
9 .json-node-toggler,
10 .json-node-stub-toggler {
11     text-decoration: none;
12     color: inherit;
13 }
14
15 .json-node-children ,
16 .json-node-root {
17     padding-left: 1em;
18 }
19
```

```
20 .json-node-header mark {  
21   background-color: rgba(199, 193, 0, 0.5);  
22   padding: 0;  
23 }  
24  
25 .json-node-header mark.highlight-active {  
26   background-color: rgba(199, 103, 46, 0.5);  
27 }  
28  
29 .json-node-label {  
30   color: #111;  
31 }  
32  
33 .json-node-number .json-node-value {  
34   color: #ff8000;  
35 }  
36  
37 .json-node-string .json-node-value {  
38   color: #808080;  
39 }  
40  
41 .json-node-boolean .json-node-value {  
42   color: #0000ff;  
43 }  
44  
45 .json-node-null .json-node-value {  
46   color: #ff8000;  
47 }  
48  
49 .json-node-number .json-node-type,  
50 .json-node-string .json-node-type,  
51 .json-node-boolean .json-node-type,  
52 .json-node-undefined .json-node-type,  
53 .json-node-null .json-node-type {  
54   display: none;  
55 }  
56  
57 .json-node-accessor {  
58   position: relative;  
59 }  
60  
61 .json-node-accessor::before {  
62   position: absolute;  
63   content: ' ';  
64   font-size: 0.7em;  
65   line-height: 1.6em;  
66   right: 0.5em;  
67   top: 0.1em;  
68   transition: transform 100ms ease-out;  
69   color: #333;  
70   opacity: 0.7;  
71 }  
72  
73 .json-node-open .json-node-accessor::before {  
74   transform: rotate(90deg);
```

```

75 }
76
77 .json-node-stub-toggler .json-node-label ,
78 .json-node-collapse {
79     color: #7f7f7f;
80 }
81 .json-node-collapse {
82     font-size: 0.8em;
83 }
84
85 @keyframes json-node-children-open {
86     from {
87         transform: scaleY(0);
88     }
89     to {
90         transform: scaleY(1);
91     }
92 }
93
94 .json-node-link {
95     display: none;
96     padding-left: 0.5em;
97     font-size: 0.8em;
98     color: #7f7f7f;
99     text-decoration: none;
100 }
```

Listing 11 - big-json-viewer-notepadpp-theme.css

```

1 #code {
2     height: calc(100vh - 74px);
3 }
4
5 .CodeMirror {
6     height: calc(100% - 6px);
7     margin: 5px;
8     border: 1px solid #f7df1e;
9     border-radius: 3px;
10    font-size: 16px;
11    margin-top: 1px;
12 }
13
14 .CodeMirror div {
15     z-index: auto;
16 }
17
18 .CodeMirror-gutter-wrapper, .CodeMirror-cursor {
19     z-index: 4!important;
20 }
21
22 .CodeMirror-vscrollbar::-webkit-scrollbar,
23 .CodeMirror-vscrollbar::-webkit-scrollbar-button,
24 .CodeMirror-vscrollbar::-webkit-scrollbar-track,
25 .CodeMirror-vscrollbar::-webkit-scrollbar-track-piece,
26 .CodeMirror-vscrollbar::-webkit-scrollbar-thumb,
27 .CodeMirror-vscrollbar::-webkit-scrollbar-corner,
```



```
28 .CodeMirror-vscrollbar::-webkit-resizer {  
29   background: transparent;  
30 }  
31 .CodeMirror-vscrollbar {  
32 }  
33 .CodeMirror-vscrollbar::-webkit-scrollbar {  
34   width: 12px;  
35   height: 12px;  
36   -webkit-border-radius: 5px;  
37   border-radius: 5px;  
38 }  
39 .CodeMirror-vscrollbar::-webkit-scrollbar-track-piece {  
40   -webkit-border-radius: 5px;  
41   border-radius: 5px;  
42 }  
43 .CodeMirror-vscrollbar::-webkit-scrollbar-thumb {  
44   background: #ddd;  
45   border-radius: 5px;  
46   background-clip: content-box;  
47   border: 2px solid transparent;  
48 }  
49 .CodeMirror-vscrollbar::-webkit-scrollbar-button {  
50   width: 0;  
51   height: 0;  
52 }  
53 .CodeMirror-ruler {  
54   z-index: 1 !important;  
55 }  
56 .CodeMirror-foldmarker {  
57   color: #000;  
58   text-shadow: none;  
59 }  
60 .CodeMirror-lint-tooltip {  
61   background-color: #fff;  
62   border: 1px solid #f7df1e;  
63   border-radius: 3px;  
64   color: #666;  
65   font-family: Roboto;  
66   line-height: 16px;  
67 }  
68 .CodeMirror-lint-tooltip {  
69   background-color: #fff;  
70   border: 1px solid #f7df1e;  
71   border-radius: 3px;  
72   color: #666;  
73 }
```

```

83   font-family: Roboto;
84   line-height: 16px;
85 }
86
87 .CodeMirror-hint {
88   line-height: 16px;
89 }
90
91 .CodeMirror-focused .cm-matchhighlight {
92   background-color: #0f0;
93 }
94
95 .CodeMirror-search-match {
96   background: gold;
97   border-top: 1px solid orange;
98   border-bottom: 1px solid orange;
99   -moz-box-sizing: border-box;
100  box-sizing: border-box;
101  opacity: .5;
102 }
103
104 .CodeMirror-selection-highlight-scrollbar {
105  background-color: #0f0;
106  width: 13px!important;
107  z-index: 100!important;
108 }
```

Listing 12 - codemirror-custom.css

```

1 #code {
2   height: calc(100vh - 74px);
3 }
4
5 .CodeMirror {
6   height: calc(100% - 6px);
7   margin: 5px;
8   border: 1px solid #f7df1e;
9   border-radius: 3px;
10  font-size: 16px;
11  margin-top: 1px;
12 }
13
14 .CodeMirror div {
15   z-index: auto;
16 }
17
18 .CodeMirror-gutter-wrapper, .CodeMirror-cursor {
19   z-index: 4!important;
20 }
21
22 .CodeMirror-vscrollbar::-webkit-scrollbar,
23 .CodeMirror-vscrollbar::-webkit-scrollbar-button,
24 .CodeMirror-vscrollbar::-webkit-scrollbar-track,
25 .CodeMirror-vscrollbar::-webkit-scrollbar-track-piece,
26 .CodeMirror-vscrollbar::-webkit-scrollbar-thumb,
27 .CodeMirror-vscrollbar::-webkit-scrollbar-corner,
```

```
28 .CodeMirror-vscrollbar::-webkit-resizer {  
29   background: transparent;  
30 }  
31  
32 .CodeMirror-vscrollbar {  
33   z-index: 3!important;  
34 }  
35  
36 .CodeMirror-vscrollbar::-webkit-scrollbar {  
37   width: 12px;  
38   height: 12px;  
39   -webkit-border-radius: 5px;  
40   border-radius: 5px;  
41 }  
42  
43 .CodeMirror-vscrollbar::-webkit-scrollbar-track-piece {  
44   -webkit-border-radius: 5px;  
45   border-radius: 5px;  
46 }  
47  
48 .CodeMirror-vscrollbar::-webkit-scrollbar-thumb {  
49   background: #ddd;  
50   border-radius: 5px;  
51   background-clip: content-box;  
52   border: 2px solid transparent;  
53 }  
54  
55 .CodeMirror-vscrollbar::-webkit-scrollbar-button {  
56   width: 0;  
57   height: 0;  
58 }  
59  
60 .CodeMirror-ruler {  
61   z-index: 1!important;  
62 }  
63  
64 .CodeMirror-foldmarker {  
65   color: #000;  
66   text-shadow: none;  
67 }  
68  
69 .CodeMirror-lint-tooltip {  
70   background-color: #fff;  
71   border: 1px solid #f7df1e;  
72   border-radius: 3px;  
73   color: #666;  
74   font-family: Roboto;  
75   line-height: 16px;  
76 }  
77  
78 .CodeMirror-lint-tooltip {  
79   background-color: #fff;  
80   border: 1px solid #f7df1e;  
81   border-radius: 3px;  
82   color: #666;
```



```
83     font-family: Roboto;
84     line-height: 16px;
85   }
86
87   .CodeMirror-hint {
88     line-height: 16px;
89   }
90
91   .CodeMirror-focused .cm-matchhighlight {
92     background-color: #0f0;
93   }
94
95   .CodeMirror-search-match {
96     background: gold;
97     border-top: 1px solid orange;
98     border-bottom: 1px solid orange;
99     -moz-box-sizing: border-box;
100    box-sizing: border-box;
101    opacity: .5;
102  }
103
104  .CodeMirror-selection-highlight-scrollbar {
105    background-color: #0f0;
106    width: 13px!important;
107    z-index: 100!important;
108  }
109
110  .CodeMirror-selected, .CodeMirror-focused .CodeMirror-selected {
111    background-color: #c0c0c0!important;
112  }
113
114  .CodeMirror-hints::-webkit-scrollbar,
115  .CodeMirror-hints::-webkit-scrollbar-button,
116  .CodeMirror-hints::-webkit-scrollbar-track,
117  .CodeMirror-hints::-webkit-scrollbar-track-piece,
118  .CodeMirror-hints::-webkit-scrollbar-thumb,
119  .CodeMirror-hints::-webkit-scrollbar-corner,
120  .CodeMirror-hints::-webkit-resizer {
121    background: transparent;
122  }
123
124  .CodeMirror-hints::-webkit-scrollbar {
125    width: 12px;
126    height: 12px;
127    -webkit-border-radius: 5px;
128    border-radius: 5px;
129  }
130
131  .CodeMirror-hints::-webkit-scrollbar-track-piece {
132    -webkit-border-radius: 5px;
133    border-radius: 5px;
134  }
135
136  .CodeMirror-hints::-webkit-scrollbar-thumb {
137    background: #ddd;
```

```
138 border-radius: 5px;
139 background-clip: content-box;
140 border: 2px solid transparent;
141 }
142
143 .CodeMirror-hints::-webkit-scrollbar-button {
144 width:0;
145 height:0;
146 }
147
148 .CodeMirror-search-dialog {
149 font-family: Roboto;
150 font-size: 16px;
151 display: none;
152 position: absolute;
153 top: 0;
154 right: 15px;
155 background-color: rgba(255, 255, 255, 0.9);
156 padding: 10px;
157 border-radius: 0 0 5px 5px;
158 box-shadow: 0 0 10px #00000026;
159 z-index: 3!important;
160 }
161
162 .CodeMirror-search-find, .CodeMirror-search-replace {
163 margin: 0;
164 border: 1px solid #ddd;
165 padding: 3px 5px;
166 display: block;
167 font-size: 16px;
168 border-radius: 3px;
169 margin-right: 10px;
170 float: left;
171 margin-bottom: 10px;
172 font-weight: 100;
173 }
174
175 .CodeMirror-search-find::placeholder, .CodeMirror-search-replace::placeholder {
176 font-weight: 100;
177 color: #999;
178 }
179
180 .CodeMirror-search-find:focus, .CodeMirror-search-replace:focus {
181 border: 1px solid #f7df1e;
182 }
183
184 .CodeMirror-search-find-prev-btn, .CodeMirror-search-find-next-btn,
185 .CodeMirror-search-replace-btn, .CodeMirror-search-replace-all-btn {
186 cursor: pointer;
187 background-size: 20px;
188 height: 20px;
189 width: 20px;
190 opacity: 0.5;
191 display: block;
```

```
192     float: left;
193     margin-top: 3px;
194 }
195
196 .CodeMirror-search-find-prev-btn:hover, .CodeMirror-search-find-next-btn:hover
197 , .CodeMirror-search-replace-btn:hover, .CodeMirror-search-replace-all-btn:hover
198 {
199     opacity: 1;
200 }
201
202 .CodeMirror-search-find-prev-btn {
203     background-image: url(../img/arrow.svg);
204     transform: rotate(90deg);
205     margin-right: 10px;
206 }
207
208 .CodeMirror-search-find-next-btn {
209     background-image: url(../img/arrow.svg);
210     transform: rotate(-90deg);
211 }
212
213 .CodeMirror-search-replace-btn {
214     background-image: url(../img/replace.svg);
215     margin-right: 10px;
216 }
217
218 .CodeMirror-search-replace-all-btn {
219     background-image: url(../img/replace-all.svg);
220 }
221
222 .CodeMirror-search-case-btn, .CodeMirror-search-regex-btn {
223     cursor: pointer;
224     background-size: 20px;
225     height: 20px;
226     width: 20px;
227     opacity: 0.5;
228     display: block;
229     float: left;
230 }
231
232 .CodeMirror-search-case-btn:hover, .CodeMirror-search-regex-btn:hover {
233     opacity: 0.7;
234 }
235
236 .CodeMirror-search-case-btn.active, .CodeMirror-search-regex-btn.active {
237     opacity: 1;
238 }
239
240 .CodeMirror-search-case-btn {
241     margin-left: 5px;
242     margin-right: 10px;
243     background-image: url(../img/match-case.svg);
244 }
```

```
245 .CodeMirror-search-regex-btn {  
246   background-image: url(../img/regex.svg);  
247 }  
248  
249 .CodeMirror-search-bottom-right {  
250   float: right;  
251   margin-top: 5px;  
252   font-weight: 100;  
253   color: #999;  
254 }  
255  
256 .CodeMirror-search-close-btn {  
257   cursor: pointer;  
258   background-size: 15px;  
259   height: 15px;  
260   width: 15px;  
261   opacity: 0.4;  
262   display: block;  
263   float: right;  
264   background-image: url(../img/close.svg);  
265   margin-left: 10px;  
266 }  
267  
268 .CodeMirror-search-close-btn:hover {  
269   opacity: 1;  
270 }
```

Listing 13 - codemirror-editor-custom.css

```
1 .CodeMirror {  
2   height: auto;  
3   font-size: 16px;  
4   font-family: 'Roboto', Arial;  
5 }  
6  
7 .CodeMirror div {  
8   z-index: auto;  
9 }  
10  
11 .CodeMirror-gutter-wrapper, .CodeMirror-cursor {  
12   z-index: 4! important;  
13 }  
14  
15 .CodeMirror-lint-tooltip {  
16   background-color: #fff;  
17   border: 1px solid #f7df1e;  
18   border-radius: 3px;  
19   color: #666;  
20   font-family: Roboto;  
21   line-height: 16px;  
22 }  
23  
24 .CodeMirror-lint-tooltip {  
25   background-color: #fff;  
26   border: 1px solid #f7df1e;  
27   border-radius: 3px;
```



```
28     color: #666;
29     font-family: 'Roboto', Arial;
30     line-height: 16px;
31   }
32
33 .CodeMirror-hint {
34   font-family: 'Roboto', Arial;
35   line-height: 16px;
36 }
37
38 .CodeMirror-focused .cm-matchhighlight {
39   background-color: #0f0;
40 }
41
42 .CodeMirror-search-match {
43   background: gold;
44   border-top: 1px solid orange;
45   border-bottom: 1px solid orange;
46   -moz-box-sizing: border-box;
47   box-sizing: border-box;
48   opacity: .5;
49 }
50
51 .CodeMirror-gutters {
52   border-right: none;
53   background-color: #fff;
54 }
55
56 .CodeMirror-selected, .CodeMirror-focused .CodeMirror-selected {
57   background-color: #c0c0c0 !important;
58 }
59
60 .CodeMirror-hints::-webkit-scrollbar,
61 .CodeMirror-hints::-webkit-scrollbar-button,
62 .CodeMirror-hints::-webkit-scrollbar-track,
63 .CodeMirror-hints::-webkit-scrollbar-track-piece,
64 .CodeMirror-hints::-webkit-scrollbar-thumb,
65 .CodeMirror-hints::-webkit-scrollbar-corner,
66 .CodeMirror-hints::-webkit-resizer {
67   background: transparent;
68 }
69
70 .CodeMirror-hints::-webkit-scrollbar {
71   width: 12px;
72   height: 12px;
73   -webkit-border-radius: 5px;
74   border-radius: 5px;
75 }
76
77 .CodeMirror-hints::-webkit-scrollbar-track-piece {
78   -webkit-border-radius: 5px;
79   border-radius: 5px;
80 }
81
82 .CodeMirror-hints::-webkit-scrollbar-thumb {
```

```
83     background: #ddd;
84     border-radius: 5px;
85     background-clip: content-box;
86     border: 2px solid transparent;
87 }
88
89 .CodeMirror-hints::-webkit-scrollbar-button {
90   width:0;
91   height:0;
92 }
```

Listing 14 - codemirror-main-custom.css

```
1  .cm-s-notepadpp span.cm-comment {
2    color: #008000;
3  }
4  .cm-s-notepadpp span.cm-keyword {
5    line-height: 1em;
6    font-weight: bold;
7    color: #0000ff;
8  }
9  .cm-s-notepadpp span.cm-string {
10   color: #808080;
11 }
12 .cm-s-notepadpp span.cm-builtin {
13   line-height: 1em;
14   font-weight: bold;
15   color: #804000;
16 }
17 .cm-s-notepadpp span.cm-special {
18   line-height: 1em;
19   font-weight: bold;
20   color: #0aa;
21 }
22 .cm-s-notepadpp span.cm-variable {
23   color: black;
24 }
25 .cm-s-notepadpp span.cm-number {
26   color: #ff8000;
27 }
28
29 .cm-s-notepadpp span.cm-atom {
30   color: #0000ff;
31 }
32 .cm-s-notepadpp span.cm-meta {
33   color: #555;
34 }
35 .cm-s-notepadpp span.cm-link {
36   color: #3a3;
37 }
38
39 .cm-s-notepadpp .CodeMirror-activeline-background {
40   background: #e8e8ff;
41 }
42 .cm-s-notepadpp .CodeMirror-matchingbracket {
43   outline: 1px solid grey;
```

```
44     color: black !important;  
45 }
```

Listing 15 - codemirror-notepadpp-theme.css

```
1 html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre, a,  
  abbr, acronym, address, big, cite, code, del, dfn, em, img, ins, kbd, q, s, samp, small,  
  strike, strong, sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol, ul, li, fieldset, form,  
  label, legend, table, caption, tbody, tfoot, thead, tr, th, td, article, aside, canvas,  
  details, embed, figure, figcaption, footer, header, hgroup, menu, nav, output, ruby  
  , section, summary, time, mark, audio, video { border: 0; font-size: 100%;  
  font: inherit; vertical-align: baseline; margin: 0; padding: 0 } article, aside,  
  details, figcaption, figure, footer, header, hgroup, menu, nav, section {  
  display: block } body { line-height: 1 } ol, ul { list-style: none } blockquote, q {  
  quotes: none } blockquote::before, blockquote::after, q::before, q::after {  
  content: none } table { border-collapse: collapse; border-spacing: 0 }  
2  
3 html {  
4   overflow: hidden;  
5 }  
6  
7 :focus {  
8   outline: 0;  
9 }  
10  
11 div {  
12   z-index: 2;  
13   box-sizing: border-box;  
14 }  
15  
16 section {  
17   position: relative;  
18   z-index: 2;  
19 }  
20  
21 body {  
22   font-family: 'Roboto', Arial;  
23   width: 100%;  
24   background: #fff;  
25 }  
26  
27 lang {  
28   display: none;  
29 }  
30  
31 .clear {  
32   clear: both;  
33 }  
34  
35 .float-left {  
36   float: left;  
37 }  
38  
39 .float-right {  
40   float: right;  
41 }
```

```
42
43 #app-logo {
44   height: 18px;
45   padding-top: 6px;
46   padding-left: 10px;
47   padding-right: 10px;
48   float: left;
49 }
50
51 #app-title {
52   float: right;
53   font-size: 16px;
54   padding-top: 8px;
55   font-weight: 100;
56   border-right: 1px solid #fffff2b;
57   padding-right: 10px;
58   height: 30px;
59 }
60
61 #window-controls {
62   -webkit-app-region: no-drag;
63   float: right;
64   padding-left: 6px
65 }
66
67 #window-controls li {
68   float: left;
69   padding: 1px 10px;
70   display: inline-block;
71   font-size: 18px;
72   font-weight: 100;
73   line-height: 24px;
74   -webkit-transition: all .2s linear;
75   transition: all .2s linear;
76   cursor: pointer;
77   -webkit-app-region: no-drag;
78 }
79
80 #window-controls li img {
81   width: 16px;
82   height: 16px;
83   user-select: none;
84   -webkit-user-drag: none;
85   user-drag: none;
86 }
87
88 #window-controls li:hover,
89 #window-controls li:active {
90   background: #fffff99;
91 }
92
93 #window-controls #win-close:hover,
94 #window-controls #win-close:active {
95   background: #cf000f;
96 }
```



```
97
98 #window-controls img {
99   width: 100%;
100  height: 100%;
101  filter: invert(1);
102 }
103
104 .tabs {
105   font-family: 'Roboto', Arial;
106   border-radius: 0;
107   padding-right: 65px;
108   background: #eee;
109   font-size: 16px;
110   padding-top: 5px;
111   height: 43px;
112 }
113
114 .tab-content, .tab-close {
115   cursor: pointer !important;
116 }
117
118 .tab-saved {
119   flex-grow: 0;
120   flex-shrink: 0;
121   position: relative;
122   width: 5px;
123   height: 5px;
124   border-radius: 50%;
125   background: transparent;
126   top: 5px;
127   right: 5px;
128 }
129
130 .changed .tab-saved {
131   background: #2e85c7;
132 }
133
134 #new-tab {
135   position: absolute;
136   top: 5px;
137   right: 35px;
138   height: 30px;
139   background: transparent;
140   padding: 5px;
141   border-radius: 15px;
142   transition: all .2s linear;
143   cursor: pointer;
144 }
145
146 #new-tab:hover {
147   background: #ccc;
148 }
149
150 #editor-more-icon {
151   position: absolute;
```



```
152 top: 5px;  
153 right: 5px;  
154 height: 30px;  
155 background: transparent;  
156 padding: 5px;  
157 border-radius: 15px;  
158 transition: all .2s linear;  
159 cursor: pointer;  
160 opacity: 0.7;  
161 }  
162  
163 #editor-more-icon:hover {  
164 background: #ccc;  
165 }  
166  
167 #editor-menu-container {  
168 background: #12568a;  
169 background: linear-gradient(to left , #2e85c7 , #12568a);  
170 color: #fff;  
171 user-select: none;  
172 -webkit-user-drag: none;  
173 user-drag: none;  
174 -webkit-app-region: drag;  
175 border-top: 2px solid #f7df1e;  
176 }  
177  
178 #editor-menu li {  
179 float: left ;  
180 padding: 5px 10px;  
181 display: inline-block;  
182 font-size: 16px;  
183 font-weight: 100;  
184 line-height: 20px;  
185 -webkit-transition: all .2s linear;  
186 transition: all .2s linear;  
187 cursor: pointer;  
188 -webkit-app-region: no-drag;  
189 }  
190  
191 #editor-menu li:hover {  
192 background: #fffff4 0;  
193 }  
194  
195 #editor-menu li img {  
196 float: left ;  
197 width: 20px;  
198 height: 20px;  
199 margin-right: 5px;  
200 user-select: none;  
201 -webkit-user-drag: none;  
202 user-drag: none;  
203 }  
204  
205 #close-dialog-cont {  
206 position: absolute;
```

```
207 top: 75px;  
208 left: 6px;  
209 right: 6px;  
210 bottom: 7px;  
211 background-color: rgba(255, 255, 255, 0.5);  
212 }  
213  
214 #close-dialog {  
215 margin: 0 auto;  
216 width: 400px;  
217 position: relative;  
218 background: #fff;  
219 box-shadow: 0 0 10px #00000026;  
220 border-radius: 10px;  
221 padding: 20px;  
222 padding-bottom: 20px;  
223 color: #333;  
224 margin-bottom: 20px;  
225 margin-top: 20px;  
226 line-height: 22px;  
227 }  
228  
229 #close-dialog-header {  
230 position: relative;  
231 border-bottom: 1px solid #666;  
232 margin-bottom: 25px;  
233 padding-bottom: 10px;  
234 user-select: none;  
235 -webkit-user-drag: none;  
236 user-drag: none;  
237 }  
238  
239 #close-dialog-header span {  
240 color: #666;  
241 display: block;  
242 font-weight: bold;  
243 font-size: 24px;  
244 border-radius: 3px;  
245 }  
246  
247 #close-file {  
248 font-weight: bold;  
249 }  
250  
251 #close-dialog-buttons {  
252 float: right;  
253 padding-top: 10px;  
254 }  
255  
256 #close-dialog-buttons button {  
257 color: #fff;  
258 background: #2e85c7;  
259 cursor: pointer;  
260 border: 2px solid #2e85c7;  
261 transition: background 0.2s linear, color 0.2s linear;
```



```
262 -webkit-transition: background 0.2s linear, color 0.2s linear;
263 border-radius: 3px;
264 padding: 2px 5px;
265 font-size: 16px;
266 margin: 3px 1px;
267 font-weight: normal;
268 text-decoration: none;
269 }
270
271 #close-dialog-buttons button:hover {
272 background: transparent;
273 color: #2e85c7;
274 }
275
276 #editor-more-popup {
277 position: absolute;
278 display: block;
279 border-radius: 5px;
280 width: 245px;
281 top: 81px;
282 right: 10px;
283 background: #ffffff;
284 border: 1px solid #f7df1e;
285 border-bottom: 1px solid #f7df1e;
286 }
287
288 #editor-more-popup .popup-triangle {
289 position: absolute;
290 width: 0;
291 height: 0;
292 border-style: solid;
293 border-width: 10px 10px 0 10px;
294 border-color: #f7df1e transparent transparent transparent;
295 top: -11px;
296 right: 5px;
297 transform: rotate(180deg);
298 }
299
300 #editor-more-popup li {
301 background: #12568a;
302 margin: 5px;
303 border-radius: 3px;
304 border: 2px solid #12568a;
305 color: #fff;
306 padding: 5px 10px;
307 font-weight: 100;
308 font-size: 18px;
309 cursor: pointer;
310 transition: background 0.2s linear;
311 -webkit-transition: background 0.2s linear;
312 }
313
314 #editor-more-popup li:hover {
315 background: #5187b0;
316 }
```

```

317
318 #editor-more-popup li img {
319   float: left;
320   width: 20px;
321   height: 20px;
322   margin-right: 10px;
323   user-select: none;
324   -webkit-user-drag: none;
325   user-drag: none;
326   margin-top: -1px;
327 }
328
329 @media only screen and (max-width: 830px) {
330   #editor-menu li img {
331     display: none;
332   }
333 }
334
335 @media only screen and (max-width: 670px) {
336   #editor-menu li {
337     padding: 5px;
338   }
339
340   #editor-menu li img {
341     display: block;
342     margin-right: 0px;
343   }
344
345   #editor-menu li str {
346     display: none;
347   }
348 }
```

Listing 16 - editor.css

```

1 html , body , div , span , applet , object , iframe , h1 , h2 , h3 , h4 , h5 , h6 , p , blockquote , pre , a ,
  abbr , acronym , address , big , cite , code , del , dfn , em , img , ins , kbd , q , s , samp , small ,
  strike , strong , sub , sup , tt , var , b , u , i , center , dl , dt , dd , ol , ul , li , fieldset , form ,
  label , legend , table , caption , tbody , tfoot , thead , tr , th , td , article , aside , canvas ,
  details , embed , figure , figcaption , footer , header , hgroup , menu , nav , output , ruby ,
  section , summary , time , mark , audio , video { border:0 ; font-size:100% ;
  font:inherit ; vertical-align:baseline ; margin:0 ; padding:0 } article , aside ,
  details , figcaption , figure , footer , header , hgroup , menu , nav , section {
  display:block } body { line-height:1 } ol , ul { list-style:none } blockquote , q {
  quotes:none } blockquote:before , blockquote:after , q:before , q:after {
  content:none } table { border-collapse:collapse ; border-spacing:0 }

2
3 html {
4   overflow: hidden;
5 }
6
7 :focus {
8   outline: 0;
9 }
10
11 div {
```

```
12     z-index: 2;
13     box-sizing: border-box;
14 }
15
16 section {
17   position: relative;
18   z-index: 2;
19 }
20
21 body {
22   font-family: 'Roboto', Arial;
23   width: 100%;
24   background: #fff;
25 }
26
27 .clear {
28   clear: both;
29 }
30
31 .float-left {
32   float: left;
33 }
34
35 .float-right {
36   float: right;
37 }
38
39 #figure-menu-button {
40   position: absolute;
41   left: 0;
42   right: 0;
43   top: -5px;
44   z-index: 9999;
45   transition: all .2s linear;
46   width: 20px;
47   height: 20px;
48   border-radius: 0 0 50% 50%;
49   background-color: #12568a;
50   margin: 5px;
51   cursor: pointer;
52 }
53
54 #figure-menu-button.active, #figure-menu-button.hovered {
55   margin-top: 25px;
56 }
57
58 #figure-menu-button img {
59   transition: transform .2s linear;
60   transform: rotate(180deg);
61   width: 60%;
62   display: block;
63   margin: 0 auto;
64   margin-top: 4px;
65 }
66
```

```
67 #figure-menu-button.active img, #figure-menu-button.hovered img {  
68   transform: rotate(0deg);  
69 }  
70  
71 #figure-menu-container {  
72   background: #12568a;  
73   background: linear-gradient(to left, #2e85c7, #12568a);  
74   color: #fff;  
75   user-select: none;  
76   -webkit-user-drag: none;  
77   user-drag: none;  
78   max-height: 5px;  
79   overflow: hidden;  
80   transition: all .2s linear;  
81   position: absolute;  
82   left: 0;  
83   right: 0;  
84   top: 0;  
85   z-index: 9998;  
86 }  
87  
88 #figure-menu-container.active, #figure-menu-container.hovered {  
89   max-height: 30px;  
90   border-bottom: 2px solid #ccc;  
91 }  
92  
93 #figure-menu ul {  
94   padding-left: 25px;  
95 }  
96  
97 #figure-menu li {  
98   float: left;  
99   padding: 5px;  
100  display: inline-block;  
101  font-size: 16px;  
102  font-weight: 100;  
103  line-height: 20px;  
104  -webkit-transition: all .2s linear;  
105  transition: all .2s linear;  
106  cursor: pointer;  
107 }  
108  
109 #figure-menu li:hover {  
110   background: #fffff4 0;  
111 }  
112  
113 #figure-menu li img {  
114   float: left;  
115   width: 20px;  
116   height: 20px;  
117   margin-right: 7px;  
118   user-select: none;  
119   -webkit-user-drag: none;  
120   user-drag: none;  
121 }
```

```

122
123 .plot-cont {
124   position: absolute;
125   top: 5px;
126   left: 0;
127   right: 0;
128   bottom: 0;
129   z-index: 1;
130 }
131
132 .modebar-container, .plotly-notifier {
133   display: none;
134 }
135
136 #figure-menu li .specific-2d , #figure-menu li .specific-3d {
137   display: none;
138 }
139
140 #figure-menu .figure-2d li .specific-2d {
141   display: initial;
142 }
143
144 #figure-menu .figure-3d li .specific-3d {
145   display: initial;
146 }
147
148 @media only screen and (max-width: 720px) {
149   #figure-menu li img {
150     display: none;
151   }
152 }
153
154 @media only screen and (max-width: 600px) {
155   #figure-menu li img {
156     display: block;
157     padding-bottom: 10px;
158   }
159
160 #figure-menu li {
161   padding: 5px;
162   width: 20px;
163   height: 20px;
164   overflow: hidden;
165 }
166 }
```

Listing 17 - figure.css

```

1 .hljs {
2   color: #000;
3 }
4
5 .hljs-built_in ,
6 .hljs-name ,
7 .hljs-selector-tag ,
8 .hljs-tag {
```

```
9      color: #00f;
10 }
11 .hljs-addition,
12 .hljs-attribute,
13 .hljs-section,
14 .hljs-template-tag,
15 .hljs-template-variable,
16 .hljs-type {
17   color: #a31515;
18 }
19
20 .hljs-literal {
21   color: #0000ff;
22 }
23
24 .hljs-deletion,
25 .hljs-meta,
26 .hljs-selector-attr,
27 .hljs-selector-pseudo {
28   color: #2b91af;
29 }
30 .hljs-doctag {
31   color: grey;
32 }
33 .hljs-attr {
34   color: red;
35 }
36 .hljs-bullet,
37 .hljs-link,
38 .hljs-symbol {
39   color: #00b0e8;
40 }
41 .hljs-emphasis {
42   font-style: italic;
43 }
44 .hljs-strong {
45   font-weight: 700;
46 }
47
48 .hljs-keyword {
49   color: #00f;
50   font-weight: bold;
51 }
52
53 .hljs-variable, .hljs-title {
54   color: black;
55 }
56
57 .hljs-number {
58   color: #ff8000;
59 }
60
61 .hljs-comment {
62   color: #008000;
63 }
```

```
64
65 .hljs-string ,
66 .hljs-quote {
67   color: #808080;
68 }
```

Listing 18 - highlight-notepadpp-theme.css

```
1 html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre, a,
2 abbr, acronym, address, big, cite, code, del, dfn, em, img, ins, kbd, q, s, samp, small,
3 strike, strong, sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol, ul, li, fieldset, form,
4 label, legend, table, caption, tbody, tfoot, thead, tr, th, td, article, aside, canvas,
5 details, embed, figure, figcaption, footer, header, hgroup, menu, nav, output, ruby,
6 section, summary, time, mark, audio, video { border: 0; font-size: 100%;
7 font: inherit; vertical-align: baseline; margin: 0; padding: 0 } article, aside,
8 details, figcaption, figure, footer, header, hgroup, menu, nav, section {
9 display: block } body { line-height: 1 } ol, ul { list-style: none } blockquote, q {
10 quotes: none } blockquote:before, blockquote:after, q:before, q:after {
11 content: none } table { border-collapse: collapse; border-spacing: 0 }
12
13
14
15 :focus {
16   outline: 0;
17 }
18
19 div {
20   z-index: 2;
21   box-sizing: border-box;
22 }
23
24 section {
25   position: relative;
26   z-index: 2;
27 }
28
29 body {
30   font-family: 'Roboto', Arial;
31   width: 100%;
32   background: #fff;
33 }
34
35 lang {
36   display: none;
37 }
```

```
39 .clear {
40   clear: both;
41 }
42
43 .float-left {
44   float: left;
45 }
46
47 .float-right {
48   float: right;
49 }
50
51 .monospaced {
52   font-family: 'RobotoMono';
53 }
54
55 #app-logo {
56   height: 24px;
57   padding-top: 8px;
58   padding-left: 10px;
59   float: left;
60 }
61
62 #app-title {
63   float: right;
64   font-size: 20px;
65   padding-top: 9px;
66   font-weight: 100;
67   border-right: 1px solid #fffff2b;
68   padding-right: 10px;
69   height: 40px;
70 }
71
72 #window-controls {
73   -webkit-app-region: no-drag;
74   float: right;
75   padding-left: 6px
76 }
77 #window-controls li {
78   float: left;
79   padding: 4px 10px;
80   display: inline-block;
81   font-size: 18px;
82   font-weight: 100;
83   line-height: 24px;
84   -webkit-transition: all .2s linear;
85   transition: all .2s linear;
86   cursor: pointer;
87   -webkit-app-region: no-drag;
88 }
89
90 #window-controls li img {
91   width: 16px;
92   height: 16px;
93   user-select: none;
```



```
94     -webkit-user-drag: none;  
95     user-drag: none;  
96 }  
97  
98 #window-controls li:hover ,  
99 #window-controls li:active {  
100    background: #fffff99 ;  
101 }  
102  
103 #window-controls #win-close:hover ,  
104 #window-controls #win-close:active {  
105    background: #cf000f ;  
106 }  
107  
108 #window-controls img {  
109    width: 100%;  
110    height: 100%;  
111    filter: invert(1);  
112 }  
113  
114 #main-menu-container {  
115    background: #12568a;  
116    background: linear-gradient(to left , #2e85c7 , #12568a);  
117    color: #fff ;  
118    user-select: none;  
119    -webkit-user-drag: none;  
120    user-drag: none;  
121    -webkit-app-region: drag;  
122    border-top: 2px solid #f7df1e;  
123 }  
124  
125 #company-logo {  
126    filter: invert(1);  
127    height: 30px;  
128    float: left ;  
129    padding: 5px;  
130    padding-left: 10px;  
131    user-select: none;  
132    -webkit-user-drag: none;  
133    user-drag: none;  
134    margin-left: 10px;  
135    border-left: 1px solid #00000054;  
136    border-right: 1px solid #00000054;  
137    padding-right: 10px;  
138 }  
139  
140 #main-menu {  
141    padding: 0 10px;  
142    -webkit-app-region: no-drag;  
143 }  
144  
145 #main-menu li {  
146    float: left ;  
147    padding: 8px 10px;  
148    display: inline-block;
```

```
149  font-size: 18px;
150  font-weight: 100;
151  line-height: 24px;
152  -webkit-transition: all .2s linear ;
153  transition: all .2s linear ;
154  cursor: pointer;
155  -webkit-app-region: no-drag;
156 }
157
158 #main-menu li:hover {
159   background: #fffff40 ;
160 }
161
162 #main-menu li img {
163   float: left ;
164   width: 24px;
165   height: 24px;
166   user-select: none;
167   -webkit-user-drag: none;
168   user-drag: none;
169   margin-right: 5px;
170 }
171
172 #paths-container li {
173   padding-bottom: 5px;
174   padding-top: 5px;
175   line-height: 22px;
176   padding-left: 10px;
177   padding-right: 10px;
178   margin: 0 10px;
179   position: relative;
180   white-space: pre-wrap;
181   border-bottom: 1px solid #f7df1e;
182   min-height: 22px;
183   cursor: pointer;
184   transition: all linear 0.3s;
185 }
186
187 #paths-container li.inactive {
188   color: #cf000f;
189 }
190
191 #paths-container li:after {
192   transition: all linear 0.3s;
193 }
194
195 #paths-container li:not (.no-paths):hover:after {
196   content: '';
197   position: absolute;
198   top: 3px;
199   left: 0px;
200   right: 0px;
201   bottom: 3px;
202   border: 1px solid #999;
203   border-radius: 3px;
```

```
204     background: #fffff69 ;
205     z-index: -1;
206 }
207
208 #paths-container li:last-child {
209     border-bottom: 1px solid transparent;
210 }
211
212 #paths-container img.remove-path {
213     height: 15px;
214     width: 15px;
215     position: absolute;
216     right: 5px;
217     top: 8px;
218     opacity: 0.3;
219     display:none;
220     transition: all linear 0.3s;
221 }
222
223 #paths-container li:hover img.remove-path {
224     display: block;
225 }
226
227 #paths-container li:hover img.remove-path:hover {
228     display: block;
229     opacity: 0.6;
230 }
231
232 #paths-container li.no-paths {
233     text-align: center;
234     font-size: 24px;
235 }
236
237 #folder-navigation-container {
238     padding: 5px 10px;
239     border-bottom: 1px solid #dadce0;
240     color: #5f6368;
241     user-select: none;
242     -webkit-user-drag: none;
243     user-drag: none;
244 }
245
246 #folder-navigation-container img, #folder-navigation-container i {
247     height: 18px;
248     width: 18px;
249     display: block;
250     margin: 0 auto;
251     -webkit-user-drag: none;
252 }
253
254 #folder-navigation-container #save-path i {
255     background-image: url(../img/star.svg);
256     background-size: 18px 18px;
257     background-position: center;
258     transition: all .2s linear;
```

```
259 }
260
261 #folder-navigation-container #save-path.saved i {
262   background-image: url(../img/star-filled.svg);
263 }
264
265 #folder-navigation-container i.i-next-folder {
266   background-image: url(../img/next.svg);
267   background-size: 14px 14px;
268   background-position: center;
269   width: 18px;
270   height: 14px;
271   display: inline-block;
272   margin-bottom: -2px;
273 }
274
275 #folder-navigation-container .next-folder {
276   transform: rotate(180deg);
277 }
278
279 #folder-navigation-container .up-folder {
280   transform: rotate(90deg);
281 }
282
283 #folder-navigation-container .previous-folder.disabled,
284 #folder-navigation-container .next-folder.disabled,
285 #folder-navigation-container .up-folder.disabled {
286   opacity: 0.3;
287   pointer-events: none;
288 }
289
290 #folder-navigation-container .address-line-container {
291   float: left;
292   width: calc(100% - 165px);
293   position: relative;
294 }
295
296 #folder-navigation-container .address-line {
297   width: calc(100% - 76px);
298   height: 20px;
299   background: #f1f3f4;
300   border-radius: 14px;
301   padding: 2px 35px;
302   margin-left: 3px;
303   border: 2px solid #f1f3f4;
304   -webkit-transition: all .2s linear;
305   transition: all .2s linear;
306   color: #5f6368;
307   font-family: 'Roboto', Arial;
308   font-size: 14px;
309 }
310
311 #folder-navigation-container .current-address-cont {
312   position: absolute;
313   left: 40px;
```

```
314     right: 35px;  
315     top: 2px;  
316     background: #f1f3f4;  
317     padding: 4px 0;  
318     pointer-events: none;  
319     white-space: nowrap;  
320     overflow: hidden;  
321     bottom: 2px;  
322 }  
323  
324 #folder-navigation-container .current-address-wrap {  
325     position: relative;  
326     max-width: 100%;  
327     display: inline-block;  
328 }  
329  
330 #folder-navigation-container .current-address {  
331     float: right;  
332 }  
333  
334 #folder-navigation-container .current-address {  
335     content: '';  
336     clear: both;  
337 }  
338  
339 #folder-navigation-container .address-line:focus {  
340     background: #fff;  
341     border: 2px solid #f7df1e;  
342 }  
343  
344 #folder-navigation-container .address-line:focus ~ .current-address-cont,  
345 #folder-navigation-container .address-line:focus ~ .save-icon {  
346     display: none;  
347 }  
348  
349 #folder-navigation-container .folder-icon {  
350     position: absolute;  
351     left: 14px;  
352     top: 4px;  
353     pointer-events: none;  
354 }  
355  
356 #folder-navigation-container span.folder {  
357     cursor: pointer;  
358     pointer-events: all;  
359 }  
360  
361 #folder-navigation-container .folder-address {  
362     font-size: 14px;  
363 }  
364  
365 #folder-navigation-container #save-path {  
366     position: absolute;  
367     top: 2.5px;  
368     padding: 2px;
```

```
369 right: 5px;
370 background: #fffff0 0 0 ;
371 margin: 0px 2px;
372 cursor: pointer;
373 width: 28px;
374 -webkit-transition: all .2s linear;
375 transition: all .2s linear;
376 display: block;
377 height: 22px;
378 -webkit-user-drag: none;
379 }
380
381 #folder-navigation-container #save-path:hover {
382 background: #fff ;
383 border-radius: 12px;
384 }
385
386 #folder-navigation-container .button {
387 padding: 5px;
388 margin: 0px 2px;
389 -webkit-transition: all .2s linear;
390 transition: all .2s linear;
391 cursor: pointer;
392 }
393
394 #folder-navigation-container .button:hover {
395 background: #f1f3f4 ;
396 border-radius: 14px;
397 }
398
399 #panels-container {
400 position: absolute;
401 top: 81px;
402 bottom: 25px;
403 left: 0px;
404 right: 0px;
405 }
406
407 #panels-container .horizontal-resizer {
408 width: 100%;
409 background-image: url(../img/hdrag.svg);
410 background-repeat: no-repeat;
411 background-position: center center;
412 background-size: auto 6px;
413 cursor: n-resize;
414 user-select: none;
415 -webkit-user-drag: none;
416 user-drag: none;
417 position: absolute;
418 }
419
420 #panels-container .horizontal-resizer:after {
421 content: '';
422 border-bottom: 1px solid transparent;
423 position: absolute;
```



```
424     top: 4px;  
425     left: 0px;  
426     right: 0px;  
427     pointer-events: none;  
428 }  
429  
430 #panels-container .vertical-resizer {  
431     height: 100%;  
432     background-image: url(../img/vdrag.svg);  
433     background-repeat: no-repeat;  
434     background-position: center center;  
435     background-size: 6px auto;  
436     cursor: w-resize;  
437     user-select: none;  
438     -webkit-user-drag: none;  
439     user-drag: none;  
440     position: absolute;  
441 }  
442  
443 #panels-container .vertical-resizer:after {  
444     content: '';  
445     border-right: 1px solid transparent;  
446     position: absolute;  
447     top: 0px;  
448     bottom: 0px;  
449     left: 4px;  
450     pointer-events: none;  
451 }  
452  
453 #panels-container .vertical-resizer:hover:after, #panels-container .  
454     horizontal-resizer:hover:after {  
455     border-right: 1px solid #f7df1e;  
456 }  
457  
458 #panels-container .horizontal-resizer:hover:after {  
459     border-bottom: 1px solid #f7df1e;  
460 }  
461  
462 #panels-container .panel-container {  
463     height: 100%;  
464 }  
465  
466 #panels-container .cell-padding {  
467     padding: 5px;  
468     height: 100%;  
469     width: 100%;  
470     position: relative;  
471 }  
472  
473 #left-panel {  
474     height: 100%;  
475     width: 20%;  
476     position: absolute;  
477     left: 0;  
478     top: 0;
```

```
478 }
479
480 #right-panel {
481   height: 100%;
482   width: 80%;
483   position: absolute;
484   left: 20%;
485   top: 0;
486 }
487
488 #left-top-panel, #left-middle-panel, #left-bottom-panel {
489   width: 100%;
490   height: 33.333%;
491 }
492
493 #panels-container .panel-title {
494   margin: 5px 10px;
495   margin-bottom: 0px;
496   color: #666;
497   padding-bottom: 3px;
498   border-bottom: 1px solid #f7df1e;
499   font-weight: 300;
500   font-size: 14px;
501   user-select: none;
502   -webkit-user-drag: none;
503   user-drag: none;
504   white-space: nowrap;
505   overflow: hidden;
506   text-overflow: ellipsis;
507 }
508
509 #left-middle-panel .panel-title, #left-bottom-panel .panel-title {
510   padding-right: 20px;
511 }
512
513 #right-panel .panel-title {
514   padding-right: 155px;
515 }
516
517 #panels-container .panel-container {
518   border: 1px solid #f1f3f4;
519   -webkit-transition: border .2s linear;
520   transition: border .2s linear;
521   position: relative;
522   border-radius: 3px;
523 }
524
525 #panels-container .panel-container:hover {
526   border: 1px solid #f7df1e;
527 }
528
529 #panels-container .panel {
530   overflow: auto;
531   position: absolute;
532   top: 23px;
```



```
533     left: 0px;  
534     right: 0px;  
535     bottom: 0px;  
536 }  
537  
538 #help-container, #info-container, #settings-container, #paths-container, #  
539     script-path-container {  
540     display: none;  
541     top: 79px;  
542     left: 0;  
543     right: 0;  
544     bottom: 25px;  
545     position: absolute;  
546     -webkit-backdrop-filter: blur(5px);  
547     backdrop-filter: blur(5px);  
548     background-color: rgba(255, 255, 255, 0.7);  
549 }  
550  
551 #info-container .app-logo {  
552     width: 150px;  
553     text-align: center;  
554     margin: 0 auto;  
555     display: block;  
556 }  
557  
558 #info-container .app-name {  
559     font-size: 34px;  
560     color: #000;  
561     text-align: center;  
562     padding-top: 10px;  
563     line-height: 46px;  
564 }  
565  
566 #info-container .app-version {  
567     font-size: 16px;  
568     text-align: center;  
569     font-weight: 100;  
570     line-height: 18px;  
571 }  
572  
573 #info-container .app-company {  
574     font-size: 24px;  
575     text-align: center;  
576     font-weight: 100;  
577     line-height: 30px;  
578 }  
579  
580 #info-container .company-logo {  
581     width: 100px;  
582     margin: 0 auto;  
583     display: block;  
584     padding: 10px;  
585 }  
586  
587 #info-container p {
```



```
587     line-height: 22px;
588     text-align: center;
589     font-weight: 100;
590   }
591
592 #info-container ul {
593   padding-top: 20px;
594   padding-left: 40px;
595   list-style: inside square;
596 }
597
598 #info-container li {
599   padding: 10px 0;
600 }
601
602 #info-container li a {
603   text-decoration: none;
604   color: #444;
605   font-size: 22px;
606 }
607
608 #info-container li a:hover {
609   opacity: 0.4;
610 }
611
612 #file-browser-cont {
613   margin: 0 10px;
614   margin-left: 0px;
615 }
616
617 #file-browser-cont ul {
618   margin-left: 15px;
619 }
620
621 #file-browser li {
622   position: relative;
623   background-repeat: no-repeat;
624   background-position: 18px 5px;
625   background-size: 18px 18px;
626 }
627
628 #file-browser li span {
629   font-size: 14px;
630   padding-bottom: 5px;
631   padding-top: 5px;
632   line-height: 18px;
633   padding-left: 40px;
634   padding-right: 0px;
635   user-select: none;
636   -webkit-user-drag: none;
637   user-drag: none;
638   white-space: nowrap;
639   overflow: hidden;
640   text-overflow: ellipsis;
641   min-height: 18px;
```

```
642     cursor: pointer;
643     color: #666;
644     position: relative;
645     display: block;
646 }
647
648 #file-browser li.folder {
649   background-image: url(../img/browser-folder.svg);
650 }
651
652 #file-browser li.link {
653   background-image: url(../img/browser-link.svg);
654 }
655
656 #file-browser li.file {
657   background-image: url(../img/browser-file.svg);
658 }
659
660 #file-browser li.file.js {
661   background-image: url(../img/browser-file-js.svg);
662 }
663
664 #file-browser li.file.jsl {
665   background-image: url(../img/browser-file-jsl.svg);
666 }
667
668 #file-browser li.file.json {
669   background-image: url(../img/browser-file-json.svg);
670 }
671
672 #file-browser li.i.expand {
673   cursor: pointer;
674   background: url(../img/expand.svg);
675   background-size: 12px;
676   height: 12px;
677   width: 12px;
678   display: block;
679   position: absolute;
680   left: 0px;
681   top: 7px;
682   transition: all .2s linear;
683   filter: grayscale(1);
684 }
685
686 #file-browser li.i.expand.expended {
687   transform: rotate(90deg);
688   filter: none;
689 }
690
691 #command-history > div {
692   font-size: 14px;
693   padding-bottom: 5px;
694   padding-top: 5px;
695   line-height: 18px;
696   padding-left: 10px;
```



```
697 padding-right: 10px;  
698 margin: 0 10px;  
699 position: relative;  
700 white-space: pre-wrap;  
701 border-bottom: 1px solid #eee;  
702 min-height: 18px;  
703 cursor: pointer;  
704 color: #666;  
705 word-break: break-word;  
706 }  
707  
708 #workspace {  
709   overflow-y: scroll !important;  
710   margin-left: 10px;  
711 }  
712  
713 #workspace .table .col {  
714   font-size: 14px;  
715   padding-bottom: 5px;  
716   padding-top: 5px;  
717   line-height: 18px;  
718   padding-left: 10px;  
719   padding-right: 10px;  
720   margin: 0 10px;  
721   position: relative;  
722   white-space: nowrap;  
723   overflow: hidden;  
724   text-overflow: ellipsis;  
725   border-bottom: 1px solid #eee;  
726   min-height: 18px;  
727   cursor: pointer;  
728   color: #666;  
729   margin: 0;  
730   float: left;  
731   border-bottom: none;  
732   left: 0 !important;  
733 }  
734  
735 #workspace-table-head .col {  
736   font-size: 14px;  
737   padding-bottom: 5px;  
738   padding-top: 5px;  
739   line-height: 18px;  
740   padding-left: 10px;  
741   padding-right: 10px;  
742   min-height: 18px;  
743   cursor: pointer;  
744   color: #666;  
745   float: left;  
746   border-right: 1px solid #ddd;  
747   white-space: nowrap;  
748   overflow: hidden;  
749   text-overflow: ellipsis;  
750   float: left;  
751   text-align: center;
```



```
752     margin-bottom: -1px;
753     margin-top: 3px;
754 }
755
756 #workspace .table .row {
757     border-bottom: 1px solid #eee;
758 }
759
760 #workspace .row:after, #workspace-table-head .row:after {
761     content: '';
762     clear: both;
763     display: block;
764 }
765
766 #workspace-table-head {
767     border-bottom: 1px solid #ddd;
768     margin: 0 10px;
769     position: absolute;
770     top: 23px;
771     left: 0;
772     right: 0;
773     background: #fff;
774     z-index: 3;
775     user-select: none;
776     -webkit-user-drag: none;
777     user-drag: none;
778     margin-right: 12px;
779     pointer-events: none;
780 }
781
782 #workspace .table {
783     margin-top: 3px;
784     text-align: center;
785     padding-top: 29px;
786     user-select: none;
787     -webkit-user-drag: none;
788     user-drag: none;
789 }
790
791 #workspace .vertical-resizer {
792     cursor: ew-resize;
793     position: absolute;
794     top: 0px;
795     bottom: 3px;
796     width: 5px;
797     left: calc(50% - 5px);
798     pointer-events: auto;
799     background: none;
800 }
801
802 #workspace .vertical-resizer:after {
803     content: '';
804     border-right: 1px solid transparent;
805     position: absolute;
806     top: 0px;
```



```
807     bottom: 0px;  
808     left: 4px;  
809     pointer-events: none;  
810   }  
811  
812 #workspace .vertical-resizer:hover:after {  
813   border-right: 1px solid #f7df1e;  
814 }  
815  
816 #workspace .col-1, #workspace-table-head .col-1 {  
817   width: 50%;  
818   text-align: left;  
819 }  
820  
821 #workspace .col-2, #workspace .col-3,  
822 #workspace-table-head .col-2, #workspace-table-head .col-3{  
823   width: 25%;  
824 }  
825  
826 #workspace .table .row {  
827   display: block;  
828   position: relative;  
829 }  
830  
831 #command-history > div.comment{  
832   color: #26a65b;  
833 }  
834  
835 #command-history > div:after , #workspace .table .row:before ,  
836 #file-browser li span:hover:after {  
837   transition: all linear 0.3s;  
838 }  
839  
840 #command-history > div:last-child , #workspace .row:last-child {  
841   border-bottom: 1px solid transparent;  
842 }  
843  
844 #workspace-table-head .col-3, #workspace .table .col {  
845   border-right: 1px solid transparent;  
846 }  
847  
848 #command-history > div:not (.comment):hover:after ,  
849 #workspace .table .row:hover:before ,  
850 #file-browser li span:hover:after  
851 {  
852   content: '';  
853   position: absolute;  
854   top: 3px;  
855   left: 0px;  
856   right: 0px;  
857   bottom: 3px;  
858   border: 1px solid #999;  
859   border-radius: 3px;  
860   z-index: -1;  
861   display: block;
```

```
862     clear: both;
863 }
864
865 #file-browser li span:hover:after {
866     left: 15px;
867 }
868
869 #command-history-options, #workspace-options, #file-browser-options {
870     position: absolute;
871     top: -1px;
872     right: 8px;
873 }
874
875 #command-history-options.options .options-right,
876 #workspace-options.options .options-right,
877 #file-browser-options.options .options-right {
878     float: right;
879 }
880
881 #command-history-options.options i.clear,
882 #workspace-options.options i.clear {
883     background: url(../img/clear.svg) no-repeat center;
884 }
885
886 #file-browser-options.options i.refresh {
887     background: url(../img/refresh.svg) no-repeat center;
888 }
889
890 #command-history-options.options i,
891 #workspace-options.options i,
892 #file-browser-options.options i {
893     width: 16px;
894     height: 18px;
895     display: block;
896     background-size: 16px !important;
897     float: left;
898     clear: none;
899     padding: 3px 5px;
900     opacity: 0.3;
901     user-select: none;
902     -webkit-user-drag: none;
903 }
904
905 #command-history-options.options i:hover,
906 #workspace-options.options i:hover,
907 #file-browser-options.options i:hover {
908     opacity: 1;
909     cursor: pointer;
910 }
911
912 #command-history-options.options i.active,
913 #workspace-options.options i.active,
914 #file-browser-options.options i.active {
915     opacity: 0.8;
916 }
```

```
917
918 #script-path-dialog-msg {
919   color: #333;
920   line-height: 22px;
921 }
922
923 #script-path {
924   font-weight: bold;
925 }
926
927 #script-path-dialog-buttons{
928   float: right;
929   padding-top: 10px;
930 }
931
932 #script-path-dialog-buttons button {
933   color: #fff;
934   background: #2e85c7;
935   cursor: pointer;
936   border: 2px solid #2e85c7;
937   transition: background 0.2s linear, color 0.2s linear;
938   -webkit-transition: background 0.2s linear, color 0.2s linear;
939   border-radius: 3px;
940   padding: 2px 5px;
941   font-size: 16px;
942   margin: 3px 1px;
943   font-weight: normal;
944   text-decoration: none;
945   width: auto;
946 }
947
948 #script-path-dialog-buttons button:hover {
949   background: transparent;
950   color: #2e85c7;
951 }
952
953 .page-cont {
954   margin: 0 auto;
955   width: 460px;
956   position: relative;
957   background: #fffff6b;
958   box-shadow: 0 0 10px #00000026;
959   border-radius: 10px;
960   padding: 20px;
961   padding-bottom: 20px;
962   color: #777;
963   margin-bottom: 20px;
964   margin-top: 20px;
965   display: flex;
966   flex-direction: column;
967   max-height: calc(100% - 40px);
968 }
969
970 .page-cont.wide {
971   width: 800px;
```



```
972    }
973
974  .page-panel {
975    overflow-y: auto;
976    padding: 10px;
977    padding-top: 25px;
978  }
979
980  .page-panel h1 {
981    font-size: 24px;
982    color: #000;
983    font-weight: 100;
984  }
985
986  .page-panel table {
987    width: 100%;
988    color: #333;
989    font-size: 18px;
990    text-align: left;
991    margin-top: 20px;
992  }
993
994  .page-panel table, .page-panel th, .page-panel td {
995    border: 1px solid #ccc;
996    border-collapse: collapse;
997  }
998
999  .page-panel th {
1000    font-weight: bold;
1001    padding: 10px;
1002  }
1003
1004  .page-panel td {
1005    padding: 10px;
1006  }
1007
1008  .panel::-webkit-scrollbar,
1009  .panel::-webkit-scrollbar-button,
1010  .panel::-webkit-scrollbar-track,
1011  .panel::-webkit-scrollbar-track-piece,
1012  .panel::-webkit-scrollbar-thumb,
1013  .panel::-webkit-scrollbar-corner,
1014  .panel::-webkit-resizer {
1015    background: transparent;
1016  }
1017
1018  .panel::-webkit-scrollbar {
1019    width: 12px;
1020    height: 12px;
1021    -webkit-border-radius: 5px;
1022    border-radius: 5px;
1023  }
1024
1025  .panel::-webkit-scrollbar-track-piece {
1026    -webkit-border-radius: 5px;
```

```
1027     border-radius: 5px;
1028 }
1029
1030 .panel::-webkit-scrollbar-thumb {
1031     background: #ddd;
1032     border-radius: 5px;
1033     background-clip: content-box;
1034     border: 2px solid transparent;
1035 }
1036
1037 .panel::-webkit-scrollbar-button {
1038     width: 0;
1039     height: 0;
1040 }
1041
1042 .options-cont input[type="text"], .options-cont input[type="number"], .
1043     options-cont input[type="submit"], .options-cont input[type="password"], .
1044     options-cont textarea, .options-cont select, .options-cont button,
1045     main-dialog input[type="text"], .main-dialog input[type="number"], .
1046     main-dialog input[type="submit"], .main-dialog input[type="password"], .
1047     main-dialog textarea, .main-dialog select, .main-dialog button {
1048     font-family: 'Roboto';
1049     border: 2px solid #ccc;
1050     color: #333;
1051     font-size: 18px;
1052     padding-left: 10px;
1053     padding-right: 10px;
1054     padding-top: 8px;
1055     padding-bottom: 8px;
1056     width: calc(100% - 24px);
1057     margin-bottom: 10px;
1058     transition: all 0.2s linear;
1059     -webkit-transition: all 0.2s linear;
1060     font-weight: bold;
1061     border-radius: 3px;
1062     -webkit-appearance: none;
1063     background: #fff;
1064     font-family: 'Roboto';
1065     -webkit-box-sizing: content-box;
1066     box-sizing: content-box;
1067     cursor: pointer;
1068     line-height: 22px;
1069 }
1070
1071 .options-cont textarea, .main-dialog textarea {
1072     resize: vertical;
1073 }
1074
1075 .options-cont input[type="submit"], .main-dialog input[type="submit"] {
1076     color: #fff;
1077     background: #2e85c7;
1078     background-size: 30px 30px;
1079     border: 0;
1080     margin-left: 2px;
1081     margin-bottom: 10px;
```

```
1078     cursor: pointer;
1079     text-transform: uppercase;
1080     border: 2px solid #2e85c7;
1081 }
1082
1083 .options-cont input[type="text"]:focus, .options-cont input[type="number"]:
1084   focus, .options-cont textarea:focus, .options-cont select:focus,
1085 .main-dialog input[type="text"]:focus, .main-dialog input[type="number"]:
1086   focus, .main-dialog textarea:focus, .main-dialog select:focus {
1087     border: 2px solid #2e85c7;
1088 }
1089
1090 .options-cont input[type="text"]:read-only:focus, .options-cont input[type="number"]:
1091   read-only:focus,
1092 .main-dialog input[type="text"]:read-only:focus, .main-dialog input[type="number"]:
1093   read-only:focus {
1094     border: 2px solid #ccc;
1095 }
1096
1097 .options-cont input[type="text"]:read-only:hover, .options-cont input[type="number"]:
1098   read-only:hover,
1099 .main-dialog input[type="text"]:read-only:hover, .main-dialog input[type="number"]:
1100   read-only:hover {
1101   cursor: auto;
1102 }
1103
1104 .options-cont input[type="submit"]:hover, .main-dialog input[type="submit"]:
1105   hover {
1106   color: #2e85c7;
1107   background: #fff;
1108 }
1109
1110 .options-cont input[type="submit"]:disabled, .main-dialog input[type="submit"]:
1111   disabled {
1112   background-image: linear-gradient(135deg, rgba(255, 255, 255, .15) 25%,
1113     transparent 25%, transparent 50%, rgba(255, 255, 255, .15) 50%, rgba(255, 255, 255, .15)
1114     75%, transparent 75%, transparent);
1115   background-size: 30px 30px;
1116   animation: animate-stripes 1s linear infinite;
1117   -webkit-animation: animate-stripes 1s linear infinite;
1118 }
1119
1120 @keyframes animate-stripes {
1121   0% {background-position: 0 0;} 100% {background-position: 60px 0;}
1122 }
```

```
1123
1124 .options-cont input:disabled, .options-cont textarea:disabled, .options-cont
1125   select:disabled,
1126 .main-dialog input:disabled, .main-dialog textarea:disabled, .
1127   main-dialog select:disabled {
1128     opacity: 0.5;
1129     pointer-events: none;
1130   }
1131
1132 .options-cont ::placeholder, .options-cont ::-webkit-input-placeholder,
1133 .main-dialog ::placeholder, .main-dialog ::-webkit-input-placeholder {
1134   color: #ccc;
1135   font-weight: normal;
1136   text-transform: none;
1137 }
1138
1139 .options-cont label, .main-dialog label {
1140   color: #12568a;
1141   margin-bottom: 2px;
1142   display: block;
1143   margin-left: 3px;
1144 }
1145
1146 .options-cont label span, .main-dialog label span {
1147   color: #555;
1148 }
1149
1150 .options-cont label.checkcont, .main-dialog label.checkcont {
1151   margin: 15px 0;
1152   font-size: 18px;
1153   padding-top: 3px;
1154   padding-bottom: 3px;
1155 }
1156
1157 .options-cont label.checkcont, .main-dialog label.checkcont {
1158   padding-top: 6px;
1159   color: #333;
1160 }
1161
1162 .checkcont {
1163   display: block;
1164   position: relative;
1165   padding-left: 35px;
1166   margin-bottom: 12px;
1167   cursor: pointer;
1168   font-size: 22px;
1169   -webkit-user-select: none;
1170   -moz-user-select: none;
1171   -ms-user-select: none;
1172   user-select: none;
1173   -webkit-user-drag: none;
1174   user-drag: none;
1175 }
1176
1177 .checkcont input {
```

```
1176 position: absolute;
1177 opacity: 0;
1178 cursor: pointer;
1179 height: 0;
1180 width: 0;
1181 }
1182
1183 .checkmark {
1184 position: absolute;
1185 top: 0;
1186 left: 0;
1187 height: 25px;
1188 width: 25px;
1189 background-color: #fff;
1190 border: 2px solid #ccc;
1191 border-radius: 3px;
1192 }
1193
1194 .checkcont:hover input ~ .checkmark {
1195 background-color: #ccc;
1196 }
1197
1198 .checkcont input:checked ~ .checkmark {
1199 background-color: #2e85c7;
1200 }
1201
1202 .checkmark:after {
1203 content: "";
1204 position: absolute;
1205 display: none;
1206 }
1207
1208 .checkcont input:checked ~ .checkmark:after {
1209 display: block;
1210 }
1211
1212 .checkcont .checkmark:after {
1213 left: 9px;
1214 top: 5px;
1215 width: 5px;
1216 height: 10px;
1217 border: solid white;
1218 border-width: 0 3px 3px 0;
1219 -webkit-transform: rotate(45deg);
1220 -ms-transform: rotate(45deg);
1221 transform: rotate(45deg);
1222 }
1223
1224 .float-input {
1225 display: flex;
1226 flex-flow: column-reverse;
1227 position: relative;
1228 margin-top: 5px;
1229 }
1230
```

```
1231 .float-select {  
1232   display: flex;  
1233   flex-flow: column-reverse;  
1234   position: relative;  
1235   margin-top: 5px;  
1236 }  
1237  
1238 .no-float-input {  
1239   position: relative;  
1240   margin-top: 5px;  
1241 }  
1242  
1243 label.float-label {  
1244   pointer-events: none;  
1245   position: absolute;  
1246   top: -13px;  
1247   margin-left: 12px !important;  
1248   padding: 1px 8px;  
1249   border-radius: 3px;  
1250   color: #777 !important;  
1251   font-weight: bold;  
1252   background: #eee;  
1253 }  
1254  
1255 .float-select label.float-label {  
1256   color: #777;  
1257   font-weight: bold;  
1258   background: #eee;  
1259 }  
1260  
1261 label.float-label, .float-input input, .float-input textarea {  
1262   transition: all 0.2s;  
1263   touch-action: manipulation;  
1264 }  
1265 .float-input input:placeholder-shown + label.float-label, .float-input  
    textarea:placeholder-shown + label.float-label {  
1266   cursor: text;  
1267   white-space: nowrap;  
1268   overflow: hidden;  
1269   text-overflow: ellipsis;  
1270   transform-origin: left bottom;  
1271   transform: translate(0, 1.7rem) scale(1.2);  
1272   background: none;  
1273 }  
1274  
1275 .float-input input::-webkit-input-placeholder, .float-input input::placeholder  
    , .float-input textarea::-webkit-input-placeholder, .float-input  
    textarea::placeholder {  
1276   opacity: 0;  
1277   transition: inherit;  
1278 }  
1279  
1280 .float-input input:focus::-webkit-input-placeholder, .float-input  
    textarea:focus::-webkit-input-placeholder {  
1281   opacity: 1;
```

```
1282 }
1283
1284 .float-input input:not(:placeholder-shown) + label.float-label,
1285 .float-input input:focus + label.float-label,
1286 .float-input textarea:not(:placeholder-shown) + label.float-label,
1287 .float-input textarea:focus + label.float-label,
1288 .float-input input:-webkit-autofill + label.float-label
1289 .float-input textarea:-webkit-autofill + label.float-label {
1290   transform: translate(0, 0) scale(1);
1291   cursor: pointer;
1292   background: #eee;
1293 }
1294
1295 .float-input input:not(:placeholder-shown) + label.float-label,
1296 .float-input input:focus + label.float-label,
1297 .float-input textarea:not(:placeholder-shown) + label.float-label,
1298 .float-input textarea:focus + label.float-label,
1299 .float-input input:-webkit-autofill + label.float-label,
1300 .float-input textarea:-webkit-autofill + label.float-label{
1301   color: #777!important;
1302   font-weight: bold;
1303   background: #eee;
1304 }
1305
1306 .options-cont button:not([disabled]):hover {
1307   color: #2e85c7;
1308   background: #fff ;
1309 }
1310
1311 .options-cont button {
1312   color: #fff ;
1313   background: #2e85c7;
1314   background-size: 30px 30px;
1315   border: 0;
1316   cursor: pointer;
1317   text-transform: uppercase;
1318   border: 2px solid #2e85c7;
1319   margin-top: 15px;
1320 }
1321
1322 .options-cont button:disabled {
1323   background-image: linear-gradient(135deg, rgba(255, 255, 255, .15) 25%,
1324     transparent 25%,
1325     transparent 50%, rgba(255, 255, 255, .15) 50%, rgba(255, 255, 255, .15)
1326     75%,
1327     transparent 75%, transparent);
1328   background-size: 30px 30px;
1329 }
1330
1331 #status-container {
1332   position: absolute;
1333   bottom: 0;
1334   left: 0;
1335   right: 0;
1336   background: #f1f3f4 ;
```



```
1335 padding: 5px 20px;  
1336 font-size: 14px;  
1337 line-height: 14px;  
1338 color: #5f6368;  
1339 }  
1340  
1341 #status {  
1342 height: 14px;  
1343 overflow: hidden;  
1344 float: left;  
1345 }  
1346  
1347 #status-icons {  
1348 float: right;  
1349 }  
1350  
1351 #status-icons img {  
1352 height: 14px;  
1353 width: 14px;  
1354 display: block;  
1355 float: left;  
1356 -webkit-user-select: none;  
1357 user-select: none;  
1358 -webkit-user-drag: none;  
1359 user-drag: none;  
1360 cursor: pointer;  
1361 }  
1362  
1363 #status-icons img:hover {  
1364 opacity: 0.5;  
1365 }  
1366  
1367 #status-icons #help-icon {  
1368 opacity: 0.4;  
1369 }  
1370  
1371 #status-icons #help-icon:hover {  
1372 opacity: 0.2;  
1373 }  
1374  
1375 #sandbox-stats-icon {  
1376 height: 14px;  
1377 overflow: hidden;  
1378 float: left;  
1379 width: 14px;  
1380 background: #999;  
1381 border-radius: 50%;  
1382 margin-right: 10px;  
1383 margin-top: -1px;  
1384 cursor: pointer;  
1385 }  
1386  
1387 #sandbox-stats-icon.ready {  
1388 background: #26a65b;  
1389 }
```



```
1390
1391 #sandbox-stats-icon.async-busy {
1392   background: #ffb40c;
1393 }
1394
1395 #sandbox-stats-icon.busy {
1396   background: #cf000f;
1397 }
1398
1399 #sandbox-stats-icon:active {
1400   opacity: 0.7;
1401 }
1402
1403 #sandbox-stats-popup {
1404   position: absolute;
1405   display: block;
1406   border-radius: 5px;
1407   width: 245px;
1408   bottom: 35px;
1409   left: 10px;
1410   background: #ffffff;
1411   border: 1px solid #f7df1e;
1412   border-bottom: 1px solid #f7df1e;
1413 }
1414
1415 #sandbox-stats-popup .popup-triangle {
1416   position: absolute;
1417   width: 0;
1418   height: 0;
1419   border-style: solid;
1420   border-width: 10px 10px 0 10px;
1421   border-color: #f7df1e transparent transparent transparent;
1422   margin-left: 6px;
1423   bottom: -11px;
1424 }
1425
1426 #sandbox-stats-header {
1427   margin: 5px 10px;
1428   margin-bottom: 0px;
1429   color: #666;
1430   padding-bottom: 3px;
1431   border-bottom: 1px solid #f7df1e;
1432   font-weight: 300;
1433   font-size: 14px;
1434   user-select: none;
1435   -webkit-user-drag: none;
1436   user-drag: none;
1437   white-space: nowrap;
1438   overflow: hidden;
1439   text-overflow: ellipsis;
1440 }
1441
1442 .sandbox-stats-cont {
1443   position: relative;
1444   padding: 8px 10px;
```

```

1445     padding-right: 100px;
1446     color: #666;
1447 }
1448
1449 .sandbox-stats-val {
1450   position: absolute;
1451   right: 10px;
1452   top: 5px;
1453   width: 45px;
1454   text-align: center;
1455   background: #999;
1456   color: #fff;
1457   padding: 3px;
1458   border-radius: 3px;
1459 }
1460
1461 #command-window-messages span.eval-code {
1462   font-weight: bold;
1463   text-decoration: underline;
1464   cursor: pointer;
1465 }
1466
1467 @media only screen and (max-width: 900px) {
1468   .page-cont.wide {
1469     width: 460px;
1470   }
1471
1472 #main-menu li img {
1473   display: none;
1474 }
1475 }
1476
1477 @media only screen and (max-width: 780px) {
1478   #main-menu li img {
1479     display: block;
1480     margin-right: 0;
1481   }
1482
1483 #main-menu li str {
1484   display: none;
1485 }
1486 }
```

Listing 19 - main.css

```

1 .cm-s-notepadpp span.cm-comment { color: #008000; }
2 .cm-s-notepadpp span.cm-keyword { line-height: 1em; font-weight: bold; color:
  #0000ff; }
3 .cm-s-notepadpp span.cm-string { color: #808080; }
4 .cm-s-notepadpp span.cm-builtin { line-height: 1em; font-weight: bold; color:
  #804000; }
5 .cm-s-notepadpp span.cm-special { line-height: 1em; font-weight: bold; color:
  #0aa; }
6 .cm-s-notepadpp span.cm-variable { color: black; }
7 .cm-s-notepadpp span.cm-number, .cm-s-notepadpp span.cm-atom { color: #ff8000;
  }
```

```

8 .cm-s-notepadapp span.cm-meta { color: #555; }
9 .cm-s-notepadapp span.cm-link { color: #3a3; }
10
11 .cm-s-notepadapp .CodeMirror-activeline-background { background: #e8e8ff; }
12 .cm-s-notepadapp .CodeMirror-matchingbracket { outline:1px solid grey;
13   color:black !important; }
```

Listing 20 - notepadapp-theme.css

```

1 .tabs {
2   box-sizing: border-box;
3   position: relative;
4   font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto,
5     Helvetica, Arial, sans-serif, "Apple Color Emoji", "Segoe UI Emoji", "
6     Segoe UI Symbol";
7   font-size: 12px;
8   height: 46px;
9   padding: 8px 3px 4px 3px;
10  background: #dee1e6;
11  border-radius: 5px 5px 0 0;
12  overflow: hidden;
13 }
14 .tabs * {
15   box-sizing: inherit;
16   font: inherit;
17 }
18 .tabs .tabs-content {
19   position: relative;
20   width: 100%;
21   height: 100%;
22 }
23 .tabs .tab {
24   position: absolute;
25   left: 0;
26   height: 36px;
27   width: 240px;
28   border: 0;
29   margin: 0;
30   z-index: 1;
31   pointer-events: none;
32 }
33 .tabs .tab,
34 .tabs .tab * {
35   user-select: none;
36   cursor: pointer;
37 }
38 .tabs .tab .tab-dividers {
39   position: absolute;
40   top: 7px;
41   bottom: 7px;
42   left: var(--tab-content-margin);
43   right: var(--tab-content-margin);
44 }
45 .tabs .tab .tab-dividers,
46 .tabs .tab .tab-dividers::before,
47 .tabs .tab .tab-dividers::after {
```

```
46     pointer-events: none;
47 }
48 .tabs .tab .tab-dividers::before ,
49 .tabs .tab .tab-dividers::after {
50   content: "";
51   display: block;
52   position: absolute;
53   top: 0;
54   bottom: 0;
55   width: 1px;
56   background: #a9adb0;
57   opacity: 1;
58   transition: opacity 0.2s ease;
59 }
60 .tabs .tab .tab-dividers::before {
61   left: 0;
62 }
63 .tabs .tab .tab-dividers::after {
64   right: 0;
65 }
66 .tabs .tab:first-child .tab-dividers::before ,
67 .tabs .tab:last-child .tab-dividers::after {
68   opacity: 0;
69 }
70 .tabs .tab .tab-background {
71   position: absolute;
72   top: 0;
73   left: 0;
74   width: 100%;
75   height: 100%;
76   overflow: hidden;
77   pointer-events: none;
78 }
79 .tabs .tab .tab-background > svg {
80   width: 100%;
81   height: 100%;
82 }
83 .tabs .tab .tab-background > svg .tab-geometry {
84   fill: #f4f5f6 ;
85 }
86 .tabs .tab[active] {
87   z-index: 5;
88 }
89 .tabs .tab[active] .tab-background > svg .tab-geometry {
90   fill: #fff ;
91 }
92 .tabs .tab:not([active]) .tab-background {
93   transition: opacity 0.2s ease;
94   opacity: 0;
95 }
96 @media (hover: hover) {
97   .tabs .tab:not([active]):hover {
98     z-index: 2;
99   }
100  .tabs .tab:not([active]):hover .tab-background {
```



```
101     opacity: 1;
102 }
103 }
104 .tabs .tab .tab-was-just-added {
105   top: 10px;
106   animation: tab-was-just-added 120ms forwards ease-in-out;
107 }
108 .tabs .tab .tab-content {
109   position: absolute;
110   display: flex;
111   top: 0;
112   bottom: 0;
113   left: var(--tab-content-margin);
114   right: var(--tab-content-margin);
115   padding: 9px 8px;
116   border-top-left-radius: 8px;
117   border-top-right-radius: 8px;
118   overflow: hidden;
119   pointer-events: all;
120 }
121 .tabs .tab[is-mini] .tab-content {
122   padding-left: 2px;
123   padding-right: 2px;
124 }
125 .tabs .tab .tab-favicon {
126   position: relative;
127   flex-shrink: 0;
128   flex-grow: 0;
129   height: 16px;
130   width: 16px;
131   background-size: 16px;
132   margin-left: 4px;
133 }
134 .tabs .tab[is-small] .tab-favicon {
135   margin-left: 0;
136 }
137 .tabs .tab[is-mini]:not([active]) .tab-favicon {
138   margin-left: auto;
139   margin-right: auto;
140 }
141 .tabs .tab[is-mini][active] .tab-favicon {
142   display: none;
143 }
144 .tabs .tab .tab-title {
145   flex: 1;
146   vertical-align: top;
147   overflow: hidden;
148   white-space: nowrap;
149   margin-left: 4px;
150   color: #5f6368;
151   -webkit-mask-image: linear-gradient(90deg, #000 0%, #000 calc(100% - 24px),
152                                     transparent);
153   mask-image: linear-gradient(90deg, #000 0%, #000 calc(100% - 24px),
154                                     transparent);
155 }
```



```
154 .tabs .tab[ is-small] .tab-title {  
155   margin-left: 0;  
156 }  
157 .tabs .tab .tab-favicon + .tab-title ,  
158 .tabs .tab[ is-small] .tab-favicon + .tab-title {  
159   margin-left: 8px;  
160 }  
161 .tabs .tab[ is-smaller] .tab-favicon + .tab-title ,  
162 .tabs .tab[ is-mini] .tab-title {  
163   display: none;  
164 }  
165 .tabs .tab[ active] .tab-title {  
166   color: #45474a;  
167 }  
168 .tabs .tab .tab-drag-handle {  
169   position: absolute;  
170   top: 0;  
171   bottom: 0;  
172   right: 0;  
173   left: 0;  
174   border-top-left-radius: 8px;  
175   border-top-right-radius: 8px;  
176 }  
177 .tabs .tab .tab-close {  
178   flex-grow: 0;  
179   flex-shrink: 0;  
180   position: relative;  
181   width: 16px;  
182   height: 16px;  
183   border-radius: 50%;  
184   background-image: url("data:image/svg+xml;utf8,<svg xmlns='http://www.w3.org  
/2000/svg' viewBox='0 0 8 8'><path stroke='rgba(0, 0, 0, .65)'  
stroke-linecap='square' stroke-width='1.5' d='M0 0 L8 8 M8 0 L0 8'></  
path></svg>");  
185   background-position: center center;  
186   background-repeat: no-repeat;  
187   background-size: 8px 8px;  
188 }  
189 @media (hover: hover) {  
190   .tabs .tab .tab-close:hover {  
191     background-color: #e8eaed ;  
192   }  
193   .tabs .tab .tab-close:hover:active {  
194     background-color: #dadce0 ;  
195   }  
196 }  
197 @media not all and (hover: hover) {  
198   .tabs .tab .tab-close:active {  
199     background-color: #dadce0 ;  
200   }  
201 }  
202 @media (hover: hover) {  
203   .tabs .tab:not([ active]) .tab-close:not(:hover):not(:active) {  
204     opacity: 0.8;  
205   }
```

```
206 }
207 .tabs .tab[is-smaller] .tab-close {
208   margin-left: auto;
209 }
210 .tabs .tab[is-mini]:not([active]) .tab-close {
211   display: none;
212 }
213 .tabs .tab[is-mini][active] .tab-close {
214   margin-left: auto;
215   margin-right: auto;
216 }
217 @-moz-keyframes tab-was-just-added {
218   to {
219     top: 0;
220   }
221 }
222 @-webkit-keyframes tab-was-just-added {
223   to {
224     top: 0;
225   }
226 }
227 @-o-keyframes tab-was-just-added {
228   to {
229     top: 0;
230   }
231 }
232 @keyframes tab-was-just-added {
233   to {
234     top: 0;
235   }
236 }
237 .tabs.tabs-is-sorting .tab:not(.tab-is-dragging),
238 .tabs:not(.tabs-is-sorting) .tab.tab-was-just-dragged {
239   transition: transform 120ms ease-in-out;
240 }
241 .tabs .tabs-bottom-bar {
242   position: absolute;
243   bottom: 0;
244   height: 4px;
245   left: 0;
246   width: 100%;
247   background: #fff;
248   z-index: 10;
249 }
250 .tabs-optional-shadow-below-bottom-bar {
251   position: relative;
252   height: 1px;
253   width: 100%;
254   background-image: url("data:image/svg+xml;utf8,<svg xmlns='http://www.w3.org
255   /2000/svg' width='1' height='1' viewBox='0 0 1 1'><rect x='0' y='0'
256   width='1' height='1' fill='rgba(0, 0, 0, .17)'></rect></svg>");
257   background-size: 1px 1px;
258   background-repeat: repeat-x;
259   background-position: 0% 0%;
```

```

259 @media only screen and (-webkit-min-device-pixel-ratio: 2), only screen and (
260   min--moz-device-pixel-ratio: 2), only screen and (-
261   o-min-device-pixel-ratio: 2/1), only screen and (min-device-pixel-ratio:
262   2), only screen and (min-resolution: 192dpi), only screen and (
263   min-resolution: 2dppx) {
264   .tabs-optional-shadow-below-bottom-bar {
265     background-image: url("data:image/svg+xml;utf8,<svg xmlns='http://www.w3.
266     org/2000/svg' width='2' height='2' viewBox='0 0 2 2'><rect x='0' y='0'
267       width='2' height='1' fill='rgba(0, 0, 0, .27)'></rect></svg>");
268   }
269 }
```

Listing 21 - tabs.css

```

1 #command-window-options {
2   position: absolute;
3   top: -1px;
4   right: 8px;
5 }
6
7 #command-window-options.options .options-right {
8   float: right;
9 }
10
11 #command-window-options.options i.settings {
12   background: url(..../img/settings.svg) no-repeat center;
13 }
14
15 #command-window-options.options i.timestamp {
16   background: url(..../img/timestamp.svg) no-repeat center;
17 }
18
19 #command-window-options.options i.autoscroll {
20   background: url(..../img/autoscroll.svg) no-repeat center;
21 }
22
23 #command-window-options.options i.clear {
24   background: url(..../img/clear.svg) no-repeat center;
25 }
26
27 #command-window-options.options i.log {
28   background: url(..../img/save-log.svg) no-repeat center;
29 }
30
31 #command-window-options.options i.to-bottom {
32   background: url(..../img/to-bottom.svg) no-repeat center;
33 }
34
35 #command-window-options.options i {
36   width: 16px;
37   height: 18px;
38   display: block;
39   background-size: 16px!important;
40   float: left;
41   clear: none;
42   padding: 3px 5px;
```



```
43 opacity: 0.3;
44 user-select: none;
45 -webkit-user-drag: none;
46 }
47
48 #command-window-options .options i:hover {
49   opacity: 1;
50   cursor: pointer;
51 }
52
53 #command-window-options .options i.active {
54   opacity: 0.8;
55 }
56
57 #command-window-input-container {
58   padding: 10px;
59   padding-left: 25px;
60   position: relative;
61   background-image: url(../img/input.svg);
62   background-position: left 12px top 12px;
63   background-size: 15px;
64   background-repeat: no-repeat;
65 }
66
67 #command-window-input {
68   border: 0;
69   color: #333;
70   font-family: 'Roboto', Arial;
71   font-size: 16px;
72   font-weight: 300;
73   padding: 0;
74   width: 100%;
75   background: transparent;
76   resize: none;
77   overflow: hidden;
78   min-height: 34px;
79 }
80
81 #command-window-messages {
82   padding-top: 10px;
83 }
84
85 #command-window-messages .welcome-message .app-logo {
86   float: left;
87   height: 200px;
88   padding: 20px;
89   padding-left: 0px;
90   display: block;
91   user-select: none;
92   -webkit-user-drag: none;
93   user-drag: none;
94 }
95
96 #command-window-messages .welcome-message .company-logo {
97   float: left;
```

```
98 height: 136px;
99 padding: 52px 10px;
100 padding-right: 10px;
101 opacity: 0.1;
102 display: block;
103 user-select: none;
104 -webkit-user-drag: none;
105 user-drag: none;
106 }
107
108 #command-window-messages .welcome-message p {
109   display: block;
110   line-height: 22px;
111   padding: 10px 0px;
112   color: #333;
113   font-weight: 300;
114   opacity: 0.8;
115   max-width: 800px;
116 }
117
118 #command-window-messages .welcome-message span:not(.timestamp) {
119   font-weight: 500;
120 }
121
122 #command-window-messages .welcome-message, #command-window-messages .
123   welcome-message * {
124     white-space: normal !important;
125   }
126
127 #command-window-messages a {
128   font-weight: 500;
129   color: #000;
130 }
131
132 #command-window-messages.messages > div {
133   padding-bottom: 5px;
134   padding-top: 5px;
135   line-height: 22px;
136   padding-left: 130px;
137   padding-right: 10px;
138   position: relative;
139   white-space: pre-wrap;
140   min-height: 32px;
141 }
142
143 #command-window-messages.messages.no-timestamp > div {
144   padding-left: 25px;
145 }
146
147 #command-window-messages.messages div.system-in {
148   background: url(../img/system-in.svg) no-repeat 4px 8px;
149   background-size: 14px;
150 }
151 #command-window-messages.messages div.data-in {
```



```
152     background: url(../img/console-in.svg) no-repeat 4px 8px;
153     background-size: 14px;
154 }
155
156 #command-window-messages.messages div.data-out {
157     background: url(../img/console-out.svg) no-repeat 4px 8px;
158     background-size: 14px;
159 }
160
161 #command-window-messages.messages div:hover {
162     background-color: #f9f9f9;
163 }
164
165 #command-window-messages.messages.no-timestamp div span.timestamp {
166     display: none;
167 }
168
169 #command-window-messages.messages div:hover span.timestamp {
170     color: #2e85c7
171 }
172
173 #command-window-messages.messages div span.timestamp {
174     color: #999;
175     display: block;
176     position: absolute;
177     left: 25px;
178 }
179
180 #command-window-messages.messages div span.log {
181     color: #666;
182 }
183
184 #command-window-messages.messages div span.msg {
185     color: #12568a;
186 }
187
188 #command-window-messages.messages div span.data {
189     color: #17d838;
190 }
191
192 #command-window-messages.messages div span.error {
193     color: #cf000f;
194 }
195
196 #command-window-messages.messages div span.warn {
197     color: #dd8f00;
198 }
199 #command-window-messages.messages:hover div:hover {
200     filter: none;
201 }
202
203 /* #command-window-messages.messages:hover div {
204     filter: grayscale(100%);
205 } */
206
```



```
207 .save-log, .change-settings {
208   margin-bottom: 0px;
209 }
210
211 .history-cont {
212   display: none;
213   left: 0px;
214   position: absolute;
215   bottom: 0px;
216   top: 23px;
217   right: 0px;
218   -webkit-backdrop-filter: blur(5px);
219   backdrop-filter: blur(5px);
220   background-color: rgba(255, 255, 255, 0.7);
221   padding: 20px;
222 }
223
224 .history-panel {
225   top: 55px !important;
226   left: 20px !important;
227   right: 20px !important;
228   bottom: 0px !important;
229   padding-top: 10px;
230   padding-bottom: 10px;
231 }
232
233 .history-panel li {
234   padding-bottom: 5px;
235   padding-top: 5px;
236   line-height: 22px;
237   padding-left: 10px;
238   padding-right: 10px;
239   margin: 0 10px;
240   position: relative;
241   white-space: pre-wrap;
242   border-bottom: 1px solid #f7df1e;
243   min-height: 22px;
244   cursor: pointer;
245   transition: all linear 0.3s;
246 }
247
248 .history-panel li:after {
249   transition: all linear 0.3s;
250 }
251
252 .history-panel li:after {
253   content: '';
254   position: absolute;
255   top: 3px;
256   left: 0px;
257   right: 0px;
258   bottom: 3px;
259   border: 1px solid transparent;
260   border-radius: 3px;
261   background: #fffff69;
```



```
262     z-index: -1;
263 }
264
265 .history-panel li.active:after {
266   border: 1px solid #999;
267 }
268
269 .history-panel li:not(.active):hover:after {
270   border: 1px solid #999;
271 }
272
273 .history-panel li:last-child {
274   border-bottom: 1px solid transparent;
275 }
276
277 .history-panel .history-empty {
278   padding-bottom: 5px;
279   padding-top: 5px;
280   line-height: 22px;
281   padding-left: 10px;
282   padding-right: 10px;
283   margin: 0 10px;
284   position: relative;
285   text-align: center;
286 }
287
288 .options-panel {
289   display: none;
290   top: 0;
291   left: 0;
292   right: 0;
293   bottom: 0;
294   position: absolute;
295   -webkit-backdrop-filter: blur(5px);
296   backdrop-filter: blur(5px);
297   background-color: rgba(255, 255, 255, 0.7);
298   overflow-y: auto;
299 }
300
301 .options-cont {
302   margin: 0 auto;
303   width: 460px;
304   position: relative;
305   background: #fffff6b;
306   box-shadow: 0 0 10px #00000026;
307   border-radius: 10px;
308   padding: 20px;
309   padding-bottom: 20px;
310   color: #777;
311   margin-bottom: 20px;
312   margin-top: 20px;
313 }
314
315 .options-header, .page-header, .history-header {
316   position: relative;
```

```

317 border-bottom: 1px solid #666;
318 margin-bottom: 25px;
319 padding-bottom: 10px;
320 user-select: none;
321 -webkit-user-drag: none;
322 user-drag: none;
323 }
324
325 .page-header {
326   margin-bottom: 0px;
327 }
328
329 .options-header span, .page-header span, .history-header span {
330   color: #666;
331   display: block;
332   font-weight: bold;
333   font-size: 24px;
334   border-radius: 3px;
335 }
336
337 .options-close, .page-close, .history-close {
338   opacity: 0.7;
339   position: absolute;
340   display: block;
341   top: 0px;
342   right: 0px;
343   cursor: pointer;
344 }
345
346 .options-close:hover, .page-close:hover, .history-close:hover {
347   opacity: 0.3;
348 }
349
350 @media only screen and (max-width: 900px) {
351   .page-cont {
352     width: 460px;
353   }
354 }
```

Listing 22 - terminal.css

```

1 html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre, a,
  abbr, acronym, address, big, cite, code, del, dfn, em, img, ins, kbd, q, s, samp, small,
  strike, strong, sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol, ul, li, fieldset, form,
  label, legend, table, caption, tbody, tfoot, thead, tr, th, td, article, aside, canvas
  , details, embed, figure, figcaption, footer, header, hgroup, menu, nav, output, ruby
  , section, summary, time, mark, audio, video { border: 0; font-size: 100% ;
  font: inherit; vertical-align: baseline; margin: 0; padding: 0 } article, aside,
  details, figcaption, figure, footer, header, hgroup, menu, nav, section {
  display: block } body { line-height: 1 } ol, ul { list-style: none } blockquote, q {
  quotes: none } blockquote::before, blockquote::after, q::before, q::after {
  content: none } table { border-collapse: collapse; border-spacing: 0 }

2
3 html {
4   overflow: hidden;
5 }
```

```

6
7  :focus {
8    outline: 0;
9  }
10
11 div {
12   z-index: 2;
13   box-sizing: border-box;
14 }
15
16 section {
17   position: relative;
18   z-index: 2;
19 }
20
21 body {
22   font-family: 'Roboto', Arial;
23   width: 100%;
24   background: #fff;
25 }
26
27 .clear {
28   clear: both;
29 }
30
31 .float-left {
32   float: left;
33 }
34
35 .float-right {
36   float: right;
37 }
```

Listing 23 - three.css

5 html

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="UTF-8">
5   <title>Window - JSLAB | PR-DC</title>
6   <meta http-equiv="Content-Security-Policy" content="script-src * 'self' '''
7     unsafe-inline ''unsafe-eval'; worker-src 'self' blob:' />
8   <link rel="stylesheet" type="text/css" href="../css/basic.css" />
9   <link rel="stylesheet" type="text/css" href="../font/roboto.css" />
10
11 </head>
12 <body>
13 </body>
14 </html>
```

Listing 24 - blank.html

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <title>Cesium Plot - JSLAB | PR-DC</title>
6      <meta http-equiv="Content-Security-Policy" content="script-src * 'unsafe-
7        inline' 'unsafe-eval' blob:;"/>
8      <link rel="stylesheet" type="text/css" href="../css/basic.css" />
9      <link rel="stylesheet" type="text/css" href="../font/roboto.css" />
10     <link rel="stylesheet" type="text/css" href="../font/LatinModern.css" />
11     <link rel="stylesheet" type="text/css" href="../lib/Cesium-1.124/Widgets/
12       widgets.css">
13   <style>
14     .cesium-viewer-bottom, .cesium-viewer-toolbar, .cesium-viewer-
15       animationContainer, .cesium-viewer-fullscreenContainer {
16         display: none !important;
17     }
18   </style>
19   <style id="dynamic-style-rules"></style>
20 </head>
21 <body>
22   <div id="map-3d-cont"></div>
23   <script type="text/javascript" src="../lib/Cesium-1.124/Cesium.js"></
24     script>
25 </body>
26 </html>
```

Listing 25 - cesium.html

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <title>D3 Canvas - JSLAB | PR-DC</title>
6      <meta http-equiv="Content-Security-Policy" content="script-src * 'self' ,
7        'unsafe-inline' 'unsafe-eval'; worker-src 'self' blob:;"/>
8      <link rel="stylesheet" type="text/css" href="../css/basic.css" />
9      <link rel="stylesheet" type="text/css" href="../font/roboto.css" />
10     <link rel="stylesheet" type="text/css" href="../font/LatinModern.css" />
11   <style id="dynamic-style-rules"></style>
12 </head>
13 <body>
14   <svg id="d3-svg"></svg>
15   <canvas id="d3-canvas"></canvas>
16   <script type="text/javascript" src="../lib/d3-7.8.5/d3-7.8.5.min.js"></
17     script>
18 </body>
19 </html>
```

Listing 26 - d3.html

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <title>Editor - JSLAB | PR-DC</title>
6      <meta http-equiv="Content-Security-Policy" content="script-src * 'self' 'unsafe-inline' 'unsafe-eval'; worker-src 'self' blob:; " />
7      <link rel="stylesheet" type="text/css" href="../css/tabs.css">
8      <link rel="stylesheet" type="text/css" href="../css/codemirror-notepadpp-theme.css">
9      <link rel="stylesheet" type="text/css" href="../lib/codemirror-5.49.2/addon/fold/foldgutter.css">
10     <link rel="stylesheet" type="text/css" href="../lib/codemirror-5.49.2/addon/lint/lint.css">
11     <link rel="stylesheet" type="text/css" href="../lib/codemirror-5.49.2/addon/hint/show-hint.css">
12     <link rel="stylesheet" type="text/css" href="../lib/codemirror-5.49.2/lib/codemirror.css">
13     <link rel="stylesheet" type="text/css" href="../css/codemirror-editor-custom.css">
14       <link rel="stylesheet" type="text/css" href="../css/editor.css" />
15     <link rel="stylesheet" type="text/css" href="../font/roboto.css" />
16
17     <style id="dynamic-style-rules"></style>
18   </head>
19   <body>
20     <div id="editor-menu-container">
21       <div>
22         
23         <ul id="editor-menu">
24           <li id="new-script" title-str="26"><str sid="10"></str></li>
25           <li id="open-menu" title-str="27"><str sid="11"></str></li>
26           <li id="save-menu" title-str="28"><str sid="12"></str></li>
27           <li id="save-as-menu" title-str="29"><str sid="13"></str></li>
28           <li id="run-menu" title-str="30"><str sid="14"></str></li>
29         </ul>
30         <ul id="window-controls">
31           <li id="win-minimize"></li>
32           <li id="win-restore"></li>
33           <li id="win-close"></li>
34         </ul>
35         <div id="app-title">JSLAB / EDITOR</div>
36         <div class="clear"></div>
37       </div>
38     </div>
39     <div class="tabs" style="--tab-content-margin: 9px">
40       <div class="tabs-content"></div>
41       
42       

```

```

43   <div class="tabs-bottom-bar"></div>
44 </div>
45 <div id="code"></div>
46 <div id="close-dialog-cont">
47   <div id="close-dialog">
48     <div id="close-dialog-header">
49       <span><str sid="47"></str></span>
50     </div>
51   <div id="close-dialog-msg">
52     <str sid="48"></str> <span id="close-file"></span> <str sid="49"></str>
53   </div>
54   <div id="close-dialog-buttons">
55     <button id="close-dialog-save"><str sid="50"></str></button>
56     <button id="close-dialog-discard"><str sid="51"></str></button>
57     <button id="close-dialog-cancel"><str sid="52"></str></button>
58   </div>
59   <br class="clear">
60 </div>
61 </div>
62
63 <div id="editor-more-popup" style="display: none;">
64   <ul>
65     <li id="search-dialog-menu" title-str="169"><str sid="168"></str></li>
67   </ul>
68   <div class="popup-triangle"></div>
69 </div>
70 <script>if (typeof module === 'object') {window.module = module; module =
71   undefined;}</script>
72 <script type="text/javascript" src="../../lib/draggabilly-2.3.0/draggabilly-
73   -2.3.0.min.js"></script>
74 <script type="text/javascript" src="../../lib/PRDC_TABS/PRDC_TABS.js"></
75   script>
76 <script type="text/javascript" src="../../lib/jshint-2.13.0/jshint-2.13.0.js"
77 ></script>
78 <script type="text/javascript" src="../../lib/codemirror-5.49.2/lib/
79   codemirror.js"></script>
80 <script type="text/javascript" src="../../lib/codemirror-5.49.2/addon/
81   selection/active-line.js"></script>
82 <script type="text/javascript" src="../../lib/codemirror-5.49.2/addon/fold/
83   foldcode.js"></script>
84 <script type="text/javascript" src="../../lib/codemirror-5.49.2/addon/fold/
85   foldgutter.js"></script>
86 <script type="text/javascript" src="../../lib/codemirror-5.49.2/addon/fold/
87   brace-fold.js"></script>
88 <script type="text/javascript" src="../../lib/codemirror-5.49.2/addon/fold/
89   indent-fold.js"></script>
90 <script type="text/javascript" src="../../lib/codemirror-5.49.2/addon/fold/
91   comment-fold.js"></script>
92 <script type="text/javascript" src="../../lib/codemirror-5.49.2/addon/display
93   /rulers.js"></script>
```

```

84 <script type="text/javascript" src="../lib/codemirror-5.49.2/addon/edit/
  matchbrackets.js"></script>
85 <script type="text/javascript" src="../lib/codemirror-5.49.2/addon/lint/
  lint.js"></script>
86 <script type="text/javascript" src="../lib/codemirror-5.49.2/addon/hint/
  show-hint.js"></script>
87 <script type="text/javascript" src="../lib/codemirror-5.49.2/addon/search/
  searchcursor.js"></script>
88 <script type="text/javascript" src="../lib/codemirror-5.49.2/addon/scroll/
  annotatescrollbar.js"></script>
89 <script type="text/javascript" src="../lib/codemirror-5.49.2/addon/search/
  matchesonscrollbar.js"></script>
90 <script type="text/javascript" src="../lib/codemirror-5.49.2/addon/search/
  jump-to-line.js"></script>
91 <script type="text/javascript" src="../lib/codemirror-5.49.2/addon/search/
  match-highlighter.js"></script>
92 <script type="text/javascript" src="../lib/codemirror-5.49.2/addon/comment/
  /comment.js"></script>
93 <script type="text/javascript" src="../lib/codemirror-5.49.2/mode/
  javascript/javascript.js"></script>
94 <script type="text/javascript" src="../lib/codemirror-5.49.2/mode/clike/
  clike.js"></script>
95
96 <script type="text/javascript" src="../js/code/custom-javascript-hint.js">
  </script>
97 <script type="text/javascript" src="../js/code/dialog-search.js"></script>
98
99 <script type="text/javascript" src="../lib/jquery-3.7.0/jquery-3.7.0.min.
  js"></script>
100
101 <script type="text/javascript" src="../js/editor/init-editor.js"></script>
102 </body>
103 </html>
```

Listing 27 - editor.html

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <title>Figure - JSLAB | PR-DC</title>
6     <meta http-equiv="Content-Security-Policy" content="script-src * 'self' '
  unsafe-inline 'unsafe-eval'; worker-src 'self' blob:;" />
7       <link rel="stylesheet" type="text/css" href="../css/figure.css
  " />
8     <link rel="stylesheet" type="text/css" href="../font/roboto.css" />
9     <link rel="stylesheet" type="text/css" href="../font/LatinModern.css" />
10
11   <style id="dynamic-style-rules"></style>
12 </head>
13 <body>
14   <div id="figure-menu-button"></div>
15   <div id="figure-menu-container">
16     <div id="figure-menu">
17       <ul>
18         <li id="save-as-menu" title-str="152">
```

```

19   ><str sid="159"></str></li>
20   <li id="zoom-menu" class="specific -2d" title-str="153"><str sid="160"></str></li>
22   <li id="pan-menu" class="specific -2d" title-str="154"><str sid="161"></str></li>
24   <li id="rotate-menu" class="specific -2d" title-str="155"><str sid="162"></str></li>
26   <li id="zoom-in-menu" class="specific -2d" title-str="156"><str sid="163"></str></li>
28   <li id="zoom-out-menu" class="specific -2d" title-str="157"><str sid="164"></str></li>
30   <li id="fit-menu" class="specific -2d" title-str="158"><str sid="165"></str></li>
32   <li id="reset-menu" class="specific -2d" title-str="158"><str sid="221"></str></li>
34   <li id="pan-menu-3d" class="specific -3d" title-str="154"><str sid="161"></str></li>
36   <li id="rotate-menu-3d" class="specific -3d" title-str="155"><str sid="162"></str></li>
38   <li id="fit-menu-3d" class="specific -3d" title-str="158"><str sid="165"></str></li>
40   <li id="reset-menu-3d" class="specific -3d" title-str="158"><str sid="221"></str></li>
42   </ul>
43   <div class="clear"></div>
44 </div>
45 <div id="figure-content">
46 </div>
47
48 <script type="text/javascript" src=".../lib/plotly-2.24.2/plotly-2.24.2.min.
49   .js"></script>
50   <script type="text/javascript" src=".../js/windows/plot.js"></script>
51 </body>
52 </html>
```

Listing 28 - figure.html

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <title>%title%</title>
6      <meta http-equiv="Content-Security-Policy" content="script-src * 'self' '
7        unsafe-inline 'unsafe-eval'; worker-src 'self' blob:; " />
8      <style>
9        html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre
10       , a, abbr, acronym, address, big, cite, code, del, dfn, em, img, ins, kbd, q, s, samp,
11       small, strike, strong, sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol, ul, li,
12       fieldset, form, label, legend, table, caption, tbody, tfoot, thead, tr, th, td,
13       article, aside, canvas, details, embed, figure, figcaption, footer, header,
14       hgroup, menu, nav, output, ruby, section, summary, time, mark, audio, video {
15         border: 0; font-size: 100%; font: inherit; vertical-align: baseline; margin: 0;
16         padding: 0} article, aside, details, figcaption, figure, footer, header, hgroup
17       , menu, nav, section { display: block } body { line-height: 1 } ol, ul { list-style:
18         none } blockquote, q { quotes: none } blockquote: before, blockquote: after, q:
```

```
before ,q:after {content:none}table{border-collapse:collapse;border-spacing:0}

9      #figure-content {
10     margin: 30px auto;
11     position: relative;
12     border: 1px solid #6f6f6f;
13     box-shadow: 0px 0px 20px #c1c1c1;
14   }
15
16
17   #figure-header {
18     background: #12568a;
19     background: linear-gradient(to left, #2e85c7, #12568a);
20     color: #fff;
21     user-select: none;
22     -webkit-user-drag: none;
23     user-drag: none;
24     max-height: 40px;
25     overflow: hidden;
26     transition: all .2s linear;
27     position: absolute;
28     left: 0;
29     right: 0;
30     top: 0;
31   }
32
33   #jslab-logo svg {
34     height: 30px;
35     float: left;
36     padding: 5px 10px;
37     user-select: none;
38     -webkit-user-drag: none;
39     user-drag: none;
40     width: 23px;
41     background: #00000047;
42   }
43
44   #company-logo svg {
45     filter: invert(1);
46     height: 30px;
47     float: left;
48     padding: 5px;
49     padding-left: 10px;
50     user-select: none;
51     -webkit-user-drag: none;
52     user-drag: none;
53     width: 54px;
54   }
55
56   #title {
57     font-family: 'Helvetica';
58     font-size: 20px;
59     padding: 3px 10px;
60     margin: 7px 0;
61     float: left;
```

```

62   font-weight: 400;
63   opacity: 0.8;
64   color: #ffffff;
65   border-left: 2px solid #fff;
66   margin-left: 5px;
67 }
68
69 #plot-cont {
70   padding-top: 40px;
71 }
72 </style>
73 </head>
74 <body>
75 <div id="figure-content">
76   <div id="figure-header">
77     <a href="https://pr-dc.com/jslab" title="JSLAB">
78       <div id="jslab-logo">
79         <svg
80           version="1.1"
81           viewBox="0 0 630 630"
82           xmlns:inkscape="http://www.inkscape.org/namespaces/inkscape"
83           xmlns:sodipodi="http://sodipodi.sourceforge.net/DTD/sodipodi-0.
84             dtd"
85           xmlns="http://www.w3.org/2000/svg"
86           xmlns:svg="http://www.w3.org/2000/svg"
87           xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
88           xmlns:cc="http://creativecommons.org/ns#"
89           xmlns:dc="http://purl.org/dc/elements/1.1/">
90         <metadata
91           id="metadata12">
92           <rdf:RDF>
93             <cc:Work
94               rdf:about="">
95               <dc:format>image/svg+xml</dc:format>
96               <dc:type
97                 rdf:resource="http://purl.org/dc/dcmitype/StillImage" />
98             </cc:Work>
99           </rdf:RDF>
100         </metadata>
101         <defs
102           id="defs10" />
103         <rect
104           id="background"
105             x="0"
106             y="0"
107             width="630"
108             height="630"
109             fill="#f7df1e"
110             rx="50"
111             ry="50" />
112         <path
d="m 132.3099,593 c 42.69727,0 71.96556,-22.72597
      71.96556,-72.65424 v -164.5911 h -48.2066 v 163.90243 c
      0,24.1033 -9.98565,30.30129 -25.82496,30.30129

```

```

113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
      -16.52798,0  -23.41463,-11.36298  -30.989963,-24.79196 L
      60,548.92539 C 71.362984,573.0287  93.744617,593
      132.3099,593 Z"
  style="font-size:341.94px;line-height:1.05;font-family:'Neutra Text';-inkscape-font-specification:'Neutra Text';word-spacing:0px;fill:#000000;fill-opacity:1;stroke-width:8.60834"
  id="path850" />
<path
  d="m 318.24908,593 c 45.79626,0 79.88522,-23.75897
    79.88522,-67.14491 0,-40.28695 -23.0703,-58.19225
    -64.04591,-75.75323 l -12.05165,-5.16499 c
    -20.65997,-8.95266 -29.61263,-14.80631
    -29.61263,-29.26829 0,-11.70732 8.95266,-20.65997
    23.0703,-20.65997 13.77332,0 22.72597,5.85365
    30.98996,20.65997 l 37.53228,-24.1033 C
    368.17734,363.67432 346.14004,353 315.49441,353 c
    -43.0416,0 -70.58823,27.54663 -70.58823,63.70158
    0,39.25394 23.0703,57.84791 57.84792,72.65422 l
    12.05165,5.165 c 22.0373,9.64132 35.12195,15.49498
    35.12195,32.02295 0,13.77332 -12.74032,23.75897
    -32.71162,23.75897 -23.75897,0 -37.18795,-12.39598
    -47.51793,-29.26829 L 230.4442,543.76039 C
    244.56185,571.65136 273.48581,593 318.24908,593 Z"
  style="font-size:341.94px;line-height:1.05;font-family:'Neutra Text';-inkscape-font-specification:'Neutra Text';word-spacing:0px;fill:#000000;fill-opacity:1;stroke-width:8.60834"
  id="path852" />
<path
  d="M 435.66613,589.901 H 589.92724 V 547.54805 H 483.87273 V
    355.75466 h -48.2066 z"
  style="font-size:341.94px;line-height:1.05;font-family:'Neutra Text';-inkscape-font-specification:'Neutra Text';word-spacing:0px;fill:#000000;fill-opacity:1;stroke-width:8.60834"
  id="path854" />
</svg>
</div>
</a>
<a href="https://pr-dc.com/" title="PR-DC Company">
  <div id="company-logo">
    <svg
      width="160mm"
      height="90mm"
      viewBox="0 0 160 90"
      version="1.1"
      id="svg869"
      xmlns="http://www.w3.org/2000/svg"
      xmlns:svg="http://www.w3.org/2000/svg"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:cc="http://creativecommons.org/ns#"
      xmlns:dc="http://purl.org/dc/elements/1.1/">
    <defs
      id="defs863" />

```

```
141 <metadata
142   id="metadata866">
143   <rdf:RDF>
144     <cc:Work
145       rdf:about=""
146       dc:format>image/svg+xml</dc:format>
147       <dc:type
148         rdf:resource="http://purl.org/dc/dcmitype/StillImage" />
149     </cc:Work>
150   </rdf:RDF>
151 </metadata>
152 <g
153   id="layer1">
154   <path
155     id="path847"
156     style="fill:#000000;fill-opacity:1;stroke:none;stroke-width:0.999999px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1"
157     d="M 483.13281 17.101562 C 483.13281 17.101562 455.74781
        17.701959 428.36328 17.615234 C 425.51072 17.606164
        422.57229 17.604649 419.5625 17.613281 C 374.41441
        17.745451 313.17548 18.456471 280.00586 28.585938 C
        219.34388 47.111121 185.96586 73.091557 158.50391
        103.79688 C 153.5363 109.42942 147.05989 117.97425
        139.72266 128.92188 C 143.64779 137.54078 145.61523
        147.65489 145.61523 159.27539 C 145.61523 172.166
        143.27648 183.00438 138.59766 191.78906 C 133.91885
        200.57382 127.9503 207.49589 120.69336 212.55664 C
        113.5319 217.52192 106.22722 220.81618 98.779297
        222.43945 C 95.807549 223.02819 92.396307 223.51965
        88.642578 223.93555 C 75.710747 253.86377 63.915338
        287.36997 55.78125 322.60352 L 449.07422 322.89844 C
        452.91989 288.9274 456.14589 259.97604 459.13281
        232.99414 C 454.31399 221.25319 451.90039 207.64528
        451.90039 192.16211 C 451.90039 165.94179 458.11321
        145.11257 470.5293 129.66602 C 474.28218 95.661149
        478.18692 60.495264 483.13281 17.101562 z M 539.49609
        115.06445 C 519.44401 115.06445 503.26564 121.67252
        490.96094 134.88867 C 478.65626 148.03972 472.50391
        166.52928 472.50391 190.35742 C 472.50391 212.88348
        478.62368 230.65689 490.86328 243.67773 C 503.10285
        256.63347 518.72784 263.11133 537.73828 263.11133 C
        553.10287 263.11133 565.76562 259.33529 575.72656
        251.7832 C 585.75262 244.16601 592.91407 232.54493
        597.21094 216.91992 L 569.18359 208.0332 C 566.77472
        218.51498 562.80339 226.19726 557.26953 231.08008 C
        551.73567 235.96289 545.12761 238.4043 537.44531
        238.4043 C 527.02866 238.4043 518.56513 234.56315
        512.05469 226.88086 C 505.54427 219.19856 502.28906
        206.30796 502.28906 188.20898 C 502.28906 171.15168
        505.57682 158.81446 512.15234 151.19727 C 518.79297
        143.58007 527.41927 139.77148 538.03125 139.77148 C
        545.71354 139.77148 552.22394 141.91992 557.5625
        146.2168 C 562.96614 150.51367 566.51431 156.37304
```

568.20703 163.79492 L 596.82031 156.95898 C 593.56511
145.50064 588.68231 136.71159 582.17188 130.5918 C
571.23438 120.24023 557.00914 115.06445 539.49609
115.06445 z M 14.216797 117.50586 L 14.216797 260.66992
L 43.123047 260.66992 L 43.123047 206.66602 L
61.970703 206.66602 C 75.05665 206.66602 85.050127
205.98242 91.951172 204.61523 C 97.0293 203.50846
102.00976 201.26237 106.89258 197.87695 C 111.8405
194.42643 115.9095 189.70638 119.09961 183.7168 C
122.28971 177.72721 123.88477 170.3379 123.88477
161.54883 C 123.88477 150.15559 121.11784 140.87826
115.58398 133.7168 C 110.05013 126.49023 103.18165
121.80273 94.978516 119.6543 C 89.639971 118.222
78.181651 117.50586 60.603516 117.50586 L 14.216797
117.50586 z M 157.9082 117.50586 L 218.74805 117.50586
C 234.04754 117.50586 245.14779 118.80794 252.04883
121.41211 C 259.01499 123.95116 264.58138 128.50844
268.74805 135.08398 C 272.91471 141.65949 274.99805
149.17908 274.99805 157.64258 C 274.99805 168.38479
271.84049 177.27149 265.52539 184.30273 C 259.21025
191.26886 249.77016 195.66342 237.20508 197.48633 C
243.45506 201.1321 248.59831 205.13606 252.63477
209.49805 C 256.73635 213.86 262.23763 221.60745
269.13867 232.74023 L 286.61914 260.66992 L 252.04883
260.66992 L 231.15039 229.51758 C 223.72849 218.38483
218.65039 211.38606 215.91602 208.52148 C 213.18164
205.59178 210.2845 203.60613 207.22461 202.56445 C
204.16471 201.45769 199.31446 200.9043 192.67383
200.9043 L 186.81445 200.9043 L 186.81445 260.66992 L
157.9082 260.66992 L 157.9082 117.50586 z M 322.95312
117.50586 L 375.78516 117.50586 C 387.69921 117.50586
396.78123 118.41729 403.03125 120.24023 C 411.4297
122.7142 418.62367 127.10873 424.61328 133.42383 C
430.60285 139.73889 435.16017 147.48634 438.28516
156.66602 C 441.41018 165.78061 442.97266 177.04361
442.97266 190.45508 C 442.97266 202.23893 441.50783
212.39517 438.57812 220.92383 C 434.99745 231.34047
429.88672 239.77153 423.24609 246.2168 C 418.23308
251.09964 411.46225 254.9082 402.93359 257.64258 C
396.55341 259.66081 388.02475 260.66992 377.34766
260.66992 L 322.95312 260.66992 L 322.95312 117.50586 z
M 43.123047 141.72461 L 57.087891 141.72461 C
67.504562 141.72461 74.43815 142.05013 77.888672
142.70117 C 82.576175 143.54753 86.449868 145.66341
89.509766 149.04883 C 92.569664 152.43425 94.099609
156.73112 94.099609 161.93945 C 94.099609 166.17123
92.99284 169.88216 90.779297 173.07227 C 88.630857
176.26237 85.636071 178.60612 81.794922 180.10352 C
77.953775 181.60091 70.336597 182.34961 58.943359
182.34961 L 43.123047 182.34961 L 43.123047 141.72461 z
M 186.81445 141.72461 L 186.81445 178.05273 L
208.20117 178.05273 C 222.06836 178.05273 230.72721
177.4668 234.17773 176.29492 C 237.62825 175.12304
240.33009 173.10485 242.2832 170.24023 C 244.23635
167.37565 245.21289 163.79492 245.21289 159.49805 C

```

245.21289 154.68032 243.91081 150.80665 241.30664
147.87695 C 238.76755 144.88217 235.15428 142.99412
230.4668 142.21289 C 228.12304 141.88747 221.0918
141.72461 209.37305 141.72461 L 186.81445 141.72461 z M
351.85938 141.72461 L 351.85938 236.54883 L 373.44141
236.54883 C 381.51433 236.54883 387.34115 236.09311
390.92188 235.18164 C 395.60935 234.00976 399.48306
232.02411 402.54297 229.22461 C 405.66796 226.42511
408.20709 221.8353 410.16016 215.45508 C 412.1133
209.00977 413.08984 200.25325 413.08984 189.18555 C
413.08984 178.11784 412.1133 169.62179 410.16016
163.69727 C 408.20709 157.77282 405.47267 153.15039
401.95703 149.83008 C 398.44139 146.50976 393.98179
144.26368 388.57812 143.0918 C 384.54168 142.18033
376.63151 141.72461 364.84766 141.72461 L 351.85938
141.72461 z M 275.35742 187.93945 L 311.72266 187.93945
L 311.72266 206.12305 L 275.35742 206.12305 L
275.35742 187.93945 z "
158   transform="scale(0.26458333)" />
159 <g
160   aria-label="PR DC"
161   id="text865"
162   style="font-style:normal;font-variant:normal;font-weight:
normal;font-stretch:normal;font-size:52.9167px;line-
height:1.25;font-family:arial;-inkscape-font-
specification:arial;letter-spacing:0px;word-spacing:0px
;fill:#000000;fill-opacity:1;stroke:none;stroke-width
:0.264583"
163   transform="translate(-26.307505,-2.9264341)" />
164   </g>
165   </svg>
166   </div>
167   <a>
168     <div id="title">
169       %title%
170     </div>
171     </div>
172     <div id="plot-cont">
173       
174     </div>
175   </div>
176
177   <script>
178     window.onload = function () {
179       var figure = document.getElementById('figure');
180       document.getElementById('figure-content').style.width = figure.width +
'px';
181     };
182   </script>
183 </body>
184 </html>

```

Listing 29 - html_figure.html

```

1 <!DOCTYPE html>
2 <html>

```

```

3  <head>
4    <meta charset="UTF-8">
5    <title>Leaflet Plot - JSLAB | PR-DC</title>
6    <meta http-equiv="Content-Security-Policy" content="script-src * 'self' ,
7      unsafe-inline 'unsafe-eval'; worker-src 'self' blob:;" />
8      <link rel="stylesheet" type="text/css" href="../css/basic.css" />
9
10   <link rel="stylesheet" type="text/css" href="../font/roboto.css" />
11   <link rel="stylesheet" type="text/css" href="../font/LatinModern.css" />
12   <link rel="stylesheet" type="text/css" href="../lib/leaflet-1.9.4/leaflet.
13     css" />
14
15   <style id="dynamic-style-rules"></style>
16 </head>
17 <body>
18   <div id="map-cont"></div>
19
20   <script type="text/javascript" src="../lib/leaflet-1.9.4/leaflet.js"></
21     script>
22   <script type="text/javascript" src="../lib/leaflet.rotatedMarker-0.2.0/
23     leaflet.rotatedMarker.js"></script>
24 </body>
25 </html>
```

Listing 30 - leaflet.html

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <title>JSLAB | PR-DC</title>
6      <meta http-equiv="Content-Security-Policy" content="script-src * 'self' ,
7        unsafe-inline 'unsafe-eval'; worker-src 'self' blob:;" />
8      <link rel="stylesheet" type="text/css" href="../css/codemirror-notepadpp-
9        theme.css">
10     <link rel="stylesheet" type="text/css" href="../css/highlight-notepadpp-
11       theme.css">
12     <link rel="stylesheet" type="text/css" href="../lib/codemirror-5.49.2/
13       addon/lint/lint.css">
14     <link rel="stylesheet" type="text/css" href="../css/big-json-viewer-
15       notepadpp-theme.css">
16     <link rel="stylesheet" type="text/css" href="../lib/codemirror-5.49.2/
17       addon/hint/show-hint.css">
18     <link rel="stylesheet" type="text/css" href="../lib/codemirror-5.49.2/lib/
19       codemirror.css">
20     <link rel="stylesheet" type="text/css" href="../css/codemirror-main-custom
21       .css">
22         <link type="text/css" rel="stylesheet" href="../css/main.css" />
23
24     <link type="text/css" rel="stylesheet" href="../css/terminal.css" />
25     <link type="text/css" rel="stylesheet" href="../font/roboto.css" />
26     <link type="text/css" rel="stylesheet" href="../font/RobotoMono.css" />
27
28     <style id="dynamic-style-rules"></style>
29 </head>
30 <body>
```

```

22 <div id="main-menu-container">
23   <div>
24     
25     
26     <ul id="main-menu">
27       <li id="editor-menu"><str sid="1"></str></li>
28       <li id="help-menu"><str sid="2"></str></li>
29       <li id="info-menu"><str sid="3"></str></li>
30       <li id="settings-menu"><str sid="9"></str></li>
31     </ul>
32     <ul id="window-controls">
33       <li id="win-minimize"></li>
34       <li id="win-restore"></li>
35       <li id="win-close"></li>
36     </ul>
37     <div id="app-title">JSLAB / MAIN</div>
38     <div class="clear"></div>
39   </div>
40 </div>
41
42 <div id="folder-navigation-container">
43   <div class="button float-left previous-folder disabled"></div>
44   <div class="button float-left next-folder disabled"></div>
45   <div class="button float-left up-folder disabled"></div>
46   <div class="button float-left open-folder"></div>
47   <div class="address-line-container">
48     <div class="folder-icon"></div>
49     <div class="folder-address">
50       <input class="address-line float-left" value="C:/ Electron/JSLAB">
51       <div class="current-address-cont">
52         <div class="current-address-wrap">
53           <div class="current-address">
54             <span class="folder">Local Disk (C:)</span>
55             <i class="i-next-folder"></i>
56             <span class="folder">Electron</span>
57             <i class="i-next-folder"></i>
58             <span class="folder">JSLAB</span>
59           </div>
60         </div>
61       </div>
62       <div id="save-path"><i title-str="35"/></i></div>
63     </div>
64   </div>
65   <div class="button float-right" id="paths-menu"></div>
66   <div class="clear"></div>
67 </div>

```

```
68
69 <div id="panels-container">
70   <div id="left-panel">
71     <div id="left-top-panel">
72       <div class="cell-padding">
73         <div class="panel-container">
74           <div class="panel-title"><str sid="4"></str></div>
75
76           <div id="file-browser-options" class="options">
77             <div class="options-right">
78               <i class="refresh" title-str="37"></i>
79             </div>
80           </div>
81
82           <div id="file-browser" class="panel">
83             <div id="file-browser-cont"></div>
84           </div>
85         </div>
86       </div>
87     </div>
88
89   <div id="left-middle-panel" >
90     <div class="cell-padding">
91       <div class="panel-container">
92         <div class="panel-title"><str sid="5"></str></div>
93
94         <div id="workspace-options" class="options">
95           <div class="options-right">
96             <i class="clear" title-str="38"></i>
97           </div>
98         </div>
99
100        <div id="workspace-table-head">
101          <div class="row">
102            <div class="col col-1"><str sid="53"></str></div>
103            <div class="col col-2"><str sid="54"></str></div>
104            <div class="col col-3"><str sid="55"></str></div>
105          </div>
106        </div>
107
108        <div id="workspace" class="panel">
109          <div class="table"></div>
110        </div>
111      </div>
112    </div>
113  </div>
114
115  <div id="left-bottom-panel">
116    <div class="cell-padding">
117      <div class="panel-container">
118        <div class="panel-title"><str sid="6"></str></div>
119
120        <div id="command-history-options" class="options">
121          <div class="options-right">
122            <i class="clear" title-str="39"></i>
```

```

123      </div>
124    </div>
125
126    <div id="command-history" class="panel">
127
128      </div>
129      </div>
130      </div>
131      </div>
132    </div>
133
134    <div id="right-panel">
135      <div class="cell-padding">
136        <div class="panel-container">
137          <div class="panel-title"><str sid="7"></str></div>
138
139        <div id="command-window-options" class="options">
140          <div class="options-right">
141            <i class="settings" title-str="40"></i>
142            <i class="timestamp" title-str="41"></i>
143            <i class="autoscroll active" title-str="42"></i>
144            <i class="clear" title-str="43"></i>
145            <i class="log" title-str="44"></i>
146            <i class="to-bottom" title-str="45"></i>
147          </div>
148        </div>
149
150        <div id="command-window" class="terminal-panel panel">
151          <div id="command-window-messages" class="messages no-timestamp">
152            </div>
153
154          <div id="command-window-input-container">
155            <div id="command-window-input-submit-cont">
156              
157            </div>
158            <textarea id="command-window-input">cmd_help;</textarea>
159          </div>
160
161        <div id="command-window-settings" class="terminal-dialog terminal-
162          settings options-panel panel" tabindex="0">
163          <div class="options-cont">
164            <div class="options-header"><span><str sid="56"></str></span>
165            
167            </div>
168            <div class="float-input" title-str="17">
169              <input autocomplete="off" type="text" name="N-messages-max"
170                class="N-messages-max" placeholder="Infinity" value="">
171              <label class="float-label" for="N-messages-max"><str sid="17">
172                </str></label>
173            </div>
174            <button class="change-settings"><str sid="57"></str></button>
175          </div>
176        </div>
177
178      </div>
179    </div>
180
181  </div>
182
183  </div>
184
185  </div>
186
187  </div>
188
189  </div>
190
191  </div>
192
193  </div>
194
195  </div>
196
197  </div>
198
199  </div>
200
201  </div>
202
203  </div>
204
205  </div>
206
207  </div>
208
209  </div>
210
211  </div>
212
213  </div>
214
215  </div>
216
217  </div>
218
219  </div>
220
221  </div>
222
223  </div>
224
225  </div>
226
227  </div>
228
229  </div>
230
231  </div>
232
233  </div>
234
235  </div>
236
237  </div>
238
239  </div>
240
241  </div>
242
243  </div>
244
245  </div>
246
247  </div>
248
249  </div>
250
251  </div>
252
253  </div>
254
255  </div>
256
257  </div>
258
259  </div>
260
261  </div>
262
263  </div>
264
265  </div>
266
267  </div>
268
269  </div>
270
271  </div>
272
273  </div>
274
275  </div>
276
277  </div>
278
279  </div>
280
281  </div>
282
283  </div>
284
285  </div>
286
287  </div>
288
289  </div>
290
291  </div>
292
293  </div>
294
295  </div>
296
297  </div>
298
299  </div>
300
301  </div>
302
303  </div>
304
305  </div>
306
307  </div>
308
309  </div>
310
311  </div>
312
313  </div>
314
315  </div>
316
317  </div>
318
319  </div>
320
321  </div>
322
323  </div>
324
325  </div>
326
327  </div>
328
329  </div>
330
331  </div>
332
333  </div>
334
335  </div>
336
337  </div>
338
339  </div>
340
341  </div>
342
343  </div>
344
345  </div>
346
347  </div>
348
349  </div>
350
351  </div>
352
353  </div>
354
355  </div>
356
357  </div>
358
359  </div>
360
361  </div>
362
363  </div>
364
365  </div>
366
367  </div>
368
369  </div>
370
371  </div>
372
373  </div>
374
375  </div>
376
377  </div>
378
379  </div>
380
381  </div>
382
383  </div>
384
385  </div>
386
387  </div>
388
389  </div>
390
391  </div>
392
393  </div>
394
395  </div>
396
397  </div>
398
399  </div>
400
401  </div>
402
403  </div>
404
405  </div>
406
407  </div>
408
409  </div>
410
411  </div>
412
413  </div>
414
415  </div>
416
417  </div>
418
419  </div>
420
421  </div>
422
423  </div>
424
425  </div>
426
427  </div>
428
429  </div>
430
431  </div>
432
433  </div>
434
435  </div>
436
437  </div>
438
439  </div>
440
441  </div>
442
443  </div>
444
445  </div>
446
447  </div>
448
449  </div>
450
451  </div>
452
453  </div>
454
455  </div>
456
457  </div>
458
459  </div>
460
461  </div>
462
463  </div>
464
465  </div>
466
467  </div>
468
469  </div>
470
471  </div>
472
473  </div>
474
475  </div>
476
477  </div>
478
479  </div>
480
481  </div>
482
483  </div>
484
485  </div>
486
487  </div>
488
489  </div>
490
491  </div>
492
493  </div>
494
495  </div>
496
497  </div>
498
499  </div>
500
501  </div>
502
503  </div>
504
505  </div>
506
507  </div>
508
509  </div>
510
511  </div>
512
513  </div>
514
515  </div>
516
517  </div>
518
519  </div>
520
521  </div>
522
523  </div>
524
525  </div>
526
527  </div>
528
529  </div>
530
531  </div>
532
533  </div>
534
535  </div>
536
537  </div>
538
539  </div>
540
541  </div>
542
543  </div>
544
545  </div>
546
547  </div>
548
549  </div>
550
551  </div>
552
553  </div>
554
555  </div>
556
557  </div>
558
559  </div>
560
561  </div>
562
563  </div>
564
565  </div>
566
567  </div>
568
569  </div>
570
571  </div>
572
573  </div>
574
575  </div>
576
577  </div>
578
579  </div>
580
581  </div>
582
583  </div>
584
585  </div>
586
587  </div>
588
589  </div>
590
591  </div>
592
593  </div>
594
595  </div>
596
597  </div>
598
599  </div>
600
601  </div>
602
603  </div>
604
605  </div>
606
607  </div>
608
609  </div>
610
611  </div>
612
613  </div>
614
615  </div>
616
617  </div>
618
619  </div>
620
621  </div>
622
623  </div>
624
625  </div>
626
627  </div>
628
629  </div>
630
631  </div>
632
633  </div>
634
635  </div>
636
637  </div>
638
639  </div>
640
641  </div>
642
643  </div>
644
645  </div>
646
647  </div>
648
649  </div>
650
651  </div>
652
653  </div>
654
655  </div>
656
657  </div>
658
659  </div>
660
661  </div>
662
663  </div>
664
665  </div>
666
667  </div>
668
669  </div>
670
671  </div>
672
673  </div>
674
675  </div>
676
677  </div>
678
679  </div>
680
681  </div>
682
683  </div>
684
685  </div>
686
687  </div>
688
689  </div>
690
691  </div>
692
693  </div>
694
695  </div>
696
697  </div>
698
699  </div>
700
701  </div>
702
703  </div>
704
705  </div>
706
707  </div>
708
709  </div>
710
711  </div>
712
713  </div>
714
715  </div>
716
717  </div>
718
719  </div>
720
721  </div>
722
723  </div>
724
725  </div>
726
727  </div>
728
729  </div>
730
731  </div>
732
733  </div>
734
735  </div>
736
737  </div>
738
739  </div>
740
741  </div>
742
743  </div>
744
745  </div>
746
747  </div>
748
749  </div>
750
751  </div>
752
753  </div>
754
755  </div>
756
757  </div>
758
759  </div>
760
761  </div>
762
763  </div>
764
765  </div>
766
767  </div>
768
769  </div>
770
771  </div>
772
773  </div>
774
775  </div>
776
777  </div>
778
779  </div>
780
781  </div>
782
783  </div>
784
785  </div>
786
787  </div>
788
789  </div>
790
791  </div>
792
793  </div>
794
795  </div>
796
797  </div>
798
799  </div>
800
801  </div>
802
803  </div>
804
805  </div>
806
807  </div>
808
809  </div>
810
811  </div>
812
813  </div>
814
815  </div>
816
817  </div>
818
819  </div>
820
821  </div>
822
823  </div>
824
825  </div>
826
827  </div>
828
829  </div>
830
831  </div>
832
833  </div>
834
835  </div>
836
837  </div>
838
839  </div>
840
841  </div>
842
843  </div>
844
845  </div>
846
847  </div>
848
849  </div>
850
851  </div>
852
853  </div>
854
855  </div>
856
857  </div>
858
859  </div>
860
861  </div>
862
863  </div>
864
865  </div>
866
867  </div>
868
869  </div>
870
871  </div>
872
873  </div>
874
875  </div>
876
877  </div>
878
879  </div>
880
881  </div>
882
883  </div>
884
885  </div>
886
887  </div>
888
889  </div>
890
891  </div>
892
893  </div>
894
895  </div>
896
897  </div>
898
899  </div>
900
901  </div>
902
903  </div>
904
905  </div>
906
907  </div>
908
909  </div>
910
911  </div>
912
913  </div>
914
915  </div>
916
917  </div>
918
919  </div>
920
921  </div>
922
923  </div>
924
925  </div>
926
927  </div>
928
929  </div>
930
931  </div>
932
933  </div>
934
935  </div>
936
937  </div>
938
939  </div>
940
941  </div>
942
943  </div>
944
945  </div>
946
947  </div>
948
949  </div>
950
951  </div>
952
953  </div>
954
955  </div>
956
957  </div>
958
959  </div>
960
961  </div>
962
963  </div>
964
965  </div>
966
967  </div>
968
969  </div>
970
971  </div>
972
973  </div>
974
975  </div>
976
977  </div>
978
979  </div>
980
981  </div>
982
983  </div>
984
985  </div>
986
987  </div>
988
989  </div>
990
991  </div>
992
993  </div>
994
995  </div>
996
997  </div>
998
999  </div>
999
1000 </div>
```

```

173
174   <div id="command-window-log" class="terminal-dialog terminal-log
175     options-panel panel" tabindex="0">
176     <div class="options-cont">
177       <div class="options-header">
178         <span><str sid="58"></str></span>
179         
181       </div>
182       <label class="checkcont"><str sid="59"></str><input class="
183         write-timestamps" type="checkbox" name="write-timestamps"
184           value="1" checked><span class="checkmark"></span></label>
185         <button class="save-log"><str sid="60"></str></button>
186       </div>
187     </div>
188
189   <div id="command-window-history" class="terminal-dialog history-
190     cont" tabindex="0">
191     <div class="history-header">
192       <span><str sid="61"></str></span>
193       
194     </div>
195     <ul class="history-panel panel" tabindex="0">
196       </div>
197
198   <div id="script-path-container" class="main-dialog" tabindex="0">
199     <div class="page-cont">
200       <div class="page-header">
201         
203         <span><str sid="62"></str></span>
204       </div>
205       <div class="page-panel panel">
206         <div id="script-path-dialog-msg">
207           <str sid="63"></str> <span id="script-path"></span> <str sid="64">
208             </str>
209         </div>
210         <div id="script-path-dialog-buttons">
211           <button id="script-path-dialog-change-dir"><str sid="65"></str><
212             button>
213             <button id="script-path-dialog-save"><str sid="66"></str></button>
214             <button id="script-path-dialog-run"><str sid="67"></str></button>
215           </div>
216           <br class="clear">
217         </div>
218       </div>
219     </div>
220
221   <div id="paths-container" class="main-dialog" tabindex="0">

```

```
219 <div class="page-cont wide">
220   <div class="page-header">
221     <span><str sid="68"></str></span>
222     
224   </div>
225   <div class="page-panel panel">
226     <ul></ul>
227   </div>
228 </div>
229
230 <div id="help-container" class="main-dialog" tabindex="0">
231   <div class="page-cont wide">
232     <div class="page-header">
233       <span><str sid="2"></str></span>
234       
236   </div>
237   <div class="page-panel panel">
238     <h1><str sid="69"></str></h1>
239     <table style="width:100%">
240       <tr>
241         <th><str sid="70"></str></th>
242         <th><str sid="71"></str></th>
243       </tr>
244       <tr>
245         <td>ESC</td>
246         <td><str sid="72"></str></td>
247       </tr>
248       <tr>
249         <td>Arrow Up</td>
250         <td><str sid="73"></str></td>
251       </tr>
252       <tr>
253         <td>Arrow Down</td>
254         <td><str sid="74"></str></td>
255       </tr>
256       <tr>
257         <td>Page Up</td>
258         <td><str sid="75"></str></td>
259       </tr>
260       <tr>
261         <td>Page Down</td>
262         <td><str sid="76"></str></td>
263       </tr>
264       <tr>
265         <td>Shift + Enter</td>
266         <td><str sid="77"></str></td>
267       </tr>
268       <tr>
269         <td>F3</td>
270         <td><str sid="78"></str></td>
271       </tr>
272 </tr>
```

```

272 <td>F7</td>
273 <td><str sid="79"></str></td>
274 </tr>
275 <tr>
276 <td>Alt + F7</td>
277 <td><str sid="80"></str></td>
278 </tr>
279 <tr>
280 <td>F8</td>
281 <td><str sid="81"></str></td>
282 </tr>
283 <tr>
284 <td>Ctrl + F</td>
285 <td><str sid="82"></str></td>
286 </tr>
287 <tr>
288 <td>Ctrl + H</td>
289 <td><str sid="83"></str></td>
290 </tr>
291 <tr>
292 <td>Ctrl + D</td>
293 <td><str sid="219"></str></td>
294 </tr>
295 <tr>
296 <td>Ctrl + S</td>
297 <td><str sid="84"></str></td>
298 </tr>
299 <tr>
300 <td>Ctrl + L</td>
301 <td><str sid="85"></str></td>
302 </tr>
303 </table>
304 </div>
305 </div>
306 </div>
307
308 <div id="info-container" class="main-dialog" tabindex="0">
309   <div class="page-cont panel">
310     <div class="page-header">
311       <span><str sid="3"></str></span>
312       
313     </div>
314     <div class="page-panel panel">
315       
316       <div class='app-name'>JavaScript LABoratory</div>
317       <div class='app-version'></div>
318       <div class='app-company'>by PR-DC company</div>
319       
320
321       <p><str sid="86"></str></p>
322       <ul>
323         <li><a href="https://www.electronjs.org/">Electron</a></li>
324         <li><a href="https://codemirror.net/">CodeMirror</a></li>
325         <li><a href="https://jquery.com/">JQuery</a></li>

```

```

326   <li><a href="https://github.com/adamschwartz/chrome-tabs">chrome-
327     tabs</a></li>
328   <li><a href="https://plotly.com/javascript/">Plotly.js</a></li>
329   <li><a href="https://mathjs.org/">Math.js</a></li>
330   <li><a href="https://www.mathjax.org/">MathJax</a></li>
331   <li><a href="https://highlightjs.org/">highlight.js</a></li>
332   <li><a href="https://github.com/dhcode/big-json-viewer/">big-json-
333     viewer</a></li>
334   <li><a href="https://threejs.org/">Three.js</a></li>
335   <li><a href="https://pyodide.org/en/stable/">pyodide</a></li>
336   <li><a href="https://www.sympy.org/en/index.html">SymPy</a></li>
337   <li><a href="https://www.cgal.org/">CGAL</a></li>
338   <li><a href="https://eigen.tuxfamily.org/index.php?title=Main_Page
339     ">Eigen</a></li>
340   <li><a href="https://github.com/benfred/fmin">fmin</a></li>
341 </div>
342 </div>
343 <div id="settings-container" class="main-dialog" tabindex="0">
344   <div class="page-cont options-cont panel">
345     <div class="page-header">
346       <span><str sid="9"></str></span>
347       
349     </div>
350     <div class="page-panel panel">
351       <div class="float-select" title-str="25">
352         <select name="set-langauge" class="set-langauge">
353           <option value="en">English</option>
354           <option value="rs">Srpski</option>
355           <option value="rsc">Српски</option>
356         </select>
357         <label class="float-label" for="set-langauge"><str sid="16"></str>
358           </label>
359       </div>
360       <div class="float-input" title-str="223">
361         <input autocomplete="off" type="text" name="N-history-max" class="N-history-max"
362           placeholder="20" value="20">
363         <label class="float-label" for="N-history-max"><str sid="223"></str></label>
364       </div>
365       <button class="change-settings"><str sid="57"></str></button>
366     </div>
367   <div id="sandbox-stats-popup" style="display: none;">
368     <div id="sandbox-stats-header">
369       <str sid="91"></str>
370     </div>
371     <div class="sandbox-stats-cont">
372       Required modules
373       <div class="sandbox-stats-val" id="sandbox-required-modules-num">0</

```

```

374         div>
375     </div>
376     <div class="sandbox-stats-cont">
377       Promises
378       <div class="sandbox-stats-val" id="sandbox-promises-num">0</div>
379     </div>
380     <div class="sandbox-stats-cont">
381       Timeouts
382       <div class="sandbox-stats-val" id="sandbox-timeouts-num">0</div>
383     </div>
384     <div class="sandbox-stats-cont">
385       Immediate
386       <div class="sandbox-stats-val" id="sandbox-immediates-num">0</div>
387     </div>
388     <div class="sandbox-stats-cont">
389       Intervals
390       <div class="sandbox-stats-val" id="sandbox-intervals-num">0</div>
391     </div>
392     <div class="sandbox-stats-cont">
393       Animation frames
394       <div class="sandbox-stats-val" id="sandbox-animation-frames-num">0</div>
395     </div>
396     <div class="sandbox-stats-cont">
397       Idle callbacks
398       <div class="sandbox-stats-val" id="sandbox-idle-callbacks-num">0</div>
399     </div>
400     <div class="popup-triangle"></div>
401   </div>
402   <div id="status-container">
403     <div id="sandbox-stats-icon" class="ready"></div>
404
405     <div id="status">
406       <str sid="87"></str>
407     </div>
408     <div id="status-icons">
409
410     </div>
411   </div>
412   <script>if (typeof module === 'object') {window.module = module; module = undefined;}</script>
413
414   <script type="text/javascript" src="../../lib/jquery-3.7.0/jquery-3.7.0.min.js"></script>
415   <script type="text/javascript" src="../../lib/jshint-2.13.0/jshint-2.13.0.js"></script>
416   <script type="text/javascript" src="../../lib/highlight-11.0.1/highlight-11.0.1.min.js"></script>
417   <script type="text/javascript" src="../../lib/tex-mml-ctml-3.2.0/tex-mml-ctml-3.2.0.js"></script>
418
419   <script type="text/javascript" src="../../lib/codemirror-5.49.2/lib/codemirror.js"></script>
420   <script type="text/javascript" src="../../lib/codemirror-5.49.2/addon/edit/

```

```

421   matchbrackets.js"></script>
422 <script type="text/javascript" src="../../lib/codemirror-5.49.2/addon/lint/
423   lint.js"></script>
424 <script type="text/javascript" src="../../lib/codemirror-5.49.2/addon/hint/
425   show-hint.js"></script>
426 <script type="text/javascript" src="../../lib/codemirror-5.49.2/addon/search/
427   searchcursor.js"></script>
428 <script type="text/javascript" src="../../lib/codemirror-5.49.2/addon/search/
429   match-highlighter.js"></script>
430 <script type="text/javascript" src="../../lib/codemirror-5.49.2/mode/
431   javascript/javascript.js"></script>
432
433 <script type="text/javascript" src="../../js/code/custom-javascript-hint.js">
434   </script>
435
436 <script type="text/javascript" src="../../js/main/init-main.js"></script>
437 </body>
438 </html>
```

Listing 31 - main.html

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <title>Plotlyjs Plot - JSLAB | PR-DC</title>
6      <meta http-equiv="Content-Security-Policy" content="script-src * 'self' ,
7          unsafe-inline , 'unsafe-eval' ; worker-src 'self' blob:;" />
8      <link rel="stylesheet" type="text/css" href="../../css/basic.css" />
9      <link rel="stylesheet" type="text/css" href="../../font/roboto.css" />
10     <link rel="stylesheet" type="text/css" href="../../font/LatinModern.css" />
11
12     <style id="dynamic-style-rules"></style>
13   </head>
14   <body>
15     <div id="plot-cont"></div>
16
17     <script type="text/javascript" src="../../lib/plotly-2.24.2/plotly-2.24.2.min.
18       .js"></script>
19   </body>
20 </html>
```

Listing 32 - plotlyjs.html

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5    </head>
6    <body>
7      <script type="text/javascript" src="../../lib/luxon-3.4.4/luxon-3.4.4.min.js"
8          "></script>
9      <script type="text/javascript" src="../../lib/math-11.8.2/math-11.8.2.min.js"
10         "></script>
11      <script type="text/javascript" src="../../lib/sprintf-1.1.3/sprintf-1.1.3.min.
12        .js"></script>
```

```

10 <script type="text/javascript" src="../../lib/Cesium-1.124/Cesium.js"></script>
11
12 <script>window.process.browser = 'Electron';</script>
13 <script type="text/javascript" src="../../lib/sympy-0.26.2/pyodide.js"></script>
14 <script type="text/javascript" src="../../node_modules/fmin/build/fmin.min.js"></script>
15 <script type="text/javascript" src="../../js/sandbox/init-sandbox-worker.js"></script>
16 </body>
17 </html>
```

Listing 33 - sandbox-worker.html

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="Content-Security-Policy" content="script-src * 'self' 'unsafe-inline' 'unsafe-eval' worker-src 'self' blob:;" />
6   </head>
7   <body>
8     <script type="text/javascript" src="../../lib/luxon-3.4.4/luxon-3.4.4.min.js"></script>
9     <script type="text/javascript" src="../../lib/math-11.8.2/math-11.8.2.min.js"></script>
10    <script type="text/javascript" src="../../lib/sprintf-1.1.3/sprintf-1.1.3.min.js"></script>
11    <script type="text/javascript" src="../../lib/Cesium-1.124/Cesium.js"></script>
12
13 <script>window.process.browser = 'Electron';</script>
14 <script type="text/javascript" src="../../lib/sympy-0.26.2/pyodide.js"></script>
15 <script type="text/javascript" src="../../node_modules/fmin/build/fmin.min.js"></script>
16 <script type="text/javascript" src="../../js/sandbox/init-sandbox.js"></script>
17 </body>
18 </html>
```

Listing 34 - sandbox.html

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="UTF-8">
5     <title>THREE - JSLAB | PR-DC</title>
6     <meta http-equiv="Content-Security-Policy" content="script-src * 'self' 'unsafe-inline' 'unsafe-eval' worker-src 'self' blob:;" />
7     <link rel="stylesheet" type="text/css" href="../../css/three.css" />
8     <link rel="stylesheet" type="text/css" href="../../font/roboto.css" />
9   </head>
10  <body>
```

```

12 <div id="scene-cont"></div>
13
14 <script type="text/javascript" src="../lib/hammer-2.0.8/hammer.min.js"></
15   script>
16 <script type="text/javascript" src="../lib/anime-3.2.1/anime-3.2.1.min.js"></
17   script>
18 <script type="text/javascript" src="../lib/tween.js-23.1.1/tween-23.1.1.js"></
19   script>
20 <script type="text/javascript" src="../lib/inflate-0.3.1/inflate-0.3.1.min.js"></
21   script>
22 <script type="importmap">
23 {
24   "imports": {
25     "three": "../lib/three.js-r162/build/three.module.js",
26     "three/addons/": "../lib/three.js-r162/examples/jsm/"
27   }
28 </script>
29 </body>
30 </html>
```

Listing 35 - three.html

6 js

```

1 /**
2  * @file Javascript helper functions
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 var { shell } = require('electron');
9
10 /**
11  * Call function when document is ready
12  * @param {function} fn - function which is called when document is ready.
13 */
14 global.ready = function(fn) {
15   if(document.readyState != 'loading') {
16     fn();
17   } else {
18     document.addEventListener('DOMContentLoaded', fn);
19   }
20 };
21
22 /**
23  * Prevents redirect
24 */
25 global.preventDefault = function() {
```

```

26  var links = $('a');
27  links.each(function() {
28      if (!$(this).hasClass('external-link')) {
29          $(this).addClass('external-link');
30          $(this).click(function(e) {
31              e.preventDefault();
32              shell.openExternal(e.target.href);
33              return false;
34          });
35      }
36  });
37 };
38
39 /**
40 * Get process arguments
41 */
42 if(process.type === 'browser' || global.is_worker) {
43     global.process_arguments = process.argv;
44 } else {
45     var { ipcRenderer } = require('electron');
46     global.process_arguments =
47         ipcRenderer.sendSync("sync-message", "get-process-arguments");
48 }

```

Listing 36 - helper.js

```

1 /**
2  * @file JSLAB init app
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7 "use strict";
8
9 const { app } = require('electron');
10 global.app_path = app.getAppPath().replace(/\\js\\?$/,'');
11
12 const helper = require("./helper.js");
13 const { PRDC_APP_CONFIG } = require('../config/config');
14
15 const config = new PRDC_APP_CONFIG();
16 global.config = config;
17
18 const { PRDC_JSLAB_MAIN } = require("./main");
19
20 // Start main
21 const main = new PRDC_JSLAB_MAIN();

```

Listing 37 - init.js

```

1 /**
2  * @file JSLAB language module
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */

```

```
7
8 const fs = require('fs');
9 const Store = require('electron-store');
10
11 const store = new Store();
12
13 /**
14 * Class for JSLAB language.
15 */
16 class PRDC_JSLAB_LANGUAGE {
17
18 /**
19 * Create JSLAB language object.
20 */
21 constructor() {
22 // Variables
23 var obj = this;
24
25 this.lang_index = store.get('lang_index');
26 if(!this.lang_index) {
27   this.lang_index = 0;
28 }
29
30 this.onLanguageChange = function() {};
31
32 var style = document.createElement('style');
33 document.head.appendChild(style);
34 this.lang_styles = style.sheet;
35
36 // Get language strings
37 this.s = JSON.parse(fs.readFileSync(app_path + '/config/lang.json'));
38
39 document.querySelectorAll('str').forEach(function(el) {
40   var id = el.getAttribute('sid');
41   el.innerHTML = obj.string(id);
42 });
43
44 this.lang = config.langs[this.lang_index];
45 this.set(this.lang);
46 }
47
48 /**
49 * Sets the application's current language and updates the UI accordingly.
50 * Saves the new language preference for future sessions.
51 * @param {string} lang The language code to set as the current language.
52 */
53 set(lang) {
54   var idx = config.langs.findIndex(function(e) {
55     return e === lang;
56   });
57   if(idx >= 0) {
58     this.lang = lang;
59     this.lang_index = idx;
60     if(this.lang_styles.cssRules.length) {
61       this.lang_styles.deleteRule(0);
```

```

62      }
63      this.lang_styles.insertRule("lang." + lang + " { display: initial }", 0);
64      this.update('html', false);

65
66      // Save language
67      store.set('lang_index', this.lang_index);
68      this.onLanguageChange(lang);
69  }
70 }

72 /**
73 * Dynamically updates text strings within the specified HTML container to
74 * the current language.
75 * Can optionally update placeholders and titles for input and option
76 * elements.
77 * @param {String} cont The selector for the container whose text strings
78 * will be updated. Defaults to 'html'.
79 * @param {boolean} flag If true, updates the container and child elements
80 * with dynamic language strings.
81 */
82 update(cont = 'html', flag = true) {
83   var obj = this;
84
85   if(flag) {
86     document.querySelectorAll(cont + ' str').forEach(function(el) {
87       var id = el.getAttribute('sid');
88       el.innerHTML = obj.string(id);
89     });
90   }
91
92   document.querySelectorAll(cont + ' input[str]').forEach(function(el) {
93     var id = el.getAttribute('str');
94     if(id in obj.s) {
95       el.setAttribute('placeholder', obj.s[id][obj.lang]);
96     }
97   });
98
99   document.querySelectorAll(cont + ' option[str]').forEach(function(el) {
100    var id = el.getAttribute('str');
101    if(id in obj.s) {
102      el.textContent = obj.s[id][obj.lang];
103    }
104  });
105
106  document.querySelectorAll(cont + ' [title-str]').forEach(function(el) {
107    var id = el.getAttribute('title-str');
108    if(id in obj.s) {
109      el.setAttribute('title', obj.s[id][obj.lang]);
110    }
111  });

112 /**
113 * Retrieves the specified language string in all available languages,
114 * wrapped in language-specific <lang> tags.

```

```

112  * @param {number} id The identifier of the string to retrieve.
113  * @returns {HTML} HTML string containing the text in all available
114  * languages.
115  */
116  string(id) {
117  var obj = this;
118  var msg = '';
119  if(id in this.s) {
120    config.langs.forEach(function(lang) {
121      msg += '<lang class="' + lang + '">' + obj.s[id][lang] + '</lang>';
122    });
123  } else {
124    config.langs.forEach(function(lang) {
125      msg += '<lang class="' + lang + '"></lang>';
126    });
127  }
128  return msg;
129 }
130 /**
131 * Retrieves the current language string for the specified identifier.
132 * @param {number} id The identifier of the string to retrieve.
133 * @returns {String} The string corresponding to the current language.
134 */
135 currentString(id) {
136  if(id in this.s) {
137    return this.s[id][this.lang];
138  } else {
139    return '';
140  }
141 }
142 /**
143 * Sets a callback function to be executed when the language is changed.
144 * @param {Function} callback The callback function to be executed on
145 * language change.
146 */
147 setOnLanguageChange(callback) {
148  if(typeof callback == 'function') {
149    this.onLanguageChange = callback;
150  }
151 }
152 }
153
154 exports.PRDC_JSLAB_LANGUAGE = PRDC_JSLAB_LANGUAGE;

```

Listing 38 - language.js

```

1 /**
2  * @file JSLAB main script
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 const { app, BrowserWindow, ipcMain, dialog, powerSaveBlocker, shell,

```

```

9  MenuItem , desktopCapturer , screen } = require( 'electron' );
10
11 const contextMenu = require( 'electron-context-menu' );
12 const fs = require( 'fs' );
13 const os = require( 'os' );
14 const Store = require( 'electron-store' );
15
16 const { PRDC_APP_LOGGER } = require( '../lib/PRDC_APP_LOGGER/PRDC_APP_LOGGER' );
17
18 var app_version = process.env.npm_package_version;
19
20 app.commandLine.appendSwitch( 'max-active-webgl-contexts' , config.
21   MAX_ACTIVE_WEBGL_CONTEXTS );
22
23 /**
24 * Class for flight control app.
25 */
26 class PRDC_JSLAB_MAIN {
27
28 /**
29 * Create app.
30 */
31 constructor() {
32   var obj = this;
33   this.show_inspect_element = false;
34   if( !config.PRODUCTION && config.DEBUG ) {
35     this.show_inspect_element = true;
36   }
37
38   this.store = new Store();
39   this.debounce_save_win_time = config.WIN_SAVE_DEBOUNCE_TIME;
40   this.debounce_save_win_bounds = [];
41
42   this.heartbeat_interval;
43   this.win_main;
44   this.win_editor;
45   this.win_sandbox;
46   this.win_opacity = 0;
47   this.editor_close_ready = false;
48   this.stop_loop_in = false;
49
50   this.app_icon = app_path + '/icons/icon.ico'; // png to ico https://
51   // icoconvert.com/
52   this.is_app_quitting = false;
53   this.known_paths = [ 'home' , 'appData' , 'userDat' , 'sessionData' ,
54   'temp' , 'exe' , 'module' , 'desktop' , 'documents' , 'downloads' ,
55   'music' , 'pictures' , 'videos' , 'recent' , 'logs' , 'crashDumps' ];
56
57   if( os.platform() == 'linux' ) {
58     this.app_icon = app_path + '/icons/icon.png';
59   }
60
61   // Create folder for app
62   this.app_folder = app.getPath( 'documents' )+ '\\ '+app.getName();

```

```

61   if (!fs.existsSync(this.app_folder)) {
62     fs.mkdirSync(this.app_folder);
63   }
64
65   // Start log
66   this.log_file = this.app_folder+'\\'+app.getName()+'.'+log';
67   this.app_logger = new PRDC_APP_LOGGER(this.log_file);
68
69   if(config.REPORT_CRASH) {
70     const Bugsnag = require('@bugsnag/electron');
71     Bugsnag.start({ apiKey: config.BUGSNAG_API_KEY });
72   }
73
74   function canToggleComment(parameters) {
75     if(parameters.formControlType === 'text-area' &&
76       parameters.pageURL.endsWith('/editor.html')) {
77       return true;
78     }
79     return false;
80   }
81
82   // Context menu
83   contextMenu({
84     append: function(default_actions, parameters) {
85       return [
86         new MenuItem({
87           label: 'Toggle Comment',
88           click: function() {
89             obj.win_editor.send('EditorWindow', 'toggle-comment');
90           },
91           visible: canToggleComment(parameters)
92         }),
93         new MenuItem({
94           role: 'selectAll',
95           label: 'Select All',
96           visible: parameters.editFlags.canSelectAll
97         }),
98         new MenuItem({
99           role: 'delete',
100          label: 'Delete',
101          visible: parameters.editFlags.canDelete
102        })
103      ];
104    },
105    showLookUpSelection: false,
106    showSearchWithGoogle: false,
107    showCopyImage: false,
108    showInspectElement: obj.show_inspect_element
109  });
110
111   // Disable renderer backgrounding
112   app.commandLine.appendSwitch('disable-renderer-backgrounding');
113   app.commandLine.appendSwitch("disable-http-cache");
114
115   // Prevent sleep

```

```
116 powerSaveBlocker.start('prevent-app-suspension');
```

```
117
118 // This method will be called when Electron has finished
119 // initialization and is ready to create browser windows.
120 // Some APIs can only be used after this event occurs.
121 app.whenReady().then(function(){
122     obj.createWindows();
123
124     app.on('activate', function() {
125         // On macOS it's common to re-create a window in the app when the
126         // dock icon is clicked and there are no other windows open.
127         if(BrowserWindow.getAllWindows().length === 0) obj.createWindows();
128     });
129 });
130
131 // Quit when all windows are closed, except on macOS. There, it's common
132 // for applications and their menu bar to stay active until the user quits
133 // explicitly with Cmd + Q.
134 app.on('window-all-closed', function () {
135     if(process.platform !== 'darwin') app.quit();
136 });
137 app.allowRendererProcessReuse = false;
138 }
139
140 /**
141 * Create all windows
142 */
143 createWindows() {
144     // Create the main window
145     this.creatMainWindow();
146
147     // Create the sandbox window
148     this.createSandboxWindow();
149
150     // Create the editor window
151     this.createEditorWindow();
152
153     // Set handlers
154     this.setHandlers();
155
156     // Handle IPC messages
157     this.handleMessages();
158 }
159
160 /**
161 * Create main window
162 */
163 creatMainWindow() {
164     var obj = this;
165     var options = {
166         title: 'JSLAB',
167         minWidth: 720,
168         minHeight: 500,
169         icon: this.app_icon, // png to ico https://icoconvert.com/
170         show: false,
```

```

171   frame: false ,
172   backgroundColor: '#ffffff' ,
173   opacity: this.win_opacity ,
174   webPreferences: {
175     nodeIntegration: true ,
176     nodeIntegrationInWorker: true ,
177     contextIsolation: false ,
178     backgroundThrottling: false
179   }
180 };
181 options = this.getWindowBounds( 'mainWinBounds' , options );
182 this.win_main = new BrowserWindow( options );
183
184 // Hide menu
185 this.win_main.setMenu( null );
186
187 // Maximize
188 if(options.maximize) {
189   this.win_main.maximize();
190 }
191
192 // and load the main.html of the app
193 this.win_main.loadFile(app_path + '/html/main.html');
194
195 // Events
196 this.win_main.on("resize" , function() { obj.saveWindowBounds(obj.win_main ,
197   'mainWinBounds'); });
198 this.win_main.on("move" , function() { obj.saveWindowBounds(obj.win_main ,
199   'mainWinBounds'); });
200 this.win_main.on("maximize" , function() { obj.saveWindowBounds(obj .
201   win_main , 'mainWinBounds'); });
202 this.win_main.on("unmaximize" , function() { obj.saveWindowBounds(obj .
203   win_main , 'mainWinBounds'); });
204
205 // Show window when ready
206 this.win_main.once('ready-to-show' , function() {
207   obj.win_main.show();
208   if(config.DEBUG) {
209     obj.openDevTools(obj.win_main);
210   }
211 });
212
213 this.win_main.webContents.on( 'render-process-gone' , function(event ,
214   details) {
215   obj.app_logger.logMessage(config.DEBUG_RENDER_GONE_ERROR, config .
216     LOG_RENDER_GONE_ERROR, config.LOG_CODES[ 'render-gone-error' ] ,
217     Render gone error' , 'Main window render gone error:' + JSON.
218     stringify(event) + ' ' + JSON.stringify(details));
219   app.exit();
220   app.relaunch();
221 });
222
223 // Close all windows
224 this.win_main.on('close' , function(e) {
225   e.preventDefault();

```

```
218     obj.win_main.webContents.executeJavaScript('win_main.close();');
219 }
220 }
221 /**
222 * Create sandbox window
223 */
224 createSandboxWindow() {
225     var obj = this;
226     this.win_sandbox = new BrowserWindow({
227         title: 'Sandbox | JSLAB',
228         minWidth: 720,
229         minHeight: 500,
230         icon: this.app_icon, // png to ico https://icoconvert.com/
231         show: false,
232         backgroundColor: '#ffffff',
233         opacity: this.win_opacity,
234         webPreferences: {
235             nodeIntegration: true,
236             nodeIntegrationInWorker: true,
237             contextIsolation: false,
238             backgroundThrottling: false
239         }
240     });
241 }
242
243 // Hide menu
244 this.win_sandbox.setMenu(null);
245
246 // and load the index.html of the app
247 this.win_sandbox.loadFile(app_path + '/html/sandbox.html');
248
249 if(config.DEBUG) {
250     // Show dev tools
251     this.openDevTools(this.win_sandbox);
252 }
253
254 // Sub windows
255 this.win_sandbox.webContents.setWindowOpenHandler(function(handler) {
256     if(handler.url.startsWith('file://')) {
257         return {
258             action: 'allow',
259             overrideBrowserWindowOptions: {
260                 minWidth: 250,
261                 minHeight: 50,
262                 icon: obj.app_icon, // png to ico https://icoconvert.com/
263                 show: false,
264                 backgroundColor: '#ffffff',
265                 opacity: obj.win_opacity,
266                 webPreferences: {
267                     nodeIntegration: true,
268                     nodeIntegrationInWorker: true,
269                     contextIsolation: false,
270                     backgroundThrottling: false
271                 }
272             }
273         }
274     }
275 }
```



```
320         }
321     });
322 }
323
324 /**
325 * Create editor window
326 */
327 createEditorWindow() {
328     var obj = this;
329     var options = {
330         title: 'Editor | JSLAB',
331         minWidth: 720,
332         minHeight: 500,
333         icon: this.app_icon, // png to ico https://icoconvert.com/
334         show: false,
335         frame: false,
336         backgroundColor: '#ffffff',
337         opacity: this.win_opacity,
338         webPreferences: {
339             nodeIntegration: true,
340             nodeIntegrationInWorker: true,
341             contextIsolation: false,
342             backgroundThrottling: false
343         }
344     };
345     options = this.getWindowBounds('editorWinBounds', options);
346     this.win_editor = new BrowserWindow(options);
347
348     // Hide menu
349     this.win_editor.setMenu(null);
350
351     // Maximize
352     if(options.maximize) {
353         this.win_editor.maximize();
354     }
355
356     // Hide menu
357     this.win_editor.setMenu(null);
358
359     // and load the index.html of the app
360     this.win_editor.loadFile(app_path + '/html/editor.html');
361
362     // Maximize
363     if(options.maximize) {
364         this.win_editor.maximize();
365     }
366
367     // Events
368     this.win_editor.on("resize", function() { obj.saveWindowBounds(obj.
369         win_editor, 'editorWinBounds'); });
370     this.win_editor.on("move", function() { obj.saveWindowBounds(obj.
371         win_editor, 'editorWinBounds'); });
372     this.win_editor.on("maximize", function() { obj.saveWindowBounds(obj.
373         win_editor, 'editorWinBounds'); });
374     this.win_editor.on("unmaximize", function() { obj.saveWindowBounds(obj.
```

```
372     win_editor, 'editorWinBounds'); });
373
374 // Hide window when ready
375 this.win_editor.hide();
376
377 this.win_editor.on('close', function(e) {
378   if(!obj.is_app_quitting) {
379     e.preventDefault();
380     obj.fadeWindowOut(obj.win_editor, 0.1, 10);
381     setTimeout(function() {
382       obj.win_editor.hide();
383     }, 100);
384   } else if(!obj.editor_close_ready) {
385     e.preventDefault();
386   }
387 });
388
389 this.win_editor.webContents.on('render-process-gone', function(event,
390   details) {
391   obj.app_logger.logMessage(config.DEBUG_RENDER_GONE_ERROR, config.
392     LOG_RENDER_GONE_ERROR, config.LOG_CODES['render-gone-error'],
393     'Render gone error', 'Editor window render gone error:' + JSON.
394     stringify(event) + ', ' + JSON.stringify(details));
395 });
396
397 /**
398 * Handle IPC messages
399 */
400 handleMessages() {
401   var obj = this;
402
403   // For MainProcess
404   ipcMain.handle('get-completions', function(e, data) {
405     obj.win_sandbox.send('SandboxWindow', 'get-completions', data);
406     return new Promise(function(resolve) {
407       ipcMain.once('completions-' + data[0], function(e, data) {
408         resolve(data);
409       });
410     });
411   });
412
413   ipcMain.handle('dialog', function(e, method, params) {
414     if(!params || params && !params.hasOwnProperty('icon')) {
415       if(!params) {
416         params = {};
417       }
418       params.icon = obj.app_icon;
419     }
420     return dialog[method](params);
421   });
422
423   ipcMain.on('get-desktop-sources', async function(e) {
424     e.returnValue = await desktopCapturer.getSources({ types: ['screen', 'window'] });
425   });
426 }
```

```
421 });
422
423 ipcMain.on('dialog', function(e, method, params) {
424   if(!params || params && !params.hasOwnProperty('icon')) {
425     if(!params) {
426       params = {};
427     }
428     params.icon = obj.app_icon;
429   }
430   e.returnValue = dialog[method](params);
431 });
432
433 ipcMain.on('sync-message', function(e, action, data) {
434   var retval;
435   switch(action) {
436     case 'get-app':
437       retval = { 'name': app.getName(), 'version': app_version, 'path':
438                 app_path, 'exe_path': app.getPath('exe') };
439       break;
440     case 'get-app-name':
441       retval = app.getName();
442       break;
443     case 'get-app-path':
444       retval = app_path;
445       break;
446     case 'get-app-version':
447       retval = app_version;
448       break;
449     case 'get-platform':
450       retval = os.platform();
451       break;
452     case 'get-debug-flag':
453       retval = config.DEBUG;
454       break;
455     case 'get-path':
456       if(obj.known_paths.includes(data)) {
457         retval = app.getPath(data);
458       } else if(data === 'root') {
459         retval = app_path;
460       } else if(data === 'includes') {
461         retval = app_path + '\\includes';
462       } else {
463         retval = true;
464       }
465       break;
466     case 'get-log-file':
467       retval = obj.log_file;
468       break;
469     case 'is-maximized-win':
470       retval = BrowserWindow.fromWebContents(e.sender).isMaximized();
471       break;
472     case 'check-stop-loop':
473       retval = obj.stop_loop_in;
474       break;
475     case 'reset-stop-loop':
```

```

475     obj.stop_loop_in = false;
476     retval = true;
477     break;
478   case 'get-process-arguments':
479     retval = process.argv;
480     break;
481   case 'call-sub-win-method':
482     var [id, method, ...args] = data;
483     if (obj.sandbox_sub_wins.hasOwnProperty(id)) {
484       retval = obj.sandbox_sub_wins[id][method](...args);
485     } else {
486       retval = false;
487     }
488     break;
489   case 'open-sub-win-devtools':
490     if (obj.sandbox_sub_wins.hasOwnProperty(data)) {
491       obj.openDevTools(obj.sandbox_sub_wins[data]);
492       retval = true;
493     } else {
494       retval = false;
495     }
496     break;
497   case 'reset-app':
498     app.exit();
499     app.relaunch();
500     break;
501   case 'reset-sandbox':
502     obj.sandbox_reload_active = true;
503     obj.win_sandbox.destroy();
504     Object.keys(obj.sandbox_sub_wins).forEach(function(wid) {
505       obj.sandbox_sub_wins[wid].destroy();
506       delete obj.sandbox_sub_wins[wid];
507     });
508     obj.createSandboxWindow();
509     break;
510   default:
511     retval = true;
512     break;
513   }
514   e.returnValue = retval;
515 });
516
517 ipcMain.on('MainProcess', function(e, action, data) {
518   switch(action) {
519     case 'show-dev-tools':
520       // Show DevTools
521       obj.openDevTools(BrowserWindow.fromWebContents(e.sender));
522       break;
523     case 'show-sandbox-dev-tools':
524       // Show Sandbox DevTools
525       obj.openDevTools(obj.win_sandbox);
526       break;
527     case 'open-dir':
528     case 'open-folder':
529     case 'show-dir':

```

```
530     case 'show-folder':
531         shell.openPath(data);
532         break;
533     case 'show-file-in-folder':
534     case 'show-file-in-dir':
535         shell.showItemInFolder(data);
536         break;
537     case 'focus-win':
538         e.sender.focus();
539         break;
540     case 'fade-in-win':
541         obj.fadeWindowIn(BrowserWindow.fromWebContents(e.sender), 0.1, 10);
542         break;
543     case 'fade-out-win':
544         obj.fadeWindowIn(BrowserWindow.fromWebContents(e.sender), 0.1, 10);
545         break;
546     case 'close-win':
547         BrowserWindow.fromWebContents(e.sender).close();
548         e.sender.destroy();
549         break;
550     case 'close-app':
551         if(obj.win_main !== undefined && !obj.win_main.isDestroyed()) {
552             obj.win_main.destroy();
553         }
554         if(obj.win_editor !== undefined && !obj.win_editor.isDestroyed()) {
555             obj.is_app_quitting = true;
556             obj.win_editor.send('EditorWindow', 'close-all');
557         }
558         if(obj.win_sandbox !== undefined && !obj.win_sandbox.isDestroyed())
559         {
560             obj.win_sandbox.destroy();
561         }
562         break;
563     case 'set-fullscreen':
564         BrowserWindow.fromWebContents(e.sender).setFullScreen(data);
565         BrowserWindow.fromWebContents(e.sender).maximize();
566         break;
567     case 'maximize-win':
568         BrowserWindow.fromWebContents(e.sender).maximize();
569         break;
570     case 'restore-win':
571         BrowserWindow.fromWebContents(e.sender).restore();
572         break;
573     case 'minimize-win':
574         BrowserWindow.fromWebContents(e.sender).minimize();
575         break;
576     case 'show-editor':
577         // Show editor
578         obj.showEditor();
579         break;
580     case 'close-editor':
581         // Close editor
582         obj.editor_close_ready = true;
583         obj.win_editor.close();
584         break;
```

```

584     case 'capture-page':
585       obj.win_main.webContents.capturePage(undefined, {
586         stayHidden: true,
587         stayAwake: true
588       }).then(function(img) {
589         var size = img.getSize();
590         obj.win_main.webContents.send('streamer', 'captured-page', {buffer
591           : img.toBitmap().buffer, width: size.width, height: size.
592           height});
593       });
594       break;
595     case 'take-screenshot':
596       obj.win_main.webContents.capturePage(undefined, {
597         stayHidden: true,
598         stayAwake: true
599       }).then(function(img) {
600         if(data) {
601           img = img.crop(data);
602         }
603         obj.win_main.webContents.send('gui', 'screenshot', {buffer: img.
604           toPNG()});
605       });
606       break;
607     case 'close-log':
608       obj.app_logger.closeLog();
609       break;
610     case 'code-evaluated':
611       Object.keys(obj.sandbox_sub_wins).forEach(function(wid) {
612         var win = obj.sandbox_sub_wins[wid];
613         if(win.forClose) {
614           setTimeout(function() {
615             win.destroy();
616             delete obj.sandbox_sub_wins[wid];
617           }, 500);
618         }
619       });
620       break;
621     case 'set-win-size':
622       obj.win_main.setSize(data[0], data[1]);
623       break;
624     case 'app-relaunch':
625       app.exit();
626       app.relaunch();
627       break;
628   });
629   // For MainWindow
630   ipcMain.on('MainWindow', function(e, action, data) {
631     if(!obj.win_main.isDestroyed()) {
632       switch(action) {
633         default:
634           obj.win_main.send('MainWindow', action, data);
635           break;
636       }
637     }
638   });
639 
```

```

636     }
637   });
638
639   // For EditorWindow
640   ipcMain.on('EditorWindow', function(e, action, data) {
641     switch(action) {
642       case 'open-script':
643         // Open file in editor
644         obj.showEditor();
645         obj.win_editor.send('EditorWindow', 'open-script', data);
646         break;
647       default:
648         // Other actions
649         obj.win_editor.send('EditorWindow', action, data);
650         break;
651     }
652   });
653
654   // For SandboxWindow
655   ipcMain.on('SandboxWindow', function(e, action, data) {
656     if(action == 'stop-loop') {
657       obj.stop_loop_in = data;
658     }
659     switch(action) {
660       default:
661         obj.win_sandbox.send('SandboxWindow', action, data);
662         break;
663     }
664   });
665 }

666 /**
667 * Show editor window
668 */
669 showEditor() {
670   this.win_editor.show();
671   this.win_editor.focus();
672   this.fadeWindowIn(this.win_editor, 0.1, 10);
673
674   if(config.DEBUG) {
675     if(!this.win_editor.devtools_win || !this.win_editor.devtools_win.
676       isVisible()) {
677       this.openDevTools(this.win_editor);
678     }
679   }
680 }

681 /**
682 * Get window bounds
683 * @param {string} store_key - key used for storage.
684 * @param {object} options - options for window.
685 * @returns {object} window bounds.
686 */
687 getWindowBounds(store_key, options) {
688   var stored_options = this.store.get(store_key);

```

```

690     if(stored_options) {
691         var bounds = stored_options.bounds;
692         var area = screen.getDisplayMatching(bounds).workArea;
693         options.maximize = stored_options.maximize;
694         if(
695             bounds.x >= area.x &&
696             bounds.y >= area.y &&
697             bounds.x + bounds.width <= area.x + area.width &&
698             bounds.y + bounds.height <= area.y + area.height
699         ) {
700             options.x = bounds.x;
701             options.y = bounds.y;
702         }
703         if(bounds.width <= area.width || bounds.height <= area.height) {
704             options.width = bounds.width;
705             options.height = bounds.height;
706         }
707     }
708     return options;
709 }
710
711 /**
712 * Save window bounds
713 * @param {BrowserWindow} win - browser window.
714 * @param {string} store_key - key used for storage.
715 */
716 saveWindowBounds(win, store_key) {
717     var obj = this;
718     if(this.debounce_save_win_bounds[store_key]) {
719         clearTimeout(this.debounce_save_win_bounds[store_key]);
720     }
721     this.debounce_save_win_bounds[store_key] = setTimeout(function() {
722         obj.debounce_save_win_bounds[store_key] = undefined;
723         var options = {};
724         options.bounds = win.getNormalBounds();
725         options.maximize = win.isMaximized();
726         obj.store.set(store_key, options);
727     }, this.debounce_save_win_time);
728 }
729
730 /**
731 * Configures session-wide handlers for certificate verification, permission
732 * checks,
733 * device permission handling, and USB protected classes handling. These
734 * handlers ensure
735 * the application's security and user privacy.
736 */
737 setHandlers() {
738     this.win_main.webContents.session.setCertificateVerifyProc(function(
739         request, callback) {
740         callback(0);
741     });
742     this.win_main.webContents.session.setPermissionCheckHandler(function() {

```

```
742     return true;
743 });
744
745 this.win_main.webContents.session.setDevicePermissionHandler(function() {
746     return true;
747 });
748
749 this.win_main.webContents.session.setUSBProtectedClassesHandler(function()
750     {
751         return [];
752     });
753 }
754
755 /**
756 * Opens the Developer Tools for the specified Electron BrowserWindow in a
757 * detached window.
758 * @param {BrowserWindow} win - The Electron 'BrowserWindow' instance for
759 * which to open the Developer Tools.
760 */
761 openDevTools(win) {
762     if (win.webContents.isDevToolsOpened()) {
763         win.webContents.closeDevTools();
764     }
765     win.webContents.openDevTools({mode: 'undocked'});
766     win.webContents.once('devtools-opened', function() {
767         win.devToolsWebContents.focus();
768     });
769 }
770
771 /**
772 * Gradually increases the opacity of a given window until it is fully
773 * opaque.
774 * @param {BrowserWindow} win - The browser window to fade in.
775 * @param {number} step - The incremental step of opacity change. Default is
776 * 0.1.
777 * @param {number} dt - The time interval in milliseconds between opacity
778 * changes. Default is 10.
779 * @returns {number} The interval identifier for the fade-in operation.
780 */
781 fadeWindowIn(win, step = 0.1, dt = 10) {
782     if (win) {
783         // Get the opacity of the window.
784         var opacity = win.getOpacity();
785         var interval = [];
786
787         if (opacity != 1) {
788             // Increase the opacity of the window by 'step' every 'dt' ms
789             interval = setInterval(function() {
790                 // Stop fading if window's opacity is 1 or greater.
791                 if (opacity >= 1) {
792                     clearInterval(interval);
793                     opacity = 1;
794                 }
795                 win.setOpacity(opacity);
796                 opacity += step;
797             });
798         }
799     }
800 }
```

```

791         }, dt);
792     }
793
794     // Return the interval. Useful if we want to stop fading at will.
795     return interval;
796   }
797   return false;
798 }
799
800 /**
801 * Gradually decreases the opacity of a given window until it is fully
802 * transparent.
803 * @param {BrowserWindow} win - The browser window to fade out.
804 * @param {number} step - The decremental step of opacity change. Default is
805 *           0.1.
806 * @param {number} dt - The time interval in milliseconds between opacity
807 *           changes. Default is 10.
808 * @returns {number} The interval identifier for the fade-out operation.
809 */
810 fadeWindowOut(win, step = 0.1, dt = 10) {
811   if(win) {
812     // Get the opacity of the window.
813     var opacity = win.getOpacity();
814     var interval = [];
815
816     if(opacity != 0) {
817       // Reduce the opacity of the window by 'step' every 'dt' ms
818       interval = setInterval(function() {
819         // Stop fading if window's opacity is 0 or lesser.
820         if(opacity <= 0) {
821           clearInterval(interval);
822           opacity = 0;
823         }
824         win.setOpacity(opacity);
825         opacity -= step;
826       }, dt);
827     }
828     // Return the interval. Useful if we want to stop fading at will.
829     return interval;
830   }
831 }
832
833 exports.PRDC_JSLAB_MAIN = PRDC_JSLAB_MAIN;

```

Listing 39 - main.js

```

1 /**
2  * @file Javascript tester module
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7

```

```
8  if(jsl) {
9      require = jsl._require;
10 }
11
12 const fs = require("fs");
13
14 /**
15 * Class for application testing.
16 */
17 class PRDC_JSLAB_TESTER {
18
19 /**
20 * Initializes the tester with the specified folder containing test modules.
21 * Loads all test modules and prepares them for execution.
22 * @param {string} folder The folder relative to the project directory where
23 * test modules are located.
24 */
25 constructor(folder) {
26     var obj = this;
27     this.path = app_path + '/js/' + folder;
28     this.modules = [];
29     this.tests = [];
30     this.total_tests = 0;
31
32     // Define function for data display
33     if(jsl) {
34         this.disp = jsl._console.log;
35     } else {
36         this.disp = console.log;
37     }
38
39     // Find all test modules
40     try {
41         var files = fs.readdirSync(this.path);
42         files.forEach(function(file) {
43             if(file.endsWith(".test.js")) {
44                 obj.modules.push(obj.path + '/' + file);
45             }
46         });
47     } catch(err) {
48         if(err) {
49             this.disp(language.currentString(92) + ': ' + err);
50         }
51     }
52
53     // Find all tests defined in test modules
54     obj.modules.forEach(function(module) {
55         var { MODULE_TESTS } = require(module);
56         obj.total_tests += MODULE_TESTS.testsNumber();
57         obj.tests.push(...MODULE_TESTS.get());
58     });
59
60     /**
61     * Executes all loaded tests , reports successes and failures , and logs the
62     */
```

```

      results.

62  */
63  runTests() {
64    var obj = this;
65    if(this.total_tests > 0) {
66      this.disp(language.currentString(93)+' '+this.total_tests+' '+language.
67        currentString(94)+'.');
68      var i = 1;
69      var passed = 0;
70      var failed = 0;
71      this.tests.forEach(function(test) {
72        var name = test.name;
73        var result = false;
74        var error = '';
75        try {
76          result = test.run();
77        } catch(e) {
78          error = e;
79        }
80        if(!result) {
81          failed += 1;
82          if(error != '') {
83            obj.disp(' ['+i+'/'+obj.total_tests+' ] '+language.currentString
84              (95)+" "+name+" - "+language.currentString(96)+': '+error)
85            ;
86          } else {
87            obj.disp(' ['+i+'/'+obj.total_tests+' ] '+language.currentString
88              (95)+" "+name+" - "+language.currentString(97)+'.');
89        }
90      });
91      this.disp(language.currentString(99)+':');
92      this.disp(' '+language.currentString(100)+': '+passed);
93      this.disp(' '+language.currentString(101)+': '+failed);
94    } else {
95      this.disp(language.currentString(102));
96    }
97  }
98 }

100 exports.PRDC_JSLAB_TESTER = PRDC_JSLAB_TESTER;
101
102 /**
103 * Represents a collection of tests for a specific module or functionality
104 * within the JSLAB application.
105 */
106 class PRDC_JSLAB_TESTS {
107 /**
108 * Creates an instance to manage and store individual tests.
109 */

```

```

110     constructor() {
111         this.tests = [];
112     }
113
114     /**
115      * Adds a new test to the collection.
116      * @param {string} name The name of the test.
117      * @param {Function} fun The test function to execute.
118      */
119     add(name, fun) {
120         this.tests.push(new PRDC_JSLAB_TEST(name, fun));
121     }
122
123     /**
124      * Returns all tests added to this collection.
125      * @returns {Array} An array of all tests within this collection.
126      */
127     get() {
128         return this.tests;
129     }
130
131     /**
132      * Returns the number of tests in the collection.
133      * @returns {number} The total number of tests.
134      */
135     testsNumber() {
136         return this.tests.length;
137     }
138 }
139
140 exports.PRDC_JSLAB_TESTS = PRDC_JSLAB_TESTS;
141
142 /**
143  * Represents an individual test within a test suite.
144 */
145 class PRDC_JSLAB_TEST {
146
147     /**
148      * Initializes a new test with a name and a test function.
149      * @param {string} name The name of the test.
150      * @param {Function} fun The function to execute as the test.
151      */
152     constructor(name, fun) {
153         this.name = name;
154         this.fun = fun;
155     }
156
157     /**
158      * Executes the test function and returns the result.
159      * @returns {boolean} The result of the test function execution, true for
160      * pass and false for fail.
161      */
162     run() {
163         return this.fun();
164     }
165 }
```

```

164 }
165
166 exports.PRDC_JSLAB_TEST = PRDC_JSLAB_TEST;

```

Listing 40 - tester.js

6.1 code

```

1 /**
2  * @file Edited CodeMirror javascript-hints.js
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 *
7 * CodeMirror, copyright (c) by Marijn Haverbeke and others
8 * Distributed under an MIT license: https://codemirror.net/LICENSE
9 *
10 */
11
12 (function(mod) {
13   if(typeof exports == "object" && typeof module == "object") // CommonJS
14     mod(require("../lib/codemirror"));
15   else if(typeof define == "function" && define.amd) // AMD
16     define(["../lib/codemirror"], mod);
17   else // Plain browser env
18     mod(CodeMirror);
19 })(function(CodeMirror) {
20   var Pos = CodeMirror.Pos;
21
22   function scriptHint(editor, keywords, getToken, options) {
23
24     // Find the token at the cursor
25     var cur = editor.getCursor(), token = getToken(editor, cur);
26     if(/\b(?:string|comment)\b/.test(token.type)) return;
27     var innerMode = CodeMirror.innerMode(editor.getLineMode(), token.state);
28     if(innerMode.mode.helperType === "json") return;
29     token.state = innerMode.state;
30
31     // If it's not a 'word-style' token, ignore the token.
32     if(!/^[\w$]*$/_.test(token.string)) {
33       token = {start: cur.ch, end: cur.ch, string: "", state: token.state,
34                 type: token.string === "." ? "property" : null};
35     } else if(token.end > cur.ch) {
36       token.end = cur.ch;
37       token.string = token.string.slice(0, cur.ch - token.start);
38     }
39
40     var tprop = token;
41     // If it is a property, find out what it is a property of.
42     while(tprop.type === "property") {
43       tprop = getToken(editor, Pos(cur.line, tprop.start));
44       if(tprop.string !== ".") return;
45       tprop = getToken(editor, Pos(cur.line, tprop.start));
46       if(!context) var context = [];

```

```

47         context.push(tprop);
48     }
49     return getCompletions(token, context, keywords, options, cur);
50 }
51
52 function javascriptHint(editor, callback, options) {
53     return scriptHint(editor, javascriptKeywords,
54         function(e, cur) { return e.getTokenAt(cur); },
55         options);
56 }
57 CodeMirror.registerHelper("hint", "javascript", javascriptHint);
58
59
60 var javascriptKeywords = ("break case catch class const continue debugger
61     default delete do else export extends false finally for function "
62     "if in import instanceof new null return super switch this
63     throw true try typeof var void while with yield").split(
64     " ");
65
66 function getCompletions(token, context, keywords, options, cur) {
67     return new Promise(function(resolve) {
68         ipcRenderer.invoke('get-completions', [token.string, JSON.stringify(
69             context), keywords]).then(function(found) {
70             resolve({list: found,
71                 from: Pos(cur.line, token.start),
72                 to: Pos(cur.line, token.end)}));
73         });
74     });
75 }
76 });

```

Listing 41 - custom-javascript-hint.js

```

1 /**
2  * @file Edited CodeMirror search.js
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  *
7  * CodeMirror, copyright (c) by Marijn Haverbeke and others
8  * Distributed under an MIT license: https://codemirror.net/LICENSE
9  */
10
11 (function(mod) {
12     if(typeof exports == "object" && typeof module == "object") // CommonJS
13         mod(require("../lib/codemirror"), require("./searchcursor"));
14     else if(typeof define == "function" && define.amd) // AMD
15         define(["../lib/codemirror", "./searchcursor"], mod);
16     else // Plain browser env
17         mod(Codemirror);
18 }) (function(Codemirror) {
19     "use strict";
20
21     function searchOverlay(query, caseInsensitive) {
22         if(typeof query == "string")

```

```

24     query = new RegExp(query.replace(/[\-\[\]\]/\\{\\}\(\)\*\+\?\\.\\\\^\\$\\|/g,
25                               "\\$JSLAB_SOURCE_CODE_DATA$"), caseInsensitive ? "gi" : "g");
26   else if (!query.global)
27     query = new RegExp(query.source, query.ignoreCase ? "gi" : "g");
28
29   return {token: function(stream) {
30     query.lastIndex = stream.pos;
31     var match = query.exec(stream.string);
32     if (match && match.index == stream.pos) {
33       stream.pos += match[0].length || 1;
34       return "searching";
35     } else if (match) {
36       stream.pos = match.index;
37     } else {
38       stream.skipToEnd();
39     }
40   }};
41
42   function SearchState() {
43     this.posFrom = this.posTo = this.lastQuery = this.query = null;
44     this.overlay = null;
45   }
46
47   function getSearchState(cm) {
48     return cm.state.search || (cm.state.search = new SearchState());
49   }
50
51   function queryCaseInsensitive(query) {
52     return typeof query == "string" && query == query.toLowerCase();
53   }
54
55   function getSearchCursor(cm, query, pos) {
56     // Heuristic: if the query string is all lowercase, do a case insensitive
57     // search.
58     return cm.getSearchCursor(query, pos, {caseFold: queryCaseInsensitive(
59       query), multiline: true});
60
61   function persistentDialog(cm, text, deflt, onEnter, onKeyDown) {
62     cm.openDialog(text, onEnter, {
63       value: deflt,
64       selectValueOnOpen: true,
65       closeOnEnter: false,
66       onClose: function() { clearSearch(cm); },
67       onKeyDown: onKeyDown
68     });
69
70   function dialog(cm, text, shortText, deflt, f) {
71     if (cm.openDialog) cm.openDialog(text, f, {value: deflt, selectValueOnOpen
72       : true});
73     else f(prompt(shortText, deflt));
74   }

```



```

127   var state = getSearchState(cm);
128   if(state.query) return findNext(cm, rev);
129   var q = cm.getSelection() || state.lastQuery;
130   if(q instanceof RegExp && q.source == "x^") q = null;
131   if(persistent && cm.openDialog) {
132     var hiding = null;
133     var searchNext = function(query, event) {
134       CodeMirror.e_stop(event);
135       if(!query) return;
136       if(query != state.queryText) {
137         startSearch(cm, state, query);
138         state.posFrom = state.posTo = cm.getCursor();
139       }
140       if(hiding) hiding.style.opacity = 1;
141       findNext(cm, event.shiftKey, function(_, to) {
142         var dialog;
143         if(to.line < 3 && document.querySelector &&
144             (dialog = cm.display.wrapper.querySelector(".CodeMirror-dialog"))
145             ) &&
146             dialog.getBoundingClientRect().bottom - 4 > cm.cursorCoords(to,
147               "window").top)
148             (hiding = dialog).style.opacity = 0.4;
149       });
150       persistentDialog(cm, getQueryDialog(cm), q, searchNext, function(event,
151         query) {
152         var keyName = CodeMirror.keyName(event);
153         var extra = cm.getOption('extraKeys'), cmd = (extra && extra[keyName])
154           || CodeMirror.keyMap[cm.getOption("keyMap")][keyName];
155         if(cmd == "findNext" || cmd == "findPrev" ||
156           cmd == "findPersistentNext" || cmd == "findPersistentPrev") {
157           CodeMirror.e_stop(event);
158           startSearch(cm, getSearchState(cm), query);
159           cm.execCommand(cmd);
160         } else if(cmd == "find" || cmd == "findPersistent") {
161           CodeMirror.e_stop(event);
162           searchNext(query, event);
163         }
164       });
165       if(immediate && q) {
166         startSearch(cm, state, q);
167         findNext(cm, rev);
168       }
169     } else {
170       dialog(cm, getQueryDialog(cm), "Search for:", q, function(query) {
171         if(query && !state.query) cm.operation(function() {
172           startSearch(cm, state, query);
173           state.posFrom = state.posTo = cm.getCursor();
174           findNext(cm, rev);
175         });
176       });
177     }
178   }
179 }

function findNext(cm, rev, callback) {cm.operation(function() {

```

```

178   var state = getSearchState(cm);
179   var cursor = getSearchCursor(cm, state.query, rev ? state.posFrom : state.
180     posTo);
181   if(!cursor.find(rev)) {
182     cursor = getSearchCursor(cm, state.query, rev ? CodeMirror.Pos(cm.
183       lastLine()) : CodeMirror.Pos(cm.firstLine(), 0));
184     if(!cursor.find(rev)) return;
185   }
186   cm.setSelection(cursor.from(), cursor.to());
187   cm.scrollIntoView({from: cursor.from(), to: cursor.to()}, 20);
188   state.posFrom = cursor.from(); state.posto = cursor.to();
189   if(callback) callback(cursor.from(), cursor.to());
190 })};

191
192 function clearSearch(cm) {cm.operation(function() {
193   var state = getSearchState(cm);
194   state.lastQuery = state.query;
195   if(!state.query) return;
196   state.query = state.queryText = null;
197   cm.removeOverlay(state.overlay);
198   if(state.annotate) { state.annotate.clear(); state.annotate = null; }
199 })};

200
201 function getQueryDialog(cm) {
202   return '<span class="CodeMirror-search-label">' + cm.phrase("Search:") +
203     '</span> <input type="text" style="width: 10em" class="CodeMirror-
204       search-field"/> <span style="color: #888" class="CodeMirror-search-
205       hint">' + cm.phrase("(Use /re/ syntax for regexp search)") + '</span>';
206 }
207 function getReplaceQueryDialog(cm) {
208   return '<input type="text" style="width: 10em" class="CodeMirror-search-
209       field"/> <span style="color: #888" class="CodeMirror-search-hint">' +
210     cm.phrase("(Use /re/ syntax for regexp search)") + '</span>';
211 }
212 function getReplacementQueryDialog(cm) {
213   return '<span class="CodeMirror-search-label">' + cm.phrase("With:") + '<
214     span> <input type="text" style="width: 10em" class="CodeMirror-search-
215       field"/>';
216 }
217 function getDoReplaceConfirm(cm) {
218   return '<span class="CodeMirror-search-label">' + cm.phrase("Replace?") +
219     '</span> <button>' + cm.phrase("Yes") + '</button> <button>' + cm.
220     phrase("No") + '</button> <button>' + cm.phrase("All") + '</button> <
221       button>' + cm.phrase("Stop") + '</button> ';
222 }

223 function replaceAll(cm, query, text) {
224   cm.operation(function() {
225     for(var cursor = getSearchCursor(cm, query); cursor.findNext();)
226       if(typeof query != "string") {
227         var match = cm.getRange(cursor.from(), cursor.to()).match(query);
228         cursor.replace(text.replace(/\\$\\d/g, function(_, i) { return match
229           [i]; }));
230       }
231   });
232 }

```

```

219         } else cursor.replace(text);
220     }
221   });
222 }
223
224 function replace(cm, all) {
225   if(cm.getOption("readOnly")) return;
226   var query = cm.getSelection() || getSearchState(cm).lastQuery;
227   var dialogText = '<span class="CodeMirror-search-label">' + (all ? cm.
228     phrase("Replace all:") : cm.phrase("Replace:")) + '</span>';
229   dialog(cm, dialogText + getReplaceQueryDialog(cm), dialogText, query,
230     function(query) {
231       if(!query) return;
232       query = parseQuery(query);
233       dialog(cm, getReplacementQueryDialog(cm), cm.phrase("Replace with:"), ""
234         , function(text) {
235           text = parseString(text);
236           if(all) {
237             replaceAll(cm, query, text);
238           } else {
239             clearSearch(cm);
240             var cursor = getSearchCursor(cm, query, cm.getCursor("from"));
241             var advance = function() {
242               var start = cursor.from(), match;
243               if(!(match = cursor.findNext())) {
244                 cursor = getSearchCursor(cm, query);
245                 if(!(match = cursor.findNext()) ||
246                   (start && cursor.from().line == start.line && cursor.from().
247                     ch == start.ch)) return;
248               }
249               cm.setSelection(cursor.from(), cursor.to());
250               cm.scrollIntoView({from: cursor.from(), to: cursor.to()});
251               confirmDialog(cm, getDoReplaceConfirm(cm), cm.phrase("Replace?"),
252                 [function() {doReplace(match);}, advance,
253                  function() {replaceAll(cm, query, text);}]);
254             };
255             var doReplace = function(match) {
256               cursor.replace(typeof query == "string" ? text :
257                 text.replace(/\$(\d)/g, function(_, i) { return
258                   match[i]; }));
259               advance();
260             };
261             advance();
262           }
263         });
264       CodeMirror.commands.showSearchDialog = function(cm) { clearSearch(cm);
265         showSearchDialog(cm); };
266       CodeMirror.commands.hideSearchDialog = function(cm) { clearSearch(cm);
267         hideSearchDialog(cm); };
268       CodeMirror.commands.find = function(cm) { clearSearch(cm); doSearch(cm); };
269       CodeMirror.commands.findPersistent = function(cm) { clearSearch(cm);
270         doSearch(cm, false, true); };

```

```

266  CodeMirror.commands.findPersistentNext = function(cm) { doSearch(cm, false,
267    true, true); };
268  CodeMirror.commands.findPersistentPrev = function(cm) { doSearch(cm, true,
269    true, true); };
270  CodeMirror.commands.findNext = doSearch;
271  CodeMirror.commands.findPrev = function(cm) { doSearch(cm, true); };
272  CodeMirror.commands.clearSearch = clearSearch;
273  CodeMirror.commands.replace = replace;
274  CodeMirror.commands.replaceAll = function(cm) { replace(cm, true); };
275 });

```

Listing 42 - custom-search.js

```

1  /**
2   * @file Search dialog based on CodeMirror search.js
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   *
7   * CodeMirror, copyright (c) by Marijn Haverbeke and others
8   * Distributed under an MIT license: https://codemirror.net/LICENSE
9   *
10  */
11
12 (function(mod) {
13   if(typeof exports == "object" && typeof module == "object") // CommonJS
14     mod(require("../lib/codemirror"), require("./searchcursor"));
15   else if(typeof define == "function" && define.amd) // AMD
16     define(["../../../lib/codemirror", "./searchcursor"], mod);
17   else // Plain browser env
18     mod(Codemirror);
19 })(function(CodeMirror) {
20   "use strict";
21
22   CodeMirror.defineOption('searchDialog', false, function(cm, val) {
23     if(val) {
24       var div = document.createElement('div');
25       div.className = 'CodeMirror-search-dialog';
26       div.innerHTML = '<input class="CodeMirror-search-find" value=""' +
27         'placeholder="Find" autocomplete="off" spellcheck="false" title="Find"' +
28         '><i class="CodeMirror-search-find-prev-btn" title="Previous match"' +
29         '></i><i class="CodeMirror-search-find-next-btn" title="Next match"' +
30         '></i><br class="clear"/><input class="CodeMirror-search-replace"' +
31         'value="" placeholder="Replace" autocomplete="off" spellcheck="false"' +
32         'title="Replace"><i class="CodeMirror-search-replace-btn" title="Replace"' +
33         '></i><i class="CodeMirror-search-replace-all-btn" title="Replace All"' +
34         '></i><br class="clear"/><i class="CodeMirror-search-case-btn"' +
35         'title="Match Case"></i><i class="CodeMirror-search-regex-btn"' +
36         'title="Use Regular Expression"></i><div class="CodeMirror-search-bottom-right"><span class="CodeMirror-search-current-match">0</span>' +
37         'of <span class="CodeMirror-search-total-matchs">0</span><i class="CodeMirror-search-close-btn"' +
38         'title="Close Search Dialog"></i></div><br class="clear"/>';
39       div.setAttribute('tabindex', 0);
40
41       var find_input = div.querySelector('.CodeMirror-search-find');

```

```

30   var replace_input = div.querySelector('.CodeMirror-search-replace');
31   var find_prev_btn = div.querySelector('.CodeMirror-search-find-prev-btn',
32     );
33   var find_next_btn = div.querySelector('.CodeMirror-search-find-next-btn',
34     );
35   var replace_btn = div.querySelector('.CodeMirror-search-replace-btn');
36   var replace_all_btn = div.querySelector('.CodeMirror-search-replace-all-
37     btn');
38   var match_case_btn = div.querySelector('.CodeMirror-search-case-btn');
39   var regex_btn = div.querySelector('.CodeMirror-search-regex-btn');
40   var close = div.querySelector('.CodeMirror-search-close-btn');

41
42   div.addEventListener('keydown', function(e) {
43     if(e.key == 'Escape') {
44       // ESC
45       if(window.jQuery) {
46         $(div).slideUp(200);
47       } else {
48         div.style.display = 'none';
49       }
50       cm.focus();
51     }
52   });
53
54   replace_input.addEventListener('keydown', function(e) {
55     if(e.key == 'Enter') {
56       // Replace
57       replace(cm, replace_input.value, false);
58     }
59   });
60
61   find_input.addEventListener('keyup', function() {
62     // Find match
63     clearSearch(cm);
64     var state = getSearchState(cm);
65     state.query = this.value;
66     doSearch(cm);
67   });
68
69   find_prev_btn.addEventListener('click', function() {
70     // Show previous match
71     doSearch(cm, true);
72   });
73
74   find_next_btn.addEventListener('click', function() {
75     // Show next match
76     doSearch(cm, false);
77   });
78
79   replace_btn.addEventListener('click', function() {
80     // Replace
81     replace(cm, replace_input.value, false)

```

```
82 });
83
84 replace_all_btn.addEventListener('click', function() {
85     // Replace all
86     replace(cm, replace_input.value, true)
87 });
88
89 match_case_btn.addEventListener('click', function() {
90     if(this.classList.contains('active')) {
91         cm.search_match_case = false;
92         this.classList.remove('active');
93     } else {
94         cm.search_match_case = true;
95         this.classList.add('active');
96     }
97 });
98
99 regex_btn.addEventListener('click', function() {
100    if(this.classList.contains('active')) {
101        cm.search_regex = false;
102        this.classList.remove('active');
103    } else {
104        cm.search_regex = true;
105        this.classList.add('active');
106    }
107 });
108
109 close.addEventListener('click', function() {
110     if(window.jQuery) {
111         $(div).slideUp(200);
112     } else {
113         div.style.display = 'none';
114     }
115     cm.focus();
116 });
117
118     cm.display.wrapper.appendChild(div);
119 }
120 });
121
122 function searchOverlay(query, caseInsensitive) {
123     if(typeof query == "string")
124         query = new RegExp(query.replace(/[-[\]]\/\{\}\(\)\*\+\?\.\\^$\[\]/g,
125             "\\\\$JSLAB_SOURCE_CODE_DATA\$"), caseInsensitive ? "gi" : "g");
126     else if(!query.global)
127         query = new RegExp(query.source, query.ignoreCase ? "gi" : "g");
128
129     return {token: function(stream) {
130         query.lastIndex = stream.pos;
131         var match = query.exec(stream.string);
132         if(match && match.index == stream.pos) {
133             stream.pos += match[0].length || 1;
134             return "searching";
135         } else if(match) {
136             stream.pos = match.index;
```

```

136     } else {
137         stream.skipToEnd();
138     }
139     return false;
140 });
141 }
142
143 function SearchState() {
144     this.posFrom = this.posTo = this.lastQuery = this.query = null;
145     this.overlay = null;
146 }
147
148 function getSearchState(cm) {
149     return cm.state.search || (cm.state.search = new SearchState());
150 }
151
152 function queryCaseInsensitive(cm, query) {
153     return !cm.search_match_case && typeof query === "string";
154 }
155
156 function getSearchCursor(cm, query, pos) {
157     return cm.getSearchCursor(query, pos, {caseFold: queryCaseInsensitive(cm,
158         query), multiline: true});
159 }
160
161 function parseString(string) {
162     return string.replace(/\[\nrt\]/g, function(match, ch) {
163         if(ch === "\n") return "\n";
164         if(ch === "\r") return "\r";
165         if(ch === "\t") return "\t";
166         if(ch === "\\") return "\\";
167         return match;
168     });
169
170 function parseQuery(cm, query) {
171     if(cm.search_regex) {
172         var isRE = query.match(/^\/(.*)\//([a-z]*$));
173         if(isRE) {
174             try { query = new RegExp(isRE[1], isRE[2].indexOf("i") === -1 ? "" : "i");
175             } catch(e) {}
176         }
177     } else {
178         query = parseString(query);
179     }
180     if(typeof query === "string" ? query === "" : query.test(""))
181         query = /x^/;
182     return query;
183 }
184
185 function startSearch(cm, state, query) {
186     state.queryText = query;
187     state.query = parseQuery(cm, query);
188     cm.removeOverlay(state.overlay, queryCaseInsensitive(cm, state.query));

```

```

189     state.overlay = searchOverlay(state.query, queryCaseInsensitive(cm, state.
190         query));
191     cm.addOverlay(state.overlay);
192     if(state.annotate) { state.annotate.clear(); state.annotate = null; }
193     state.annotate = cm.showMatchesOnScrollbar(state.query,
194         queryCaseInsensitive(cm, state.query));
195
196     function showSearchDialog(cm) {
197       var div = cm.display.wrapper.querySelector('.CodeMirror-search-dialog');
198       if(window.jQuery) {
199         $(div).slideDown(200);
200       } else {
201         div.style.display = 'block';
202       }
203       var query = cm.getSelection();
204       var find_input = cm.display.wrapper.querySelector('.CodeMirror-search-find');
205       if(query) {
206         find_input.value = query;
207         clearSearch(cm);
208         var state = getSearchState(cm);
209         state.query = query;
210         doSearch(cm);
211       }
212       find_input.focus();
213     }
214
215     function hideSearchDialog(cm) {
216       var div = cm.display.wrapper.querySelector('.CodeMirror-search-dialog');
217       if(window.jQuery) {
218         $(div).slideUp(200);
219       } else {
220         div.style.display = 'none';
221       }
222     }
223
224     function doSearch(cm, rev) {
225       var state = getSearchState(cm);
226       var q = state.query;
227       if(q != state.queryText) {
228         startSearch(cm, state, q);
229         state.posFrom = state.posTo = cm.getCursor();
230       }
231       if(q) {
232         findNext(cm, rev);
233       }
234       if(state.annotate) {
235         var current_match_index = state.annotate.matches.findIndex(function(e) {
236           return e.from.line == state.posFrom.line &&
237             e.from.ch == state.posFrom.ch && e.to.line == state.posTo.line &&
238             e.to.ch == state.posTo.ch;
239         })+1;
240         var total_matches = cm.display.wrapper.querySelector('.CodeMirror-search-
total-matches');
```

```

240     total_matchs.innerText = state.annotate.matches.length;
241     var current_match = cm.display.wrapper.querySelector('.CodeMirror-search-
242         -current-match');
243     current_match.innerText = current_match_index;
244   }
245
246   function findNext(cm, rev, callback) {cm.operation(function() {
247     var state = getSearchState(cm);
248     var cursor = getSearchCursor(cm, state.query, rev ? state.posFrom : state.
249         posTo);
250     if(!cursor.find(rev)) {
251       cursor = getSearchCursor(cm, state.query, rev ? CodeMirror.Pos(cm.
252           lastLine()) : CodeMirror.Pos(cm.firstLine(), 0));
253     if(!cursor.find(rev)) return;
254     cm.setSelection(cursor.from(), cursor.to());
255     cm.scrollIntoView({from: cursor.from(), to: cursor.to()}, 20);
256     state.posFrom = cursor.from(); state.posto = cursor.to();
257     if(callback) callback(cursor.from(), cursor.to());
258   });
259
260   function clearSearch(cm) {cm.operation(function() {
261     var state = getSearchState(cm);
262     state.lastQuery = state.query;
263     if(!state.query) return;
264     state.query = state.queryText = null;
265     cm.removeOverlay(state.overlay);
266     if(state.annotate) { state.annotate.clear(); state.annotate = null; }
267   });
268
269   function replaceAll(cm, query, text) {
270     clearSearch(cm);
271     cm.operation(function() {
272       for(var cursor = getSearchCursor(cm, query); cursor.findNext();) {
273         if(typeof query != "string") {
274           var match = cm.getRange(cursor.from(), cursor.to()).match(query);
275           cursor.replace(text.replace(/\$(\d)/g, function(_, i) { return match
276               [i]; }));
277         } else cursor.replace(text);
278       });
279
280   function replace(cm, text, all) {
281     if(cm.getOption("readOnly")) return;
282     var query = getSearchState(cm).query;
283     if(query) {
284       text = parseString(text);
285       if(all) {
286         replaceAll(cm, query, text);
287       } else {
288         var cursor = getSearchCursor(cm, query, cm.getCursor());
289         var start = cursor.from(), match;
290         if(!(match = cursor.findNext())) {

```

```

291     cursor = getSearchCursor(cm, query);
292     if (!(match = cursor.findNext()) || 
293         (start && cursor.from().line == start.line && cursor.from().ch == 
294         start.ch)) return;
295     cm.setSelection(cursor.from(), cursor.to());
296     cursor.replace(typeof query == "string" ? text : 
297       text.replace(/\$(\d)/, function(w, i) {return match[i];}));
298   }
299 }
300 }
301
302 CodeMirror.commands.showSearchDialog = function(cm) {clearSearch(cm); 
303   showSearchDialog(cm);}
303 CodeMirror.commands.hideSearchDialog = function(cm) {clearSearch(cm); 
304   hideSearchDialog(cm);}
304 CodeMirror.commands.find = function(cm) {clearSearch(cm); doSearch(cm);}
305 CodeMirror.commands.findNext = function(cm) {doSearch(cm, false);}
306 CodeMirror.commands.findPrev = function(cm) {doSearch(cm, true);}
307 CodeMirror.commands.clearSearch = clearSearch;
308 CodeMirror.commands.replace = replace;
309 CodeMirror.commands.replaceAll = function(cm) {replace(cm, true);}
310 });

```

Listing 43 - dialog-search.js

6.2 dev

```

1 /**
2  * @file Native modules build
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  * @version 0.0.1
7 */
8
9 // Import the filesystem module
10 const fs = require('fs');
11
12 global.process_arguments = process.argv;
13 const { PRDC_APP_CONFIG } = require('../config/config');
14
15 console.log('[build-configure.js] Started');
16 var t = performance.now();
17
18 var config = new PRDC_APP_CONFIG();
19
20 // Change package.json
21 console.log('Changing package.json... ');
22 var data = JSON.parse(fs.readFileSync('package.json'));
23 var filename = data.name+'_'+data.version;
24 if(process.argv.length > 3) {
25   var action = process.argv[3];
26   if(action === 'dist-portable') {

```

```

27     filename += '_portable';
28   }
29 }
30
31 data.build.artifactName = filename + '.' + ext;
32
33 if(config.SIGN_BUILD) {
34   data.build.win.sign = "js/dev/build-sign.js";
35 } else {
36   delete data.build.win.sign;
37 }
38
39 fs.writeFileSync('package.json', JSON.stringify(data, null, 2));
40
41 // Creating binding.gyp
42 console.log('Creating binding.gyp... ');
43 var binding_data = {targets: []};
44
45 // - Native module
46 if(fs.existsSync('cpp/binding.gyp')) {
47   var data = JSON.parse(fs.readFileSync('cpp/binding.gyp'));
48   if(!Array.isArray(data)) {
49     data = [data];
50   }
51   for(var i = 0; i < data.length; i++) {
52     binding_data.targets[i] = data[i];
53     binding_data.targets[i].defines = ["NAPI_DISABLE_CPP_EXCEPTIONS"];
54   }
55 }
56
57 // - Libs binding.gyp
58 config.COMPILE_LIBS.forEach(function(lib) {
59   var data = JSON.parse(fs.readFileSync('lib/' + lib + '/binding.gyp'));
60   binding_data.targets.push(data);
61 });
62
63 // - Save binding.gyp
64 fs.writeFileSync('binding.gyp', JSON.stringify(binding_data, null, 2));
65
66 // Create bin folder
67 if(!fs.existsSync('bin/')) {
68   fs.mkdirSync('bin/');
69 }
70
71 console.log('[build-configure.js] ' + ((performance.now() - t) / 1000).toFixed(3)
72   + ' s');
```

Listing 44 - build-configure.js

```

1 /**
2  * @file Build sign
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  * @version 0.0.1
7  */
```

```

8
9 // Import the filesystem module
10 const { PRDC_APP_CONFIG } = require('../config/config.js');
11
12 console.log('[build-sign.js] Started');
13 var t = performance.now();
14
15 if (!process.env.COMPANY_NAME) {
16   console.error('Environment variable COMPANY_NAME must be defined!');
17 }
18 if (!process.env.TIMESTAMP_SERVER) {
19   console.error('Environment variable TIMESTAMP_SERVER must be defined!');
20 }
21 if (!process.env.SIGN_TOOL_PATH) {
22   console.error('Environment variable SIGN_TOOL_PATH must be defined!');
23 }
24
25 var config = new PRDC_APP_CONFIG();
26
27 const { execSync } = require('child_process');
28
29
30 /**
31 * Signs a specified file using the digital signature tool and parameters
32 * defined in the application's configuration. The
33 * function reads the configuration to obtain the path to the signing tool,
34 * the company name for the signature, and other
35 * necessary parameters. It then constructs the command to execute the signing
36 * process and runs it using 'execSync'.
37 *
38 * @param {Object} data - An object containing parameters for the signing
39 * process.
40 * @param {string} data.path - The path to the file that needs to be signed.
41 * @param {string} data.hash - The hash algorithm to use for signing (e.g., 'sha256').
42 */
43 exports.default = function(data) {
44   console.info(`Signing ${data.path} with ${data.hash} to ${config.COMPANY_NAME}`);
45
46   const sha256 = data.hash === 'sha256';
47   const appendCert = sha256 ? '/as' : null;
48   const timestamp = sha256 ? '/tr' : '/t';
49   const appendTd = sha256 ? '/td sha256' : null;
50
51   let args = [
52     `${config.SIGN_TOOL_PATH}`,
53     'sign',
54     '/debug',
55     '/n',
56     `${config.COMPANY_NAME}`,
57     '/a',
58     appendCert,
59     '/fd',
60     data.hash,
61   ];
62
63   if (appendTd) {
64     args.push(appendTd);
65   }
66
67   return execSync(args.join(' '));
68 }

```

```

57     timestamp ,
58     config.TIMESTAMP_SERVER,
59     appendTd ,
60     '/v' ,
61     `${data.path}` "
62   ];
63
64   try {
65     const { stdout } = execSync(args.join(' '));
66     console.log(stdout);
67   } catch(err) {
68     throw err;
69   }
70
71   console.log(`[ build-sign.js ] ${((performance.now()-t)/1000).toFixed(3)} s`);
72 };

```

Listing 45 - build-sign.js

```

1  /**
2   * @file Clear app data
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   * @version 0.0.1
7   */
8
9 // Import modules
10 const fs = require('fs');
11 const os = require('os');
12 const rimraf = require('rimraf');
13 const readline = require('readline').createInterface({
14   input: process.stdin,
15   output: process.stdout
16 });
17
18 var data = JSON.parse(fs.readFileSync('package.json'));
19
20 console.log(`[ clear-app-data.js ] Started`);
21 var t = performance.now();
22
23 // Variables
24 var confirm = process.argv.includes('--confirm');
25
26 if(confirm) {
27   readline.question("Clear app data? yes/[no]: ", function(answer) {
28     if(answer === 'yes') {
29       main();
30     } else {
31       console.log('Action aborted.');
32       console.log(`[ clear-app-data.js ] Execution done in ${((performance.now() - t)/1000).toFixed(3)} s`);
33       process.exit();
34     }
35   }
36   readline.close();

```

```

36      });
37  } else {
38   main();
39 }
40
41 function main() {
42   var path = os.homedir()+'\\AppData\\Roaming\\'+data.name;
43   console.log('Clearing app data from '+path);
44   rimraf.sync(path);
45
46   console.log('[clear-app-data.js] Execution done in ' + ((performance.now()-t
47     )/1000).toFixed(3) + ' s');
48   process.exit();
49 }
```

Listing 46 - clear-app-data.js

```

1  /**
2   * @file Download libs
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   * @version 0.0.1
7   */
8
9 // Import modules
10 const fs = require('fs');
11 const rimraf = require('rimraf');
12 const readline = require('readline').createInterface({
13   input: process.stdin,
14   output: process.stdout
15 });
16 const path = require('path');
17 const { extractFull } = require('node-7z');
18 const bin_path = require('7zip-bin').path7za;
19
20 const { PRDC_APP_CONFIG } = require("../config/config");
21
22 console.log('[dev-download-libs.js] Started');
23 var t = performance.now();
24
25 // Variables
26 var config = new PRDC_APP_CONFIG();
27
28 var force = process.argv.includes('--force');
29 var confirm = process.argv.includes('--confirm');
30
31 if(confirm) {
32   readline.question("Download libs? yes/[no]: ", function(answer) {
33     if(answer === 'yes') {
34       main();
35     } else {
36       console.log('Libs download aborted.');
37       console.log('[dev-download-libs.js] Execution done in ' + ((performance.
38         now() - t) / 1000).toFixed(3) + ' s');
39       process.exit();
40     }
41   });
42 }
43
44 function main() {
45   rimraf.sync(path);
46   console.log('Libs download completed');
47 }
```

```
39     }
40     readline.close();
41   });
42 } else {
43   main();
44 }
45
46 async function main() {
47   if(force) {
48     console.log('Deleting all libs... ');
49     rimraf.sync('./lib');
50   }
51
52   for(const lib of config.USED_LIBS) {
53     const libDir = path.join('./lib', lib);
54     const serverLibDir = path.join(config.SERVER_LIBS_PATH, lib);
55     const serverLib7z = path.join(config.SERVER_LIBS_PATH, `${lib}.7z`);
56     const lib7z = path.join('./lib', `${lib}.7z`);
57
58     // Check if lib folder exists locally
59     if(!fs.existsSync(libDir)) {
60       if(fs.existsSync(serverLibDir)) {
61         process.stdout.write('Downloading ${lib}... ');
62         await fs.promises.cp(serverLibDir, libDir, { recursive: true });
63         process.stdout.clearLine(0);
64         process.stdout.cursorTo(0);
65         console.log(`Downloading of ${lib} complete.`);
66       } else if(fs.existsSync(serverLib7z)) {
67         await copyWithProgress(lib, serverLib7z, lib7z);
68         await extractWithProgress(lib, lib7z, './lib');
69       } else {
70         console.log(`[ERROR] Neither folder nor .7z found for ${lib}.`);
71       }
72     } else {
73       console.log(`Lib ${lib} already downloaded.`);
74     }
75   }
76
77   console.log(`\nAll libraries downloaded.`);
78   console.log(`[download-libs.js] Execution done in ' + ((performance.now() - t) / 1000).toFixed(3) + ' s' );
79   process.exit();
80 }
81
82 // Function to copy files with progress
83 async function copyWithProgress(lib, src, dest) {
84   return new Promise((resolve, reject) => {
85     const total_size = fs.statSync(src).size;
86     let copied = 0;
87
88     const read_stream = fs.createReadStream(src);
89     const write_stream = fs.createWriteStream(dest);
90
91     read_stream.on('data', (chunk) => {
92       copied += chunk.length;
93     })
94   })
95 }
```

```

93   const progress = ((copied / total_size) * 100).toFixed(2);
94
95   process.stdout.clearLine(0);
96   process.stdout.cursorTo(0);
97   process.stdout.write(`Downloading ${lib}: ${progress}% (${copied}/${total_size})`);
98 });
99
100 read_stream.pipe(write_stream);
101
102 write_stream.on('finish', () => {
103   process.stdout.clearLine(0);
104   process.stdout.cursorTo(0);
105   console.log(`Downloading of ${lib} complete.`);
106   resolve();
107 });
108
109 read_stream.on('error', reject);
110 write_stream.on('error', reject);
111 });
112 }
113
114 // Function to extract .7z with progress
115 async function extractWithProgress(lib, archive, dest) {
116   return new Promise((resolve, reject) => {
117     const extractor = extractFull(archive, dest, {
118       $bin: bin_path,
119       $progress: true
120     });
121
122     extractor.on('progress', (progress) => {
123       const percent = progress.percent.toFixed(0);
124       process.stdout.clearLine(0);
125       process.stdout.cursorTo(0);
126       process.stdout.write(`Extracting ${lib}: ${percent}%`);
127     });
128
129     extractor.on('end', () => {
130       process.stdout.clearLine(0);
131       process.stdout.cursorTo(0);
132       console.log(`Extraction of ${lib} complete.`);
133       fs.unlinkSync(archive);
134       resolve();
135     });
136
137     extractor.on('error', (err) => {
138       console.error(`Extraction error for ${lib}:`, err);
139       fs.unlinkSync(archive);
140       reject(err);
141     });
142   });
143 }

```

Listing 47 - download-libs.js

```

2  * @file Generate sandbox documentation for JSLAB
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  * @version 0.0.1
7  */
8
9 const jsdoc = require('jsdoc-api');
10 const fs = require('fs');
11 const cp = require("child_process");
12 const rimraf = require('rimraf');
13
14 const { PRDC_APP_CONFIG } = require('../.../config/config');
15
16 console.log('[make-doc.js] Started');
17 var t = performance.now();
18
19 var package_data = JSON.parse(fs.readFileSync('package.json'));
20 var app_version = package_data.version;
21 var year = new Date().getFullYear();
22
23 var config = new PRDC_APP_CONFIG();
24 var jslab_doc = {
25   'global': [],
26   'lib': []
27 };
28
29 async function processDoc(module) {
30   var docs_out;
31   if(config.OUTPUT_COMPLETE_JSDOC) {
32     docs_out = [];
33   } else {
34     docs_out = {};
35   }
36   var docs = await jsdoc.explain({ files: `js/sandbox/${module.file}.js` });
37   docs.forEach(function(doc) {
38     if(config.OUTPUT_COMPLETE_JSDOC) {
39       docs_out.push(doc);
40     } else if((doc.kind === 'function' || doc.kind === 'member') &&
41       doc.memberof === module.className &&
42       !doc.name.startsWith('_') &&
43       !doc.name.startsWith('#') &&
44       doc.name !== 'jsl' &&
45       doc.description) {
46       var doc_output = {};
47       doc_output.name = doc.name;
48       doc_output.kind = doc.kind;
49       doc_output.description = doc.description;
50       if(doc.params) {
51         doc_output.params = doc.params;
52       }
53       if(doc.returns) {
54         doc_output.returns = doc.returns;
55       }
56       if(doc.async) {

```

```

57         doc_output.async = doc.async;
58     }
59     if(doc.meta) {
60         doc_output.source_filename = doc.meta.filename;
61         doc_output.source_lineno = doc.meta.lineno;
62         doc_output.source_range = doc.meta.range;
63     }
64     docs_out[doc.name] = doc_output;
65   }
66 });
67 return docs_out;
68 }
69
70 (async function main() {
71   config.SUBMODULES['builtin'].push(...config.DOC_SUBMODULES_ADDITIONAL);
72   for(var module of config.SUBMODULES['builtin']) {
73     console.log(' Generating documentation for: global/' + module.name);
74     jslab_doc['global'][module.name] = await processDoc(module);
75   }
76   for(var lib of config.SUBMODULES['lib']) {
77     console.log(' Generating documentation for: lib/' + lib.name);
78     jslab_doc['lib'][lib.name] = await processDoc(lib);
79   }
80   var jslab_doc_flat = Object.values(jslab_doc).reduce((acc, innerObj) => {
81     return { ...acc, ...innerObj };
82   }, {});
83   if(config.OUTPUT_COMPLETE_JSDOC) {
84     return;
85   }
86
87   // Make JSON documentation
88   console.log(' Making documentation.json');
89   fs.writeFileSync('docs/documentation.json', JSON.stringify(jslab_doc, null,
90                   2));
91
92   // Make HTML documentation
93   console.log(' Making documentation.html');
94   var html_template = fs.readFileSync('dev/documentation_template.html',
95                                     'utf8');
96   html_template = html_template.replaceAll(`$JSLAB_CODE_DATA
97
98 % -----
99 % End of document
100 % -----
101
102 % -----
103 % End of document
104 % -----
105 \end{document}, JSON.stringify(jslab_doc_flat, null, 2));
106   html_template = html_template.replaceAll(`$JSLAB_APP_VERSION
107
108 % -----
109 % End of document

```

```

110  % -----
111  \end{document}, "'"+year+"'");
112  fs.writeFileSync('docs/documentation.html', html_template);
113
114  // Make tex
115  console.log(' Making documentation.tex');
116  var latex_template = fs.readFileSync('dev/documentation_template.tex', 'utf8');
117  latex_template = latex_template.replaceAll('$JSLAB_APP_VERSION'
118
119  % -----
120  % End of document
121  % -----
122  \end{document}, 'v'+app_version+'');
123  latex_template = latex_template.replaceAll('$JSLAB_PUBLISH_YEAR'
124
125  % -----
126  % End of document
127  % -----
128  \end{document}, year);
129
130  function getLatexByCodeCategory(category) {
131      // Helper function to escape LaTeX special characters
132      function escapeLatex(string) {
133          if(typeof string !== 'string') {
134              return string;
135          }
136          return string
137              .replace(/\\"/g, '\\textbackslash{}')
138              .replace(/\&/g, '\\&')
139              .replace(/\%/g, '\\%')
140              .replace(/\$/g, '\\'
141
142  % -----
143  % End of document
144  % -----
145  \end{document})
146      .replace(/\#/g, '\\#')
147      .replace(/\_ /g, '\\_')
148      .replace(/\{/g, '\\{')
149      .replace(/\}/g, '\\}')
150      .replace(/\~/g, '\\textasciitilde{}')
151      .replace(/\^/g, '\\textasciicircum{}')
152      .replace(/\`/g, '\\textasciigrave{}');
153  }
154
155  let latex = '';
156
157  // Iterate over each item in the specified category
158  latex += '\\subsection{$\{escapeLatex(category)}$\n';
159
160  for(const item_name in jslab_doc_flat[category]) {
161      const item = jslab_doc_flat[category][item_name];
162
163      if(item.kind === 'function') {

```

```

164 // Generate the function signature
165 const params = item.params
166   ? item.params
167     .map(param => {
168       if(param.type === 'Object' && param.properties) {
169         // Generate nested object structure for 'opts'
170         const nestedSignature = param.properties
171           .map(prop =>
172             prop.optional
173               ? `[$ {escapeLatex(prop.name)} ]`
174               : `${escapeLatex(prop.name)} `
175           )
176         .join(', ');
177
178       // Format the 'opts' parameter
179       return param.optional
180         ? `[$ {escapeLatex(param.name)} ]\\{ ${nestedSignature} ]]`
181         : `${escapeLatex(param.name)} ]\\{ ${nestedSignature} ]`;
182     } else {
183       // Handle simple parameters with optionality and default
184       values
185       return param.optional
186         ? `[$ {escapeLatex(param.name)} ${param.default ? `=$ {`
187           escapeLatex(param.default)} ` : ''}]`
188         : `${escapeLatex(param.name)} ${param.default ? `=$ {`
189           escapeLatex(param.default)} ` : ''}`;
190     }
191   }
192
193   // Add a title for the function
194   latex += '\\vspace{5mm}\\n\\noindent \\codeBlock{\\texttt{${escapeLatex(
195     (item.name) }(${params}) }}}{\\color{jsl-gray}\\vspace{2mm}\\hrule\\
196     \\vspace{4mm}}\\n\\n';
197
198   // Add metadata if available
199   if(item.since) {
200     latex += '\\textbf{Added in:} \\${escapeLatex(item.since)}\\\\\\n';
201   }
202
203   // Add parameters section
204   if(item.params && item.params.length > 0) {
205     latex += '\\n\\noindent \\textbf{Parameters:}\\n\\begin{itemize}\\n';
206     item.params.forEach(param => {
207       const type = param.type && param.type.names && param.type.names[0]
208         ? param.type.names[0] : 'Unknown';
209       latex += ' \\item \\texttt{${escapeLatex(param.name)}} \\texttt{<
210         ${escapeLatex(type)}>}: ${escapeLatex(param.description || '')}\\n';
211     });
212     latex += '\\end{itemize}\\n';
213   }
214
215   // Add returns section

```

```

211     if(item.returns && item.returns.length > 0) {
212         const returnType = item.returns[0].type && item.returns[0].type.names
213             && item.returns[0].type.names[0] ? item.returns[0].type.names[0] : 'Unknown';
214         latex += '\n\\noindent \\textbf{Returns:} \\texttt{<${escapeLatex(
215             returnType)}>}: ${escapeLatex(item.returns[0].description)}\n';
216     }
217
218     // Add description
219     if(item.description) {
220         latex += '\n\\noindent ${escapeLatex(item.description)}\n\n';
221     }
222
223     // Add examples if available
224     if(item.examples && item.examples.length > 0) {
225         latex += '\\begin{verbatim}\n${escapeLatex(item.examples[0])}\n\\end{verbatim}\n';
226     }
227
228 } else if(item.kind === 'member') {
229     // Add a title for the member
230     latex += '\\vspace{5mm}\\noindent \\code{\\texttt{$\\{escapeLatex(item.name)}$}}{\\color{jsl-gray}\\vspace{2mm}\\hrule\\vspace{4mm}}\n';
231
232     // Add metadata if available
233     if(item.since) {
234         latex += '\\textbf{Added in:} \\${escapeLatex(item.since)}\\\\\\n';
235     }
236
237     // Add type information
238     const type = item.type && item.type.names && item.type.names[0] ? item.type.names[0] : 'Unknown';
239     latex += '\n\\noindent \\textbf{Type:} \\texttt{<${escapeLatex(type)}>}\n';
240
241     // Add description
242     if(item.description) {
243         latex += '\n\\noindent ${escapeLatex(item.description)}\n\n';
244     }
245
246     latex += '\n';
247
248     return latex;
249 }
250
251 var latex_text = '';
252 for(const category in jslab_doc_flat) {
253     latex_text += getLatexByCodeCategory(category);
254 }
255 latex_template = latex_template.replaceAll('$JSLAB_CODE_DATA'
256

```

```

257 % -----
258 % End of document
259 %
260 \end{document}, latex_text);
261 fs.writeFileSync('docs/documentation.tex', latex_template);
262
263 // Make pdf
264 console.log(' Making documentation.pdf');
265
266 var dir = __dirname + '/../../dev/latex/';
267 var file = dir + 'documentation.tex';
268 fs.mkdirSync(dir, { recursive: true });
269 fs.writeFileSync('docs/documentation.tex', file);
270 try {
271   for(var i = 0; i < config.DOC_LATEX_RERUNS_NUMBER; i++) {
272     console.log(' Run ' + (i+1) + '/' + config.DOC_LATEX_RERUNS_NUMBER);
273     cp.execSync('cd ' + dir + '& pdflatex documentation.tex --interaction=nonstopmode', { cwd: dir });
274   }
275   fs.writeFileSync(dir + 'documentation.pdf', 'docs/documentation.pdf');
276 } catch(e) {
277   console.log(' LaTeX compile failed! Add path to pdflatex to environment
278   variables and try again with MiKTeX 24.1 or later. Output: ');
279   if(e.output) {
280     console.log(e.output.toString());
281   } else {
282     console.log(e);
283   }
284   rimraf.sync(dir);
285
286   console.log('[make-doc.js] ' + ((performance.now()-t)/1000).toFixed(3) + ' s
287   ');
288 }());

```

Listing 48 - make-doc.js

```

1 /**
2  * @file Generate source code book for JSLAB
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  * @version 0.0.1
7 */
8
9 const fs = require('fs');
10 const path = require('path');
11 const cp = require("child_process");
12 const rimraf = require('rimraf');
13
14 const { PRDC_APP_CONFIG } = require('../config/config');
15
16 console.log('[make-source-code-book.js] Started');
17 var t = performance.now();
18
19 var package_data = JSON.parse(fs.readFileSync('package.json'));

```



```
20 var app_version = package_data.version;
21 var year = new Date().getFullYear();
22
23 var config = new PRDC_APP_CONFIG();
24 var levels = [ '\\\\section', '\\\\subsection', '\\\\subsubsection'];
25 var latex = ',';
26
27 // Helper function to escape LaTeX special characters
28 function escapeLatex(string) {
29     if(typeof string !== 'string') {
30         return string;
31     }
32     return string
33         .replace(/\\/g, '\\textbackslash{}')
34         .replace(/\&/g, '\\&')
35         .replace(/\%/g, '\\%')
36         .replace(/\$/g, '\\'
37
38 % -----
39 % End of document
40 %
41 \end{document}
42     .replace(/\#/g, '\\#')
43     .replace(/\_g, '\\_')
44     .replace(/\{g, '\\{')
45     .replace(/\}g, '\\}')
46     .replace(/\^g, '\\textasciitilde{}')
47     .replace(/\^g, '\\textasciicircum{}')
48     .replace(/\`g, '\\textasciigrave{}');
49 }
50
51 /**
52 * Recursively traverses directories and reads file contents.
53 * @param {string[]} pathsList - List of file or folder paths to process.
54 * @returns {Promise<Object>} - An object representing the directory structure
55 * with file details.
56 */
57 async function buildFileStructure(pathsList) {
58     const result = {root:{}};
59
60     for(const itemPath of pathsList) {
61         const absolutePath = path.resolve(itemPath);
62         const stats = await fs.promises.lstat(absolutePath);
63
64         if(stats.isDirectory()) {
65             // Recursively process the directory
66             result[path.basename(absolutePath)] = await traverseDirectory(
67                 absolutePath);
68         } else if(stats.isFile()) {
69             result['root'][path.basename(absolutePath)] = await fs.promises.readFile(
70                 absolutePath, 'utf-8');
71         }
72     }
73
74     return result;
75 }
```

```
72  }
73
74 /**
75  * Helper function to traverse a directory recursively.
76  * @param {string} dirPath - The directory path to traverse.
77  * @returns {Promise<Object>} - An object representing the directory structure
78  */
79 async function traverseDirectory(dirPath) {
80   const dirContents = await fs.promises.readdir(dirPath, { withFileTypes: true
81   });
82   const dirObject = {};
83
84   // Separate files and directories
85   const files = dirContents.filter(dirent => dirent.isFile());
86   const directories = dirContents.filter(dirent => dirent.isDirectory());
87
88   // Process files first
89   for(const file of files) {
90     const fullPath = path.join(dirPath, file.name);
91     dirObject[file.name] = await fs.promises.readFile(fullPath, 'utf-8');
92   }
93
94   // Then process directories
95   for(const directory of directories) {
96     const fullPath = path.join(dirPath, directory.name);
97     // Recursively process subdirectories
98     dirObject[directory.name] = await traverseDirectory(fullPath);
99   }
100
101   return dirObject;
102 }
103 /**
104  * Determines the programming language based on the file extension.
105  * @param {string} filename - The name of the file (e.g., "main.cpp").
106  * @returns {string|null} - The corresponding language name or null if unknown
107  */
108 function getLanguage(filename) {
109   const extensionMatch = filename.match(/\.([^.]+)$/);
110   if (!extensionMatch) {
111     return null;
112   }
113
114   const extension = extensionMatch[1].toLowerCase();
115
116   // Mapping of extensions to languages
117   const extensionToLanguageMap = {
118     'cpp': 'C++',
119     'h': 'C++',
120     'js': 'JavaScript',
121     'json': 'JavaScript',
122     'gyp': 'JavaScript',
123     'css': 'CSS',
```

```

124     'html' : 'HTML'
125   };
126
127   return extensionToLanguageMap[ extension ] || null;
128 }
129
130 /**
131 * Recursively traverses an object and performs actions based on the type of
132 * each property.
133 * @param {Object} obj - The object to traverse.
134 * @param {Function} callback - A function to execute on each property.
135 * @param {number} [level=0] - The current recursion level (used internally).
136 */
137 function traverseObject(obj, level = 0) {
138   for(const key in obj) {
139     if(obj.hasOwnProperty(key)) {
140       if(typeof obj[key] === 'object' && obj[key] !== null) {
141         latex += '\n%' + '-----'.repeat(4-level) + '\n' + levels[level]
142           + '{' + escapeLatex(key) + '}\\n\\n';
143         traverseObject(obj[key], level + 1);
144       } else {
145         latex += '
146 \\begin{lstlisting}[style=${getLanguage(key)}Style, caption=${escapeLatex(key)
147 )}]
148 ${obj[key]}
149 \\end{lstlisting}
150 ';
151     }
152   }
153   (async function main() {
154     // Make tex
155     console.log(' Making source-code-book.tex');
156     var latex_template = fs.readFileSync('dev/source_code_template.tex', 'utf8')
157     ;
158     latex_template = latex_template.replaceAll('$JSLAB_APP_VERSION
159 % -----
160 % End of document
161 % -----
162 \\end{document}, 'v'+app_version+'');
163     latex_template = latex_template.replaceAll('$JSLAB_PUBLISH_YEAR
164 % -----
165 % End of document
166 % -----
167 \\end{document}, year);
168
169     var app_path = __dirname + '/../../';
170     var fileStructure = await buildFileStructure(config.SOURCE_CODE_BOOK_FILES);
171     traverseObject(fileStructure);
172     latex_template = latex_template.replaceAll('$JSLAB_SOURCE_CODE_DATA
173
174

```

```

175 % -----
176 % End of document
177 %
178 \end{document}, latex);
179 fs.writeFileSync('docs/source-code-book.tex', latex_template);
180
181 // Make pdf
182 console.log(' Making source-code-book.pdf');
183
184 var dir = __dirname + '/.../dev/latex/';
185 var file = dir + 'source-code-book.tex';
186 fs.mkdirSync(dir, { recursive: true });
187 fs.writeFileSync('docs/source-code-book.tex', file);
188
189 for(var i = 0; i < config.SOURCE_CODE_BOOK_LATEX_RERUNS_NUMBER; i++) {
190     console.log(' Run ' + (i+1) + '/' + config.
191         SOURCE_CODE_BOOK_LATEX_RERUNS_NUMBER);
192     try {
193         cp.execSync('cd ' + dir + '& pdflatex source-code-book.tex --'
194             'interaction=nonstopmode', { cwd: dir });
195     } catch(e) {
196         if(!fs.existsSync(dir + 'source-code-book.pdf') || (e.output &&
197             e.output.toString().trim().split('\n').pop().toLowerCase().includes(
198                 'fatal'))) {
199             console.log(' LaTeX compile failed! Add path to pdflatex to
200                 environment variables and try again with MiKTeX 24.1 or later.
201                 Output: ');
202             if(e.output) {
203                 console.log(e.output.toString());
204             } else {
205                 console.log(e);
206             }
207             break;
208         }
209     rimraf.sync(dir);
210
211     console.log('[make-source-code-book.js.js] ' + ((performance.now() - t) / 1000).
212        toFixed(3) + ' s');
213 }())

```

Listing 49 - make-source-code-book.js

```

1 /**
2 * @file Prepare libs
3 * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4 * PR-DC, Republic of Serbia
5 * info@pr-dc.com
6 * @version 0.0.1
7 */
8

```

```

9 // Import modules
10 const fs = require('fs');
11 const rimraf = require('rimraf');
12 const path = require('path');
13 const { extractFull } = require('node-7z');
14 const bin_path = require('7zip-bin').path7za;
15
16 const { PRDC_APP_CONFIG } = require("../config/config");
17
18 console.log('[prepare-libs.js] Started');
19 var t = performance.now();
20
21 // Variables
22 var config = new PRDC_APP_CONFIG();
23
24 async function main() {
25   for(const lib of config.COMPRESSED_LIBS) {
26     const lib_dir = path.join('./lib', lib);
27     const lib_7z = path.join('./lib', `${lib}.7z`);
28
29     // Check if lib folder exists locally
30     if(!fs.existsSync(lib_dir)) {
31       await extractWithProgress(lib, lib_7z, './lib');
32     } else {
33       console.log(`Lib ${lib} already uncompressed.`);
34     }
35   }
36
37   console.log('\nAll libraries ready.');
38   console.log(`[prepare-libs.js] Execution done in ${((performance.now() - t) / 1000).toFixed(3)} s`);
39 }
40
41 main();
42
43 // Function to extract .7z with progress
44 async function extractWithProgress(lib, archive, dest) {
45   return new Promise((resolve, reject) => {
46     const extractor = extractFull(archive, dest, {
47       $bin: bin_path,
48       $progress: true
49     });
50
51     extractor.on('progress', (progress) => {
52       const percent = progress.percent.toFixed(0);
53       process.stdout.clearLine(0);
54       process.stdout.cursorTo(0);
55       process.stdout.write(`Extracting ${lib}: ${percent}%`);
56     });
57
58     extractor.on('end', () => {
59       process.stdout.clearLine(0);
60       process.stdout.cursorTo(0);
61       console.log(`Extraction of ${lib} complete.`);
62       fs.unlinkSync(archive);
63     });
64   });
65 }

```

```

63         resolve();
64     });
65
66     extractor.on('error', (err) => {
67       console.error(`Extraction error for ${lib}:`, err);
68       fs.unlinkSync(archive);
69       reject(err);
70   });
71 });
72 }

```

Listing 50 - prepare-libs.js

```

1  /**
2   * @file Upload source code
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   * @version 0.0.1
7   */
8
9 // Import modules
10 const fs = require('fs');
11 const dircompare = require('dir-compare');
12 const path = require('path');
13
14 const { PRDC_APP_CONFIG } = require("../config/config");
15
16 console.log('[upload-source-code.js] Started');
17 var t = performance.now();
18
19 if(!process.env.SERVER_PATH) {
20   console.error('Environment variable SERVER_PATH must be defined!');
21 }
22 if(!process.env.SERVER_LIBS_PATH) {
23   console.error('Environment variable SERVER_LIBS_PATH must be defined!');
24 }
25
26 // Variables
27 var config = new PRDC_APP_CONFIG();
28
29 const options = {
30   compareSize: config.UPLOAD_COMPARE_SIZE,
31   compareContent: config.UPLOAD_COMPARE_CONTENT,
32   compareDate: config.UPLOAD_COMPARE_DATE,
33   excludeFilter: config.SOURCE_UPLOAD_EXCLUDE.join(', ')
34 };
35
36 // Compare directories
37 console.log('Comparing directories... ');
38 var result = dircompare.compareSync('.', config.SERVER_SOURCE_PATH, options);
39
40 console.log(`Statistics: \n Equal entries: ${result.equal}, \n Distinct entries: ${result.distinct}, \n Left only entries: ${result.left}, \n Right only entries: ${result.right}, \n Differences: ${result.differences}`);
41

```

```

42
43 if(result.differences) {
44   // Analyzing different files
45   console.log('Analyzing different files...');

46   result.diffSet.forEach(function(dif) {
47     if(dif.type1 == 'file' || dif.type2 == 'file') {
48       if(dif.state != 'equal') {
49         switch(dif.state) {
50           case 'distinct':
51             if(!config.UPLOAD_COMPARE_SIZE_ON_DISTINCT || dif.date1 >= dif.
52                 date2) {
53               console.log('File ' + path.join(dif.relativePath, dif.name1) + ' '
54                 changed.');
55             } else {
56               console.log('File ' + path.join(dif.relativePath, dif.name1) + ' '
57                 last modified on server.');
58             }
59             break;
60           case 'left':
61             console.log('File ' + path.join(dif.relativePath, dif.name1) + ' '
62               missing on server.');
63             break;
64           case 'right':
65             console.log('File ' + path.join(dif.relativePath, dif.name2) + ' '
66               exists only on server.');
67             break;
68           default:
69             console.log('Unhandled state for: ');
70             console.log(dif);
71         }
72       if(['distinct', 'left'].includes(dif.state)) {
73         if(!dif.date2 || !config.UPLOAD_COMPARE_SIZE_ON_DISTINCT || dif.
74           date1 >= dif.date2) {
75           console.log(' Uploading file to server...');

76           if(!dif.date2) {
77             fs.mkdirSync(path.join(config.SERVER_SOURCE_PATH, dif.
78               relativePath), { recursive: true });
79             fs.writeFileSync(path.join(dif.path1, dif.name1),
80               path.join(config.SERVER_SOURCE_PATH, dif.relativePath, dif.
81               name1));
82           } else {
83             fs.mkdirSync(dif.path2, { recursive: true });
84             fs.writeFileSync(path.join(dif.path1, dif.name1),
85               path.join(dif.path2, dif.name2));
86           }
87         }
88       }
89     }
90   })
91 }

92 // console.log(dif);
93
94 });
95 });

96 });
97 });

98 } else {
99   console.log('No different files...');


```

```

89  }
90 console.log('[upload-source-code.js] Execution done in ' + ((performance.now()
- t)/1000).toFixed(3) + ' s');
```

Listing 51 - upload-source-code.js

6.3 editor

```

1  /**
2   * @file JSLAB editor module
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   */
7
8  /**
9   * Class for JSLAB editor.
10  */
11 class PRDC_JSLAB_EDITOR {
12
13 /**
14  * Initializes the editor, setting up event listeners for UI interactions
15  * and window controls.
16  * @param {object} win The window object representing the current Electron
17  * window.
18  */
19 constructor(win) {
20   var obj = this;
21   this.win = win;
22
23   // On menu click
24   $("#save-menu").click(function() { obj.win.script_manager.saveScript(); });
25   $("#save-as-menu").click(function() { obj.win.script_manager.saveAsScript();
26   });
27   $("#open-menu").click(function() { obj.win.script_manager.openScriptFile();
28   });
29   $("#run-menu").click(function() { obj.win.script_manager.runScript(); });
30   $("#new-tab").click(function() { obj.win.script_manager.createScript(); });
31   ;
32   $("#new-script").click(function() { obj.win.script_manager.createScript();
33   });
34   $("#close-dialog-save").click(function() { obj.win.script_manager.
35   closingDialogButton(2); });
36   $("#close-dialog-discard").click(function() { obj.win.script_manager.
37   closingDialogButton(1); });
38   $("#close-dialog-cancel").click(function() { obj.win.script_manager.
39   closingDialogButton(0); });
40
41   $("#search-dialog-menu").click(function() {
```

```

37   obj.win.script_manager.openSearchDialog();
38   obj.win.editor_more_popup.close();
39 });
40
41 // Keydown actions
42 document.addEventListener("keydown", function(e) {
43   if(e.key === 'F11') {
44     obj.toggleFullscreen();
45   } else if(e.ctrlKey && e.key.toLowerCase() === "n") {
46     obj.win.script_manager.createScript();
47   } else if(e.ctrlKey && e.key === "F4") {
48     obj.win.script_manager.removeScriptByTab(obj.win.script_manager.active_tab);
49   } else if(e.key === "F5") {
50     obj.win.script_manager.runScript();
51   } else if(e.ctrlKey && e.key.toLowerCase() === "o") {
52     obj.win.script_manager.openScriptFile();
53   } else if(e.ctrlKey && e.key.toLowerCase() === "s" && !e.shiftKey) {
54     obj.win.script_manager.saveScript();
55   } else if(e.ctrlKey && e.key.toLowerCase() === "s" && e.shiftKey) {
56     obj.win.script_manager.saveAsScript();
57   }
58 });
59
60 // Window controls
61 window.addEventListener('resize', function() {
62   // Detect change of maximize
63   obj.maximized = ipcRenderer.sendSync('sync-message', 'is-maximized-win');
64   ;
65   if(obj.maximized) {
66     $('#win-restore img').attr('src', '../img/win-restore.svg');
67   } else {
68     $('#win-restore img').attr('src', '../img/win-maximize.svg');
69   }, true);
70
71 $('#win-close').click(function() {
72   ipcRenderer.send('MainProcess', 'close-editor');
73 });
74
75 $('#win-restore').click(function() {
76   obj.toggleFullscreen(false);
77   obj.maximized = !obj.maximized;
78   if(obj.maximized) {
79     ipcRenderer.send('MainProcess', 'maximize-win');
80   } else {
81     ipcRenderer.send('MainProcess', 'restore-win');
82   }
83 });
84
85 $('#win-minimize').click(function() {
86   obj.toggleFullscreen(false);
87   ipcRenderer.send('MainProcess', 'minimize-win');
88 });
89

```

```

90     window.dispatchEvent(new Event('resize'));
91   }
92
93   /**
94    * Toggles the fullscreen state of the editor window, or sets it explicitly
95    * if a parameter is provided.
96    * @param {boolean} [fullscreen] Optional. If specified, sets the fullscreen
97    * state to the provided value. If omitted, toggles the current state.
98    */
99   toggleFullscreen(fullscreen) {
100     if(fullscreen == null) {
101       fullscreen = !this.fullscreen;
102     }
103     if(fullscreen) {
104       ipcRenderer.send('MainProcess', 'set-fullscreen', true);
105     } else {
106       ipcRenderer.send('MainProcess', 'set-fullscreen', false);
107     }
108
109   /**
110    * Displays a message to the user through the application's main window. Can
111    * optionally request to focus the window.
112    * @param {string} msg The message to display.
113    * @param {boolean} [focus=true] Optional. Whether to focus the application
114    * window when displaying the message. Defaults to true.
115    */
116   disp(msg, focus = true) {
117     ipcRenderer.send("MainWindow", "editor-disp", [msg, focus]);
118
119   /**
120    * Displays an internal message within the application's main window. Can
121    * optionally request to focus the window.
122    * @param {string} msg - The internal message to display.
123    * @param {boolean} [focus=true] - Optional. Whether to focus the
124    * application window when displaying the message. Defaults to true.
125    */
126   dispInternal(msg, focus = true) {
127     ipcRenderer.send("MainWindow", "disp-internal", [msg, focus]);
128
129   /**
130    * Displays an error message through the application's main window.
131    * Typically used for internal errors.
132    * @param {string} msg The error message to display.
133    */
134   errorInternal(msg) {
135     ipcRenderer.send("MainWindow", "internal-error", msg);
136   }
137 }
138
139 exports.PRDC_JSLAB_EDITOR = PRDC_JSLAB_EDITOR;

```

Listing 52 - editor.js



```
1  /**
2   * @file JSLAB editor window init file
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   */
7
8 // Modules
9 // -----
10 const { ipcRenderer } = require('electron');
11
12 const helper = require("../js/helper.js");
13 const { PRDC_APP_CONFIG } = require('../config/config');
14 const { PRDC_APP_LOGGER } = require('../lib/PRDC_APP_LOGGER/PRDC_APP_LOGGER');
15 const { PRDC_JSLAB_LANGUAGE } = require('../js/language');
16
17 global.app_path = process.argv.find(e => e.startsWith('--app-path=')).split('=')
18   [1].replace(/\.\.\\js\\?$/,'');
19
20 const { PRDC_JSLAB_WIN_EDITOR } = require('../js/editor/win-editor');
21
22 // Start log
23 const log_file = ipcRenderer.sendSync('sync-message', 'get-log-file');
24 const app_logger = new PRDC_APP_LOGGER(log_file);
25
26 // Global variables
27 const config = new PRDC_APP_CONFIG();
28 var language = new PRDC_JSLAB_LANGUAGE();
29 var win_editor = new PRDC_JSLAB_WIN_EDITOR();
30
31 // When document is ready
32 // -----
33 ready(function() {
34   // Jquery ready
35   $(document).ready(function() {
36     win_editor.onReady();
37   });
38});
```

Listing 53 - init-editor.js

```
1 /**
2  * @file JSLAB editor script manager module
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 const { PRDC_JSLAB_EDITOR_SCRIPT } = require('../script');
9
10 const fs = require('fs');
11 const { pathEqual } = require('path-equal');
12 const Store = require('electron-store');
13 const { ESLint } = require("eslint");
14
15 const store = new Store();
```

```
16 //**
17 * Class for JSLAB editor script.
18 */
19 class PRDC_JSLAB_EDITOR_SCRIPT_MANAGER {
20
21     /**
22      * Initializes the script manager with references to the application window
23      * and sets up event listeners for tab interactions.
24      * @param {object} win The application window where the editor is hosted.
25      */
26     constructor(win) {
27         var obj = this;
28         this.win = win;
29
30         this.scripts = [];
31         this.last_script_paths = [];
32         this.last_active_script;
33         this.active_tab;
34         this.close_window = false;
35         this.tabs = new PRDC_TABS();
36
37         this.eslint = new ESLint(config.LINT_OPTIONS);
38
39         // Tabs
40         this.tabs_cont = document.querySelector(".tabs");
41         this.tabs.init(this.tabs_cont);
42
43         // On tab add
44         this.tabs_cont.addEventListener("tabAdd", function({ detail }) {
45             $("#close-dialog-cont").hide();
46             detail.tabEl.onmousedown = function (e) {
47                 if(e && (e.which === 2 || e.button === 4)) {
48                     obj.removeScriptByTab(detail.tabEl);
49                 }
50             };
51         });
52
53         // On tab remove
54         this.tabs_cont.addEventListener("tabRemove", function() {
55             if(obj.scripts.length === 0) {
56                 if(obj.close_window) {
57                     store.set("last_script_paths", obj.last_script_paths);
58                     store.set("last_active_script", obj.last_active_script);
59                     ipcRenderer.send("MainProcess", "close-editor");
60                 } else {
61                     obj.createScript();
62                 }
63             }
64         });
65
66         // On tab close
67         this.tabs_cont.addEventListener("tabClose", function({ detail }) {
68             obj.removeScriptByTab(detail.tabEl);
69         });
70     }
71 }
```

```

70
71 // On tab change
72 this.tabs_cont.addEventListener("activeTabChange", function({ detail }) {
73   if(obj.active_tab !== undefined) {
74     // Update code to last active_tab
75     var [script] = obj.getScriptByTab(obj.active_tab);
76     if(script !== undefined) {
77       script.update();
78     }
79   }
80   obj.active_tab = detail.tabEl;
81   var [script] = obj.getScriptByTab(obj.active_tab);
82   if(script !== undefined) {
83     if(!script.closing) {
84       $("#close-dialog-cont").hide();
85     }
86     script.show();
87   }
88 });
89
90
91 // On start up
92 var argv = ipcRenderer.sendSync('sync-message', 'get-process-arguments');
93 var file_path = argv.find(arg => arg.endsWith('.jsl')); // path as
94 argument
95 var script_paths = store.get("last_script_paths"); // last opened scripts
96 if(file_path) {
97   if(script_paths) {
98     script_paths.push(file_path);
99   } else {
100    script_paths = [file_path];
101  }
102 }
103 if(script_paths) {
104   script_paths.forEach(function(script_path) {
105     var lstat = fs.lstatSync(script_path, {throwIfNoEntry: false});
106     if(lstat && lstat.isFile()) {
107       var [script, i] = obj.getScriptByPath(script_path);
108       if(script === undefined) {
109         obj.createScript(script_path);
110       } else {
111         script.activate();
112       }
113     });
114 }
115
116 if(this.scripts.length > 0) {
117   if(file_path) {
118     this.activateScriptByPath(file_path);
119   } else {
120     var last_active_script = store.get("last_active_script");
121     if(last_active_script) {
122       this.activateScriptByPath(last_active_script);
123     }
124 }

```

```

124         }
125         this.win.showEditor();
126     } else {
127         // New tab
128         this.createScript();
129     }
130 }
131
132 /**
133 * Saves the currently active script.
134 */
135 saveScript() {
136     this.getScriptByTab(this.active_tab)[0].save();
137 }
138
139 /**
140 * Saves the currently active script under a new file name.
141 */
142 saveAsScript() {
143     this.getScriptByTab(this.active_tab)[0].saveAs();
144 }
145
146 /**
147 * Opens search dialog for currently active script.
148 */
149 openSearchDialog() {
150     this.getScriptByTab(this.active_tab)[0].openSearchDialog();
151 }
152
153 /**
154 * Opens a script file from the filesystem into the editor.
155 */
156 openScriptFile() {
157     var obj = this;
158     let options = {
159         title: language.currentString(146),
160         buttonLabel: language.currentString(147),
161         filters: [
162             { name: "jsl", extensions: ["jsl", "txt"] },
163             { name: "All Files", extensions: ["*"] },
164         ],
165     };
166
167     ipcRenderer.invoke("dialog", "showOpenDialog", options)
168     .then(function(result) {
169         if(result.canceled) return obj.win.editor.disp("@editor/openScriptFile"
170             : "+language.string(132)");
171         var open_path = result.filePaths[0];
172         var [script, i] = obj.getScriptByPath(open_path);
173         if(script == undefined) {
174             obj.createScript(open_path);
175         } else {
176             script.activate();
177         }
178     }).catch(function(err) {

```

```
178         obj.win.editor.errorInternal(err);
179     });
180 }
181 /**
182 * Opens a script by its path if not already open, or activates it if it is
183 * already open.
184 * @param {Array<string, number>} data - An array where the first element is
185 * the script path and the second is the line number.
186 */
187 openScript(data) {
188     var script_path = data[0];
189     var script_lineno = data[1];
190     var lstat = fs.lstatSync(script_path, {throwIfNoEntry: false});
191     if(lstat && lstat.isFile()) {
192         var [script, i] = this.getScriptByPath(script_path);
193         if(script == undefined) {
194             this.createScript(script_path, script_lineno);
195         } else {
196             this.win.editor.disp("@editor/openScript: "+language.string(133)+" "+
197                 script_path+" "+language.string(134)+"!", false);
198             script.activate();
199             script.setLine(script_lineno);
200         }
201     }
202 }
203 /**
204 * Executes the currently active script.
205 */
206 runScript() {
207     this.getScriptByTab(this.active_tab)[0].run();
208 }
209 /**
210 * Creates a new script tab and editor instance, loading the script from the
211 * given path.
212 * @param {string} path - The file path of the script to load.
213 * @param {number} lineno - The line number to navigate to in the newly
214 * created script.
215 */
216 createScript(path, lineno) {
217     this.tab = this.tabs.addTab();
218     var script = new PRDC_JSLAB_EDITOR_SCRIPT(this.win, this, path, this.tab);
219     script.setLine(lineno);
220     this.scripts.push(script);
221 }
222 /**
223 * Toggles the comment state of the selected lines in the currently active
224 * script tab.
225 * This action uses the active tab's associated script editor to apply or
226 * remove comments.
227 */
228 toggleComment() {
```

```
226     this.getScriptByTab(this.active_tab)[0].toggleComment();  
227 }  
228  
229 /**
230 * Activates the script associated with the given tab element.  
231 * @param {HTMLElement} tab The tab element associated with the script to  
232 activate.  
233 */  
234 activateScriptByTab(tab) {  
235     this.tabs.setCurrentTab(tab);  
236 }  
237 /**
238 * Activates the script associated with the given filesystem path.  
239 * @param {string} script_path The path to the script to activate.  
240 */  
241 activateScriptByPath(script_path) {  
242     var [script, i] = this.getScriptByPath(script_path);  
243     if(script != undefined) {  
244         this.activateScriptByTab(script.tab);  
245     }  
246 }  
247 /**
248 * Retrieves the script object and its index associated with the given tab  
249 element.  
250 * @param {HTMLElement} tab The tab element associated with the script.  
251 * @returns {Array} An array containing the script object and its index in  
252 the scripts array.  
253 */  
254 getScriptByTab(tab) {  
255     var i = this.scripts.findIndex(function(script) {  
256         return script.tab == tab;  
257     });  
258     if(i > -1) {  
259         return [this.scripts[i], i];  
260     } else {  
261         return [undefined, -1];  
262     }  
263 }  
264 /**
265 * Retrieves the script object and its index associated with the given  
266 filesystem path.  
267 * @param {string} script_path The path to the script.  
268 * @returns {Array} An array containing the script object and its index in  
269 the scripts array.  
270 */  
271 getScriptByPath(script_path) {  
272     var i = this.scripts.findIndex(function(script) {  
273         return pathEqual(script.path, script_path);  
274     });  
275     if(i > -1) {  
276         return [this.scripts[i], i];  
277     } else {
```

```

276         return [undefined, -1];
277     }
278 }
279 /**
280 * Updates the name displayed on a script's tab.
281 * @param {HTMLElement} tab The tab element associated with the script.
282 * @param {string} name The new name to display on the tab.
283 */
284 setScriptNameByTab(tab, name) {
285     this.tabs.updateTab(tab, {
286         title: name,
287         favicon: false
288     });
289 }
290 /**
291 * Removes the script associated with the given tab element from the manager
292 * and UI.
293 * @param {HTMLElement} tab The tab element associated with the script to
294 * remove.
295 */
296 removeScriptByTab(tab) {
297     var [script, i] = this.getScriptByTab(tab);
298     script.activate();
299     if(script.remove()) {
300         this.scripts[i].removeCodeEditor();
301         this.scripts.splice(i, 1);
302         this.tabs.removeTab(tab);
303     }
304 }
305 /**
306 * Initiates the closure of all open scripts, optionally persisting their
307 * state for future sessions.
308 */
309 closeAllScripts() {
310     var obj = this;
311     this.close_window = true;
312     this.last_script_paths = [];
313     this.last_active_script = this.active_tab.getAttribute('title');
314
315     this.tabs_cont.querySelectorAll('.tabs-content .tab').forEach(function(tab) {
316         var path = tab.getAttribute('title');
317         if(path) {
318             obj.last_script_paths.push(path);
319         }
320     });
321
322     ipcRenderer.send("MainProcess", "focus-win");
323     $("#close-dialog-cancel").hide();
324     for(var i = this.scripts.length - 1; i >= 0; i--) {
325         var tab = this.scripts[i].tab;
326         if(this.scripts[i].remove()) {

```

```

327         this.scripts[i].removeCodeEditor();
328         this.scripts.splice(i, 1);
329         this.tabs.removeTab(tab);
330     }
331 }
332 }

334 /**
335 * Checks if a script with the given filesystem path is already open in the
336 * editor.
337 * @param {string} script_path The path to the script to check.
338 * @returns {boolean} True if the script is already open, false otherwise.
339 */
340 checkScriptOpenByPath(script_path) {
341     var [script, i] = this.getScriptByPath(script_path);
342     return !(script === undefined);
343 }

344 /**
345 * Handles user interaction with the script closing dialog, determining
346 * whether to save changes, discard them, or cancel the closure.
347 * @param {number} s - button state
348 */
349 closingDialogButton(s) {
350     var [script, i] = this.getScriptByTab(this.active_tab);
351     var tab = this.scripts[i].tab;
352     var script_path = this.scripts[i].path;
353     if(s == 2 || s == 1) {
354         if(s == 2) {
355             this.scripts[i].save();
356             script_path = this.scripts[i].path;
357         }
358         this.scripts[i].removeCodeEditor();
359         this.scripts.splice(i, 1);
360         this.tabs.removeTab(tab);
361     } else {
362         this.scripts[i].closing = false;
363         $("#close-dialog-cont").hide();
364     }
365     if(this.close_window && script_path !== undefined) {
366         this.last_script_paths.push(script_path);
367     }
368 }
369 }

370 exports.PRDC_JSLAB_EDITOR_SCRIPT_MANAGER = PRDC_JSLAB_EDITOR_SCRIPT_MANAGER

```

Listing 54 - script-manager.js

```

1 /**
2 * @file JSLAB editor script module
3 * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4 * PR-DC, Republic of Serbia
5 * info@pr-dc.com
6 */
7

```

```
8 const fs = require('fs');
9 const path = require('path');
10
11 /**
12 * Class for JSLAB editor script.
13 */
14 class PRDC_JSLAB_EDITOR_SCRIPT {
15
16 /**
17 * Initializes a new instance of the editor script, setting up the code
18 * editor and loading any specified script.
19 * @param {object} win The window object where the editor is embedded.
20 * @param {object} script_manager The script manager handling multiple
21 * scripts.
22 * @param {string} script_path The path to the script file to be loaded into
23 * the editor.
24 * @param {HTMLElement} tab The tab element associated with this script in
25 * the UI.
26 */
27 constructor(win, script_manager, script_path, tab) {
28     var obj = this;
29     this.win = win;
30     this.script_manager = script_manager;
31     this.tab = tab;
32
33     this.code_editor;
34     this.code = "";
35     this.path;
36     this.name;
37     this.saved_code = "";
38     this.closing = false;
39
40     if(script_path !== undefined) {
41         this.path = script_path;
42         this.name = path.basename(this.path);
43         this.loadCode(script_path);
44     } else {
45         this.path = undefined;
46         this.name = "Unknown";
47     }
48
49     this.tab.setAttribute("title", this.path);
50     this.script_manager.setScriptNameByTab(this.tab, this.name);
51
52     this.code_editor = CodeMirror(document.getElementById("code"), {
53         theme: "notepadpp",
54         rulers: [{ color: "#aff", column: 75, lineStyle: "solid" }],
55         indentUnit: 2,
56         tabSize: 2,
57         lineNumbers: true,
58         lineWrapping: true,
59         styleActiveLine: true,
60         matchBrackets: true,
61         gutter: true,
62         gutters: [
63             { id: "script-name", label: this.name },
64             { id: "script-path", label: this.path }
65         ]
66     });
67
68     this.code_editor.on("change", () => {
69         this.saved_code = this.code_editor.getValue();
70     });
71
72     this.tab.addEventListener("click", () => {
73         this.code_editor.focus();
74     });
75
76     this.tab.addEventListener("contextmenu", (e) => {
77         e.preventDefault();
78     });
79
80     this.tab.addEventListener("dragstart", (e) => {
81         e.dataTransfer.setData("text/plain", this.name);
82     });
83
84     this.tab.addEventListener("dragover", (e) => {
85         e.preventDefault();
86     });
87
88     this.tab.addEventListener("drop", (e) => {
89         e.preventDefault();
90
91         const file = e.dataTransfer.files[0];
92         if(file) {
93             const reader = new FileReader();
94             reader.onload = (e) => {
95                 this.loadCode(e.target.result);
96             };
97             reader.readAsText(file);
98         }
99     });
100 }
```

```

59         "CodeMirror-linenumbers",
60         "CodeMirror-foldgutter",
61         "CodeMirror-lint-markers",
62     ],
63     foldGutter: true,
64     searchDialog: true,
65     highlightSelectionMatches: { annotateScrollbar: true },
66 });
67
68 CodeMirror.keyMap.default["Shift-Tab"] = "indentLess";
69 CodeMirror.keyMap.default["Tab"] = "indentMore";
70 CodeMirror.keyMap.default["Ctrl-F"] = "showSearchDialog";
71 CodeMirror.keyMap.default['Ctrl-G'] = 'findNext';
72 CodeMirror.keyMap.default['Shift-Ctrl-G'] = 'findPrev';
73 CodeMirror.keyMap.default['Shift-Ctrl-F'] = 'replace';
74 CodeMirror.keyMap.default['Shift-Ctrl-R'] = 'replaceAll';
75 CodeMirror.keyMap.default['Ctrl-/'] = 'toggleComment';
76
77 // Keypress events
78 this.code_editor.on("keypress", function(cm, event) {
79   if(
80     !cm.state.completionActive &&
81     !event.ctrlKey &&
82     event.key != "Enter" &&
83     event.key != ";" &&
84     event.key != " " &&
85     (event.key != "{" & (event.key != "}"))
86   ) {
87     CodeMirror.commands.autocomplete(cm, null, { completeSingle: false });
88   }
89 });
90
91 // Drop events
92 this.code_editor.on("drop", function(data, e) {
93   e.preventDefault();
94 });
95
96 this.code_editor.setValue(this.code);
97 this.code_editor.clearHistory();
98 this.code_editor.on("change", function() {
99   obj.codeChanged();
100 });
101 this.updateEditorMode();
102
103 this.show();
104 }
105
106 /**
107 * Loads the code from the specified script path into the editor.
108 * @param {string} script_path The path to the script file to be loaded.
109 */
110 loadCode(script_path) {
111   try {
112     var data = fs.readFileSync(script_path);
113     this.code = data.toString();

```

```

114     } catch (err) {
115         this.win.editor.errorInternal(err.stack);
116     }
117     this.saved_code = this.code;
118 }
119
120 /**
121 * Displays the editor for this script, focusing it and hiding other scripts
122 * editors.
123 */
124 show() {
125     $(".CodeMirror").hide();
126     if(this.closing) {
127         $("#close-file").text(this.name);
128         $("#close-dialog-cont").fadeIn(300, "linear");
129     }
130     $(this.code_editor.display.wrapper).show();
131     this.code_editor.focus();
132 }
133 /**
134 * Updates the editor's stored code value based on the current content in
135 * the editor.
136 */
137 update() {
138     this.code = this.code_editor.getValue();
139 }
140 /**
141 * Saves the current script to its associated file path. If the script does
142 * not have a path, prompts for one.
143 * @returns {boolean} True if the save operation was successful, false
144 * otherwise.
145 */
146 save() {
147     if(this.path !== undefined) {
148         if(this.isActive()) {
149             this.update();
150         }
151         this.tab.classList.remove("changed");
152         if(this.code != this.saved_code) {
153             try {
154                 fs.writeFileSync(this.path, this.code);
155             } catch (err) {
156                 this.win.editor.errorInternal(err.stack);
157                 return false;
158             }
159             this.saved_code = this.code;
160             this.tab.classList.remove("changed");
161             return true;
162         } else {
163             return true;
164         }
165     } else {
166         return this.saveAs();
167     }
168 }

```

```

165     }
166   }
167
168 /**
169 * Saves the current script to a new file , prompting the user for the file
170 path .
171 * @returns {boolean} True if the save operation was successful , false
172 otherwise .
173 */
174 saveAs() {
175   var script_path ;
176   if(this .path === undefined) {
177     script_path = "script.jsl";
178   } else {
179     script_path = this .path;
180   }
181   let options = {
182     title: language.currentString(144) ,
183     defaultPath: script_path ,
184     buttonLabel: language.currentString(145) ,
185     filters: [
186       { name: "jsl" , extensions: [ "jsl" , "txt" ] } ,
187       { name: "All Files" , extensions: [ "*" ] } ,
188     ] ,
189   };
190
191   if(this .isActive()) {
192     this .update();
193   }
194
195   script_path = ipcRenderer.sendSync("dialog" , "showSaveDialogSync" , options
196 );
197   if(script_path === undefined) {
198     this .win.editor.disp("@editor/saveAs: "+language.string(129));
199     return false ;
200   }
201
202   if(script_path != this .path) {
203     if(!this .script_manager.checkScriptOpenByPath(script_path)) {
204       try {
205         fs.writeFileSync(script_path , this .code);
206       } catch (err) {
207         this .win.editor.errorInternal(err .stack);
208         return false ;
209       }
210       this .path = script_path;
211       this .name = this .path.replace(/.*[\\\/]/ , "");
212       this .tab.setAttribute("title" , this .path);
213       this .tab.querySelector(".tab-title").innerHTML = this .name;
214       this .saved_code = this .code;
215       this .tab.classList.remove("changed");
216       this .updateEditorMode();
217       return true ;
218     } else {
219       this .win.editor.errorInternal(language.string(130));
220     }
221   }
222 }
```

```
217         return false;
218     }
219 } else {
220     return false;
221 }
222 }

223 /**
224 * Prepares the script for removal, checking if changes are unsaved and
225 * potentially prompting the user.
226 * @returns {boolean} True if the script can be safely removed, false if
227 * there are unsaved changes.
228 */
229 remove() {
230     this.closing = true;
231     if(this.isActive()) {
232         this.update();
233     }
234     if(this.code.replace(/[\r]/g, '') != this.saved_code.replace(/[\r]/g, '')) {
235         // Fade in window
236         ipcRenderer.send("MainProcess", "fade-in-win");
237         if(this.isActive()) {
238             $("#close-file").text(this.name);
239             $("#close-dialog-cont").fadeIn(300, "linear");
240         }
241         return false;
242     } else {
243         return true;
244     }
245 }

246 /**
247 * Removes the code editor associated with this script from the DOM.
248 */
249 removeCodeEditor() {
250     $(this.code_editor.display.wrapper).remove();
251 }

252 /**
253 * Checks if the script's tab is currently active in the UI.
254 * @returns {boolean} True if this script's tab is active, false otherwise.
255 */
256 isActive() {
257     return this.tab == this.script_manager.active_tab;
258 }

259 }

260 /**
261 * Activates this script's tab in the UI, showing its editor and hiding
262 * others.
263 */
264 activate() {
265     if(!this.isActive()) {
266         this.script_manager.activateScriptByTab(this.tab);
267         this.show();
268     }
269 }
```

```
268     }
269 }
270
271 /**
272 * Sets the cursor in the code editor to the specified line number.
273 * @param {number} lineno - The line number to navigate to (1-based index).
274 */
275 setLine(lineno) {
276     if(isFinite(lineno)) {
277         this.code_editor.setCursor({ line: lineno -1, ch: 0 });
278     }
279 }
280
281 /**
282 * Saves the script and then runs it , typically in an external execution
283 * environment .
284 * @param {array} lines The specific lines or sections of the script to
285 * execute , if applicable .
286 */
287 run(lines) {
288     if(this.save()) {
289         ipcRenderer.send("MainWindow", "run", [this.path, lines]);
290     } else {
291         this.win.editor.disp("@editor/run: "+language.string(131));
292     }
293 }
294 /**
295 * Marks the script as having unsaved changes , updating the UI accordingly .
296 */
297 codeChanged() {
298     this.tab.classList.add("changed");
299 }
300 /**
301 * Toggles the comment state of the currently selected lines in the code
302 * editor .
303 */
304 toggleComment() {
305     this.code_editor.execCommand('toggleComment');
306 }
307 /**
308 * Update editor mode based on script extension
309 */
310 updateEditorMode() {
311     var obj = this;
312     if(typeof this.path === 'string' || this.path instanceof String) {
313         var file_extension = path.extname(this.path);
314         if(['.cpp', '.c', '.ino', '.h', '.hpp'].includes(file_extension.toLowerCase())) {
315             this.code_editor.setOption("mode", "clike");
316             this.code_editor.setOption("lint", {});
317             return;
318         }
319     }
320 }
```

```

319     }
320     this.code_editor.setOption("mode", "javascript");
321     this.code_editor.setOption("lint", {
322       getAnnotations: async function(text, callback) {
323         var results = await obj.script_manager.eslint.lintText(text);
324         callback(results[0].messages.map(message => {
325           from: CodeMirror.Pos(message.line - 1, message.column - 1),
326           to: CodeMirror.Pos(
327             message.endLine ? message.endLine - 1 : message.line - 1,
328             message.endColumn ? message.endColumn - 1 : message.column
329           ),
330           severity: message.severity === 2 ? "error" : "warning",
331           message: message.message,
332         }));
333       },
334       async: true
335     });
336   }
337
338 /**
339 * Opens search dialog in the code editor.
340 */
341 openSearchDialog() {
342   this.code_editor.execCommand('showSearchDialog');
343 }
344
345 }
346
347 exports.PRDC_JSLAB_EDITOR_SCRIPT = PRDC_JSLAB_EDITOR_SCRIPT;

```

Listing 55 - script.js

```

1 /**
2  * @file JSLAB Editor
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 // Modules
9 // -----
10 const { ipcRenderer } = require('electron');
11
12 const { PRDC_JSLAB_EDITOR } = require('../editor');
13 const { PRDC_JSLAB_EDITOR_SCRIPT_MANAGER } = require('../script-manager');
14
15 const { PRDC_POPUP } = require('../.../lib/PRDC_POPUP/PRDC_POPUP');
16
17 const fs = require("fs");
18
19 /**
20  * Class for JSLAB editor win.
21  */
22 class PRDC_JSLAB_WIN_EDITOR {
23
24 /**

```

```
25 * Create editor win.  
26 */  
27 constructor() {  
28     var obj = this;  
29  
30     // Classes  
31     this.editor = new PRDC_JSLAB_EDITOR(this);  
32     this.script_manager = new PRDC_JSLAB_EDITOR_SCRIPT_MANAGER(this);  
33     this.editor_more_popup = new PRDC_POPUP(document.getElementById('editor-  
            more-icon'),  
            document.getElementById('editor-more-popup'));  
34  
35     // Prevent redirects  
36     preventRedirect();  
37  
38     // Events  
39     // -----  
40     // Toggle DevTools  
41     document.addEventListener("keydown", function (e) {  
42         if(e.key == "F12") {  
43             ipcRenderer.send("MainProcess", "toggle-dev-tools");  
44         } else if(e.ctrlKey && e.key.toLowerCase() == 'c') {  
45             if(obj.getSelectionText() == "") {  
46                 // No selected text  
47                 obj.editor.dispInternal(language.string(89));  
48                 ipcRenderer.send('SandboxWindow', 'stop-loop', true);  
49                 e.stopPropagation();  
50                 e.preventDefault();  
51             }  
52         }  
53     });  
54  
55     // Error handle  
56     // -----  
57     window.addEventListener("unhandledrejection", function (e) {  
58         obj.editor.errorInternal(e.reason.stack);  
59         e.preventDefault();  
60     });  
61  
62     window.addEventListener("error", function (e) {  
63         obj.editor.errorInternal(e.error.stack);  
64         e.preventDefault();  
65     });  
66  
67     // On IPC message  
68     ipcRenderer.on("EditorWindow", function(event, action, data) {  
69         switch (action) {  
70             case "open-script":  
71                 // On open script  
72                 obj.script_manager.openScript(data);  
73                 break;  
74             case "close-all":  
75                 // On close all  
76                 obj.script_manager.closeAllScripts();  
77                 break;  
78         }  
79     });  
80 }
```

```

79      case "set-language":
80        language.set(data);
81        break;
82      case "toggle-comment":
83        obj.script_manager.toggleComment();
84        break;
85    }
86  );
87
88 // On file drop
89 document.addEventListener("drop", function(e) {
90   e.stopPropagation();
91   e.preventDefault();
92   console.log(e.dataTransfer.files);
93
94   for(const f of e.dataTransfer.files) {
95     obj.script_manager.openScript([f.path]);
96   }
97 }, false);
98
99 /**
100 * Retrieves selected text within the application, if any.
101 * @return {string} The currently selected text.
102 */
103 getSelectionText() {
104   var text = "";
105   if(window.getSelection) {
106     text = window.getSelection().toString();
107   } else if(document.selection && document.selection.type != "Control") {
108     text = document.selection.createRange().text;
109   }
110   return text;
111 }
112
113 /**
114 * Method used to execute code when gui is ready.
115 */
116 onReady() {
117   // Fade in window
118   ipcRenderer.send('MainProcess', 'fade-in-win');
119 }
120
121 /**
122 * Show editor
123 */
124 showEditor() {
125   ipcRenderer.send("MainProcess", "show-editor");
126 }
127
128 }
129 exports.PRDC_JSLAB_WIN_EDITOR = PRDC_JSLAB_WIN_EDITOR;

```

Listing 56 - win-editor.js

6.4 main

```

1  /**
2   * @file JSLAB app module
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   */
7
8  const os = require('os');
9  const cp = require('child_process');
10
11 /**
12  * Class for JSLAB app.
13 */
14 class PRDC_JSLAB_APP {
15
16 /**
17  * Initializes application properties, fetching system and environment
18  * information relevant to the application.
19  * @param {object} win The window object representing the current Electron
20  * window.
21  */
22 constructor(win) {
23   var obj = this;
24   this.win = win;
25
26   this.version;
27   this.user;
28   this.documents_path;
29
30   // Platform specific code
31   this.computer_name;
32   switch(process.platform) {
33     case 'win32':
34       this.computer_name = process.env.COMPUTERNAME;
35       break;
36     case 'darwin':
37       this.computer_name = cp.execSync('scutil --get ComputerName')
38         .toString().trim();
39       break;
40     default:
41       this.computer_name = os.hostname();
42       break;
43   }
44   this.version = ipcRenderer.sendSync('sync-message', 'get-app-version');
45   this.documents_path = ipcRenderer.sendSync('sync-message',
46     'get-path', 'documents');
47   this.exe_path = ipcRenderer.sendSync('sync-message',
48     'get-path', 'exe');
49   this.user = os.userInfo().username + '@' + this.computer_name;
50 }
51 /**
52  * Generates a formatted string representing the current date and time,

```

```

  suitable for timestamps and logging.
52  * @param {number} [t=Date.now()] Optional timestamp to format, defaults to
53   the current time if not provided.
54  * @returns {string} A string formatted to represent a date and time,
55   structured as DD_MM_YYYY. HH_MM_SS_mmm.
56  */
57  getDateTimeFullStr(t = Date.now()) {
58    var d = new Date(t);
59    return ('0' + d.getDate()).slice(-2) + "_" +
60      ('0' + (d.getMonth() + 1)).slice(-2) + "_" +
61      d.getFullYear() + '.' + ('0' + d.getHours()).slice(-2) + "_" +
62      ('0' + d.getMinutes()).slice(-2) + "_" +
63      ('0' + d.getSeconds()).slice(-2) + "_" +
64      ('00' + d.getMilliseconds()).slice(-3);
65  }
66 exports.PRDC_JSLAB_APP = PRDC_JSLAB_APP;

```

Listing 57 - app.js

```

1 /**
2  * @file JSLAB command history module
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 const Store = require('electron-store');
9
10 const store = new Store();
11
12 /**
13  * Class for JSLAB command history.
14  */
15 class PRDC_JSLAB_COMMAND_HISTORY {
16
17 /**
18  * Initializes the command history.
19  * @param {object} win The window object representing the current Electron
20   window.
21  */
22  constructor(win) {
23    var obj = this;
24    this.win = win;
25    this.history_cont = document.getElementById('command-history');
26    this.full_history = store.get('full_history') || [];
27    this.history = [];
28
29    this.N_history_max = Number(store.get('N_history_max'));
30    if (!isFinite(this.N_history_max)) {
31      this.setMaxSize(20);
32    }
33
34    this.renderHistory();

```

```

35   // Append initialization command
36   const init_cmd = `// JSLAB ${this.win.app.version}, ${new Date()} [${this.
37     win.app.user}]`;
38   this.updateHistory(init_cmd);
39
40   // History clear button click
41   const clear_button = document.querySelector('#command-history-options .
42     clear');
43   if(clear_button) {
44     clear_button.addEventListener('click', function() {
45       obj.clearHistory()
46     });
47
48   // Event delegation for command interactions
49   this.history_cont.addEventListener('click', function(e) {
50     if(e.target && !e.target.classList.contains('comment')) {
51       obj.selectCommand(e.target.textContent);
52     }
53   });
54
55   this.history_cont.addEventListener('dblclick', function(e) {
56     if(e.target && !e.target.classList.contains('comment')) {
57       obj.evalSelectedCommand(e.target.textContent);
58     }
59   });
60
61 /**
62 * Renders the entire history efficiently using DocumentFragment.
63 */
64 renderHistory() {
65   const fragment = document.createDocumentFragment();
66
67   this.full_history.forEach((cmd) => {
68     const div = document.createElement('div');
69     if(cmd.startsWith('//')) {
70       div.classList.add('comment');
71     }
72     div.textContent = cmd.replace(/\//g, '\\\\');
73     fragment.appendChild(div);
74   });
75
76   this.history_cont.appendChild(fragment);
77   this.scrollToBottom();
78 }
79
80 /**
81 * Adds a command to the history, updating the UI and internal storage
82 * accordingly.
83 * @param {string} cmd The command to be added to the history.
84 */
85 updateHistory(cmd) {
86   if(this.full_history.length >= this.N_history_max) {
87     this.full_history.shift();

```

```
87     if(this.history_cont.firstChild) {
88         this.history_cont.removeChild(this.history_cont.firstChild);
89     }
90 }
91
92 const div = document.createElement('div');
93 if(cmd.startsWith('///')) {
94     div.classList.add('comment');
95 } else {
96     this.history.unshift(cmd)
97 }
98 div.textContent = cmd;
99 this.history_cont.appendChild(div);
100 this.full_history.push(cmd);
101
102 this.scrollToBottom();
103 this.saveHistory();
104 }
105
106 /**
107 * Scrolls the history container to the bottom.
108 */
109 scrollToBottom() {
110     this.history_cont.scrollTop = this.history_cont.scrollHeight;
111 }
112
113 /**
114 * Selects a command from the history, placing it in the command input for
115 * potential modification and re-execution.
116 * @param {string} cmd The command to select.
117 */
118 selectCommand(cmd) {
119     const code_input = this.win.command_window.code_input;
120     code_input.setValue(cmd);
121     code_input.focus();
122     code_input.setCursor(code_input.lineCount(), 0);
123 }
124
125 /**
126 * Evaluates a selected command from the history, directly executing it
127 * without modification.
128 * @param {string} cmd The command to evaluate.
129 */
130 evalSelectedCommand(cmd) {
131     this.win.eval.evalCommand(cmd);
132 }
133
134 /**
135 * Clears the command history.
136 */
137 clearHistory() {
138     this.full_history = [];
139     this.history_cont.innerHTML = '';
140     this.saveHistory();
141 }
```

```

140
141  /**
142   * Saves the current history to storage asynchronously.
143   */
144 saveHistory() {
145   store.set('full_history', this.full_history);
146 }
147
148 /**
149  * Sets the maximum number of commands to retain in the history.
150  * @param {number} N - The desired maximum number of history entries
151  */
152 setMaxSize(N) {
153   if(N < 5) {
154     N = 5;
155   }
156   this.N_history_max = N;
157   $('#settings-container .N-history-max').val(N);
158   while(this.full_history.length >= this.N_history_max) {
159     this.full_history.shift();
160     if(this.history_cont.firstChild) {
161       this.history_cont.removeChild(this.history_cont.firstChild);
162     }
163   }
164   store.set('N_history_max', this.N_history_max);
165   this.saveHistory();
166 }
167 }
168
169 exports.PRDC_JSLAB_COMMAND_HISTORY = PRDC_JSLAB_COMMAND_HISTORY;

```

Listing 58 - command-history.js

```

1 /**
2  * @file JSLAB command window module
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 const { shell } = require('electron');
9 const { BigJsonViewerDom } = require('big-json-viewer');
10 const Store = require('electron-store');
11 const { ESLint } = require("eslint");
12
13 const store = new Store();
14
15 /**
16  * Class for JSLAB command window.
17  */
18 class PRDC_JSLAB_COMMAND_WINDOW {
19
20 /**
21  * Initializes the command window, setting up the UI components, event
22  * listeners, and loading settings from storage.
23  * @param {object} win The window object representing the current Electron

```

```
        window.  
23     */  
24     constructor(win) {  
25         var obj = this;  
26         this.win = win;  
27  
28         this.terminal_cont = document.getElementById('right-panel');  
29  
30         this.terminal_history_cont;  
31         this.terminal_options_cont;  
32         this.messages;  
33         this.autoscroll = true;  
34         this.show_timestamp = false;  
35         this.write_timestamps = true;  
36         this.log = [];  
37         this.log_dialog;  
38         this.settings_dialog;  
39         this.N_messages = 1;  
40         this.N_messages_max = Infinity;  
41         this.code_input;  
42         this.textarea;  
43         this.ignore_output = false;  
44         this.no_ans = false;  
45         this.i_history = -1;  
46  
47         this.last_class;  
48         this.last_tic;  
49  
50         this.eslint = new ESLint(config.LINT_OPTIONS);  
51  
52         // Load settings  
53         this.autoscroll = store.get('autoscroll');  
54         if(!this.autoscroll) {  
55             this.autoscroll = true;  
56         }  
57         this.show_timestamp = store.get('show_timestamp');  
58         if(!this.show_timestamp) {  
59             this.show_timestamp = false;  
60         }  
61         this.N_messages_max = Number(store.get('N_messages_max'));  
62         if(isFinite(this.N_messages_max) || this.N_messages_max == 0) {  
63             this.N_messages_max = Infinity;  
64         }  
65         this.write_timestamps = store.get('write_timestamps');  
66         if(!this.write_timestamps) {  
67             this.write_timestamps = true;  
68         }  
69  
70         // Create terminal DOM  
71         this.messages = $('#right-panel .messages');  
72  
73         // Welcome message  
74         var ts = this.getTimestamp();  
75         $(this.messages).html('<div class="welcome-message system-in message"><  
span class="timestamp">' + ts + '</span><
77 div class="clear"></div><p><span>JSLAB</span>, '+language.string(8)+'
78   ' + this.win.app.version + '</p><p>' +language.string(136)+ ' + new
79 Date().getFullYear() + '@ <span>PR-DC</span> info@pr-dc.com</p><p>' +
80 language.string(137)+ '</p><p>' +language.string(138)+ '</p><p>' +language.
81 string(139)+' <span>cmd_help</span></p><p>' +language.string(135)+' <a
82 href="https://pr-dc.com/jslab">pr-dc.com/jslab </a></p></div>');
83
84 // Command history
85 this.terminal_history_cont =
86 $( '#right-panel .history-cont' );
87 $( '#right-panel .history-close' ).click( function () {
88   obj.closeTerminalDialog( obj.terminal_history_cont );
89 });
90 this.terminal_history_cont.on( 'keydown', function (e) {
91   // https://keycode.info/
92   function activateCommand() {
93     var el_a = $( '#right-panel .history-cont .history-panel li.active' );
94     var el = $( '#right-panel .history-cont .history-panel li[i="'+obj.
95       i_history+'"]');
96     el_a.removeClass('active');
97     el.addClass('active');
98     el[0].scrollIntoView({block: 'center', inline: 'center'});
99     obj.code_input.setValue( obj.win.command_history.history[obj.i_history
100      ] );
101    obj.code_input.setCursor( obj.code_input.lineCount(), 0 );
102  }
103
104  if( e.key === 'Enter' && !e.shiftKey ) {
105    // Enter
106    var el_a = $( '#right-panel .history-cont .history-panel li.active' );
107    var cmd = el_a.html();
108    obj.win.eval.evalCommand(cmd);
109    obj.closeTerminalDialog( obj.terminal_history_cont );
110    e.stopPropagation();
111    e.preventDefault();
112  } else if( e.key === 'Escape' ) {
113    // ESC
114    obj.closeTerminalDialog( obj.terminal_history_cont );
115    e.stopPropagation();
116    e.preventDefault();
117  } else if( e.key === 'ArrowUp' ) {
118    // Arrow up
119    if( obj.i_history > 0 ) {
120      obj.i_history -= 1;
121      activateCommand();
122    }
123    e.stopPropagation();
124    e.preventDefault();
125  } else if( e.key === 'ArrowDown' ) {
126    // Arrow down
127    if( obj.i_history < (obj.win.command_history.history.length-1) ) {
128      obj.i_history += 1;
129      activateCommand();
130    }
131  }
132}
133
```

```

122     e.stopPropagation();
123     e.preventDefault();
124 } else if(e.key == 'PageUp') {
125   // Page up
126   if(obj.win.command_history.history.length) {
127     obj.i_history = 0;
128     activateCommand();
129   }
130   e.stopPropagation();
131   e.preventDefault();
132 } else if(e.key == 'PageDown') {
133   // Page down
134   if(obj.win.command_history.history.length) {
135     obj.i_history = obj.win.command_history.history.length - 1;
136     activateCommand();
137   }
138   e.stopPropagation();
139   e.preventDefault();
140 }
141 });
142
143 // Create settings dialog
144 this.settings_dialog =
145 $('#right-panel .terminal-settings');
146 this.settings_dialog.on('keydown', function(e) {
147   if(e.key == 'Escape') {
148     // ESC
149     obj.closeTerminalDialog(obj.settings_dialog);
150     e.stopPropagation();
151     e.preventDefault();
152   }
153 });
154 $('#right-panel .terminal-settings .options-close').click(function(){
155   obj.closeTerminalDialog(obj.settings_dialog);
156 });
157 this.setNMessagesMax();
158 $('#right-panel .terminal-settings .change-settings').click(function(){
159   obj.closeTerminalDialog(obj.settings_dialog);
160   obj.N_messages_max =
161     Number($('#right-panel .terminal-settings .N-messages-max').val());
162   obj.setNMessagesMax();
163   if(obj.N_messages > obj.N_messages_max) {
164     while(obj.N_messages > obj.N_messages_max) {
165       obj.messages[0].firstChild.remove();
166       obj.N_messages -= 1;
167     }
168   }
169 });
170
171 // Create log save dialog
172 this.log_dialog = $('#right-panel .terminal-log');
173 this.log_dialog.on('keydown', function(e) {
174   if(e.key == 'Escape') {
175     // ESC
176     obj.closeTerminalDialog(obj.log_dialog);

```

```

177         e.stopPropagation();
178         e.preventDefault();
179     }
180 });
181 $('#right-panel .terminal-log .options-close').click(function() {
182     obj.closeTerminalDialog(obj.log_dialog);
183 });
184 $('#right-panel .terminal-log .write-timestamps').click(function() {
185     obj.write_timestamps = this.checked;
186     obj.setWriteTimestamps();
187 });
188 this.setWriteTimestamps();
189 $('#right-panel .terminal-log .save-log').click(function() {
190     obj.closeTerminalDialog(obj.log_dialog);
191     obj.saveLog();
192 });
193
194 // Document keydown callbacks
195 $(document).on('keydown', function(e) {
196     if(e.key.toLowerCase() === 'f' && e.ctrlKey) {
197         // Ctrl + F
198         obj.scrollToBottom();
199         obj.code_input.focus();
200         obj.code_input.setCursor(obj.code_input.lineCount(), 0);
201         e.stopPropagation();
202         e.preventDefault();
203     }
204 });
205
206 // Create code input
207 this.code_input = CodeMirror.fromTextArea(
208     document.getElementById('command-window-input'), {
209     mode: 'javascript',
210     theme: 'notepadpp',
211     indentUnit: 2,
212     tabSize: 2,
213     lineWrapping: true,
214     matchBrackets: true,
215     gutter: true,
216     gutters: ['CodeMirror-lint-markers'],
217     lint: {
218         getAnnotations: async function(text, callback) {
219             var results = await obj.eslint.lintText(text);
220             callback(results[0].messages.map(message => ({
221                 from: CodeMirror.Pos(message.line - 1, message.column - 1),
222                 to: CodeMirror.Pos(
223                     message.endLine ? message.endLine - 1 : message.line - 1,
224                     message.endColumn ? message.endColumn - 1 : message.column
225                 ),
226                 severity: message.severity === 2 ? "error" : "warning",
227                 message: message.message,
228             })));
229         },
230         async: true
231     },

```

```

232     highlightSelectionMatches: { annotateScrollbar: true },
233     viewportMargin: Infinity ,
234     extraKeys: { Enter: function() {
235       sendCommand();
236     } }
237   });
238
239   CodeMirror.keyMap.default['Shift-Tab'] = 'indentLess';
240   CodeMirror.keyMap.default['Tab'] = 'indentMore';
241
242   this.code_input.on('keypress', function(cm, event) {
243     if (!cm.state.completionActive && !event.ctrlKey &&
244       event.key != 'Enter' &&
245       event.key != ';' && event.key != ',' &&
246       event.key != '{' & event.key != '}') {
247       CodeMirror.commands.autocomplete(cm, null,
248         {completeSingle: false});
249     }
250   });
251
252   // Code input keydown callbacks
253   function historyUp() {
254     if (obj.i_history < (obj.win.command_history.history.length - 1)) {
255       obj.i_history += 1;
256       obj.code_input.setValue(obj.win.command_history.history[obj.i_history]);
257       obj.code_input.setCursor(obj.code_input.lineCount(), 0);
258       obj.scrollToBottom();
259     }
260   }
261
262   function historyDown() {
263     if (obj.i_history > 0) {
264       obj.i_history -= 1;
265       obj.code_input.setValue(obj.win.command_history.history[obj.i_history]);
266       obj.code_input.setCursor(obj.code_input.lineCount(), 0);
267     }
268   }
269
270   function sendCommand() {
271     var cmd = obj.code_input.getValue();
272     obj.win.eval.evalCommand(cmd);
273   }
274
275   this.code_input.on('keydown', function(cm, e) {
276     // https://keycode.info/
277     if (e.key == 'Escape' &&
278       !obj.code_input.state.completionActive) {
279       // ESC
280       obj.code_input.setValue('');
281       obj.resetHistoryIndex();
282       obj.scrollToBottom();
283       e.stopPropagation();
284       e.preventDefault();

```

```

285 } else if(e.key == 'ArrowUp' &&
286   !obj.code_input.state.completionActive) {
287   // Arrow up
288
289   var cursor = obj.code_input.getCursor();
290   var line = obj.code_input.lineCount() - 1;
291   var position = obj.code_input.getLine(line).length;
292   if(cursor.line == 0 && (cursor.ch == position || cursor.ch == 0)) {
293     historyUp(e);
294     e.stopPropagation();
295     e.preventDefault();
296   }
297 } else if(e.key == 'ArrowDown' &&
298   !obj.code_input.state.completionActive) {
299   // Arrow down
300
301   var cursor = obj.code_input.getCursor();
302   var line = obj.code_input.lineCount() - 1;
303   var position = obj.code_input.getLine(line).length;
304   if(cursor.line == line && (cursor.ch == position || cursor.ch == 0)) {
305     historyDown(e);
306     e.stopPropagation();
307     e.preventDefault();
308   }
309 } else if(e.key == 'PageUp') {
310   // Page up
311   if(obj.win.command_history.history.length) {
312     obj.i_history = obj.win.command_history.history.length - 1;
313     obj.code_input.setValue(obj.win.command_history.history[obj.win.
314       command_history.history.length - 1]);
315     obj.code_input.setCursor(obj.code_input.lineCount(), 0);
316     obj.scrollToBottom();
317   }
318   e.stopPropagation();
319   e.preventDefault();
320 } else if(e.key == 'PageDown') {
321   // Page down
322   if(obj.win.command_history.history.length) {
323     obj.i_history = 0;
324     obj.code_input.setValue(obj.win.command_history.history[0]);
325     obj.code_input.setCursor(obj.code_input.lineCount(), 0);
326     obj.scrollToBottom();
327   }
328   e.stopPropagation();
329   e.preventDefault();
330 } else if(e.key == 'F3') {
331   // F3
332   if(obj.win.command_history.history.length) {
333     var new_cmd = obj.win.command_history.history[0];
334     obj.win.evalCommand(new_cmd);
335   }
336   e.stopPropagation();
337   e.preventDefault();
338 } else if(e.key == 'F7' && e.altKey) {
339   // Alt + F7

```

```

339     obj.resetHistoryIndex();
340     obj.win.command_history.history = [];
341     e.stopPropagation();
342     e.preventDefault();
343 } else if(e.key === 'F7') {
344   // F7
345   obj.openTerminalDialog(obj.terminal_history_cont);
346   var cmds_cont = $('#right-panel .history-cont .history-panel');
347   cmds_cont.html('');
348   if(obj.win.command_history.history.length) {
349     obj.win.command_history.history.forEach(function(e, i) {
350       if(i === obj.i_history) {
351         cmds_cont.append('<li i="' + i + '" class="active">' + e + '</li>');
352       } else {
353         cmds_cont.append('<li i="' + i + '">' + e + '</li>');
354       }
355     });
356   var el_a = $('#right-panel .history-cont .history-panel li.active');
357   if(el_a.length > 0) {
358     el_a[0].scrollIntoView({block: 'center', inline: 'center'});
359   }
360   var cmds = cmds_cont.find('li');
361   cmds.on('click', function() {
362     cmds.removeClass('active');
363     $(this).addClass('active');
364     obj.i_history = Number($(this).attr('i'));
365   });
366 } else {
367   cmds_cont
368     .append('<div class="history-empty">History is empty!</div>');
369 }
370 e.stopPropagation();
371 e.preventDefault();
372 } else if(e.key === 'F8') {
373   // F8
374   var cursor = obj.code_input.getCursor();
375   var line = cursor.line;
376   var position = cursor.ch;
377   var cmd = obj.code_input.getLine(line).substring(0, position);
378   var j = -1;
379   for(var i = 0; i < obj.win.command_history.history.length; i++) {
380     if(obj.win.command_history.history[i].startsWith(cmd)) {
381       if(j > -1) {
382         if(i > obj.i_history) {
383           j = i;
384           break;
385         }
386       } else {
387         j = i;
388         if(obj.i_history == -1) {
389           break;
390         }
391       }
392     }
393   }

```

```

394
395   }
396   if(j > -1) {
397     obj.i_history = j;
398     obj.code_input.setValue(obj.win.command_history.history[obj.i_history]);
399     if(obj.win.command_history.history[obj.i_history].length > position)
400       {
401         obj.code_input.focus();
402         obj.code_input.setCursor(cursor);
403       }
404     e.stopPropagation();
405     e.preventDefault();
406   } else if(e.key.toLowerCase() == 's' && e.ctrlKey) {
407     // Ctrl + S
408     obj.openTerminalDialog(obj.settings_dialog);
409     e.stopPropagation();
410     e.preventDefault();
411   } else if(e.key.toLowerCase() == 'l' && e.ctrlKey) {
412     // Ctrl + L
413     obj.openTerminalDialog(obj.log_dialog);
414     e.stopPropagation();
415     e.preventDefault();
416   });
417
418 // Focus code input
419 $('#right-panel .terminal-panel').click(function(e){
420   if(e.target != this) return;
421   obj.code_input.focus();
422   obj.code_input.setCursor(obj.code_input.lineCount(), 0);
423 });
424
425 $('#command-window-input-container').click(function(e){
426   if(e.target != this) return;
427   obj.code_input.focus();
428   obj.code_input.setCursor(obj.code_input.lineCount(), 0);
429 });
430
431 // Terminal options cont
432 this.terminal_options_cont = $('#right-panel .options');
433
434 // Terminal settings button click
435 $('#right-panel .options .settings').click(function(){
436   obj.openTerminalDialog(obj.settings_dialog);
437 });
438
439 // Terminal timestamp button click
440 $('#right-panel .options .timestamp').click(function(){
441   if(obj.show_timestamp) {
442     obj.show_timestamp = false;
443   } else {
444     obj.show_timestamp = true;
445   }
446   obj.setTimestamp();

```

```

447 });
448 this.setTimestamp();
449
450 // Terminal auto scroll button click
451 $('#right-panel .options .autoscroll').click(function(){
452     if(obj.autoscroll) {
453         obj.autoscroll = false;
454     } else {
455         obj.autoscroll = true;
456     }
457     obj.setAutoscroll();
458 });
459 this.setAutoscroll();
460
461 // Terminal clear button click
462 $('#right-panel .options .clear').click(function(){
463     obj.clear();
464 });
465
466 // Terminal save log button click
467 $('#right-panel .options .log').click(function(){
468     obj.openTerminalDialog(obj.log_dialog);
469 });
470
471 // Terminal scroll to bottom button click
472 $('#right-panel .options .to-bottom').click(function(){
473     obj.scrollToBottom();
474     obj.code_input.focus();
475     obj.code_input.setCursor(obj.code_input.lineCount(), 0);
476 });
477
478 // Prevent all redirects
479 $(document.body).on('click', '#command-window-messages a', function(e) {
480     if(e.target.href) {
481         e.preventDefault();
482         shell.openExternal(e.target.href);
483         return false;
484     }
485     return true;
486 });
487
488 // Commands for evaluation
489 $(document.body).on('click', '#command-window-messages span.eval-code',
490     function() {
491         obj.win.eval.evalCommand(this.innerText);
492     });
493
494 /**
495 * Resets the index used for navigating through command history to its
496 * default state.
497 */
498 resetHistoryIndex() {
499     this.i_history = -1;
500 }

```

```

500
501 /**
502 * Applies syntax highlighting to a given snippet of code, returning HTML
503 * markup with syntax highlighting styles applied.
504 * @param {string} data The code snippet to which syntax highlighting should
505 * be applied.
506 * @returns {string} HTML string representing the highlighted code. Special
507 * HTML characters like '<' and '>' are properly escaped.
508 */
509 highlightCode(data) {
510     return hljs.highlight(data,
511         {language: 'javascript'}).value
512         .replaceAll('<', '<').replaceAll('>', '>'); // Return < and >
513 }
514
515 /**
516 * Clears all messages from the command window.
517 */
518 clear() {
519     this.N_messages = 0;
520     $(this.messages).html(' ');
521 }
522
523 /**
524 * Displays an error message in the command window.
525 * @param {string} msg The error message to display.
526 */
527 error(msg) {
528     this.win.workspace.updateWorkspace();
529     return this.addMessage('data-in', '<span class="error">' +
530         this.prettyPrint(msg) + '</span>');
531 }
532
533 /**
534 * Displays a warning message in the command window.
535 * @param {string} msg The warning message to display.
536 */
537 warn(msg) {
538     this.win.workspace.updateWorkspace();
539     return this.addMessage('data-in', '<span class="warn">' +
540         this.prettyPrint(msg) + '</span>');
541 }
542
543 /**
544 * Highlights and displays a response message in the command window,
545 * particularly used for 'ans' variable responses.
546 * @param {Array|String} data The data to be highlighted and displayed.
547 */
548 highlightAnsMessage(data) {
549     if(data[1]) {
550         var el = this.message('ans = ', 'ans = '+data[0]);
551         BigJsonViewerDom.fromData(data[0]).then(function(viewer) {
552             const node = viewer.getRootElement();
553             el[0].appendChild(node);
554             node.openAll(1);
555     }

```

```

551     }) . catch(function(err) {
552         console.log(err);
553     });
554 } else {
555     this.message(this.highlightCode('ans = ' + data[0]), 'ans = ' + data[0])
556     ;
557 }
558 /**
559 * Displays a general message in the command window.
560 * @param {string} msg The message to display.
561 * @param {string} raw Raw message to log.
562 */
563 message(msg, raw) {
564     this.win.workspace.updateWorkspace();
565     return this.addMessage('data-in', this.prettyPrint(msg), raw);
566 }
567 /**
568 * Displays a general message in the command window with monospaced font.
569 * @param {string} msg The message to display.
570 * @param {string} raw Raw message to log.
571 */
572 messageMonospaced(msg, raw) {
573     this.win.workspace.updateWorkspace();
574     return this.addMessage('data-in', '<div class="monospaced">' + this.
575         prettyPrint(msg) + '</div>', raw);
576 }
577 /**
578 * Displays a general message in the command window.
579 * @param {string} msg The message to display.
580 */
581 messageLatex(msg) {
582     this.win.workspace.updateWorkspace();
583     var el = this.addMessage('data-in', '\\\\(' + msg + '\\\\)');
584     MathJax.typeset(el);
585     return el;
586 }
587 /**
588 * Displays a message from internal operations in the command window.
589 * @param {string} msg The internal message to display.
590 */
591 messageInternal(msg) {
592     this.addMessage('system-in', this.prettyPrint(msg));
593 }
594 /**
595 * Displays an error message originating from internal operations or system
596 * errors.
597 * @param {string} msg The error message to display.
598 */
599 errorInternal(msg) {
600
601
602

```

```

603     this.addMessage('system-in', '<span class="error">' +
604       this.prettyPrint(msg) + '</span>');
605   }
606
607 /**
608 * Displays a message related to editor operations in the command window.
609 * @param {string} msg The editor message to display.
610 */
611 messageEditor(msg) {
612   this.addMessage('system-in', '<span class="log">Editor: ' +
613     this.prettyPrint(msg) + '</span>');
614 }
615
616 /**
617 * Opens a specified dialog related to the terminal, like settings or
618 * history.
619 * @param {jQuery} e The jQuery object representing the dialog to open.
620 */
621 openTerminalDialog(e) {
622   if(!e.is(':visible')) {
623     $('.terminal-dialog').fadeOut(300, 'linear');
624     e.fadeIn(300, 'linear', function() {
625       e.focus();
626     });
627   }
628
629 /**
630 * Closes a specified dialog related to the terminal.
631 * @param {jQuery} e The jQuery object representing the dialog to close.
632 */
633 closeTerminalDialog(e) {
634   e.fadeOut(300, 'linear');
635   this.code_input.focus();
636   this.code_input.setCursor(this.code_input.lineCount(), 0);
637 }
638
639 /**
640 * Adds a command as a message to the terminal, applying syntax highlighting
641 * and pretty printing.
642 * @param {string} cmd The command to add to the terminal output.
643 */
644 addMessageCmd(cmd) {
645   var txt = this.prettyPrint(cmd);
646   this.addMessage('data-out', this.highlightCode(txt), txt);
647 }
648
649 /**
650 * Adds a message to the terminal output. Supports merging messages for
651 * continuous output scenarios.
652 * @param {string} msg_class The CSS class to apply to the message, defining
653 * its type (e.g., 'system-in', 'data-out').
654 * @param {string} data The message content, which can include HTML markup.
655 * @param {boolean} [merge_messages=false] Whether to merge this message
656 * with the previous one if they are of the same class.

```

```

653   * @returns { Object } A jQuery object representing the created message
654   * element.
655   */
656 addMessage(msg_class, data, raw, merge_messages = false) {
657   if(typeof raw == 'undefined') {
658     raw = data;
659   }
660   var t = performance.now();
661   var el;
662   if(msg_class != this.last_class) {
663     this.last_class = msg_class;
664     this.last_tic = t;
665     var ts = this.getTimestamp();
666     if(this.N_messages < this.N_messages_max) {
667       this.N_messages += 1;
668     } else {
669       this.messages[0].firstChild.remove();
670     }
671     el = $(`<div class="${msg_class} +
672       ${ts}><span class="timestamp"> +
673         ${ts}</span> + ${data}</div>`);
674     $(this.messages).append(el);
675     this.log.push({ 'class': msg_class, 'timestamp': ts, 'data': raw });
676   } else {
677     if(!merge_messages || t - this.last_tic > 1) {
678       this.last_tic = t;
679       var ts = this.getTimestamp();
680       if(this.N_messages < this.N_messages_max) {
681         this.N_messages += 1;
682       } else {
683         this.messages[0].firstChild.remove();
684       }
685       el = $(`<div class="${msg_class} +
686         ${ts}><span class="timestamp"> +
687           ${ts}</span> + ${data}</div>`);
688       $(this.messages).append(el);
689       this.log.push({ 'class': msg_class, 'timestamp': ts, 'data': raw });
690     } else {
691       el = $(this.messages).find('div').last();
692       el.append(data);
693       this.log[this.log.length - 1].data += raw;
694     }
695   }
696   if(this.autoscroll) {
697     this.scrollToBottom();
698   }
699   return el;
700 }
701 /**
702 * Toggles the visibility of timestamps in the command window.
703 */
704 setTimestamp() {
705   var timestamp_button =
706     $('#right-panel .options .timestamp');

```

```

707     if (this.show_timestamp) {
708         if (!$(this.messages).hasClass('no-timestamp')) {
709             $(this.messages).removeClass('no-timestamp');
710             $(timestamp_button).addClass('active');
711             $(timestamp_button).attr('title', language.currentString(41));
712             $(timestamp_button).attr('title-str', 41);
713         }
714     } else {
715         if (!$(this.messages).hasClass('no-timestamp')) {
716             $(this.messages).addClass('no-timestamp');
717             $(timestamp_button).removeClass('active');
718             $(timestamp_button).attr('title', language.currentString(166));
719             $(timestamp_button).attr('title-str', 166);
720         }
721     }
722     store.set('show_timestamp', this.show_timestamp);
723 }
724
725 /**
726 * Toggles the autoscroll feature of the command window, ensuring the latest
727 messages are always in view.
728 */
729 setAutoscroll() {
730     var autoscroll_button =
731         $('#right-panel.options.autoscroll');
732     if (this.autoscroll) {
733         if (!$(autoscroll_button).hasClass('active')) {
734             $(autoscroll_button).addClass('active');
735             $(autoscroll_button).attr('title', language.currentString(42));
736             $(autoscroll_button).attr('title-str', 42);
737         }
738     } else {
739         if ($(autoscroll_button).hasClass('active')) {
740             $(autoscroll_button).removeClass('active');
741             $(autoscroll_button).attr('title', language.currentString(167));
742             $(autoscroll_button).attr('title-str', 167);
743         }
744     }
745     store.set('autoscroll', this.autoscroll);
746 }
747
748 /**
749 * Sets the maximum number of messages to display in the command window
750 before older messages are removed.
751 */
752 setNMessagesMax() {
753     var N_messages_max_input =
754         $('#right-panel.terminal-settings.N-messages-max');
755     if (this.N_messages_max < 5) {
756         this.N_messages_max = 5;
757     }
758     if ($(N_messages_max_input).val() != this.N_messages_max) {
759         $(N_messages_max_input).val(this.N_messages_max);
760     }
761     store.set('N_messages_max', this.N_messages_max);

```

```

760 }
761
762 /**
763 * Toggles whether timestamps are written to the log file.
764 */
765 setWriteTimestamps() {
766   var write_timestamps_input =
767     $('#right-panel .terminal-log .write-timestamps')[0];
768   if(this.write_timestamps) {
769     if(!write_timestamps_input.checked) {
770       write_timestamps_input.checked = true;
771     }
772   } else {
773     if(write_timestamps_input.checked) {
774       write_timestamps_input.checked = false;
775     }
776   }
777   store.set('write_timestamps', this.write_timestamps);
778 }
779
780 /**
781 * Saves the current log of the command window to a file.
782 */
783 saveLog() {
784   let options = {
785     title: language.currentString(150),
786     defaultPath: 'jslab_' + this.win.app.getDateTimeFullStr() + '.log',
787     buttonLabel: language.currentString(151),
788     filters: [
789       {name: 'Log', extensions: ['log', 'txt']},
790       {name: 'All Files', extensions: ['*']}
791     ]
792   };
793
794   var obj = this;
795   ipcRenderer.invoke('dialog', 'showSaveDialog', options).then(function(
796     result) {
797     if(!result.canceled) {
798       var data = '';
799       obj.log.forEach(function(x) {
800         data += x.class + ': ';
801         if(obj.write_timestamps) {
802           data += '[' + x.timestamp + '] ';
803         }
804         data += x.data + '\r\n';
805       });
806
807       const fs = require('fs');
808       fs.writeFile(result.filePath, data, function(err) {
809         if(err) {
810           obj.errorInternal(err);
811         }
812       });
813     })
814   }).catch(function(err) {

```

```

814     obj.errorInternal(err);
815   });
816 }
817
818 /**
819 * Scrolls the command window to the bottom, ensuring the latest messages
820 * are visible.
821 */
822 scrollToBottom() {
823   var bcr = this.messages[0].getBoundingClientRect();
824   $(this.messages).parent()[0].scrollTop = bcr.height;
825 }
826
827 /**
828 * Generates a timestamp for use in the command window.
829 * @returns {string} A string representing the current timestamp.
830 */
831 getTimestamp() {
832   var date = new Date();
833   var pad = function(num, size) {
834     return ('000' + num).slice(size * -1);
835   };
836   var time = parseFloat(date.getTime() / 1000).toFixed(3);
837   var hours = date.getHours();
838   var minutes = Math.floor(time / 60) % 60;
839   var seconds = Math.floor(time - minutes * 60);
840   var milliseconds = time.slice(-3);
841
842   return pad(hours, 2) + ':' + pad(minutes, 2) + ':' +
843         pad(seconds, 2) + '.' + pad(milliseconds, 3);
844 }
845
846 /**
847 * Provides a function to replace circular references while stringifying an
848 * object. Useful for logging objects with circular references.
849 * @returns {Function} A replacer function for JSON.stringify.
850 */
851 getCircularReplacer() {
852   const seen = new WeakSet();
853   return function(key, value) {
854     if(typeof value === 'object' && value !== null) {
855       if(seen.has(value)) {
856         return;
857       }
858       seen.add(value);
859     }
860     return value;
861   };
862 }
863
864 /**
865 * Formats and pretty-prints data for display in the command window.
866 * @param {object|string} data The data to format.
867 * @returns {string} The formatted data as a string.
868 */

```

```

867   prettyPrint(data) {
868     if(typeof data == 'string') {
869       return data.replace(/\n/g, '<br/>');
870     } else if(typeof data == 'object') {
871       if(!Object.keys(data).length){
872         if(data.constructor.name == 'Error') {
873           return data.stack.toString();
874         } else {
875           return data.toString();
876         }
877       } else {
878         return JSON.stringify(data, this.getCircularReplacer(), 2);
879       }
880     } else {
881       return String(data);
882     }
883   }
884 }
885
886 exports.PRDC_JSLAB_COMMAND_WINDOW = PRDC_JSLAB_COMMAND_WINDOW;

```

Listing 59 - command-window.js

```

1  /**
2   * @file JSLAB eval module
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   */
7
8 const path = require('path');
9
10 /**
11  * Class for JSLAB eval.
12  */
13 class PRDC_JSLAB_EVAL {
14
15 /**
16  * Initializes the script evaluation functionality, setting up the necessary
17  * environment and variables.
18  * @param {object} win The window object representing the current Electron
19  * window.
20  */
21 constructor(win) {
22   var obj = this;
23   this.win = win;
24
25   this.last_script_path;
26   this.jslfilename = 'jslcmdwindow';
27 }
28
29 /**
30  * Evaluates a script from the specified path, optionally focusing on
31  * specific lines. Manages the evaluation process to prevent overlap with
32  * ongoing evaluations.
33  * @param {string} script_path The path to the script file to be evaluated.

```

```

30 * @param {Array<number>} [lines] Optional. Specifies the lines of the
31   script to focus the evaluation on.
32 */
33 evalScript(script_path, lines) {
34   if(!this.win.evaluting) {
35     this.last_script_path = script_path;
36     script_path = script_path.replace(/\//g, '\\\\');
37     var cmd;
38     if(lines !== undefined) {
39       cmd = `run("${script_path}", ${lines.toString()})`;
40     } else {
41       cmd = `run("${script_path}")`;
42     }
43     this.evalCommand(cmd);
44   } else {
45     this.win.command_window.message('@evalScript: Sandbox is busy...');
46   }
47 }
48 /**
49 * Evaluates a given command, optionally displaying the output in the
50   application's command window. Manages the command evaluation process to
51   prevent overlap with ongoing evaluations.
52 * @param {string} cmd The command to be evaluated.
53 * @param {boolean} [show_output=true] Specifies whether the output of the
54   command should be displayed in the command window.
55 * @param {string} [jsl_file_name='jslcmdwindow'] Specifies the file name
56   context for the command evaluation.
57 */
58 evalCommand(cmd, show_output = true, jsl_file_name = 'jslcmdwindow') {
59   if(!this.win.evaluting) {
60     if(cmd.length) {
61       this.win.command_window.addMessageCmd(cmd);
62       this.win.command_history.updateHistory(cmd);
63       this.win.command_window.code_input.setValue('');
64       this.win.command_window.scrollToBottom();
65       this.win.command_window.resetHistoryIndex();
66       this.evalCode(cmd, show_output, jsl_file_name);
67     }
68   } else {
69     this.win.command_window.message('@evalCommand: Sandbox is busy...');
70   }
71 }
72 /**
73 * Directly evaluates code, interacting with the sandbox environment for
74   execution. Ensures that only one evaluation is happening at any time.
75 * @param {string} code The code snippet to be evaluated.
76 * @param {boolean} [show_output=true] Specifies whether the output of the
77   code evaluation should be shown.
78 * @param {string} [jsl_file_name='jslcmdwindow'] The context file name for
79   the code evaluation.
80 */
81 evalCode(code, show_output = true, jsl_file_name = 'jslcmdwindow') {
82   if(!this.win.evaluting) {
83

```

```

77     this.win.evaluating = true;
78     ipcRenderer.send('SandboxWindow', 'eval-code',
79       [code, show_output, jsl_file_name]);
80   } else {
81     this.win.command_window.message('@evalCode: Sandbox is busy...');
82   }
83 }
84
85 /**
86 * Handles actions for the script directory dialog buttons, including
87 changing the active directory, saving the directory to paths, and
88 running the last script.
89 * @param {number} s The button state indicating the action to be performed.
90 */
91 scriptDirDialogButton(s) {
92   this.win.gui.closeDialog($('#script-path-container'));
93   var script_dir = path.dirname(this.last_script_path);
94   if(s == 2) {
95     // Change active directory
96     this.win.folder_navigation.setPath(script_dir);
97   } else if(s == 1) {
98     // Add directory to saved paths
99     this.win.folder_navigation.savePath(script_dir);
100   }
101   ipcRenderer.send('SandboxWindow', 'run-last-script');
102 }
103 }
104
105 exports.PRDC_JSLAB_EVAL = PRDC_JSLAB_EVAL;

```

Listing 60 - eval.js

```

1 /**
2  * @file JSLAB file browser module
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 const fs = require('fs');
9 const path = require('path');
10
11 /**
12  * Class for JSLAB file browser.
13 */
14 class PRDC_JSLAB_FILE_BROWSER {
15
16 /**
17  * Initializes the file browser, setting up the UI component and event
18  listeners for file browser interactions.
19  * @param {object} win The window object representing the current Electron
20  window.
21 */
22 constructor(win) {

```

```

21   var obj = this;
22   this.win = win;
23
24   this.file_browser_cont = document.getElementById('file-browser-cont');
25
26   // File browser refresh button click
27   $('#file-browser-options .refresh').click(function(e){
28     obj.updateFileBrowser();
29   });
30 }
31
32 /**
33 * Displays the contents of a specified folder within the file browser UI,
34 * optionally clearing the current display and replacing it with the new
35 * content.
36 * @param {string} folder_path The path to the folder whose contents should
37 * be displayed.
38 * @param {HTMLElement} [element=this.file_browser_cont] The HTML element
39 * where the folder contents should be displayed. Defaults to the file
40 * browser container.
41 * @param {boolean} [root=false] Indicates whether the folder is the root
42 * folder being displayed. If true, the browser will clear its current
43 * content.
44 */
45 showFolderContent(folder_path, element = this.file_browser_cont, root =
46   false) {
47   var obj = this;
48   var ul = document.createElement('ul');
49   ul.setAttribute('path', folder_path.replace(/\//g, '/'));
50   if(root) {
51     $(element).html('');
52   } else {
53     $(element).find('ul').remove();
54   }
55   fs.readdir(folder_path, {withFileTypes: true}, function(err, dirents) {
56     if(err) {
57       obj.win.command_window.errorInternal(language.string(92) + ': ' + err)
58       ;
59     }
60     dirents.sort((a, b) => {
61       if(a.isDirectory() && !b.isDirectory()) return -1;
62       if(!a.isDirectory() && b.isDirectory()) return 1;
63
64       return a.name.localeCompare(b.name);
65     });
66     dirents.forEach(function(dirent) {
67       var absolute_path = path.join(folder_path, dirent.name);
68       obj.addFileBrowserItem(absolute_path, dirent, ul);
69     });
70     $(ul).appendTo(element).hide().slideDown(300, 'linear');
71   });
72 }

```

```
67
68  /**
69   * Adds an item to the file browser UI, such as a file or folder, including
70   * its name and an icon indicating its type.
71   * @param {string} absolute_path The absolute path to the file or folder to
72   * add.
73   * @param {HTMLElement} ul The unordered list (UL) HTML element to which the
74   * item should be added.
75   */
76 addFileBrowserItem(absolute_path, dirent, ul) {
77   var obj = this;
78   var type_folder = false;
79   var li = document.createElement('li');
80   li.setAttribute('path', absolute_path.replace(/\\/g, '/'));
81   li.innerHTML = '<span>' + path.basename(absolute_path) + '</span>';
82
83   if(dirent.isDirectory()) {
84     li.className = 'folder';
85     type_folder = true;
86   } else if(dirent.isSymbolicLink()) {
87     li.className = 'link';
88     absolute_path = fs.readlinkSync(absolute_path);
89     type_folder = true;
90   } else {
91     li.className = 'file';
92     li.onclick = function(e) {
93       e.stopPropagation();
94       e.preventDefault();
95       ipcRenderer.send('EditorWindow', 'open-script', [absolute_path]);
96     };
97     var ext = absolute_path.split('.').pop();
98     if(ext === 'jsl') {
99       li.classList.add('jsl');
100    } else if(ext === 'js') {
101      li.classList.add('js');
102    } else if(ext === 'json') {
103      li.classList.add('json');
104    }
105  }
106  if(type_folder) {
107    li.onclick = function(e) {
108      e.stopPropagation();
109      e.preventDefault();
110      obj.win.folder_navigation.setPath(absolute_path);
111    };
112    var expand = document.createElement('i');
113    expand.className = 'expand';
114    expand.onclick = function(e) {
115      e.stopPropagation();
116      e.preventDefault();
117      if($(this).hasClass('expended')) {
118        $(this).removeClass('expended');
119        var parent_ul = $(this).parent().find('ul');
120        $(parent_ul).slideUp(300, 'linear', function() {
121          $(parent_ul).remove();
122        });
123      }
124    };
125  }
126}
```

```

119         });
120     } else {
121         $(this).addClass('expanded');
122         obj.showFolderContent(absolute_path, $(this).parent());
123     }
124 };
125 li.appendChild(expand);
126 }
127 ul.appendChild(li);
128 }
129 /**
130 * Refreshes the file browser to reflect the current state of the filesystem
131 * or the contents of the current directory.
132 */
133 updateFileBrowser() {
134     this.win.folder_navigation.setPath(this.win.folder_navigation.current_path
135 );
136 }
137 }
138
139 exports.PRDC_JSLAB_FILE_BROWSER = PRDC_JSLAB_FILE_BROWSER;

```

Listing 61 - file-browser.js

```

1 /**
2  * @file JSLAB folder navigation module
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 const fs = require('fs');
9 const path = require('path');
10 const { pathEqual } = require('path-equal');
11 const Store = require('electron-store');
12
13 const store = new Store();
14
15 /**
16 * Class for JSLAB folder navigation.
17 */
18 class PRDC_JSLAB_FOLDER_NAVIGATION {
19
20 /**
21 * Initializes folder navigation, setting up UI components and event
22 * listeners for folder navigation actions.
23 * @param {object} win The window object representing the current Electron
24 * window.
25 */
26 constructor(win) {
27     var obj = this;
28     this.win = win;
29
30     this.current_path = undefined;
31
32     // Set up UI components and event listeners here...
33 }
34
35 /**
36 * Returns the current path.
37 */
38 get currentPath() {
39     return this.current_path;
40 }
41
42 /**
43 * Sets the current path.
44 */
45 set currentPath(path) {
46     this.current_path = path;
47 }
48
49 /**
50 * Shows the folder content at the specified absolute path.
51 */
52 showFolderContent(absolute_path) {
53     // Implementation...
54 }
55
56 /**
57 * Refreshes the file browser to reflect the current state of the filesystem
58 * or the contents of the current directory.
59 */
60 updateFileBrowser() {
61     this.win.folder_navigation.setPath(this.win.folder_navigation.current_path);
62 }
63
64 /**
65 * Returns the current window object.
66 */
67 get win() {
68     return this._win;
69 }
70
71 /**
72 * Sets the current window object.
73 */
74 set win(win) {
75     this._win = win;
76 }
77
78 /**
79 * Returns the current folder navigation object.
80 */
81 get folder_navigation() {
82     return this._folder_navigation;
83 }
84
85 /**
86 * Sets the current folder navigation object.
87 */
88 set folder_navigation(folder_navigation) {
89     this._folder_navigation = folder_navigation;
90 }
91
92 /**
93 * Returns the current store object.
94 */
95 get store() {
96     return this._store;
97 }
98
99 /**
100 * Sets the current store object.
101 */
102 set store(store) {
103     this._store = store;
104 }
105
106 /**
107 * Returns the current UI component object.
108 */
109 get ui() {
110     return this._ui;
111 }
112
113 /**
114 * Sets the current UI component object.
115 */
116 set ui(ui) {
117     this._ui = ui;
118 }
119
120 /**
121 * Returns the current UI component object.
122 */
123 get ui() {
124     return this._ui;
125 }
126
127 /**
128 * Sets the current UI component object.
129 */
130 set ui(ui) {
131     this._ui = ui;
132 }
133
134 /**
135 * Returns the current UI component object.
136 */
137 get ui() {
138     return this._ui;
139 }
140
141 /**
142 * Sets the current UI component object.
143 */
144 set ui(ui) {
145     this._ui = ui;
146 }
147
148 /**
149 * Returns the current UI component object.
150 */
151 get ui() {
152     return this._ui;
153 }
154
155 /**
156 * Sets the current UI component object.
157 */
158 set ui(ui) {
159     this._ui = ui;
160 }
161
162 /**
163 * Returns the current UI component object.
164 */
165 get ui() {
166     return this._ui;
167 }
168
169 /**
170 * Sets the current UI component object.
171 */
172 set ui(ui) {
173     this._ui = ui;
174 }
175
176 /**
177 * Returns the current UI component object.
178 */
179 get ui() {
180     return this._ui;
181 }
182
183 /**
184 * Sets the current UI component object.
185 */
186 set ui(ui) {
187     this._ui = ui;
188 }
189
190 /**
191 * Returns the current UI component object.
192 */
193 get ui() {
194     return this._ui;
195 }
196
197 /**
198 * Sets the current UI component object.
199 */
200 set ui(ui) {
201     this._ui = ui;
202 }
203
204 /**
205 * Returns the current UI component object.
206 */
207 get ui() {
208     return this._ui;
209 }
210
211 /**
212 * Sets the current UI component object.
213 */
214 set ui(ui) {
215     this._ui = ui;
216 }
217
218 /**
219 * Returns the current UI component object.
220 */
221 get ui() {
222     return this._ui;
223 }
224
225 /**
226 * Sets the current UI component object.
227 */
228 set ui(ui) {
229     this._ui = ui;
230 }
231
232 /**
233 * Returns the current UI component object.
234 */
235 get ui() {
236     return this._ui;
237 }
238
239 /**
240 * Sets the current UI component object.
241 */
242 set ui(ui) {
243     this._ui = ui;
244 }
245
246 /**
247 * Returns the current UI component object.
248 */
249 get ui() {
250     return this._ui;
251 }
252
253 /**
254 * Sets the current UI component object.
255 */
256 set ui(ui) {
257     this._ui = ui;
258 }
259
260 /**
261 * Returns the current UI component object.
262 */
263 get ui() {
264     return this._ui;
265 }
266
267 /**
268 * Sets the current UI component object.
269 */
270 set ui(ui) {
271     this._ui = ui;
272 }
273
274 /**
275 * Returns the current UI component object.
276 */
277 get ui() {
278     return this._ui;
279 }
280
281 /**
282 * Sets the current UI component object.
283 */
284 set ui(ui) {
285     this._ui = ui;
286 }
287
288 /**
289 * Returns the current UI component object.
290 */
291 get ui() {
292     return this._ui;
293 }
294
295 /**
296 * Sets the current UI component object.
297 */
298 set ui(ui) {
299     this._ui = ui;
300 }
301
302 /**
303 * Returns the current UI component object.
304 */
305 get ui() {
306     return this._ui;
307 }
308
309 /**
310 * Sets the current UI component object.
311 */
312 set ui(ui) {
313     this._ui = ui;
314 }
315
316 /**
317 * Returns the current UI component object.
318 */
319 get ui() {
320     return this._ui;
321 }
322
323 /**
324 * Sets the current UI component object.
325 */
326 set ui(ui) {
327     this._ui = ui;
328 }
329
330 /**
331 * Returns the current UI component object.
332 */
333 get ui() {
334     return this._ui;
335 }
336
337 /**
338 * Sets the current UI component object.
339 */
340 set ui(ui) {
341     this._ui = ui;
342 }
343
344 /**
345 * Returns the current UI component object.
346 */
347 get ui() {
348     return this._ui;
349 }
350
351 /**
352 * Sets the current UI component object.
353 */
354 set ui(ui) {
355     this._ui = ui;
356 }
357
358 /**
359 * Returns the current UI component object.
360 */
361 get ui() {
362     return this._ui;
363 }
364
365 /**
366 * Sets the current UI component object.
367 */
368 set ui(ui) {
369     this._ui = ui;
370 }
371
372 /**
373 * Returns the current UI component object.
374 */
375 get ui() {
376     return this._ui;
377 }
378
379 /**
380 * Sets the current UI component object.
381 */
382 set ui(ui) {
383     this._ui = ui;
384 }
385
386 /**
387 * Returns the current UI component object.
388 */
389 get ui() {
390     return this._ui;
391 }
392
393 /**
394 * Sets the current UI component object.
395 */
396 set ui(ui) {
397     this._ui = ui;
398 }
399
400 /**
401 * Returns the current UI component object.
402 */
403 get ui() {
404     return this._ui;
405 }
406
407 /**
408 * Sets the current UI component object.
409 */
410 set ui(ui) {
411     this._ui = ui;
412 }
413
414 /**
415 * Returns the current UI component object.
416 */
417 get ui() {
418     return this._ui;
419 }
420
421 /**
422 * Sets the current UI component object.
423 */
424 set ui(ui) {
425     this._ui = ui;
426 }
427
428 /**
429 * Returns the current UI component object.
430 */
431 get ui() {
432     return this._ui;
433 }
434
435 /**
436 * Sets the current UI component object.
437 */
438 set ui(ui) {
439     this._ui = ui;
440 }
441
442 /**
443 * Returns the current UI component object.
444 */
445 get ui() {
446     return this._ui;
447 }
448
449 /**
450 * Sets the current UI component object.
451 */
452 set ui(ui) {
453     this._ui = ui;
454 }
455
456 /**
457 * Returns the current UI component object.
458 */
459 get ui() {
460     return this._ui;
461 }
462
463 /**
464 * Sets the current UI component object.
465 */
466 set ui(ui) {
467     this._ui = ui;
468 }
469
470 /**
471 * Returns the current UI component object.
472 */
473 get ui() {
474     return this._ui;
475 }
476
477 /**
478 * Sets the current UI component object.
479 */
480 set ui(ui) {
481     this._ui = ui;
482 }
483
484 /**
485 * Returns the current UI component object.
486 */
487 get ui() {
488     return this._ui;
489 }
490
491 /**
492 * Sets the current UI component object.
493 */
494 set ui(ui) {
495     this._ui = ui;
496 }
497
498 /**
499 * Returns the current UI component object.
500 */
501 get ui() {
502     return this._ui;
503 }
504
505 /**
506 * Sets the current UI component object.
507 */
508 set ui(ui) {
509     this._ui = ui;
510 }
511
512 /**
513 * Returns the current UI component object.
514 */
515 get ui() {
516     return this._ui;
517 }
518
519 /**
520 * Sets the current UI component object.
521 */
522 set ui(ui) {
523     this._ui = ui;
524 }
525
526 /**
527 * Returns the current UI component object.
528 */
529 get ui() {
530     return this._ui;
531 }
532
533 /**
534 * Sets the current UI component object.
535 */
536 set ui(ui) {
537     this._ui = ui;
538 }
539
540 /**
541 * Returns the current UI component object.
542 */
543 get ui() {
544     return this._ui;
545 }
546
547 /**
548 * Sets the current UI component object.
549 */
550 set ui(ui) {
551     this._ui = ui;
552 }
553
554 /**
555 * Returns the current UI component object.
556 */
557 get ui() {
558     return this._ui;
559 }
560
561 /**
562 * Sets the current UI component object.
563 */
564 set ui(ui) {
565     this._ui = ui;
566 }
567
568 /**
569 * Returns the current UI component object.
570 */
571 get ui() {
572     return this._ui;
573 }
574
575 /**
576 * Sets the current UI component object.
577 */
578 set ui(ui) {
579     this._ui = ui;
580 }
581
582 /**
583 * Returns the current UI component object.
584 */
585 get ui() {
586     return this._ui;
587 }
588
589 /**
590 * Sets the current UI component object.
591 */
592 set ui(ui) {
593     this._ui = ui;
594 }
595
596 /**
597 * Returns the current UI component object.
598 */
599 get ui() {
600     return this._ui;
601 }
602
603 /**
604 * Sets the current UI component object.
605 */
606 set ui(ui) {
607     this._ui = ui;
608 }
609
610 /**
611 * Returns the current UI component object.
612 */
613 get ui() {
614     return this._ui;
615 }
616
617 /**
618 * Sets the current UI component object.
619 */
620 set ui(ui) {
621     this._ui = ui;
622 }
623
624 /**
625 * Returns the current UI component object.
626 */
627 get ui() {
628     return this._ui;
629 }
630
631 /**
632 * Sets the current UI component object.
633 */
634 set ui(ui) {
635     this._ui = ui;
636 }
637
638 /**
639 * Returns the current UI component object.
640 */
641 get ui() {
642     return this._ui;
643 }
644
645 /**
646 * Sets the current UI component object.
647 */
648 set ui(ui) {
649     this._ui = ui;
650 }
651
652 /**
653 * Returns the current UI component object.
654 */
655 get ui() {
656     return this._ui;
657 }
658
659 /**
660 * Sets the current UI component object.
661 */
662 set ui(ui) {
663     this._ui = ui;
664 }
665
666 /**
667 * Returns the current UI component object.
668 */
669 get ui() {
670     return this._ui;
671 }
672
673 /**
674 * Sets the current UI component object.
675 */
676 set ui(ui) {
677     this._ui = ui;
678 }
679
680 /**
681 * Returns the current UI component object.
682 */
683 get ui() {
684     return this._ui;
685 }
686
687 /**
688 * Sets the current UI component object.
689 */
690 set ui(ui) {
691     this._ui = ui;
692 }
693
694 /**
695 * Returns the current UI component object.
696 */
697 get ui() {
698     return this._ui;
699 }
700
701 /**
702 * Sets the current UI component object.
703 */
704 set ui(ui) {
705     this._ui = ui;
706 }
707
708 /**
709 * Returns the current UI component object.
710 */
711 get ui() {
712     return this._ui;
713 }
714
715 /**
716 * Sets the current UI component object.
717 */
718 set ui(ui) {
719     this._ui = ui;
720 }
721
722 /**
723 * Returns the current UI component object.
724 */
725 get ui() {
726     return this._ui;
727 }
728
729 /**
730 * Sets the current UI component object.
731 */
732 set ui(ui) {
733     this._ui = ui;
734 }
735
736 /**
737 * Returns the current UI component object.
738 */
739 get ui() {
740     return this._ui;
741 }
742
743 /**
744 * Sets the current UI component object.
745 */
746 set ui(ui) {
747     this._ui = ui;
748 }
749
750 /**
751 * Returns the current UI component object.
752 */
753 get ui() {
754     return this._ui;
755 }
756
757 /**
758 * Sets the current UI component object.
759 */
760 set ui(ui) {
761     this._ui = ui;
762 }
763
764 /**
765 * Returns the current UI component object.
766 */
767 get ui() {
768     return this._ui;
769 }
770
771 /**
772 * Sets the current UI component object.
773 */
774 set ui(ui) {
775     this._ui = ui;
776 }
777
778 /**
779 * Returns the current UI component object.
780 */
781 get ui() {
782     return this._ui;
783 }
784
785 /**
786 * Sets the current UI component object.
787 */
788 set ui(ui) {
789     this._ui = ui;
790 }
791
792 /**
793 * Returns the current UI component object.
794 */
795 get ui() {
796     return this._ui;
797 }
798
799 /**
800 * Sets the current UI component object.
801 */
802 set ui(ui) {
803     this._ui = ui;
804 }
805
806 /**
807 * Returns the current UI component object.
808 */
809 get ui() {
810     return this._ui;
811 }
812
813 /**
814 * Sets the current UI component object.
815 */
816 set ui(ui) {
817     this._ui = ui;
818 }
819
820 /**
821 * Returns the current UI component object.
822 */
823 get ui() {
824     return this._ui;
825 }
826
827 /**
828 * Sets the current UI component object.
829 */
830 set ui(ui) {
831     this._ui = ui;
832 }
833
834 /**
835 * Returns the current UI component object.
836 */
837 get ui() {
838     return this._ui;
839 }
840
841 /**
842 * Sets the current UI component object.
843 */
844 set ui(ui) {
845     this._ui = ui;
846 }
847
848 /**
849 * Returns the current UI component object.
850 */
851 get ui() {
852     return this._ui;
853 }
854
855 /**
856 * Sets the current UI component object.
857 */
858 set ui(ui) {
859     this._ui = ui;
860 }
861
862 /**
863 * Returns the current UI component object.
864 */
865 get ui() {
866     return this._ui;
867 }
868
869 /**
870 * Sets the current UI component object.
871 */
872 set ui(ui) {
873     this._ui = ui;
874 }
875
876 /**
877 * Returns the current UI component object.
878 */
879 get ui() {
880     return this._ui;
881 }
882
883 /**
884 * Sets the current UI component object.
885 */
886 set ui(ui) {
887     this._ui = ui;
888 }
889
890 /**
891 * Returns the current UI component object.
892 */
893 get ui() {
894     return this._ui;
895 }
896
897 /**
898 * Sets the current UI component object.
899 */
900 set ui(ui) {
901     this._ui = ui;
902 }
903
904 /**
905 * Returns the current UI component object.
906 */
907 get ui() {
908     return this._ui;
909 }
910
911 /**
912 * Sets the current UI component object.
913 */
914 set ui(ui) {
915     this._ui = ui;
916 }
917
918 /**
919 * Returns the current UI component object.
920 */
921 get ui() {
922     return this._ui;
923 }
924
925 /**
926 * Sets the current UI component object.
927 */
928 set ui(ui) {
929     this._ui = ui;
930 }
931
932 /**
933 * Returns the current UI component object.
934 */
935 get ui() {
936     return this._ui;
937 }
938
939 /**
940 * Sets the current UI component object.
941 */
942 set ui(ui) {
943     this._ui = ui;
944 }
945
946 /**
947 * Returns the current UI component object.
948 */
949 get ui() {
950     return this._ui;
951 }
952
953 /**
954 * Sets the current UI component object.
955 */
956 set ui(ui) {
957     this._ui = ui;
958 }
959
960 /**
961 * Returns the current UI component object.
962 */
963 get ui() {
964     return this._ui;
965 }
966
967 /**
968 * Sets the current UI component object.
969 */
970 set ui(ui) {
971     this._ui = ui;
972 }
973
974 /**
975 * Returns the current UI component object.
976 */
977 get ui() {
978     return this._ui;
979 }
980
981 /**
982 * Sets the current UI component object.
983 */
984 set ui(ui) {
985     this._ui = ui;
986 }
987
988 /**
989 * Returns the current UI component object.
990 */
991 get ui() {
992     return this._ui;
993 }
994
995 /**
996 * Sets the current UI component object.
997 */
998 set ui(ui) {
999     this._ui = ui;
1000 }
1001
1002 /**
1003 * Returns the current UI component object.
1004 */
1005 get ui() {
1006     return this._ui;
1007 }
1008
1009 /**
1010 * Sets the current UI component object.
1011 */
1012 set ui(ui) {
1013     this._ui = ui;
1014 }
1015
1016 /**
1017 * Returns the current UI component object.
1018 */
1019 get ui() {
1020     return this._ui;
1021 }
1022
1023 /**
1024 * Sets the current UI component object.
1025 */
1026 set ui(ui) {
1027     this._ui = ui;
1028 }
1029
1030 /**
1031 * Returns the current UI component object.
1032 */
1033 get ui() {
1034     return this._ui;
1035 }
1036
1037 /**
1038 * Sets the current UI component object.
1039 */
1040 set ui(ui) {
1041     this._ui = ui;
1042 }
1043
1044 /**
1045 * Returns the current UI component object.
1046 */
1047 get ui() {
1048     return this._ui;
1049 }
1050
1051 /**
1052 * Sets the current UI component object.
1053 */
1054 set ui(ui) {
1055     this._ui = ui;
1056 }
1057
1058 /**
1059 * Returns the current UI component object.
1060 */
1061 get ui() {
1062     return this._ui;
1063 }
1064
1065 /**
1066 * Sets the current UI component object.
1067 */
1068 set ui(ui) {
1069     this._ui = ui;
1070 }
1071
1072 /**
1073 * Returns the current UI component object.
1074 */
1075 get ui() {
1076     return this._ui;
1077 }
1078
1079 /**
1080 * Sets the current UI component object.
1081 */
1082 set ui(ui) {
1083     this._ui = ui;
1084 }
1085
1086 /**
1087 * Returns the current UI component object.
1088 */
1089 get ui() {
1090     return this._ui;
1091 }
1092
1093 /**
1094 * Sets the current UI component object.
1095 */
1096 set ui(ui) {
1097     this._ui = ui;
1098 }
1099
1100 /**
1101 * Returns the current UI component object.
1102 */
1103 get ui() {
1104     return this._ui;
1105 }
1106
1107 /**
1108 * Sets the current UI component object.
1109 */
1110 set ui(ui) {
1111     this._ui = ui;
1112 }
1113
1114 /**
1115 * Returns the current UI component object.
1116 */
1117 get ui() {
1118     return this._ui;
1119 }
1120
1121 /**
1122 * Sets the current UI component object.
1123 */
1124 set ui(ui) {
1125     this._ui = ui;
1126 }
1127
1128 /**
1129 * Returns the current UI component object.
1130 */
1131 get ui() {
1132     return this._ui;
1133 }
1134
1135 /**
1136 * Sets the current UI component object.
1137 */
1138 set ui(ui) {
1139     this._ui = ui;
1140 }
1141
1142 /**
1143 * Returns the current UI component object.
1144 */
1145 get ui() {
1146     return this._ui;
1147 }
1148
1149 /**
1150 * Sets the current UI component object.
1151 */
1152 set ui(ui) {
1153     this._ui = ui;
1154 }
1155
1156 /**
1157 * Returns the current UI component object.
1158 */
1159 get ui() {
1160     return this._ui;
1161 }
1162
1163 /**
1164 * Sets the current UI component object.
1165 */
1166 set ui(ui) {
1167     this._ui = ui;
1168 }
1169
1170 /**
1171 * Returns the current UI component object.
1172 */
1173 get ui() {
1174     return this._ui;
1175 }
1176
1177 /**
1178 * Sets the current UI component object.
1179 */
1180 set ui(ui) {
1181     this._ui = ui;
1182 }
1183
1184 /**
1185 * Returns the current UI component object.
1186 */
1187 get ui() {
1188     return this._ui;
1189 }
1190
1191 /**
1192 * Sets the current UI component object.
1193 */
1194 set ui(ui) {
1195     this._ui = ui;
1196 }
1197
1198 /**
1199 * Returns the current UI component object.
1200 */
1201 get ui() {
1202     return this._ui;
1203 }
1204
1205 /**
1206 * Sets the current UI component object.
1207 */
1208 set ui(ui) {
1209     this._ui = ui;
1210 }
1211
1212 /**
1213 * Returns the current UI component object.
1214 */
1215 get ui() {
1216     return this._ui;
1217 }
1218
1219 /**
1220 * Sets the current UI component object.
1221 */
1222 set ui(ui) {
1223     this._ui = ui;
1224 }
1225
1226 /**
1227 * Returns the current UI component object.
1228 */
1229 get ui() {
1230     return this._ui;
1231 }
1232
1233 /**
1234 * Sets the current UI component object.
1235 */
1236 set ui(ui) {
1237     this._ui = ui;
1238 }
1239
1240 /**
1241 * Returns the current UI component object.
1242 */
1243 get ui() {
1244     return this._ui;
1245 }
1246
1247 /**
1248 * Sets the current UI component object.
1249 */
1250 set ui(ui) {
1251     this._ui = ui;
1252 }
1253
1254 /**
1255 * Returns the current UI component object.
1256 */
1257 get ui() {
1258     return this._ui;
1259 }
1260
1261 /**
1262 * Sets the current UI component object.
1263 */
1264 set ui(ui) {
1265     this._ui = ui;
1266 }
1267
1268 /**
1269 * Returns the current UI component object.
1270 */
1271 get ui() {
1272     return this._ui;
1273 }
1274
1275 /**
1276 * Sets the current UI component object.
1277 */
1278 set ui(ui) {
1279     this._ui = ui;
1280 }
1281
1282 /**
1283 * Returns the current UI component object.
1284 */
1285 get ui() {
1286     return this._ui;
1287 }
1288
1289 /**
1290 * Sets the current UI component object.
1291 */
1292 set ui(ui) {
1293     this._ui = ui;
1294 }
1295
1296 /**
1297 * Returns the current UI component object.
1298 */
1299 get ui() {
1300     return this._ui;
1301 }
1302
1303 /**
1304 * Sets the current UI component object.
1305 */
1306 set ui(ui) {
1307     this._ui = ui;
1308 }
1309
1310 /**
1311 * Returns the current UI component object.
1312 */
1313 get ui() {
1314     return this._ui;
1315 }
1316
1317 /**
1318 * Sets the current UI component object.
1319 */
1320 set ui(ui) {
1321     this._ui = ui;
1322 }
1323
1324 /**
1325 * Returns the current UI component object.
1326 */
1327 get ui() {
1328     return this._ui;
1329 }
1330
1331 /**
1332 * Sets the current UI component object.
1333 */
1334 set ui(ui) {
1335     this._ui = ui;
1336 }
1337
1338 /**
1339 * Returns the current UI component object.
1340 */
1341 get ui() {
1342     return this._ui;
1343 }
1344
1345 /**
1346 * Sets the current UI component object.
1347 */
1348 set ui(ui) {
1349     this._ui = ui;
1350 }
1351
1352 /**
1353 * Returns the current UI component object.
1354 */
1355 get ui() {
1356     return this._ui;
1357 }
1358
1359 /**
1360 * Sets the current UI component object.
1361 */
1362 set ui(ui) {
1363     this._ui = ui;
1364 }
1365
1366 /**
1367 * Returns the current UI component object.
1368 */
1369 get ui() {
1370     return this._ui;
1371 }
1372
1373 /**
1374 * Sets the current UI component object.
1375 */
1376 set ui(ui) {
1377     this._ui = ui;
1378 }
1379
1380 /**
1381 * Returns the current UI component object.
1382 */
1383 get ui() {
1384     return this._ui;
1385 }
1386
1387 /**
1388 * Sets the current UI component object.
1389 */
1390 set ui(ui) {
1391     this._ui = ui;
1392 }
1393
1394 /**
1395 * Returns the current UI component object.
1396 */
1397 get ui() {
1398     return this._ui;
1399 }
1400
1401 /**
1402 * Sets the current UI component object.
1403 */
1404 set ui(ui) {
1405     this._ui = ui;
1406 }
1407
1408 /**
1409 * Returns the current UI component object.
1410 */
1411 get ui() {
1412     return this._ui;
1413 }
1414
1415 /**
1416 * Sets the current UI component object.
1417 */
1418 set ui(ui) {
1419     this._ui = ui;
1420 }
1421
1422 /**
1423 * Returns the current UI component object.
1424 */
1425 get ui() {
1426     return this._ui;
1427 }
1428
1429 /**
1430 * Sets the current UI component object.
1431 */
1432 set ui(ui) {
1433     this._ui = ui;
1434 }
1435
1436 /**
1437 * Returns the current UI component object.
1438 */
1439 get ui() {
1440     return this._ui;
1441 }
1442
1443 /**
1444 * Sets the current UI component object.
1445 */
1446 set ui(ui) {
1447     this._ui = ui;
1448 }
1449
1450 /**
1451 * Returns the current UI component object.
1452 */
1453 get ui() {
1454     return this._ui;
1455 }
1456
1457 /**
1458 * Sets the current UI component object.
1459 */
1460 set ui(ui) {
1461     this._ui = ui;
1462 }
1463
1464 /**
1465 * Returns the current UI component object.
1466 */
1467 get ui() {
1468     return this._ui;
1469 }
1470
1471 /**
1472 * Sets the current UI component object.
1473 */
1474 set ui(ui) {
1475     this._ui = ui;
1476 }
1477
1478 /**
1479 * Returns the current UI component object.
1480 */
1481 get ui() {
1482     return this._ui;
1483 }
1484
1485 /**
1486 * Sets the current UI component object.
1487 */
1488 set ui(ui) {
1489     this._ui = ui;
1490 }
1491
1492 /**
1493 * Returns the current UI component object.
1494 */
1495 get ui() {
1496     return this._ui;
1497 }
1498
1499 /**
1500 * Sets the current UI component object.
1501 */
1502 set ui(ui) {
1503     this._ui = ui;
1504 }
1505
1506 /**
1507 * Returns the current UI component object.
1508 */
1509 get ui() {
1510     return this._ui;
1511 }
1512
1513 /**
1514 * Sets the current UI component object.
1515 */
1516 set ui(ui) {
1517     this._ui = ui;
1518 }
1519
1520 /**
1521 * Returns the current UI component object.
1522 */
1523 get ui() {
1524     return this._ui;
1525 }
1526
1527 /**
1528 * Sets the current UI component object.
1529 */
1530 set ui(ui) {
1531     this._ui = ui;
1532 }
1533
1534 /**
1535 * Returns the current UI component object.
1536 */
1537 get ui() {
1538     return this._ui;
1539 }
1540
1541 /**
1542 * Sets the current UI component object.
1543 */
1544 set ui(ui) {
1545     this._ui = ui;
1546 }
1547
1548 /**
1549 * Returns the current UI component object.
1550 */
1551 get ui() {
1552     return this._ui;
1553 }
1554
1555 /**
1556 * Sets the current UI component object.
1557 */
1558 set ui(ui) {
1559     this._ui = ui;
1560 }
1561
1562 /**
1563 * Returns the current UI component object.
1564 */
1565 get ui() {
1566     return this._ui;
1567 }
1568
1569 /**
1570 * Sets the current UI component object.
1571 */
1572 set ui(ui) {
1573     this._ui = ui;
1574 }
1575
1576 /**
1577 * Returns the current UI component object.
1578 */
1579 get ui() {
1580     return this._ui;
1581 }
1582
1583 /**
1584 * Sets the current UI component object.
1585 */
1586 set ui(ui) {
1587     this._ui = ui;
1588 }
1589
1590 /**
1591 * Returns the current UI component object.
1592 */
1593 get ui() {
1594     return this._ui;
1595 }
1596
1597 /**
1598 * Sets the current UI component object.
1599 */
1600 set ui(ui) {
1601     this._ui = ui;
1602 }
1603
1604 /**
1605 * Returns the current UI component object.
1606 */
1607 get ui() {
1608     return this._ui;
1609 }
1610
1611
```

```

29   this.saved_paths = [];
30   this.path_history = [];
31   this.i_path_history = 0;
32
33   this.folder_navigation_cont = document.getElementById('folder-navigation-
34   container');
35
36   // Folder navigation
37   $('#folder-navigation-container .address-line').blur(function() {
38     obj.onPathInput(this);
39   });
39   $('#folder-navigation-container .address-line').on('keydown', function(e)
40   {
41     if(e.key == 'Enter' && !e.shiftKey) {
42       // Enter
43       e.stopPropagation();
44       e.preventDefault();
45       obj.onPathInput(this);
46     }
47   });
47   $('#folder-navigation-container .address-line').focus(function() {
48     this.setSelectionRange(0, this.value.length);
49   });
50   $('#folder-navigation-container .open-folder').click(function() {
51     let options = {
52       title: language.currentString(148),
53       defaultPath: obj.current_path,
54       buttonLabel: language.currentString(149),
55       properties: ['openDirectory']
56     };
57     ipcRenderer.invoke('dialog', 'showOpenDialog', options).then(function(
58       result) {
59       if(!result.canceled) {
60         obj.setPath(result.filePaths[0]);
61     }
61     }).catch(function(err) {
62       obj.win.command_window.errorInternal(err);
63     });
64   });
65   $('#folder-navigation-container .up-folder').click(function() {
66     var folders = obj.current_path.split(path.sep);
67     folders = folders.filter(function(el) { return el != ''; });
68     folders.pop();
69     if(folders.length == 1) {
70       obj.setPath(folders);
71     } else {
72       obj.setPath(path.join(...folders));
73     }
74   });
75   $('#folder-navigation-container .previous-folder').click(function() {
76     obj.setPath(obj.current_path, obj.i_path_history+1);
77   });
78   $('#folder-navigation-container .next-folder').click(function() {
79     obj.setPath(obj.current_path, obj.i_path_history-1);
80   });

```

```

81
82 // Paths logic
83 $('#paths-menu').click(function() {
84   obj.updatePathsList();
85   obj.win.gui.openPathsMenu();
86 });
87 $('#paths-close').click(function() {
88   obj.win.gui.closePathsMenu();
89 });
90 $('#paths-container').on('keydown', function(e) {
91   if(e.key === 'Escape') {
92     // ESC
93     obj.win.gui.closePathsMenu();
94     e.stopPropagation();
95     e.preventDefault();
96   }
97 });
98 $('#save-path').click(function() {
99   obj.toggleSavePath();
100 });

101 // Script path logic
102 $('#script-path-close').click(function() {
103   obj.win.gui.closeDialog($('#script-path-container'));
104 });
105 $('#script-path-container').on('keydown', function(e) {
106   if(e.key === 'Escape') {
107     // ESC
108     obj.win.gui.closeDialog($('#script-path-container'));
109     e.stopPropagation();
110     e.preventDefault();
111   }
112 });
113 });

114 // Saved paths
115 this.saved_paths = store.get('saved_paths');
116 if(!this.saved_paths) {
117   this.saved_paths = [];
118 }
119 ipcRenderer.send('SandboxWindow', 'set-saved-paths', this.saved_paths);
120

121 // Set path
122 var current_path = store.get('current_path');
123 if(!current_path || !(fs.existsSync(current_path) &&
124   fs.lstatSync(current_path, {throwIfNoEntry: false}).isDirectory())) {
125   current_path = this.win.app.documents_path;
126 }
127 this.setPath(current_path);
128 }

129

130 /**
131 * Processes the input from the address line, navigating to the specified
132 * path if it is different from the current path.
133 * @param {HTMLElement} e The HTML input element containing the path.
134 */

```

```

135  onPathInput(e) {
136    var new_path = $(e).val();
137    new_path = this.addPathSep(new_path);
138    if(!pathEqual(new_path, this.current_path)) {
139      $(e).val(new_path);
140      this.setPath(new_path);
141    }
142  }
143
144 /**
145 * Sets the current path for navigation, updating the UI and internal state
146 * accordingly. Supports navigation through history via index.
147 * @param {string} new_path The new path to set as the current directory.
148 * @param {number} [i=undefined] Optional index for navigation through the
149 * path history.
150 * @param {boolean} [inform_sandbox=true] Whether to inform the sandbox
151 * process of the path change.
152 */
153 setPath(new_path, i = undefined, inform_sandbox = true) {
154   new_path = this.addPathSep(new_path);
155   if(this.checkDirectory(new_path)) {
156     if(!pathEqual(new_path, this.current_path) && i === undefined) {
157       this.path_history.unshift(new_path);
158       if(this.path_history.length > 1) {
159         $('#folder-navigation-container .previous-folder').removeClass(
160           'disabled');
161         $('#folder-navigation-container .next-folder').addClass('disabled');
162         this.i_path_history = 0;
163       }
164     } else if(i !== undefined && i >= 0 &&
165       i < this.path_history.length) {
166       this.i_path_history = i;
167       new_path = this.path_history[i];
168       if(i == 0) {
169         $('#folder-navigation-container .next-folder').addClass('disabled');
170       } else {
171         $('#folder-navigation-container .next-folder').removeClass('disabled');
172       }
173     }
174   }
175
176   this.current_path = new_path;
177   if(inform_sandbox) {
178     ipcRenderer.send('SandboxWindow', 'set-current-path', new_path);
179   }
180   if(this.saved_paths.indexOf(this.current_path) >= 0) {
181     $('#save-path').addClass('saved');
182   } else {

```

```

183         $( '#save-path' ).removeClass( 'saved' );
184     }
185     this.win.file_browser.showFolderContent( this.current_path, undefined,
186                                         true );
187 } else {
188   this.setPath( this.win.app.documents_path );
189 }
190 this.showCurrentPath();
191 }

192 /**
193 * Saves the current path to the list of saved paths for quick access.
194 * @param {string} new_path The path to save.
195 * @param {boolean} [inform_sandbox=true] Whether to inform the sandbox
196   process of the update.
197 */
198 savePath( new_path, inform_sandbox = true ) {
199   new_path = this.addPathSep( new_path );
200   var i = this.saved_paths.indexOf( new_path );
201   if( i < 0 ) {
202     this.saved_paths.push( new_path );
203   }
204   if( inform_sandbox ) {
205     ipcRenderer.send( 'SandboxWindow', 'set-saved-paths', this.saved_paths );
206   }
207 }

208 /**
209 * Removes a path from the list of saved paths.
210 * @param {string} saved_path The path to remove.
211 * @param {boolean} [inform_sandbox=true] Whether to inform the sandbox
212   process of the update.
213 */
214 removePath( saved_path, inform_sandbox = true ) {
215   saved_path = this.addPathSep( saved_path );
216   var i = this.saved_paths.indexOf( saved_path );
217   if( i >= 0 ) {
218     this.saved_paths.splice( i, 1 );
219   }
220   if( inform_sandbox ) {
221     ipcRenderer.send( 'SandboxWindow', 'set-saved-paths', this.saved_paths );
222   }
223 }

224 /**
225 * Toggles the current path between being saved and not saved, updating the
226   UI and stored paths accordingly.
227 */
228 toggleSavePath() {
229   var i = this.saved_paths.indexOf( this.current_path );
230   if( i >= 0 ) {
231     this.removePath( this.current_path );
232     $( '#save-path' ).removeClass( 'saved' );
233   } else {
234     this.savePath( this.current_path );

```

```

234     $( '#save-path' ).addClass( 'saved' );
235   }
236   this.updatePathsList();
237 }
238
239 /**
240 * Updates the UI to display the current path in the address line and
241 * navigation breadcrumbs.
242 */
243 showCurrentPath() {
244   var obj = this;
245   $('#folder-navigation-container .address-line').val(this.current_path);
246   var folders = this.current_path.split(path.sep);
247   folders = folders.filter(function(el) { return el != ''; });
248   var address = $('#folder-navigation-container .current-address')[0];
249   address.innerHTML = '';
250   var full_path = folders[0];
251   if(folders.length == 1) {
252     $('#folder-navigation-container .up-folder').addClass('disabled');
253   } else {
254     $('#folder-navigation-container .up-folder').removeClass('disabled');
255   }
256   for(var i = 0; i < folders.length; i++) {
257     if(i > 0) {
258       full_path += path.sep;
259       full_path += folders[i];
260     }
261     if(folders[i] != '') {
262       var span = document.createElement('span');
263       span.className = 'folder';
264       span.title = full_path;
265       if(i == 0) {
266         span.title += path.sep;
267       }
268       span.textContent = folders[i];
269       span.onclick = function() {
270         obj.setPath(this.title);
271       };
272       address.appendChild(span);
273       if(i != (folders.length-1)) {
274         $(address).append('<i class="i-next-folder"></i>');
275       }
276     }
277   }
278
279 /**
280 * Updates the list of saved paths in the UI, allowing users to quickly
281 * navigate to frequently accessed directories.
282 */
283 updatePathsList() {
284   var obj = this;
285   var cont = $('#paths-container .page-panel ul');
286   if(this.saved_paths.length > 0) {
287     $(cont).html('');

```

```

287   this.saved_paths.forEach(function(saved_path) {
288     var row = document.createElement('li');
289     row.textContent = saved_path;
290     row.onclick = function() {
291       obj.win.gui.closePathsMenu();
292       obj.setPath(saved_path);
293     };
294     if (!obj.checkDirectory(saved_path)) {
295       row.className = 'inactive';
296     }
297     var btn = document.createElement('img');
298     btn.src = '../img/close.svg';
299     btn.className = 'remove-path';
300     btn.onclick = function(event) {
301       event.preventDefault();
302       event.stopPropagation();
303       if (pathEqual(saved_path, obj.current_path)) {
304         $('#save-path').removeClass('saved');
305       }
306       obj.removePath(saved_path);
307       $(this).parent().remove();
308       if (obj.saved_paths.length == 0) {
309         $(cont).html('<li class="no-paths">' + language.string(140) + '</li>');
310         ;
311       }
312       row.appendChild(btn);
313       $(cont).append(row);
314     });
315   } else {
316     $(cont).html('<li class="no-paths">' + language.string(140) + '</li>');
317   }
318 }
319
320 /**
321 * Handles UI and state updates when a script directory is unknown,
322 * prompting the user for action.
323 */
324 unknownScriptDir() {
325   var script_dir = this.addPathSep(path.dirname(this.win.eval(
326     last_script_path)));
327   $('#script-path').text(script_dir);
328   this.win.gui.openDialog($('#script-path-container'));
329   return true;
330 }
331
332 /**
333 * Appends a path separator to the end of a path string if it is not already
334 * present.
335 * @param {string} path_str The path string to modify.
336 * @return {string} The modified path string with a trailing separator.
337 */
338 addPathSep(path_str) {
339   if (path_str && path_str[path_str.length - 1] != path.sep) {
340     path_str += path.sep;

```

```

338     }
339     return path_str;
340   }
341
342 /**
343 * Checks if the specified directory exists and is a directory.
344 * @param {string} directory The path to check.
345 * @return {boolean} True if the directory exists and is a directory, false
346   otherwise.
347 */
348 checkDirectory(directory) {
349   var lstat = fs.lstatSync(directory, {throwIfNoEntry: false});
350   if(lstat != undefined && lstat.isDirectory()) {
351     return true;
352   } else {
353     return false;
354   }
355
356 /**
357 * Checks if the specified file exists and is a file.
358 * @param {string} file_path The path to the file to check.
359 * @return {boolean} True if the file exists and is a file, false otherwise.
360 */
361 checkFile(file_path) {
362   var lstat = fs.lstatSync(file_path, {throwIfNoEntry: false});
363   if(lstat != undefined && lstat.isFile()) {
364     return true;
365   } else {
366     return false;
367   }
368 }
369 }
370
371 exports.PRDC_JSLAB_FOLDER_NAVIGATION = PRDC_JSLAB_FOLDER_NAVIGATION;

```

Listing 62 - folder-navigation.js

```

1 /**
2  * @file JSLAB GUI module
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 /**
9  * Class for JSLAB GUI.
10 */
11 class PRDC_JSLAB_GUI {
12
13 /**
14  * Initializes the GUI, setting up event listeners for window controls, menu
15    actions, and dialog interactions.
16  * @param {object} win The window object representing the current Electron
17    window.
18 */

```



```

68     if (obj.maximized) {
69         ipcRenderer.send('MainProcess', 'maximize-win');
70     } else {
71         ipcRenderer.send('MainProcess', 'restore-win');
72     });
73 });
74
75 $("#win-minimize").click(function() {
76     obj.toggleFullscreen(false);
77     ipcRenderer.send('MainProcess', 'minimize-win');
78 });
79 window.dispatchEvent(new Event('resize'));
80 }
81
82 /**
83 * Invoked when the GUI is ready, performing initial UI setup tasks such as
84 * fading in the window.
85 */
86 onReady() {
87     // Fade in window
88     ipcRenderer.send('MainProcess', 'fade-in-win');
89 }
90 /**
91 * Toggles the fullscreen state of the application window.
92 * @param {boolean} [fullscreen] Optional. Specifies the fullscreen state.
93 * If not provided, the state is toggled based on the current state.
94 */
95 toggleFullscreen(fullscreen) {
96     if(fullscreen == null) {
97         fullscreen = !this.fullscreen;
98     }
99     if(fullscreen) {
100         ipcRenderer.send('MainProcess', 'set-fullscreen', true);
101     } else {
102         ipcRenderer.send('MainProcess', 'set-fullscreen', false);
103     }
104     this.fullscreen = fullscreen;
105 }
106 /**
107 * Updates the status displayed in the status bar of the application.
108 * @param {string} state The current state to display.
109 * @param {string} txt The text to display in the status bar.
110 */
111 setStatus(state, txt) {
112     this.state = state;
113     $(this.status_cont).html(txt);
114     this.setStatsIcon();
115 }
116
117 /**
118 * Resets the stats data to initial values.
119 */
120 resetStats() {

```

```

121     this.setStatus('ready', language.string(87));
122     var stats = {
123         'required_modules': 0,
124         'promises': 0,
125         'timeouts': 0,
126         'immediates': 0,
127         'intervals': 0,
128         'animation_frames': 0,
129         'idle_callbacks': 0
130     };
131     this.setStats(stats);
132 }
133
134 /**
135 * Updates the statistics displayed in the GUI, such as the number of active
136     promises, timeouts, and intervals.
137 * @param {object} stats An object containing statistical information to
138     display.
139 */
140 setStats(stats) {
141     this.stats = stats;
142     $('#sandbox-required-modules-num').text(stats['required_modules']);
143     $('#sandbox-promises-num').text(stats['promises']);
144     $('#sandbox-timeouts-num').text(stats['timeouts']);
145     $('#sandbox-immediates-num').text(stats['immediates']);
146     $('#sandbox-intervals-num').text(stats['intervals']);
147     $('#sandbox-animation-frames-num').text(stats['animation_frames']);
148     $('#sandbox-idle-callbacks-num').text(stats['idle_callbacks']);
149     this.stats_num = stats['promises']+stats['timeouts']+stats['immediates']+stats['intervals']+stats['animation_frames']+stats['idle_callbacks'];
150     this.setStatsIcon();
151 }
152 /**
153 * Updates the visibility and appearance of the statistics icon based on the
154     current application state and stats.
155 */
156 setStatsIcon() {
157     this.sandbox_stats_icon.className = '';
158     if(this.state == 'ready') {
159         if(this.stats_num > 0) {
160             this.sandbox_stats_icon.classList.add('async-busy');
161         } else {
162             this.sandbox_stats_icon.classList.add('ready');
163         }
164     } else {
165         this.sandbox_stats_icon.classList.add('busy');
166     }
167 }
168 /**
169 * Opens a specified dialog within the application.
170 * @param {jQuery} e The jQuery object representing the dialog to open.
171 */
172 openDialog(e) {

```

```
173     if (!e.is(':visible')) {
174         this.last_focus = document.activeElement;
175         $('#main-dialog').fadeOut(300, 'linear');
176         e.fadeIn(300, 'linear', function() {
177             e.focus();
178         });
179     }
180 }
181 /**
182 * Closes a specified dialog within the application.
183 * @param {jQuery} e The jQuery object representing the dialog to close.
184 */
185 closeDialog(e) {
186     e.fadeOut(300, 'linear');
187     $(this.last_focus).focus();
188 }
189 /**
190 * Opens the paths menu dialog, providing quick access to saved and
191 * frequently used paths.
192 */
193 openPathsMenu() {
194     this.openDialog($('#paths-container'));
195 }
196
197 /**
198 * Closes the paths menu dialog.
199 */
200 closePathsMenu() {
201     this.closeDialog($('#paths-container'));
202 }
203
204 /**
205 * Opens the help dialog, providing access to application documentation or
206 * assistance.
207 */
208 help() {
209     this.openDialog($('#help-container'));
210 }
211
212 /**
213 * Closes the help dialog.
214 */
215 closeHelp() {
216     this.closeDialog($('#help-container'));
217 }
218
219 /**
220 * Opens the application info dialog, displaying information about the
221 * application.
222 */
223 info() {
224     this.openDialog($('#info-container'));
225 }
```

```

225
226  /**
227   * Closes the application info dialog.
228   */
229 closeInfo() {
230   this.closeDialog($('#info-container'));
231 }
232
233 /**
234  * Opens the settings dialog, allowing the user to configure application
235   settings.
236  */
237 settings() {
238   this.openDialog($('#settings-container'));
239 }
240
241 /**
242  * Closes the settings dialog.
243  */
244 closeSettings() {
245   this.closeDialog($('#settings-container'));
246 }
247
248 /**
249  * Changes the application language to the specified language ID.
250  * @param {number} id The ID of the language to switch to.
251  */
252 changeLangauge(id) {
253   language.set(id);
254   ipcRenderer.send('EditorWindow', 'set-language', id);
255   ipcRenderer.send('SandboxWindow', 'set-language', id);
256 }
257
258 exports.PRDC_JSLAB_GUI = PRDC_JSLAB_GUI;

```

Listing 63 - gui.js

```

1 /**
2  * @file JSLAB help module
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 /**
9  * Class for JSLAB help.
10 */
11 class PRDC_JSLAB_HELP {
12
13 /**
14  * Initializes the help functionality, setting up event listeners for help
15   menu actions and dialog interactions.
16  * @param {object} win The window object representing the current Electron
17   window.
18  */

```



```
17 constructor(win) {
18     var obj = this;
19     this.win = win;
20
21     // Help logic
22     $('#help-menu').click(function() { obj.win.gui.help(); });
23
24     $('#help-close').click(function() { obj.win.gui.closeHelp(); });
25
26     $('#help-container').on('keydown', function(e) {
27         if(e.key === 'Escape') {
28             // ESC
29             obj.win.gui.closeHelp();
30             e.stopPropagation();
31             e.preventDefault();
32         }
33     });
34 }
35
36 }
37
38 exports.PRDC_JSLAB_HELP = PRDC_JSLAB_HELP
```

Listing 64 - help.js

```
1 /**
2  * @file JSLAB info module
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 /**
9  * Class for JSLAB info .
10 */
11 class PRDC_JSLAB_INFO {
12
13 /**
14  * Initializes the information display functionality , setting up event
15  * listeners for information menu actions and dialog interactions .
16  * @param {object} win The window object representing the current Electron
17  * window .
18 */
19 constructor(win) {
20     var obj = this;
21     this.win = win;
22
23     // Info logic
24     $('#info-container .app-version').text('version ' + this.win.app.version);
25     $('#info-menu').click(function() { obj.win.gui.info(); });
26     $('#info-close').click(function() { obj.win.gui.closeInfo(); });
27     $('#info-container').on('keydown', function(e) {
28         if(e.key === 'Escape') {
29             // ESC
30             obj.win.gui.closeInfo();
31             e.stopPropagation();
32         }
33     });
34 }
```



```
30         e.preventDefault();
31     }
32   });
33 }
34
35 }
36
37 exports.PRDC_JSLAB_INFO = PRDC_JSLAB_INFO
```

Listing 65 - info.js

```
1 /**
2  * @file JSLAB main window init file
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 // Modules
9 // -----
10 const { ipcRenderer } = require('electron');
11
12
13 const helper = require('../js/helper.js');
14 const { PRDC_APP_CONFIG } = require('../config/config');
15 const { PRDC_APP_LOGGER } = require('../lib/PRDC_APP_LOGGER/PRDC_APP_LOGGER');
16 const { PRDC_JSLAB_LANGUAGE } = require('../js/language');
17
18 global.app_path = process.argv.find(e => e.startsWith('--app-path=')).split('=')
19   [1].replace(/\.\.js\?\$/,'');
20
21 const { PRDC_JSLAB_WIN_MAIN } = require('../js/main/win-main');
22
23 // Start log
24 const log_file = ipcRenderer.sendSync('sync-message', 'get-log-file');
25 const app_logger = new PRDC_APP_LOGGER(log_file);
26
27 // Global variables
28 const config = new PRDC_APP_CONFIG();
29 var language = new PRDC_JSLAB_LANGUAGE();
30 var win_main = new PRDC_JSLAB_WIN_MAIN();
31
32 // When document is ready
33 // -----
34 ready(function() {
35   // Jquery ready
36   $(document).ready(function() {
37     win_main.onReady();
38   });
39});
```

Listing 66 - init-main.js

```
1 /**
2  * @file JSLAB panels module
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
```

```

4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 const { PRDC_PANEL } = require('../lib/PRDC_PANEL/PRDC_PANEL');
9 const Store = require('electron-store');
10
11 const store = new Store();
12
13 /**
14  * Class for JSLAB panels.
15 */
16 class PRDC_JSLAB_PANELS {
17
18 /**
19  * Initializes main application panels and configures their default sizes
20  * and orientations.
21  * @param {object} win The window object representing the current Electron
22  * window.
23 */
24 constructor(win) {
25   var obj = this;
26   this.win = win;
27
28   this.columns = new PRDC_PANEL('columns', 'vertical', document.
29     getElementById('panels-container'), [document.getElementById('left-
30     panel'), document.getElementById('right-panel')], config.
31     PANEL_DEFAULT_COLUMNS);
32
33   this.left_rows = new PRDC_PANEL('left-rows', 'horizontal', document.
34     getElementById('left-panel'), [document.getElementById('left-top-panel').
35     getElementById('left-middle-panel'), document.
36     getElementById('left-bottom-panel')], config.PANEL_DEFAULT_LEFT_ROWS);
37
38   this.workspace_columns = new PRDC_PANEL('workspace-columns', 'vertical',
39     document.getElementById('workspace'), ['#left-middle-panel.col-1',
40     '#left-middle-panel.col-2', '#left-middle-panel.col-3'], config.
41     PANEL_DEFAULT_WORKSPACE_COLUMNS);
42
43   this.columns.addSubPanel(this.left_rows);
44   this.left_rows.addSubPanel(this.workspace_columns);
45
46   /**
47    * Invoked when the application window is ready, triggering an initial
48    * resize event to ensure panels are correctly laid out.
49   */
50   onReady() {
51     this.columns.onResize();
52   }
53
54   /**
55    * Invoked when the application window is resized, triggering an update
56    * event to ensure panels are correctly laid out.
57   */
58   onResize() {
59     this.columns.onResize();
60   }
61
62   /**
63    * Invoked when the application window is closed, triggering a close
64    * event to ensure panels are correctly laid out.
65   */
66   onClose() {
67     this.columns.onClose();
68   }
69
70   /**
71    * Invoked when the application window is moved, triggering a move
72    * event to ensure panels are correctly laid out.
73   */
74   onMove() {
75     this.columns.onMove();
76   }
77
78   /**
79    * Invoked when the application window is maximized, triggering a maximize
80    * event to ensure panels are correctly laid out.
81   */
82   onMaximize() {
83     this.columns.onMaximize();
84   }
85
86   /**
87    * Invoked when the application window is restored from a minimized state,
88    * triggering a restore event to ensure panels are correctly laid out.
89   */
90   onRestore() {
91     this.columns.onRestore();
92   }
93
94   /**
95    * Invoked when the application window is activated, triggering an activate
96    * event to ensure panels are correctly laid out.
97   */
98   onActivate() {
99     this.columns.onActivate();
100   }
101
102   /**
103    * Invoked when the application window is deactivated, triggering a deactivate
104    * event to ensure panels are correctly laid out.
105   */
106   onDeactivate() {
107     this.columns.onDeactivate();
108   }
109
110   /**
111    * Invoked when the application window is focused, triggering a focus
112    * event to ensure panels are correctly laid out.
113   */
114   onFocus() {
115     this.columns.onFocus();
116   }
117
118   /**
119    * Invoked when the application window is unfocused, triggering an unfocus
120    * event to ensure panels are correctly laid out.
121   */
122   onUnfocus() {
123     this.columns.onUnfocus();
124   }
125
126   /**
127    * Invoked when the application window is blurred, triggering a blur
128    * event to ensure panels are correctly laid out.
129   */
130   onBlur() {
131     this.columns.onBlur();
132   }
133
134   /**
135    * Invoked when the application window is focused, triggering a focus
136    * event to ensure panels are correctly laid out.
137   */
138   onFocus() {
139     this.columns.onFocus();
140   }
141
142   /**
143    * Invoked when the application window is unfocused, triggering an unfocus
144    * event to ensure panels are correctly laid out.
145   */
146   onUnfocus() {
147     this.columns.onUnfocus();
148   }
149
150   /**
151    * Invoked when the application window is blurred, triggering a blur
152    * event to ensure panels are correctly laid out.
153   */
154   onBlur() {
155     this.columns.onBlur();
156   }
157
158   /**
159    * Invoked when the application window is destroyed, triggering a destroy
160    * event to ensure panels are correctly laid out.
161   */
162   onDestroy() {
163     this.columns.onDestroy();
164   }
165
166   /**
167    * Invoked when the application window is closed, triggering a close
168    * event to ensure panels are correctly laid out.
169   */
170   onClose() {
171     this.columns.onClose();
172   }
173
174   /**
175    * Invoked when the application window is maximized, triggering a maximize
176    * event to ensure panels are correctly laid out.
177   */
178   onMaximize() {
179     this.columns.onMaximize();
180   }
181
182   /**
183    * Invoked when the application window is restored from a minimized state,
184    * triggering a restore event to ensure panels are correctly laid out.
185   */
186   onRestore() {
187     this.columns.onRestore();
188   }
189
190   /**
191    * Invoked when the application window is activated, triggering an activate
192    * event to ensure panels are correctly laid out.
193   */
194   onActivate() {
195     this.columns.onActivate();
196   }
197
198   /**
199    * Invoked when the application window is deactivated, triggering a deactivate
200    * event to ensure panels are correctly laid out.
201   */
202   onDeactivate() {
203     this.columns.onDeactivate();
204   }
205
206   /**
207    * Invoked when the application window is focused, triggering a focus
208    * event to ensure panels are correctly laid out.
209   */
210   onFocus() {
211     this.columns.onFocus();
212   }
213
214   /**
215    * Invoked when the application window is unfocused, triggering an unfocus
216    * event to ensure panels are correctly laid out.
217   */
218   onUnfocus() {
219     this.columns.onUnfocus();
220   }
221
222   /**
223    * Invoked when the application window is blurred, triggering a blur
224    * event to ensure panels are correctly laid out.
225   */
226   onBlur() {
227     this.columns.onBlur();
228   }
229
230   /**
231    * Invoked when the application window is destroyed, triggering a destroy
232    * event to ensure panels are correctly laid out.
233   */
234   onDestroy() {
235     this.columns.onDestroy();
236   }
237
238   /**
239    * Invoked when the application window is closed, triggering a close
240    * event to ensure panels are correctly laid out.
241   */
242   onClose() {
243     this.columns.onClose();
244   }
245
246   /**
247    * Invoked when the application window is maximized, triggering a maximize
248    * event to ensure panels are correctly laid out.
249   */
250   onMaximize() {
251     this.columns.onMaximize();
252   }
253
254   /**
255    * Invoked when the application window is restored from a minimized state,
256    * triggering a restore event to ensure panels are correctly laid out.
257   */
258   onRestore() {
259     this.columns.onRestore();
260   }
261
262   /**
263    * Invoked when the application window is activated, triggering an activate
264    * event to ensure panels are correctly laid out.
265   */
266   onActivate() {
267     this.columns.onActivate();
268   }
269
270   /**
271    * Invoked when the application window is deactivated, triggering a deactivate
272    * event to ensure panels are correctly laid out.
273   */
274   onDeactivate() {
275     this.columns.onDeactivate();
276   }
277
278   /**
279    * Invoked when the application window is focused, triggering a focus
280    * event to ensure panels are correctly laid out.
281   */
282   onFocus() {
283     this.columns.onFocus();
284   }
285
286   /**
287    * Invoked when the application window is unfocused, triggering an unfocus
288    * event to ensure panels are correctly laid out.
289   */
290   onUnfocus() {
291     this.columns.onUnfocus();
292   }
293
294   /**
295    * Invoked when the application window is blurred, triggering a blur
296    * event to ensure panels are correctly laid out.
297   */
298   onBlur() {
299     this.columns.onBlur();
300   }
301
302   /**
303    * Invoked when the application window is destroyed, triggering a destroy
304    * event to ensure panels are correctly laid out.
305   */
306   onDestroy() {
307     this.columns.onDestroy();
308   }
309
310   /**
311    * Invoked when the application window is closed, triggering a close
312    * event to ensure panels are correctly laid out.
313   */
314   onClose() {
315     this.columns.onClose();
316   }
317
318   /**
319    * Invoked when the application window is maximized, triggering a maximize
320    * event to ensure panels are correctly laid out.
321   */
322   onMaximize() {
323     this.columns.onMaximize();
324   }
325
326   /**
327    * Invoked when the application window is restored from a minimized state,
328    * triggering a restore event to ensure panels are correctly laid out.
329   */
330   onRestore() {
331     this.columns.onRestore();
332   }
333
334   /**
335    * Invoked when the application window is activated, triggering an activate
336    * event to ensure panels are correctly laid out.
337   */
338   onActivate() {
339     this.columns.onActivate();
340   }
341
342   /**
343    * Invoked when the application window is deactivated, triggering a deactivate
344    * event to ensure panels are correctly laid out.
345   */
346   onDeactivate() {
347     this.columns.onDeactivate();
348   }
349
350   /**
351    * Invoked when the application window is focused, triggering a focus
352    * event to ensure panels are correctly laid out.
353   */
354   onFocus() {
355     this.columns.onFocus();
356   }
357
358   /**
359    * Invoked when the application window is unfocused, triggering an unfocus
360    * event to ensure panels are correctly laid out.
361   */
362   onUnfocus() {
363     this.columns.onUnfocus();
364   }
365
366   /**
367    * Invoked when the application window is blurred, triggering a blur
368    * event to ensure panels are correctly laid out.
369   */
370   onBlur() {
371     this.columns.onBlur();
372   }
373
374   /**
375    * Invoked when the application window is destroyed, triggering a destroy
376    * event to ensure panels are correctly laid out.
377   */
378   onDestroy() {
379     this.columns.onDestroy();
380   }
381
382   /**
383    * Invoked when the application window is closed, triggering a close
384    * event to ensure panels are correctly laid out.
385   */
386   onClose() {
387     this.columns.onClose();
388   }
389
390   /**
391    * Invoked when the application window is maximized, triggering a maximize
392    * event to ensure panels are correctly laid out.
393   */
394   onMaximize() {
395     this.columns.onMaximize();
396   }
397
398   /**
399    * Invoked when the application window is restored from a minimized state,
400    * triggering a restore event to ensure panels are correctly laid out.
401   */
402   onRestore() {
403     this.columns.onRestore();
404   }
405
406   /**
407    * Invoked when the application window is activated, triggering an activate
408    * event to ensure panels are correctly laid out.
409   */
410   onActivate() {
411     this.columns.onActivate();
412   }
413
414   /**
415    * Invoked when the application window is deactivated, triggering a deactivate
416    * event to ensure panels are correctly laid out.
417   */
418   onDeactivate() {
419     this.columns.onDeactivate();
420   }
421
422   /**
423    * Invoked when the application window is focused, triggering a focus
424    * event to ensure panels are correctly laid out.
425   */
426   onFocus() {
427     this.columns.onFocus();
428   }
429
430   /**
431    * Invoked when the application window is unfocused, triggering an unfocus
432    * event to ensure panels are correctly laid out.
433   */
434   onUnfocus() {
435     this.columns.onUnfocus();
436   }
437
438   /**
439    * Invoked when the application window is blurred, triggering a blur
440    * event to ensure panels are correctly laid out.
441   */
442   onBlur() {
443     this.columns.onBlur();
444   }
445
446   /**
447    * Invoked when the application window is destroyed, triggering a destroy
448    * event to ensure panels are correctly laid out.
449   */
450   onDestroy() {
451     this.columns.onDestroy();
452   }
453
454   /**
455    * Invoked when the application window is closed, triggering a close
456    * event to ensure panels are correctly laid out.
457   */
458   onClose() {
459     this.columns.onClose();
460   }
461
462   /**
463    * Invoked when the application window is maximized, triggering a maximize
464    * event to ensure panels are correctly laid out.
465   */
466   onMaximize() {
467     this.columns.onMaximize();
468   }
469
470   /**
471    * Invoked when the application window is restored from a minimized state,
472    * triggering a restore event to ensure panels are correctly laid out.
473   */
474   onRestore() {
475     this.columns.onRestore();
476   }
477
478   /**
479    * Invoked when the application window is activated, triggering an activate
480    * event to ensure panels are correctly laid out.
481   */
482   onActivate() {
483     this.columns.onActivate();
484   }
485
486   /**
487    * Invoked when the application window is deactivated, triggering a deactivate
488    * event to ensure panels are correctly laid out.
489   */
490   onDeactivate() {
491     this.columns.onDeactivate();
492   }
493
494   /**
495    * Invoked when the application window is focused, triggering a focus
496    * event to ensure panels are correctly laid out.
497   */
498   onFocus() {
499     this.columns.onFocus();
500   }
501
502   /**
503    * Invoked when the application window is unfocused, triggering an unfocus
504    * event to ensure panels are correctly laid out.
505   */
506   onUnfocus() {
507     this.columns.onUnfocus();
508   }
509
510   /**
511    * Invoked when the application window is blurred, triggering a blur
512    * event to ensure panels are correctly laid out.
513   */
514   onBlur() {
515     this.columns.onBlur();
516   }
517
518   /**
519    * Invoked when the application window is destroyed, triggering a destroy
520    * event to ensure panels are correctly laid out.
521   */
522   onDestroy() {
523     this.columns.onDestroy();
524   }
525
526   /**
527    * Invoked when the application window is closed, triggering a close
528    * event to ensure panels are correctly laid out.
529   */
530   onClose() {
531     this.columns.onClose();
532   }
533
534   /**
535    * Invoked when the application window is maximized, triggering a maximize
536    * event to ensure panels are correctly laid out.
537   */
538   onMaximize() {
539     this.columns.onMaximize();
540   }
541
542   /**
543    * Invoked when the application window is restored from a minimized state,
544    * triggering a restore event to ensure panels are correctly laid out.
545   */
546   onRestore() {
547     this.columns.onRestore();
548   }
549
550   /**
551    * Invoked when the application window is activated, triggering an activate
552    * event to ensure panels are correctly laid out.
553   */
554   onActivate() {
555     this.columns.onActivate();
556   }
557
558   /**
559    * Invoked when the application window is deactivated, triggering a deactivate
560    * event to ensure panels are correctly laid out.
561   */
562   onDeactivate() {
563     this.columns.onDeactivate();
564   }
565
566   /**
567    * Invoked when the application window is focused, triggering a focus
568    * event to ensure panels are correctly laid out.
569   */
570   onFocus() {
571     this.columns.onFocus();
572   }
573
574   /**
575    * Invoked when the application window is unfocused, triggering an unfocus
576    * event to ensure panels are correctly laid out.
577   */
578   onUnfocus() {
579     this.columns.onUnfocus();
580   }
581
582   /**
583    * Invoked when the application window is blurred, triggering a blur
584    * event to ensure panels are correctly laid out.
585   */
586   onBlur() {
587     this.columns.onBlur();
588   }
589
590   /**
591    * Invoked when the application window is destroyed, triggering a destroy
592    * event to ensure panels are correctly laid out.
593   */
594   onDestroy() {
595     this.columns.onDestroy();
596   }
597
598   /**
599    * Invoked when the application window is closed, triggering a close
600    * event to ensure panels are correctly laid out.
601   */
602   onClose() {
603     this.columns.onClose();
604   }
605
606   /**
607    * Invoked when the application window is maximized, triggering a maximize
608    * event to ensure panels are correctly laid out.
609   */
610   onMaximize() {
611     this.columns.onMaximize();
612   }
613
614   /**
615    * Invoked when the application window is restored from a minimized state,
616    * triggering a restore event to ensure panels are correctly laid out.
617   */
618   onRestore() {
619     this.columns.onRestore();
620   }
621
622   /**
623    * Invoked when the application window is activated, triggering an activate
624    * event to ensure panels are correctly laid out.
625   */
626   onActivate() {
627     this.columns.onActivate();
628   }
629
630   /**
631    * Invoked when the application window is deactivated, triggering a deactivate
632    * event to ensure panels are correctly laid out.
633   */
634   onDeactivate() {
635     this.columns.onDeactivate();
636   }
637
638   /**
639    * Invoked when the application window is focused, triggering a focus
640    * event to ensure panels are correctly laid out.
641   */
642   onFocus() {
643     this.columns.onFocus();
644   }
645
646   /**
647    * Invoked when the application window is unfocused, triggering an unfocus
648    * event to ensure panels are correctly laid out.
649   */
650   onUnfocus() {
651     this.columns.onUnfocus();
652   }
653
654   /**
655    * Invoked when the application window is blurred, triggering a blur
656    * event to ensure panels are correctly laid out.
657   */
658   onBlur() {
659     this.columns.onBlur();
660   }
661
662   /**
663    * Invoked when the application window is destroyed, triggering a destroy
664    * event to ensure panels are correctly laid out.
665   */
666   onDestroy() {
667     this.columns.onDestroy();
668   }
669
670   /**
671    * Invoked when the application window is closed, triggering a close
672    * event to ensure panels are correctly laid out.
673   */
674   onClose() {
675     this.columns.onClose();
676   }
677
678   /**
679    * Invoked when the application window is maximized, triggering a maximize
680    * event to ensure panels are correctly laid out.
681   */
682   onMaximize() {
683     this.columns.onMaximize();
684   }
685
686   /**
687    * Invoked when the application window is restored from a minimized state,
688    * triggering a restore event to ensure panels are correctly laid out.
689   */
690   onRestore() {
691     this.columns.onRestore();
692   }
693
694   /**
695    * Invoked when the application window is activated, triggering an activate
696    * event to ensure panels are correctly laid out.
697   */
698   onActivate() {
699     this.columns.onActivate();
700   }
701
702   /**
703    * Invoked when the application window is deactivated, triggering a deactivate
704    * event to ensure panels are correctly laid out.
705   */
706   onDeactivate() {
707     this.columns.onDeactivate();
708   }
709
710   /**
711    * Invoked when the application window is focused, triggering a focus
712    * event to ensure panels are correctly laid out.
713   */
714   onFocus() {
715     this.columns.onFocus();
716   }
717
718   /**
719    * Invoked when the application window is unfocused, triggering an unfocus
720    * event to ensure panels are correctly laid out.
721   */
722   onUnfocus() {
723     this.columns.onUnfocus();
724   }
725
726   /**
727    * Invoked when the application window is blurred, triggering a blur
728    * event to ensure panels are correctly laid out.
729   */
730   onBlur() {
731     this.columns.onBlur();
732   }
733
734   /**
735    * Invoked when the application window is destroyed, triggering a destroy
736    * event to ensure panels are correctly laid out.
737   */
738   onDestroy() {
739     this.columns.onDestroy();
740   }
741
742   /**
743    * Invoked when the application window is closed, triggering a close
744    * event to ensure panels are correctly laid out.
745   */
746   onClose() {
747     this.columns.onClose();
748   }
749
750   /**
751    * Invoked when the application window is maximized, triggering a maximize
752    * event to ensure panels are correctly laid out.
753   */
754   onMaximize() {
755     this.columns.onMaximize();
756   }
757
758   /**
759    * Invoked when the application window is restored from a minimized state,
760    * triggering a restore event to ensure panels are correctly laid out.
761   */
762   onRestore() {
763     this.columns.onRestore();
764   }
765
766   /**
767    * Invoked when the application window is activated, triggering an activate
768    * event to ensure panels are correctly laid out.
769   */
770   onActivate() {
771     this.columns.onActivate();
772   }
773
774   /**
775    * Invoked when the application window is deactivated, triggering a deactivate
776    * event to ensure panels are correctly laid out.
777   */
778   onDeactivate() {
779     this.columns.onDeactivate();
780   }
781
782   /**
783    * Invoked when the application window is focused, triggering a focus
784    * event to ensure panels are correctly laid out.
785   */
786   onFocus() {
787     this.columns.onFocus();
788   }
789
790   /**
791    * Invoked when the application window is unfocused, triggering an unfocus
792    * event to ensure panels are correctly laid out.
793   */
794   onUnfocus() {
795     this.columns.onUnfocus();
796   }
797
798   /**
799    * Invoked when the application window is blurred, triggering a blur
800    * event to ensure panels are correctly laid out.
801   */
802   onBlur() {
803     this.columns.onBlur();
804   }
805
806   /**
807    * Invoked when the application window is destroyed, triggering a destroy
808    * event to ensure panels are correctly laid out.
809   */
810   onDestroy() {
811     this.columns.onDestroy();
812   }
813
814   /**
815    * Invoked when the application window is closed, triggering a close
816    * event to ensure panels are correctly laid out.
817   */
818   onClose() {
819     this.columns.onClose();
820   }
821
822   /**
823    * Invoked when the application window is maximized, triggering a maximize
824    * event to ensure panels are correctly laid out.
825   */
826   onMaximize() {
827     this.columns.onMaximize();
828   }
829
830   /**
831    * Invoked when the application window is restored from a minimized state,
832    * triggering a restore event to ensure panels are correctly laid out.
833   */
834   onRestore() {
835     this.columns.onRestore();
836   }
837
838   /**
839    * Invoked when the application window is activated, triggering an activate
840    * event to ensure panels are correctly laid out.
841   */
842   onActivate() {
843     this.columns.onActivate();
844   }
845
846   /**
847    * Invoked when the application window is deactivated, triggering a deactivate
848    * event to ensure panels are correctly laid out.
849   */
850   onDeactivate() {
851     this.columns.onDeactivate();
852   }
853
854   /**
855    * Invoked when the application window is focused, triggering a focus
856    * event to ensure panels are correctly laid out.
857   */
858   onFocus() {
859     this.columns.onFocus();
860   }
861
862   /**
863    * Invoked when the application window is unfocused, triggering an unfocus
864    * event to ensure panels are correctly laid out.
865   */
866   onUnfocus() {
867     this.columns.onUnfocus();
868   }
869
870   /**
871    * Invoked when the application window is blurred, triggering a blur
872    * event to ensure panels are correctly laid out.
873   */
874   onBlur() {
875     this.columns.onBlur();
876   }
877
878   /**
879    * Invoked when the application window is destroyed, triggering a destroy
880    * event to ensure panels are correctly laid out.
881   */
882   onDestroy() {
883     this.columns.onDestroy();
884   }
885
886   /**
887    * Invoked when the application window is closed, triggering a close
888    * event to ensure panels are correctly laid out.
889   */
890   onClose() {
891     this.columns.onClose();
892   }
893
894   /**
895    * Invoked when the application window is maximized, triggering a maximize
896    * event to ensure panels are correctly laid out.
897   */
898   onMaximize() {
899     this.columns.onMaximize();
900   }
901
902   /**
903    * Invoked when the application window is restored from a minimized state,
904    * triggering a restore event to ensure panels are correctly laid out.
905   */
906   onRestore() {
907     this.columns.onRestore();
908   }
909
910   /**
911    * Invoked when the application window is activated, triggering an activate
912    * event to ensure panels are correctly laid out.
913   */
914   onActivate() {
915     this.columns.onActivate();
916   }
917
918   /**
919    * Invoked when the application window is deactivated, triggering a deactivate
920    * event to ensure panels are correctly laid out.
921   */
922   onDeactivate() {
923     this.columns.onDeactivate();
924   }
925
926   /**
927    * Invoked when the application window is focused, triggering a focus
928    * event to ensure panels are correctly laid out.
929   */
930   onFocus() {
931     this.columns.onFocus();
932   }
933
934   /**
935    * Invoked when the application window is unfocused, triggering an unfocus
936    * event to ensure panels are correctly laid out.
937   */
938   onUnfocus() {
939     this.columns.onUnfocus();
940   }
941
942   /**
943    * Invoked when the application window is blurred, triggering a blur
944    * event to ensure panels are correctly laid out.
945   */
946   onBlur() {
947     this.columns.onBlur();
948   }
949
950   /**
951    * Invoked when the application window is destroyed, triggering a destroy
952    * event to ensure panels are correctly laid out.
953   */
954   onDestroy() {
955     this.columns.onDestroy();
956   }
957
958   /**
959    * Invoked when the application window is closed, triggering a close
960    * event to ensure panels are correctly laid out.
961   */
962   onClose() {
963     this.columns.onClose();
964   }
965
966   /**
967    * Invoked when the application window is maximized, triggering a maximize
968    * event to ensure panels are correctly laid out.
969   */
970   onMaximize() {
971     this.columns.onMaximize();
972   }
973
974   /**
975    * Invoked when the application window is restored from a minimized state,
976    * triggering a restore event to ensure panels are correctly laid out.
977   */
978   onRestore() {
979     this.columns.onRestore();
980   }
981
982   /**
983    * Invoked when the application window is activated, triggering an activate
984    * event to ensure panels are correctly laid out.
985   */
986   onActivate() {
987     this.columns.onActivate();
988   }
989
990   /**
991    * Invoked when the application window is deactivated, triggering a deactivate
992    * event to ensure panels are correctly laid out.
993   */
994   onDeactivate() {
995     this.columns.onDeactivate();
996   }
997
998   /**
999    * Invoked when the application window is focused, triggering a focus
1000   * event to ensure panels are correctly laid out.
1001  */
1002  onFocus() {
1003    this.columns.onFocus();
1004  }
1005
1006  /**
1007   * Invoked when the application window is unfocused, triggering an unfocus
1008   * event to ensure panels are correctly laid out.
1009  */
1010  onUnfocus() {
1011    this.columns.onUnfocus();
1012  }
1013
1014  /**
1015   * Invoked when the application window is blurred, triggering a blur
1016   * event to ensure panels are correctly laid out.
1017  */
1018  onBlur() {
1019    this.columns.onBlur();
1020  }
1021
1022  /**
1023   * Invoked when the application window is destroyed, triggering a destroy
1024   * event to ensure panels are correctly laid out.
1025  */
1026  onDestroy() {
1027    this.columns.onDestroy();
1028  }
1029
1030  /**
1031   * Invoked when the application window is closed, triggering a close
1032   * event to ensure panels are correctly laid out.
1033  */
1034  onClose() {
1035    this.columns.onClose();
1036  }
1037
1038  /**
1039   * Invoked when the application window is maximized, triggering a maximize
1040   * event to ensure panels are correctly laid out.
1041  */
1042  onMaximize() {
1043    this.columns.onMaximize();
1044  }
1045
1046  /**
1047   * Invoked when the application window is restored from a minimized state,
1048   * triggering a restore event to ensure panels are correctly laid out.
1049  */
1050  onRestore() {
1051    this.columns.onRestore();
1052  }
1053
1054  /**
1055   * Invoked when the application window is activated, triggering an activate
1056   * event to ensure panels are correctly laid out.
1057  */
1058  onActivate() {
1059    this.columns.onActivate();
1060  }
1061
1062  /**
1063   * Invoked when the application window is deactivated, triggering a deactivate
1064   * event to ensure panels are correctly laid out.
1065  */
1066  onDeactivate() {
1067    this.columns.onDeactivate();
1068  }
1069
1070  /**
1071   * Invoked when the application window is focused, triggering a focus
1072   * event to ensure panels are correctly laid out.
1073  */
1074  onFocus() {
1075    this.columns.onFocus();
1076  }
1077
1078  /**
1079   * Invoked when the application window is unfocused, triggering an unfocus
1080   * event to ensure panels are correctly laid out.
1081  */
1082  onUnfocus() {
1083    this.columns.onUnfocus();
1084  }
1085
1086  /**
1087   * Invoked when the application window is blurred, triggering a blur
1088   * event to ensure panels are correctly laid out.
1089  */
1090  onBlur() {
1091    this.columns.onBlur();
1092  }
1093
1094  /**
1095   * Invoked when the application window is destroyed, triggering a destroy
1096   * event to ensure panels are correctly laid out.
1097  */
1098  onDestroy() {
1099    this.columns.onDestroy();
1100  }
1101
1102  /**
1103   * Invoked when the application window is closed, triggering a close
1104   * event to ensure panels are correctly laid out.
1105  */
1106  onClose() {
1107    this.columns.onClose();
1108  }
1109
1110  /**
1111   * Invoked when the application window is maximized, triggering a maximize
1112   * event to ensure panels are correctly laid out.
1113  */
1114  onMaximize() {
1115    this.columns.onMaximize();
1116  }
1117
1118  /**
1119   * Invoked when the application window is restored from a minimized state,
1120   * triggering a restore event to ensure panels are correctly laid out.
1121  */
1122  onRestore() {
1123    this.columns.onRestore();
1124  }
1125
1126  /**
1127   * Invoked when the application window is activated, triggering an activate
1128   * event to ensure panels are correctly laid out.
1129  */
1130  onActivate() {
1131    this.columns.onActivate();
1132  }
1133
1134  /**
1135   * Invoked when the application window is deactivated, triggering a deactivate
1136   * event to ensure panels are correctly laid out.
1137  */
1138  onDeactivate() {
1139    this.columns.onDeactivate();
1140  }
1141
1142  /**
1143   * Invoked when the application window is focused, triggering a focus
1144   * event to ensure panels are correctly laid out.
1145  */
1146  onFocus() {
1147    this.columns.onFocus();
1148  }
1149
1150  /**
1151   * Invoked when the application window is unfocused, triggering an unfocus
1152   * event to ensure panels are correctly laid out.
1153  */
1154  onUnfocus() {
1155    this.columns.onUnfocus();
1156  }
1157
1158  /**
1159   * Invoked when the application window is blurred, triggering a blur
1160   * event to ensure panels are correctly laid out.
1161  */
1162  onBlur() {
1163    this.columns.onBlur();
1164  }
1165
1166  /**
1167   * Invoked when the application window is destroyed, triggering a destroy
1168   * event to ensure panels are correctly laid out.
1169  */
1170  onDestroy() {
1171    this.columns.onDestroy();
1172  }
1173
1174  /**
1175   * Invoked when the application window is closed, triggering a close
1176   * event to ensure panels are correctly laid out.
1177  */
1178  onClose() {
1179    this.columns.onClose();
1180  }
1181
1182  /**
1183   * Invoked when the application window is maximized, triggering a maximize
1184   * event to ensure panels are correctly laid out.
1185  */
1186  onMaximize() {
1187    this.columns.onMaximize();
1188  }
1189
1190  /**
1191   * Invoked when the application window is restored from a minimized state,
1192   * triggering a restore event to ensure panels are correctly laid out.
1193  */
1194  onRestore() {
1195    this.columns.onRestore();
1196  }
1197
1198  /**
1199   * Invoked when the application window is activated, triggering an activate
1200   * event to ensure panels are correctly laid out.
1201  */
1202  onActivate() {
1203    this.columns.onActivate();
1204  }
1205
1206  /**
1207   * Invoked when the application window is deactivated, triggering a deactivate
1208   * event to ensure panels are correctly laid out.
1209  */
1210  onDeactivate() {
1211    this.columns.onDeactivate();
1212  }
1213
1214  /**
1215   * Invoked when the application window is focused, triggering a focus
1216   * event to ensure panels are correctly laid out.
1217  */
1218  onFocus() {
1219    this.columns.onFocus();
1220  }
1221
1222  /**
1223   * Invoked when the application window is unfocused, triggering an unfocus
1224   * event to ensure panels are correctly laid out.
1225  */
1226  onUnfocus() {
1227    this.columns.onUnfocus();
1228  }
1229
1230  /**
1231   * Invoked when the application window is blurred, triggering a blur
1232   * event to ensure panels are correctly laid out.
1233  */
1234  onBlur() {
1235    this.columns.onBlur();
1236  }
1237
1238  /**
1239   * Invoked when the application window is destroyed, triggering a destroy
1240   * event to ensure panels are correctly laid out.
1241  */
1242  onDestroy() {
1243    this.columns.onDestroy();
1244  }
1245
1246  /**
1247   * Invoked when the application window is closed, triggering a close
1248   * event to ensure panels are correctly laid out.
1249  */
1250  onClose() {

```

```
47  }
48
49 exports.PRDC_JSLAB_PANELS = PRDC_JSLAB_PANELS;
50
51 /**
52 * Represents a single resizable panel within the application. Supports
53 * resizing, orientation configuration, and nesting of sub-panels.
54 */
55 class PRDC_JSLAB_PANEL {
56
57     /**
58      * Initializes a new panel with specified configuration.
59      * @param {string} id Unique identifier for the panel.
60      * @param {string} orientation Panel orientation ('vertical' or 'horizontal').
61      * @param {HTMLElement} container DOM element serving as the container for
62      * this panel.
63      * @param {Array<HTMLElement|string>} elements Array of DOM elements or
64      * selectors representing the child elements of this panel.
65      * @param {Array<number>} default_size Array of numbers representing the
66      * default sizes (in percentages) of the panel's child elements.
67      */
68     constructor(id, orientation, container, elements, default_size) {
69         let obj = this;
70         this.id = id;
71         this.orientation = orientation;
72         this.container = container;
73         this.elements = elements;
74         this.default_size = default_size;
75
76         this.container_bcr;
77         this.cells = elements.length;
78         this.cells_size = [];
79         this.sub_panels = [];
80
81         this.setContainerBCR();
82
83         if(orientation == 'vertical') {
84             this.attr_pos = 'left';
85             this.attr_size = 'width';
86             this.attr_axis = 'X';
87             this.attr_cont_size = 'clientWidth';
88         } else if(orientation == 'horizontal') {
89             this.attr_pos = 'top';
90             this.attr_size = 'height';
91             this.attr_axis = 'Y';
92             this.attr_cont_size = 'clientHeight';
93         }
94
95         // Get cell sizes in %
96         this.cells_size = store.get('panel-' + this.id);
97         if(!this.cells_size) {
98             this.cells_size = this.default_size;
99         }
100 }
```

```

97   var pos_resizer = 0;
98   var pos_cell = 0;
99   for(let i = 0; i < this.cells; i++) {
100     // Create resizer element
101     if(i < this.cells - 1) {
102       let resizer = document.createElement('div');
103       resizer.classList.add(orientation + '-resizer');
104       pos_resizer += this.cells_size[i];
105       resizer.style[this.attr_pos] = 'calc('+pos_resizer+'% - '+config.
106         PANEL_RESIZER_WIDTH/2+'px)';
107       resizer.style[this.attr_size] = config.PANEL_RESIZER_WIDTH+'px';
108       resizer.setAttribute('index', i);
109       this.container.appendChild(resizer);
110
111     // Create onDrag listener
112     let fun = function(e) {
113       obj.onResizerDrag(e, resizer);
114     };
115     resizer.addEventListener('mousedown', function() {
116       document.addEventListener('mousemove', fun, false);
117       document.addEventListener('mouseup', function mouseUp() {
118         document.removeEventListener('mousemove', fun, false);
119         document.removeEventListener('mousemove', mouseUp, false);
120       }, false);
121     }, false);
122
123     // Set cells size and position
124     if(typeof this.elements[i] === 'string') {
125       document.querySelectorAll(this.elements[i]).forEach(function(element)
126         {
127           element.style[obj.attr_pos] = pos_cell+'%';
128           element.style[obj.attr_size] = obj.cells_size[i]+'%';
129         });
130     } else {
131       this.elements[i].style[this.attr_pos] = pos_cell+'%';
132       this.elements[i].style[this.attr_size] = this.cells_size[i]+'%';
133     }
134     pos_cell += obj.cells_size[i];
135   }
136
137 /**
138 * Handles resizing of the panel, adjusting child elements and sub-panels as
139 * necessary.
140 */
141 onResize() {
142   this.setContainerBCR();
143   this.sub_panels.forEach(function(sub_panel) {
144     sub_panel.onResize();
145   });
146
147 /**
148 * Adds a sub-panel to the current panel, establishing a parent-child

```

```

        relationship for nested panel layouts.
    * @param {PRDC_JSLAB_PANEL} panel The sub-panel to add.
    */
addSubPanel(panel) {
    this.sub_panels.push(panel);
}

/**
 * Handles drag events for resizing the panel, updating sizes and positions
 * of child elements and persisting changes.
 * @param {Event} e The mouse event associated with the drag.
 * @param {HTMLElement} resizer The resizer element being dragged.
 */
onResizerDrag(e, resizer) {
    var obj = this;
    var i = Number(resizer.getAttribute('index'));

    var pos = (e['page'+this.attr_axis]-this.container_bcr[this.attr_pos])/this.container[this.attr_cont_size]*100;
    var total_size = this.cells_size[i]+this.cells_size[i+1];
    var size_pre = this.cells_size.slice(0, i).reduce(function(sum, num) {
        return sum + num; }, 0);
    if(pos < size_pre+config.PANEL_MIN_SIZE) {
        pos = size_pre+config.PANEL_MIN_SIZE;
    } else if(pos > size_pre+total_size-config.PANEL_MIN_SIZE) {
        pos = size_pre+total_size-config.PANEL_MIN_SIZE;
    }
    this.cells_size[i] = pos-size_pre;
    this.cells_size[i+1] = total_size-this.cells_size[i];

    // Set resizer position
    resizer.style[this.attr_pos] = 'calc('+pos+'% - '+config.PANEL_RESIZER_WIDTH/2+'px)';

    // Set first cell size and position
    if(typeof this.elements[i] === 'string') {
        document.querySelectorAll(this.elements[i]).forEach(function(element) {
            element.style[obj.attr_size] = obj.cells_size[i]+'%';
        });
    } else {
        this.elements[i].style[this.attr_size] = this.cells_size[i]+'%';
    }

    // Set second cell size and position
    if(typeof this.elements[i+1] === 'string') {
        document.querySelectorAll(this.elements[i+1]).forEach(function(element) {
            element.style[obj.attr_pos] = pos+'%';
            element.style[obj.attr_size] = obj.cells_size[i+1]+'%';
        });
    } else {
        this.elements[i+1].style[this.attr_pos] = pos+'%';
        this.elements[i+1].style[this.attr_size] = this.cells_size[i+1]+'%';
    }
}

```

```

198     this.sub_panels.forEach(function(sub_panel) {
199         sub_panel.onResize();
200     });
201     store.set('panel-' + this.id, this.cells_size);
202 }
203
204 /**
205 * Updates the bounding client rectangle (BCR) of the panel container, used
206 * in calculating sizes and positions.
207 */
208 setContainerBCR() {
209     this.container_bcr = this.container.getBoundingClientRect();
210 }
211
212 /**
213 * Updates the sizes and positions of the panel's child elements based on
214 * the current panel configuration.
215 */
216 updateCells() {
217     var pos_cell = 0;
218     for(let i = 0; i < this.cells; i++) {
219         // Set cells size and position
220         if(typeof this.elements[i] === 'string') {
221             document.querySelectorAll(this.elements[i]).forEach(function(element) {
222                 element.style[obj.attr_pos] = pos_cell+'%';
223                 element.style[obj.attr_size] = obj.cells_size[i]+'%';
224             });
225         } else {
226             this.elements[i].style[this.attr_pos] = pos_cell+'%';
227             this.elements[i].style[this.attr_size] = this.cells_size[i]+'%';
228         }
229         pos_cell += obj.cells_size[i];
230     }
231 }

```

Listing 67 - panels.js

```

1 /**
2 * @file JSLAB settings module
3 * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4 * PR-DC, Republic of Serbia
5 * info@pr-dc.com
6 */
7
8 /**
9 * Class for JSLAB settings.
10 */
11 class PRDC_JSLAB_SETTINGS {
12
13 /**
14 * Initializes the settings management functionality, setting up event
15 * listeners for settings menu actions and dialog interactions.
16 * @param {object} win The window object representing the current Electron
17 * window.

```

```

16   */
17 constructor(win) {
18   var obj = this;
19   this.win = win;
20
21   // Settings logic
22   $('#settings-menu').click(function() { obj.win.gui.settings(); });
23   $('#settings-close').click(function() { obj.win.gui.closeSettings(); });
24   $('#settings-container').on('keydown', function(e) {
25     if(e.key === 'Escape') {
26       // ESC
27       obj.win.gui.closeSettings();
28       e.stopPropagation();
29       e.preventDefault();
30     }
31   });
32
33   $('#settings-container .set-langauge').on('change', function() {
34     obj.win.gui.changeLangauge($('#this').val());
35   });
36   $('#settings-container .set-langauge').val(language.lang);
37
38   $('#settings-container .change-settings').on('click', function() {
39     obj.win.command_history.setMaxSize($('#settings-container .N-history-max').val());
40     obj.win.gui.closeSettings();
41   });
42 }
43
44
45 exports.PRDC_JSLAB_SETTINGS = PRDC_JSLAB_SETTINGS

```

Listing 68 - settings.js

```

1 /**
2  * @file JSLAB GUI script
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 // Modules
9 // -----
10 const { ipcRenderer } = require('electron');
11
12 const { PRDC_JSLAB_HELP } = require('./help');
13 const { PRDC_JSLAB_INFO } = require('./info');
14 const { PRDC_JSLAB_SETTINGS } = require('./settings');
15 const { PRDC_JSLAB_EVAL } = require('./eval');
16 const { PRDC_JSLAB_COMMAND_WINDOW } = require('./command-window');
17 const { PRDC_JSLAB_COMMAND_HISTORY } = require('./command-history');
18 const { PRDC_JSLAB_WORKSPACE } = require('./workspace');
19 const { PRDC_JSLAB_FOLDER_NAVIGATION } = require('./folder-navigation');
20 const { PRDC_JSLAB_FILE_BROWSER } = require('./file-browser');
21 const { PRDC_JSLAB_PANELS } = require('./panels');
22 const { PRDC_JSLAB_GUI } = require('./gui');

```

```

23 const { PRDC_JSLAB_APP } = require('./app');
24
25 const { PRDC_POPUP } = require('../..../lib/PRDC_POPUP/PRDC_POPUP');
26
27 const fs = require('fs');
28 const path = require('path');
29 const Store = require('electron-store');
30
31 const store = new Store();
32
33 /**
34 * Class for JSLAB main win.
35 */
36 class PRDC_JSLAB_WIN_MAIN {
37
38 /**
39 * Create main win.
40 */
41 constructor() {
42   var obj = this;
43
44 // Classes
45 this.eval = new PRDC_JSLAB_EVAL(this);
46 this.workspace = new PRDC_JSLAB_WORKSPACE(this);
47 this.file_browser = new PRDC_JSLAB_FILE_BROWSER(this);
48 this.panels = new PRDC_JSLAB_PANELS(this);
49 this.gui = new PRDC_JSLAB_GUI(this);
50 this.help = new PRDC_JSLAB_HELP(this);
51 this.app = new PRDC_JSLAB_APP(this);
52 this.command_history = new PRDC_JSLAB_COMMAND_HISTORY(this);
53 this.command_window = new PRDC_JSLAB_COMMAND_WINDOW(this);
54 this.info = new PRDC_JSLAB_INFO(this);
55 this.settings = new PRDC_JSLAB_SETTINGS(this);
56 this.folder_navigation = new PRDC_JSLAB_FOLDER_NAVIGATION(this);
57
58 this.flight_commands_popup = new PRDC_POPUP(document.getElementById(
      'sandbox-stats-icon'),
      document.getElementById('sandbox-stats-popup')));
59
60 // Prevent redirects
61 preventRedirect();
62
63
64 // Events
65 // -----
66 // Catch errors
67 window.addEventListener('unhandledrejection', function(e) {
68   obj.command_window.errorInternal(e.reason.stack);
69   e.preventDefault();
70 });
71
72 window.addEventListener('error', function(e) {
73   obj.command_window.errorInternal(e.error.stack);
74   e.preventDefault();
75 });
76

```

```
77 // On IPC message
78 ipcRenderer.on('MainWindow', function(event, action, data) {
79   switch(action) {
80     case 'editor-disp':
81       var msg = data[0];
82       var focus = data[1];
83       obj.command_window.messageEditor(msg);
84       if(focus) {
85         ipcRenderer.send('MainProcess', 'focus-win');
86       }
87       break;
88     case 'disp':
89       obj.command_window.message(data);
90       break;
91     case 'disp-monospaced':
92       obj.command_window.messageMonospaced(data);
93       break;
94     case 'disp-latex':
95       obj.command_window.messageLatex(data);
96       break;
97     case 'error':
98       obj.command_window.error(data);
99       ipcRenderer.send('MainProcess', 'focus-win');
100      break;
101    case 'warn':
102      obj.command_window.warn(data);
103      ipcRenderer.send('MainProcess', 'focus-win');
104      break;
105    case 'internal-error':
106      obj.command_window.errorInternal(data);
107      ipcRenderer.send('MainProcess', 'focus-win');
108      break;
109    case 'run':
110      var script_path = data[0];
111      var lines = data[1];
112      obj.eval.evalScript(script_path, lines);
113      ipcRenderer.send('MainProcess', 'focus-win');
114      break;
115    case 'help':
116      obj.gui.help();
117      break;
118    case 'info':
119      obj.gui.info();
120      break;
121    case 'settings':
122      obj.gui.settings();
123      break;
124    case 'clear':
125      obj.command_window.clear();
126      break;
127    case 'save-path':
128      var new_path = data;
129      obj.folder_navigation.savePath(new_path, false);
130      break;
131    case 'remove-path':
```

```

132     var saved_path = data;
133     obj.folder_navigation.removePath(saved_path, false);
134     break;
135   case 'set-workspace':
136     obj.workspace.setWorkspace(data);
137     break;
138   case 'update-workspace':
139     obj.workspace.updateWorkspace();
140     break;
141   case 'update-file-browser':
142     obj.file_browser.updateFileBrowser();
143     break;
144   case 'code-evaluating':
145     obj.evaluating = true;
146     break;
147   case 'code-evaluated':
148     obj.evaluating = false;
149     break;
150   case 'show-ans':
151     obj.command_window.highlightAnsMessage(data);
152     break;
153   case 'set-status':
154     var state = data[0];
155     var txt = data[1];
156     obj.gui.setStatus(state, txt);
157     break;
158   case 'set-stats':
159     var stats = data;
160     obj.gui.setStats(stats);
161     break;
162   case 'clear-storage':
163     store.clear();
164     break;
165   case 'unknown-script-dir':
166     obj.folder_navigation.unknownScriptDir();
167     break;
168   case 'set-current-path':
169     obj.folder_navigation.setPath(data);
170     break;
171   }
172 });
173
174 // Keyboard events
175 document.addEventListener('keydown', function(e) {
176   // Show Dev Tools
177   if(e.key == 'F12') {
178     ipcRenderer.send('MainProcess', 'show-dev-tools');
179     ipcRenderer.send('MainProcess', 'show-sandbox-dev-tools');
180     e.stopPropagation();
181     e.preventDefault();
182   } else if(e.ctrlKey && e.key.toLowerCase() == 'c') {
183     if(obj.getSelectionText() == "") {
184       // No selected text
185       obj.command_window.messageInternal(language.string(89));
186       ipcRenderer.send('SandboxWindow', 'stop-loop', true);

```

```

187         e.stopPropagation();
188         e.preventDefault();
189     }
190 } else if(e.key.toLowerCase() == 'h' && e.ctrlKey) {
191 // Ctrl + H
192 obj.gui.help();
193 e.stopPropagation();
194 e.preventDefault();
195 } else if(e.key.toLowerCase() == 'i' && e.ctrlKey) {
196 // Ctrl + I
197 obj.gui.info();
198 e.stopPropagation();
199 e.preventDefault();
200 } else if(e.key.toLowerCase() == 's' && e.ctrlKey) {
201 // Ctrl + S
202 obj.gui.settings();
203 e.stopPropagation();
204 e.preventDefault();
205 } else if(e.key.toLowerCase() == 'd' && e.ctrlKey) {
206 // Ctrl + D
207 obj.eval.evalCommand('openDoc()');
208 e.stopPropagation();
209 e.preventDefault();
210 }
211 });
212 }
213
214 /**
215 * Method called when the program is ready to start.
216 */
217 onReady() {
218     var obj = this;
219     obj.processArguments();
220     obj.panels.onReady();
221
222     // Fade in window
223     ipcRenderer.send('MainProcess', 'fade-in-win');
224
225     // Focus code input
226     obj.command_window.code_input.focus();
227     obj.command_window.code_input.setCursor(
228         obj.command_window.code_input.lineCount(), 0);
229 }
230
231 /**
232 * Processes startup arguments, opening files or setting the workspace as
233 * needed.
234 */
235 processArguments() {
236     // Process arguments
237     if(process.argv.length > 0) {
238         var arg = process.argv[1];
239
240         // Check if there is scripts in argument
241         if(arg && this.folder_navigation.checkFile(arg)) {

```

```

241     // Open script in editor
242     ipcRenderer.send('EditorWindow', 'open-script', [arg]);
243     var dir = path.dirname(arg);
244     if(dir !== undefined) {
245         this.folder_navigation.setPath(dir);
246     }
247 }
248 }
249 }
250
251 /**
252 * Retrieves selected text within the application, if any.
253 * @return {string} The currently selected text.
254 */
255 getSelectionText() {
256     var text = "";
257     if(window.getSelection) {
258         text = window.getSelection().toString();
259     } else if(document.selection && document.selection.type != "Control") {
260         text = document.selection.createRange().text;
261     }
262     return text;
263 }
264
265 /**
266 * Resets the sandbox and updates paths and settings.
267 */
268 onSandboxReset() {
269     ipcRenderer.send('SandboxWindow', 'set-language', language.lang);
270     ipcRenderer.send('SandboxWindow', 'set-saved-paths', this.
271         folder_navigation.saved_paths);
272     ipcRenderer.send('SandboxWindow', 'set-current-path', this.
273         folder_navigation.current_path);
274     this.command_window.clear();
275     this.workspace.updateWorkspace();
276     this.gui.resetStats();
277     this.evaluating = false;
278 }
279
280 /**
281 * Gracefully closes the application after performing necessary cleanup
282 * operations.
283 */
284 close() {
285     store.set('full_history', this.command_history.full_history);
286     store.set('current_path', this.folder_navigation.current_path);
287     store.set('saved_paths', this.folder_navigation.saved_paths);
288     ipcRenderer.send('MainProcess', 'close-app');
289 }
290
291 exports.PRDC_JSLAB_WIN_MAIN = PRDC_JSLAB_WIN_MAIN;

```

Listing 69 - win-main.js

```
2  * @file JSLAB workspace module
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 /**
9  * Class for JSLAB workspace.
10 */
11 class PRDC_JSLAB_WORKSPACE {
12
13 /**
14  * Initializes the workspace, setting up the UI component and event
15  * listeners for workspace interactions.
16  * @param {object} win The window object representing the current Electron
17  * window.
18 */
19 constructor(win) {
20     var obj = this;
21     this.win = win;
22
23     this.data = [];
24     this.workspace_cont = document.getElementById('workspace');
25
26     // Workspace clear button click
27     $('#workspace-options .clear').click(function(){
28         obj.win.evalCommand('clear');
29     });
30
31 /**
32  * Updates the workspace with new data, refreshing the display of variables
33  * and their values.
34  * @param {Array} data The data to be displayed in the workspace, typically
35  * an array of variable information.
36 */
37 setWorkspace(data) {
38     this.data = data;
39     this.updateWorkspace();
40
41 /**
42  * Refreshes the workspace display based on the current data, updating the
43  * layout and content of the workspace area.
44 */
45 updateWorkspace() {
46     var obj = this;
47
48     var workspace_table = $('#workspace .table')[0];
49     workspace_table.innerHTML = '';
50     var cells_size = this.win.panels.workspace_columns.cells_size;
51     this.data.forEach(function(v) {
52         var row = document.createElement('div');
53         row.onclick = function() {
54             obj.win.command_window.code_input.setValue(this.getAttribute('variable'));
```

```

      ') );
52   obj.win.command_window.code_input.focus();
53   obj.win.command_window.code_input.setCursor(obj.win.command_window.
54     code_input.lineCount(), 0);
55 };
56 row.onclick = function() {
57   obj.win.eval.evalCommand(this.getAttribute('variable'));
58 };
59 row.setAttribute('variable', v[0]);
60 row.className = 'row';
61 row.innerHTML = '<div class="col col-1" style="width:' + cells_size[0] +
62   '%">' + v[0] +
63   '</div><div class="col col-2" style="width:' + cells_size[1] + '%">' + v
64   [1] +
65   '</div><div class="col col-3" style="width:' + cells_size[2] + '%">' + v
66   [2] + '</div>';
67 workspace_table.appendChild(row);
68 });
69 }
70 }

68 exports.PRDC_JSLAB_WORKSPACE = PRDC_JSLAB_WORKSPACE;

```

Listing 70 - workspace.js

6.5 sandbox

```

1 /**
2  * @file JSLAB library array submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 /**
9  * Class for JSLAB array submodule.
10 * Column-major order for matrix operation
11 */
12 class PRDC_JSLAB_LIB_ARRAY {
13
14 /**
15  * Constructs an array submodule object for JSLAB.
16  * @param {Object} jsl - Reference to the main JSLAB object.
17 */
18 constructor(jsl) {
19   var obj = this;
20   this.jsl = jsl;
21 }
22
23 /**
24  * Converts an iterable or array-like object into a standard array.
25  * @param {Iterable|ArrayLike} A - The iterable or array-like object to
26  * convert.
27  * @returns {Array} A new array containing the elements from the input.

```



```
27      */
28  array(A) {
29      return Array.from(A);
30  }
31
32 /**
33 * Retrieves the last element from the provided array.
34 * @param {Array} A - The array from which the last element is to be
35 * retrieved.
36 * @returns {*} The last element of the array. If the array is empty,
37 * returns undefined.
38 */
39 end(A) {
40     return A[A.length - 1];
41 }
42
43 /**
44 * Returns the index of the last element in the array.
45 * @param {Array} A - The array to evaluate.
46 * @returns {number} The index of the last element.
47 */
48 endi(A) {
49     return A.length - 1;
50 }
51
52 /**
53 * Retrieves a specific column from the provided matrix.
54 * @param {Array} A - The matrix array.
55 * @param {number} col - The column index to retrieve.
56 * @returns {Array} The specified column as an array.
57 */
58 column(A, col) {
59     return A.map(function(x) { return x[col]; });
60 }
61
62 /**
63 * Retrieves a specific row from the provided matrix.
64 * @param {Array} A - The matrix array.
65 * @param {number} row - The row index to retrieve.
66 * @returns {Array} The specified row as an array.
67 */
68 row(A, row) {
69     return A[row];
70 }
71
72 /**
73 * Returns the first N elements from an array.
74 * @param {Array} A - The input array.
75 * @param {number} [N=1] - The number of elements to return from the start.
76 * @returns {Array} - An array containing the first N elements.
77 */
78 first(A, N = 1) {
79     return A.slice(0, N);
}
```

```

80  /**
81   * Returns the last N elements from an array.
82   * @param {Array} A - The input array.
83   * @param {number} [N=1] - The number of elements to return from the end.
84   * @returns {Array} - An array containing the last N elements.
85   */
86  last(A, N = 1) {
87    return A.slice(-N);
88  }
89
90 /**
91  * Generates an array of indices based on rows and columns.
92  * @param {number|number[]} rows - The row index or array of row indices.
93  * @param {number|number[]} cols - The column index or array of column
94  *     indices.
95  * @param {number} rows_max - The maximum number of rows.
96  * @returns {number[]} An array of calculated indices.
97  */
98  index(rows, cols, rows_max) {
99    if(!Array.isArray(rows)) {
100      rows = [rows];
101    }
102    if(!Array.isArray(cols)) {
103      cols = [cols];
104    }
105    var A = zeros(rows.length * cols.length);
106
107    var k = 0;
108    for(var j = 0; j < cols.length; j++) {
109      for(var i = 0; i < rows.length; i++) {
110        A[k++] = rows[i] + cols[j] * rows_max;
111      }
112    }
113    return A;
114  }
115
116 /**
117  * Finds all indices of a specified value in the array.
118  * @param {Array} A - The array to search.
119  * @param {*} value - The value to find.
120  * @returns {number[]} An array of indices where the value is found.
121  */
122  indexOfAll(A, value) {
123    return A.reduce(function(a, e, i) {
124      if(e === value) {
125        a.push(i);
126      }
127      return a;
128    }, []);
129  }
130
131 /**
132  * Finds the index of a sequence of elements in the array.
133  * @param {Array} A - The array to search.

```

```

134  * @param {Array} search_elements - The sequence of elements to find.
135  * @param {number} [from_index=0] - The index to start the search from.
136  * @returns {number} The starting index of the found sequence, or -1 if not
137  *   found.
138  */
139  indexOfMulti(A, search_elements, from_index) {
140    from_index = from_index || 0;
141
142    var index = Array.prototype.indexOf.call(A, search_elements[0],
143      from_index);
144    if(search_elements.length === 1 || index === -1) {
145      // Not found or no other elements to check
146      return index;
147    }
148
149    for(var i = index, j = 0;
150        j < search_elements.length && i < A.length; i++, j++) {
151      if(A[i] !== search_elements[j]) {
152        return indexOfMulti(A, search_elements, index + 1);
153      }
154    }
155
156    return (i === index + search_elements.length) ? index : -1;
157  }
158 /**
159  * Shuffles indices of an array using the Fisher-Yates algorithm.
160  * @param {Array} array Array whose indices are to be shuffled.
161  * @return {Array} Shuffled array of indices.
162  */
163 shuffleIndices(array) {
164   const idx = Array.from(array.keys());
165   for(let i = idx.length - 1; i > 0; i--) {
166     const j = Math.floor(Math.random() * (i + 1));
167     [idx[i], idx[j]] = [idx[j], idx[i]];
168   }
169   return idx;
170 }
171 /**
172  * Sets a subset of array elements based on provided indices.
173  * @param {Array} A - The target array.
174  * @param {number[]} indices - The indices at which to set values.
175  * @param {Array} B - The array of values to set.
176  */
177 setSub(A, indices, B) {
178   var j = 0;
179   if(!Array.isArray(B)) {
180     B = createFilledArray(indices.length, B);
181   }
182   for(var i = 0; i < indices.length; i++) {
183     A[indices[i]] = B[j++];
184   }
185 }
186
187

```

```

188  /**
189   * Sets a subset of array elements based on provided boolean indices.
190   * @param {Array} A - The target array to modify.
191   * @param {boolean[]} b - A boolean array indicating which elements to set (true = set, false = skip).
192   * @param {Array} B - The array of values to set at the indices determined by 'b'.
193   */
194  setSubB(A, b, B) {
195    this.setSub(A, b2i(b), B);
196  }
197
198  /**
199   * Retrieves a subset of array elements based on provided indices.
200   * @param {Array} A - The source array.
201   * @param {number[]} indices - The indices of elements to retrieve.
202   * @returns {Array} An array containing the retrieved elements.
203   */
204  getSub(A, indices) {
205    var B = zeros(indices.length);
206    var j = 0;
207    for(var i = 0; i < indices.length; i++) {
208      B[j++] = A[indices[i]];
209    }
210    return B;
211  }
212
213  /**
214   * Retrieves a subset of array elements based on provided indices.
215   * @param {Array} A - The source array.
216   * @param {boolean[]} b - A boolean array indicating which elements to retrieve.
217   * @returns {Array} An array containing the retrieved elements.
218   */
219  getSubB(A, b) {
220    return this.getSub(A, b2i(b));
221  }
222
223  /**
224   * Moves an element within the array from one index to another.
225   * @param {Array} A - The array to modify.
226   * @param {number} from_index - The index of the element to move.
227   * @param {number} to_index - The target index where the element should be moved.
228   * @returns {Array} The modified array with the element moved.
229   */
230  moveElement(A, from_index, to_index) {
231    const start_index = from_index < 0 ? A.length + from_index : from_index;
232
233    if(start_index >= 0 && start_index < A.length) {
234      const end_index = to_index < 0 ? A.length + to_index : to_index;
235
236      const [item] = A.splice(from_index, 1);
237      A.splice(end_index, 0, item);
238    }
  
```

```
239     return A;
240 }
241
242 /**
243 * Removes an element from the array at the specified index.
244 * @param {Array} A - The array to modify.
245 * @param {number} index - The index of the element to remove.
246 * @returns {Array} The array after the element has been removed.
247 */
248 removeElement(A, index) {
249     return A.splice(index, 1);
250 }
251
252 /**
253 * Removes the first occurrence of a specified value from the array.
254 * @param {Array} A - The array to modify.
255 * @param {*} value - The value to remove.
256 * @returns {Array} The array after the value has been removed.
257 */
258 removeElementByValue(A, value) {
259     var index = A.indexOf(value);
260     if(index !== -1) {
261         A.splice(index, 1);
262     }
263     return A;
264 }
265
266 /**
267 * Removes elements from array A that have properties listed in array B.
268 * @param {Array<Object>} A - The array to filter.
269 * @param {Array} B - The array of properties or values to remove.
270 * @param {string} [prop] - The property name to check in objects within A.
271 * @returns {Array} The filtered array.
272 */
273 removeElementProp(A, B, prop) {
274     if(prop) {
275         return A.filter(function(e) { return !B.includes(e[prop]); });
276     } else {
277         return A.filter(function(e) { return !B.includes(e); });
278     }
279 }
280
281 /**
282 * Finds the index of the first object in the array where the specified
283 * property matches the given value.
284 * @param {Array<Object>} A - The array to search.
285 * @param {string} property - The property name to compare.
286 * @param {*} value - The value to match.
287 * @returns {number} The index of the matching object, or -1 if not found.
288 */
289 findIndexProp(A, property, value) {
290     return A.findIndex(function(object) {
291         return object[property] === value;
292     });
293 }
```

```
293
294  /**
295   * Sets a value at the specified multi-dimensional indices in the array.
296   * @param {Array} A - The target array.
297   * @param {number[]} indices - An array of indices representing the position
298   *
299   * @param {*} value - The value to set.
300   * @returns {*} The value that was set.
301   */
302   setValueAt(A, indices, value) {
303     var reference = A;
304     for(var i = 0; i < indices.length - 1; i++) {
305       reference = reference[indices[i]];
306     }
307     reference[indices.length - 1] = value;
308     return reference;
309   }
310 /**
311  * Retrieves a value from the array at the specified multi-dimensional
312  * indices.
313  * @param {Array} A - The source array.
314  * @param {number[]} indices - An array of indices representing the position
315  *
316  * @returns {*} The value at the specified indices, or undefined if out of
317  * bounds.
318  */
319 getValueAt(A, indices) {
320   if(Array.isArray(A)) {
321     var reference = A;
322     for(var i = 0; i < indices.length; i++) {
323       if(indices[i] < reference.length) {
324         reference = reference[indices[i]];
325       } else {
326         return undefined;
327       }
328     }
329     return reference;
330   } else {
331     return undefined;
332   }
333 /**
334  * Alias for setValueAt.
335  * @param {Array} A - The target array.
336  * @param {number[]} indices - An array of indices representing the position
337  *
338  * @param {*} value - The value to set.
339  * @returns {*} The value that was set.
340  */
341 setVal(A, indices, value) {
342   return this.setValueAt(A, indices, value);
343 }
```

```
343  /**
344  * Alias for getValueAt.
345  * @param {Array} A - The source array.
346  * @param {number[]} indices - An array of indices representing the position
347  *
348  * @returns {*} The value at the specified indices, or undefined if out of
349  * bounds.
350  */
351 getVal(A, indices) {
352   return this.getValueAt(A, indices);
353 }
354 /**
355 * Creates a shallow copy of the provided array.
356 * @param {Array} A - The array to clone.
357 * @returns {Array} A new array containing all elements from A.
358 */
359 cloneArray(A) {
360   return [...A];
361 }
362 /**
363 * Creates an n-dimensional array.
364 * @param {number} length - The size of the first dimension.
365 * @param {...number} [dimensions] - Sizes of subsequent dimensions.
366 * @returns {Array} The newly created n-dimensional array.
367 */
368 createArray(length) {
369   var A = new Array(length || 0);
370   var i = length;
371
372   if(arguments.length > 1) {
373     var args = Array.prototype.slice.call(arguments, 1);
374     while(i--) A[(length-1)-i] = createArray.apply(this, args);
375   }
376
377   return A;
378 }
379 /**
380 * Creates an n-dimensional array filled with a specific value.
381 * @param {number} length - The size of the first dimension.
382 * @param {*} val - The value to fill the array with.
383 * @param {...number} [dimensions] - Sizes of subsequent dimensions.
384 * @returns {Array} The filled n-dimensional array.
385 */
386 createFilledArray(length, val) {
387   var A = new Array(length || 0);
388   var i = length;
389
390   if(arguments.length > 2) {
391     var args = Array.prototype.slice.call(arguments, 1);
392     while(i--) A[(length-1)-i] = this.createFilledArray.apply(this, args);
393   } else {
394     A.fill(val);
395   }
396 }
```

```
396     }
397
398     return A;
399 }
400
401 /**
402 * Fills the array with a specific value.
403 * @param {*} val - The value to fill the array with.
404 * @param {Array} A - The array to fill.
405 * @param {number} length - The number of elements to fill.
406 */
407 fill(val, A, length) {
408     for(var i = 0; i < length; i++) {
409         A[i] = val;
410     }
411 }
412
413 /**
414 * Creates an n-dimensional array filled with NaN.
415 * @param {...number} size - The size of each dimension.
416 * @returns {Array} The NaN-filled n-dimensional array.
417 */
418 NaNs(...size) {
419     return this.createFilledArray(...size, NaN);
420 }
421
422 /**
423 * Creates an n-dimensional array filled with zeros.
424 * @param {...number} size - The size of each dimension.
425 * @returns {Array} The zero-filled n-dimensional array.
426 */
427 zeros(...size) {
428     return this.createFilledArray(...size, 0);
429 }
430
431 /**
432 * Creates an n-dimensional array filled with ones.
433 * @param {...number} size - The size of each dimension.
434 * @returns {Array} The one-filled n-dimensional array.
435 */
436 ones(...size) {
437     return this.createFilledArray(...size, 1);
438 }
439
440 /**
441 * Creates a 2-dimensional identity matrix.
442 * @param {number} size - The size of the identity matrix.
443 * @returns {Array} The identity matrix as a 2D array.
444 */
445 eye(size) {
446     return this.diag(this.ones(size), size);
447 }
448
449 /**
450 * Scales an array by a scalar.
```

```

451  * @param {Array<number>} A - The array to scale.
452  * @param {number} s - The scalar value.
453  * @returns {Array<number>} The scaled array.
454  */
455  scale(A, s) {
456    var obj = this;
457    return A.map(function(x) {
458      if(Array.isArray(x)) {
459        return obj.scale(x, s);
460      } else {
461        return x * s;
462      }
463    });
464  }
465
466 /**
467  * Creates a linearly spaced vector.
468  * @param {number} x1 - The start value.
469  * @param {number} x2 - The end value.
470  * @param {number} [N=100] - The number of points.
471  * @returns {Array<number>} The linearly spaced vector.
472  */
473 linspace(x1, x2, rows = 100) {
474   var A = new Array(rows);
475   var dx = (x2-x1)/(rows-1);
476   for(var i = 0; i < rows; i++) {
477     A[i] = x1+dx*i;
478   }
479   return A;
480 }
481
482 /**
483  * Generates an array of numbers within a specified range.
484  * @param {...*} args - The start, end, and optional step for the range.
485  * @returns {Array<number>} An array containing numbers within the specified
486  * range.
487  */
488 range(...args) {
489   return [...this.jsl.env.math.range(...args, true).toArray()];
490 }
491
492 /**
493  * Generates a sequence of numbers from 'x1' to 'x2' with increments of 'dx'.
494  * @param {number} x1 - The starting value of the sequence.
495  * @param {number} x2 - The ending value of the sequence.
496  * @param {number} dx - The increment between values in the sequence.
497  * @returns {number[]} An array of numbers from 'x1' to 'x2' with step size
498  * 'dx'.
499  */
500 colon(x1, x2, dx) {
501   if(dx === 0) {
502     this.jsl.env.error('@colon: '+language.string(189));
503   }
504   const x = [];

```

```

503     const tolerance = 1e-14; // Tolerance for floating-point comparisons
504     let n = 0;
505     let xi = x1;
506     const increasing = dx > 0;
507
508     while(  

509         (increasing && xi <= x2 + tolerance) ||  

510         (!increasing && xi >= x2 - tolerance)  

511     ) {  

512         x.push(xi);  

513         xi = x1 + dx * ++n;  

514     }  

515
516     return x;
517 }
518
519 /**
520 * Applies a function to corresponding elements of one or more arrays.
521 * @param {Function} func - The function to apply to the elements.
522 * @param {...Array} arrays - One or more arrays to process.
523 * @returns {Array} A new array with the function applied to each
524   corresponding element.
525 */
526 elementWise(func, ...arrays) {
527     // Ensure all arrays have the same length
528     const length = arrays[0].length;
529
530     // Initialize the result array
531     const result = [];
532
533     // Apply the function element-wise
534     for(let i = 0; i < length; i++) {
535         // Get the corresponding elements from all arrays
536         const values = arrays.map(array => array[i]);
537
538         // Apply the function to the values and store in the result array
539         result.push(func(...values));
540     }
541
542     return result;
543 }
544
545 /**
546 * Applies a function to each element of an array or matrix along a
547   specified dimension.
548 * @param {Array|Matrix} A - The array or matrix to process.
549 * @param {number} dim - The dimension along which to apply the function.
550 * @param {Function} fun - The function to apply to each element.
551 * @returns {Array|Matrix} The result of applying the function to A.
552 */
553 arrayfun(A, dim, fun) {
554     return this.jsl.env.math.apply(A, dim, fun);
555 }
556
557 /**

```

```

556  * Performs element-wise division on two arrays or matrices.
557  * @param {Array|Matrix} x - The numerator array or matrix.
558  * @param {Array|Matrix} y - The denominator array or matrix.
559  * @returns {Array|Matrix} The result of the element-wise division.
560  */
561 divideEl(x, y) {
562   return this.jsl.env.math.dotDivide(x, y);
563 }
564
565 /**
566  * Performs element-wise multiplication on two arrays or matrices.
567  * @param {Array|Matrix} x - The first array or matrix.
568  * @param {Array|Matrix} y - The second array or matrix.
569  * @returns {Array|Matrix} The result of the element-wise multiplication.
570  */
571 multiplyEl(x, y) {
572   return this.jsl.env.math.dotMultiply(x, y);
573 }
574
575 /**
576  * Raises elements of an array or matrix to the power of elements in another
577  * array or matrix, element-wise.
578  * @param {Array|Matrix} x - The base array or matrix.
579  * @param {Array|Matrix|number} y - The exponent array, matrix, or scalar.
580  * @returns {Array|Matrix} The result of the element-wise exponentiation.
581  */
582 powEl(x, y) {
583   return this.jsl.env.math.dotPow(x, y);
584 }
585
586 /**
587  * Calculates the dot product of two vectors or matrices.
588  * @param {number[]} x - The first input vector or flat array.
589  * @param {number[]} y - The second input vector or flat array.
590  * @param {number} [cols] - Optional number of columns to reshape the inputs
591  *   into matrices.
592  * @returns {number | number[][][] } The resulting dot product, either as a
593  *   scalar or a matrix.
594  */
595 dot(x, y, cols) {
596   if(cols) {
597     var rows = x.length/cols;
598     var x_in = reshape(transpose(x, cols, rows), cols, rows);
599     var y_in = reshape(transpose(y, cols, rows), cols, rows);
600     return this.elementWise((a, b) => this.jsl.env.math.dot(a, b), x_in,
601       y_in);
602   } else {
603     return this.jsl.env.math.dot(x, y);
604   }
605 }
606
607 /**
608  * Determines if the element is greater than zero.
609  * @param {number} e - The element to check.
610  * @returns {boolean} True if greater than zero, otherwise false.

```

```

607  */
608  lZero(e) {
609    return e > 0;
610  }
611
612 /**
613 * Converts a boolean array into an array of indices where the values are ‘
614 * true’.
615 * @param {boolean[]} arr – The input array of boolean values.
616 * @returns {number[]} An array of indices corresponding to ‘true’ values in
617 * the input array.
618 */
619 b2i(arr) {
620   return arr.map((value, index) => value ? index : -1).filter(index => index
621   !== -1);
622 }
623
624 /**
625 * Calculates the average value of an array.
626 * @param {Array<number>} arr – The array to average.
627 * @returns {number} The average value.
628 */
629 average(arr) {
630   return arr.reduce(function(p, c) {
631     return p + c;
632   }, 0) / arr.length;
633 }
634
635 /**
636 * Calculates the Exponential Moving Average of the data.
637 * @param {number[]} data – The data array.
638 * @param {number} alpha – The smoothing factor between 0 and 1.
639 * @returns {number[]} The Exponential Moving Average of the data.
640 */
641 averageEM(data, alpha) {
642   var result = [];
643   var ema = data[0];
644   result.push(ema);
645   for(var i = 1; i < data.length; i++) {
646     ema = alpha * data[i] + (1 - alpha) * ema;
647     result.push(ema);
648   }
649   return result;
650 }
651
652 /**
653 * Applies a moving average filter to an input array while keeping the
654 * output array the same size.
655 * @param {number[]} inputArray – The array of numbers to filter.
656 * @param {number} windowSize – The size of the moving window (number of
657 * elements to average).
658 * @returns {number[]} The filtered array with the same length as the input
659 * array.
660 */
661 averageMoving(inputArray, windowSize) {

```

```

656     var result = [];
657     var len = inputArray.length;
658     var halfWindow = Math.floor(windowSize / 2);
659
660     for(var i = 0; i < len; i++) {
661         var start = i - halfWindow;
662         var end = i + halfWindow;
663
664         // Adjust the window if it goes beyond the array bounds
665         if(start < 0) start = 0;
666         if(end >= len) end = len - 1;
667
668         var sum = 0;
669         var count = 0;
670         for(var j = start; j <= end; j++) {
671             sum += inputArray[j];
672             count++;
673         }
674         result.push(sum / count);
675     }
676     return result;
677 }
678
679 /**
680 * Applies a moving average filter to an input array while keeping the
681 * output array the same size.
682 * @param {number[]} inputArray - The array of numbers to filter.
683 * @param {number} windowHeight - The size of the moving window (number of
684 * elements to average).
685 * @returns {number[]} The filtered array with the same length as the input
686 * array.
687 */
688 movmean(inputArray, windowHeight) {
689     return this.averageMoving(inputArray, windowHeight);
690 }
691
692 /**
693 * Determines if two arrays are equal.
694 * @param {Array} A1 - The first array.
695 * @param {Array} A2 - The second array.
696 * @returns {boolean} True if arrays are equal, otherwise false.
697 */
698 isequal(A1, A2) {
699     let n;
700     if((n = A1.length) != A2.length) return false;
701     for(let i = 0; i < n; i++) if(A1[i] !== A2[i]) return false;
702     return true;
703 }
704
705 /**
706 * Negates the boolean values in an array.
707 * @param {Array<boolean>} A - The array to negate.
708 * @returns {Array<boolean>} The negated array.
709 */
710 neg(A) {

```

```
708     var B = new Array(A.length);
709     for(var i = 0; i < A.length; i++) {
710         B[i] = !A[i];
711     }
712     return B;
713 }
714
715 /**
716 * Determines if all elements in an array evaluate to true.
717 * @param {Array<boolean>} A - The array to check.
718 * @returns {boolean} True if all elements are true, otherwise false.
719 */
720 all(A) {
721     let n = A.length;
722     for(let i = 0; i < n; i++) {
723         if(!A[i]) {
724             return false;
725         }
726     }
727     return true;
728 }
729
730 /**
731 * Determines if any element in an array evaluates to true.
732 * @param {Array<boolean>} A - The array to check.
733 * @returns {boolean} True if any element is true, otherwise false.
734 */
735 any(A) {
736     let n = A.length;
737     for(let i = 0; i < n; i++) {
738         if(A[i]) {
739             return true;
740         }
741     }
742     return false;
743 }
744
745 /**
746 * Checks if the array contains a specified item.
747 * @param {Array} arr - The array to search.
748 * @param {*} item - The item to search for.
749 * @returns {boolean} True if the item is found, otherwise false.
750 */
751 arrayContains(arr, item) {
752     if(!Array.prototype.indexOf) {
753         var i = arr.length;
754         while(i--) {
755             if(arr[i] === item) {
756                 return true;
757             }
758         }
759         return false;
760     }
761     return arr.indexOf(item) !== -1;
762 }
```

```

763
764  /**
765   * Checks if an array contains any duplicate elements.
766   * @param {Array} array - The array to check for duplicates.
767   * @returns {boolean} True if duplicates are found, false otherwise.
768   */
769 hasDuplicates(array) {
770   return new Set(array).size !== array.length;
771 }
772
773 /**
774  * Removes duplicate values from an array.
775  * @param {Array} arr - The array from which duplicates are to be removed.
776  * @returns {Array} An array containing only unique elements from the
777  * original array.
778  */
779 removeDuplicates(arr) {
780   return arr.filter(function(item, index) { return arr.indexOf(item) ===
781     index; });
782 }
783 /**
784  * Reverses the order of elements in each row of a matrix.
785  * @param {Array} array - The matrix array to flip horizontally.
786  * @returns {Array} The horizontally flipped matrix.
787  */
788 fliplr(array) {
789   return array.reverse();
790 }
791 /**
792  * Moves the first 'n' elements of the array to the end.
793  * @param {Array} array - The array to be modified.
794  * @param {number} n - The number of elements to move from the start to the
795  * end.
796  * @returns {Array} The modified array with the first 'n' elements moved to
797  * the end.
798 */
799 moveLR(array, n) {
800   var A = [...array];
801   if(A.length === 0 || n <= 0) return array;
802   n = n % A.length;
803   const elements_to_move = A.splice(0, n);
804   A.push(...elements_to_move);
805   return A;
806 }
807 /**
808  * Adds multiple operands, which can be either scalars or arrays.
809  * If multiple operands are arrays, they must be of the same length.
810  * @param {...number|Array<number>} args - The operands, scalar or arrays.
811  * @returns {number|Array<number>} The result of adding all operands.
812  * @throws {Error} If input types are invalid or arrays have different
813  * lengths.
814  */

```

```

813 plus(... args) {
814   if(args.length === 0) {
815     this.jsl.env.error('@plus: No arguments provided');
816   }
817
818   // Helper function to add two operands
819   const addTwo = (a, b) => {
820     if(Array.isArray(a) && Array.isArray(b)) {
821       if(a.length !== b.length) {
822         this.jsl.env.error('@plus: ' + language.string(176)); // Arrays have
823         different lengths
824       }
825       return a.map((val, idx) => this.plus(val, b[idx]));
826     } else if(Array.isArray(a) && typeof b === 'number') {
827       return a.map(val => this.plus(val, b));
828     } else if(typeof a === 'number' && Array.isArray(b)) {
829       return b.map(val => this.plus(a, val));
830     } else if(typeof a === 'number' && typeof b === 'number') {
831       return a + b;
832     } else {
833       this.jsl.env.error('@plus: ' + language.string(177)); // Invalid type
834     }
835     return false;
836   };
837
838   // Iterate through all arguments and accumulate the result
839   return args.reduce((acc, current) => addTwo(acc, current));
840 }
841 /**
842 * Adds multiple operands, which can be either scalars or arrays.
843 * If multiple operands are arrays, they must be of the same length.
844 * @param {...number|Array<number>} args - The operands, scalar or arrays.
845 * @returns {number|Array<number>} The result of adding all operands.
846 * @throws {Error} If input types are invalid or arrays have different
847 * lengths.
848 */
849 add(... args) {
850   return this.plus(... args);
851 }
852 /**
853 * Subtracts multiple operands from the first one, which can be either
854 * scalars or arrays.
855 * If multiple operands are arrays, they must be of the same length.
856 * @param {...number|Array<number>} args - The operands, scalar or arrays.
857 * @returns {number|Array<number>} The result of subtracting all subsequent
858 * operands from the first one.
859 * @throws {Error} If input types are invalid or arrays have different
860 * lengths.
861 */
862 minus(... args) {
863   if(args.length === 0) {
864     this.jsl.env.error('@minus: No arguments provided');
865   }

```

```

863
864 // Helper function to subtract two operands
865 const subtractTwo = (a, b) => {
866   if(Array.isArray(a) && Array.isArray(b)) {
867     if(a.length !== b.length) {
868       this.jsl.env.error('@minus: ' + language.string(176)); // Arrays
869       have different lengths
870     }
871     return a.map((val, idx) => this.minus(val, b[idx]));
872   } else if(Array.isArray(a) && typeof b === 'number') {
873     return a.map(val => this.minus(val, b));
874   } else if(typeof a === 'number' && Array.isArray(b)) {
875     // Subtracting an array from a scalar: scalar - array
876     return b.map(val => this.minus(a, val));
877   } else if(typeof a === 'number' && typeof b === 'number') {
878     return a - b;
879   } else {
880     this.jsl.env.error('@minus: ' + language.string(177)); // Invalid type
881   }
882   return false;
883 };
884
885 // The first argument is the initial value
886 let result = args[0];
887
888 // Iterate through the rest of the arguments and subtract each from the
889 // result
890 for(let i = 1; i < args.length; i++) {
891   result = subtractTwo(result, args[i]);
892 }
893
894 return result;
895 }
896 /**
897 * Subtracts multiple operands from the first one, which can be either
898 * scalars or arrays.
899 * If multiple operands are arrays, they must be of the same length.
900 * @param {...number|Array<number>} args - The operands, scalar or arrays.
901 * @returns {number|Array<number>} The result of subtracting all subsequent
902 * operands from the first one.
903 * @throws {Error} If input types are invalid or arrays have different
904 * lengths.
905 */
906 subtract(...args) {
907   return this.minus(...args);
908 }
909
910 /**
911 * Computes the cross product of two 3D vectors.
912 * @param {number[]} A - The first 3D vector.
913 * @param {number[]} B - The second 3D vector.
914 * @param {number} cols - The number of columns (typically 1 for single
915 * vectors).
916 * @returns {number[]} The cross product vector.

```

```

912  */
913 cross3D(A, B, cols = 1) {
914   var C = new Array(3 * cols).fill(0);
915   for(var j = 0; j < cols; j++) {
916     C[j * 3] = A[j * 3 + 1] * B[j * 3 + 2] - A[j * 3 + 2] * B[j * 3 + 1];
917     C[j * 3 + 1] = A[j * 3 + 2] * B[j * 3] - A[j * 3] * B[j * 3 + 2];
918     C[j * 3 + 2] = A[j * 3] * B[j * 3 + 1] - A[j * 3 + 1] * B[j * 3];
919   }
920   return C;
921 }
922
923 /**
924 * Computes the inverse of a matrix represented as a flat array.
925 * @param {number[]} A - The input matrix in a flat array format.
926 * @param {number} size - The number of rows and columns in the matrix.
927 * @returns {number[]|false} The inverted matrix as a flat array, or 'false'
928         if the matrix is non-invertible.
929 */
930 inv(A, size) {
931   var A_mat = reshape(A, size, size);
932   try {
933     return reshape(this.jsl.env.math.inv(A_mat), size*size, 1);
934   } catch {
935     return false;
936   }
937
938 /**
939 * Concatenates multiple matrices row-wise.
940 * @param {number} cols_C - The number of columns in the concatenated matrix
941 *
942 * @param {...Array} args - The matrices to concatenate.
943 * @returns {Array} The concatenated matrix.
944 */
945 concatRow(cols_C, ...args) {
946   var N = args.length;
947   var rows_C = args.reduce((a, e) => a += e.length, 0) / cols_C;
948
949   var C = new Array(rows_C * cols_C).fill(0);
950
951   var p = 0;
952   for(var j = 0; j < cols_C; j++) {
953     for(var k = 0; k < N; k++) {
954       var P = args[k].length / cols_C;
955       for(var i = 0; i < P; i++) {
956         C[p++] = args[k][j*P+i];
957       }
958     }
959   }
960   return C;
961 }
962 /**
963 * Concatenates multiple matrices column-wise.
964 * @param {number} rows_C - The number of rows in the concatenated matrix.

```

```

965  * @param {...Array} args - The matrices to concatenate.
966  * @returns {Array} The concatenated matrix.
967  */
968 concatCol(rows_C, ...args) {
969   return args.flat();
970 }
971
972 /**
973  * Concatenates multiple vectors.
974  * @param {...Array} args - The vectors to concatenate.
975  * @returns {Array} The concatenated vector.
976  */
977 concat(...args) {
978   return args.flat();
979 }
980
981 /**
982  * Repeats a row vector multiple times.
983  * @param {Array} A - The row vector to repeat.
984  * @param {number} rows - The number of times to repeat the row.
985  * @returns {Array} The repeated row matrix.
986  */
987 repRow(A, rows) {
988   var cols = A.length;
989   var C = new Array(cols * rows).fill(0);
990   for(var i = 0; i < cols; i++) {
991     for(var j = 0; j < rows; j++) {
992       C[i*rows+j] = A[i];
993     }
994   }
995   return C;
996 }
997
998 /**
999  * Repeats a column vector multiple times.
1000  * @param {Array} A - The column vector to repeat.
1001  * @param {number} cols - The number of times to repeat the column.
1002  * @returns {Array} The repeated column matrix.
1003  */
1004 repCol(A, cols) {
1005   var rows = A.length;
1006   var C = new Array(cols * rows).fill(0);
1007   for(var i = 0; i < cols; i++) {
1008     for(var j = 0; j < rows; j++) {
1009       C[i*rows+j] = A[j];
1010     }
1011   }
1012   return C;
1013 }
1014
1015 /**
1016  * Sums the elements of each row in a matrix.
1017  * @param {Array<number>} A - The matrix array.
1018  * @param {number} rows - The number of rows in the matrix.
1019  * @param {number} cols - The number of columns in the matrix.

```

```
1020 * @returns {number[]} An array containing the sum of each row.
1021 */
1022 sumRow(A, rows, cols) {
1023     var C = new Array(rows).fill(0);
1024     for(var i = 0; i < rows; i++) {
1025         for(var j = 0; j < cols; j++) {
1026             C[i] += A[i*cols+j];
1027         }
1028     }
1029     return C;
1030 }
1031 /**
1032 * Sums the elements of each column in a matrix.
1033 * @param {Array<number>} A - The matrix array.
1034 * @param {number} rows - The number of rows in the matrix.
1035 * @param {number} cols - The number of columns in the matrix.
1036 * @returns {number[]} An array containing the sum of each column.
1037 */
1038 sumCol(A, rows, cols) {
1039     var C = new Array(cols).fill(0);
1040     for(var j = 0; j < cols; j++) {
1041         for(var i = 0; i < rows; i++) {
1042             C[j] += A[j*rows+i];
1043         }
1044     }
1045     return C;
1046 }
1047 /**
1048 * Calculates the Euclidean norm of each row in a matrix.
1049 * @param {Array<number>} A - The matrix array.
1050 * @param {number} rows - The number of rows in the matrix.
1051 * @param {number} cols - The number of columns in the matrix.
1052 * @returns {number[]} An array containing the norm of each row.
1053 */
1054 normRow(A, rows, cols) {
1055     var C = new Array(rows).fill(0);
1056     for(var i = 0; i < rows; i++) {
1057         for(var j = 0; j < cols; j++) {
1058             C[i] += pow(A[i*cols+j], 2);
1059         }
1060         C[i] = sqrt(C[i]);
1061     }
1062     return C;
1063 }
1064 /**
1065 * Calculates the Euclidean norm of each column in a matrix.
1066 * @param {Array<number>} A - The matrix array.
1067 * @param {number} rows - The number of rows in the matrix.
1068 * @param {number} cols - The number of columns in the matrix.
1069 * @returns {number[]} An array containing the norm of each column.
1070 */
1071 normCol(A, rows, cols) {
```

```

1075     var C = new Array(cols).fill(0);
1076     for(var j = 0; j < cols; j++) {
1077         for(var i = 0; i < rows; i++) {
1078             C[j] += pow(A[j*rows+i], 2);
1079         }
1080         C[j] = sqrt(C[j]);
1081     }
1082     return C;
1083 }
1084
1085 /**
1086 * Transposes a matrix.
1087 * @param {Array<number>} A - The matrix array to transpose.
1088 * @param {number} rows - The number of rows in the original matrix.
1089 * @param {number} cols - The number of columns in the original matrix.
1090 * @returns {Array<number>} The transposed matrix array.
1091 */
1092 transpose(A, rows, cols) {
1093     var C = new Array(rows * cols).fill(0);
1094     for(var i = 0; i < rows; i++) {
1095         for(var j = 0; j < cols; j++) {
1096             C[j * rows + i] = A[i * cols + j];
1097         }
1098     }
1099     return C;
1100 }
1101
1102 /**
1103 * Multiplies two matrices.
1104 * @param {Array<number>} A - The first matrix array.
1105 * @param {Array<number>} B - The second matrix array.
1106 * @param {number} rows_A - The number of rows in matrix A.
1107 * @param {number} cols_A - The number of columns in matrix A.
1108 * @param {number} cols_B - The number of columns in matrix B.
1109 * @returns {Array<number>} The resulting matrix array after multiplication.
1110 */
1111 multiply(A, B, rows_A, cols_A, cols_B) {
1112     var C = new Array(rows_A * cols_B).fill(0);
1113     for(var i = 0; i < rows_A; i++) {
1114         for(var j = 0; j < cols_A; j++) {
1115             for(var k = 0; k < cols_B; k++) {
1116                 C[k * rows_A + i] += A[j * rows_A + i] * B[k * cols_A + j];
1117             }
1118         }
1119     }
1120     return C;
1121 }
1122
1123 /**
1124 * Performs element-wise dot product on columns of two matrices.
1125 * @param {Array<number>} A - The first matrix array.
1126 * @param {Array<number>} B - The second matrix array.
1127 * @param {number} rows - The number of rows in each matrix.
1128 * @param {number} cols - The number of columns in each matrix.
1129 * @returns {Array<number>} An array containing the dot product of each

```

```

    column.

1130  */
1131 dotColumn(A, B, rows, cols) {
1132   var C = new Array(cols).fill(0);
1133   for(var j = 0; j < cols; j++) {
1134     for(var i = 0; i < rows; i++) {
1135       C[j] += A[j * rows + i] * B[j * rows + i];
1136     }
1137   }
1138   return C;
1139 }

1140 /**
1141 * Creates a diagonal matrix from a given array.
1142 * @param {number[]} A - The array to form the diagonal.
1143 * @param {number} length - The size of the square matrix.
1144 * @returns {Array<number>} The diagonal matrix as a 1D array.
1145 */
1146 diag(A, length) {
1147   var B = new Array(length*length).fill(0);
1148   for(var i = 0; i < length; i++) {
1149     B[i * length + i] = A[i];
1150   }
1151   return B;
1152 }

1153 /**
1154 * Solves a linear system of equations using LU decomposition.
1155 * @param {Array<number>} A - The coefficient matrix.
1156 * @param {Array<number>} B - The constant terms.
1157 * @param {number} N - The size of the matrix (NxN).
1158 * @returns {Array<number>} The solution vector.
1159 */
1160 linsolve(A, B, N) {
1161   return this.reshape(this.jsl.env.math.lusolve(this.reshape(A, N, N), B),
1162                      1, N);
1163 }

1164 /**
1165 * Computes the reciprocal of each element in the array.
1166 * @param {Array<number>} A - The array of numbers.
1167 * @param {number} length - The number of elements in the array.
1168 * @returns {Array<number>} An array containing the reciprocals of the
1169 * original elements.
1170 */
1171 reciprocal(A, length) {
1172   var B = new Array(length).fill(0);
1173   for(var i = 0; i < length; i++) {
1174     B[i] = 1 / A[i];
1175   }
1176   return B;
1177 }

1178 /**
1179 * Reshapes an array into a new dimension.
1180 */
1181 *

```

```

1182  * @param {Array} A - The array to reshape.
1183  * @param {number} rows - The number of rows in the new shape.
1184  * @param {number} cols - The number of columns in the new shape.
1185  * @returns {Array} The reshaped array.
1186  * @throws {Error} If the total number of elements does not match.
1187  */
1188 reshape(A, rows, cols) {
1189   if(rows === _) {
1190     rows = A.length / cols;
1191   }
1192   if(cols === _) {
1193     cols = A.length / rows;
1194   }
1195   let flat_arr = [];
1196
1197   if(Array.isArray(A[0])) {
1198     var numRows = A.length;
1199     var numCols = A[0].length;
1200
1201     for(let j = 0; j < numCols; j++) {
1202       for(let i = 0; i < numRows; i++) {
1203         flat_arr.push(A[i][j]);
1204       }
1205     }
1206   } else {
1207     flat_arr = A.slice();
1208   }
1209
1210   if(flat_arr.length !== rows * cols) {
1211     this.jsl.env.error('@reshape: '+language.string(178));
1212   }
1213
1214   var reshaped = Array.from({ length: rows }, () => Array(cols));
1215
1216   for(let j = 0; j < cols; j++) {
1217     for(let i = 0; i < rows; i++) {
1218       var index = j * rows + i;
1219       reshaped[i][j] = flat_arr[index];
1220     }
1221   }
1222
1223   if(rows === 1 || cols === 1) {
1224     reshaped = reshaped.flat();
1225   }
1226   return reshaped;
1227 }
1228
1229 /**
1230  * Finds the maximum element in the array and its index, excluding NaN
1231  * values.
1232  * @param {number[]} A - The array to search.
1233  * @returns {Array} An array containing the max value and its index.
1234  * @throws {TypeError} If the input is not an array.
1235  * @throws {Error} If the array is empty or only contains NaN values.
1236  */

```

```

1236 maxi(A) {
1237   if (!Array.isArray(A)) {
1238     this.jsl.env.error('@maxi: '+language.string(190));
1239   }
1240
1241   const filtered_A = A.filter(num => !isNaN(num));
1242   if(filtered_A.length === 0) {
1243     this.jsl.env.error('@maxi: '+language.string(191));
1244   }
1245
1246   let max = filtered_A[0];
1247   let index = A.indexOf(max);
1248
1249   for(let i = 1; i < A.length; i++) {
1250     const current = A[i];
1251     if(!isNaN(current) && current > max) {
1252       max = current;
1253       index = i;
1254     }
1255   }
1256
1257   return [max, index];
1258 }
1259
1260 /**
1261 * Finds the minimum element in the array and its index, excluding NaN
1262 * values.
1263 * @param {number[]} A - The array to search.
1264 * @returns {Array} An array containing the min value and its index.
1265 * @throws {TypeError} If the input is not an array.
1266 * @throws {Error} If the array is empty or only contains NaN values.
1267 */
1268 mini(A) {
1269   if (!Array.isArray(A)) {
1270     this.jsl.env.error('@mini: '+language.string(190));
1271   }
1272
1273   const filtered_A = A.filter(num => !isNaN(num));
1274   if(filtered_A.length === 0) {
1275     this.jsl.env.error('@mini: '+language.string(191));
1276   }
1277
1278   let min = filtered_A[0];
1279   let index = A.indexOf(min);
1280
1281   for(let i = 1; i < A.length; i++) {
1282     const current = A[i];
1283     if(!isNaN(current) && current < min) {
1284       min = current;
1285       index = i;
1286     }
1287   }
1288
1289   return [min, index];
1290 }
```

```

1290
1291 /**
1292 * Sorts the array in ascending order with indices , excluding NaN values .
1293 * @param {number[]} A - The array to sort .
1294 * @returns {Array} An array containing the sorted values and their original
1295 * indices .
1296 * @throws {TypeError} If the input is not an array .
1297 */
1298 sorti(A) {
1299   if(!Array.isArray(A)) {
200   this.jsl.env.error('@sorti: '+language.string(190));
1300 }
1301
1302 // Create an array of indices and sort based on values in A ,
1303 // with NaNs sorted to the end .
1304 const indices = A.map((_, idx) => idx);
1305 indices.sort((i, j) => {
1306   if(isNaN(A[i]) && isNaN(A[j])) return 0;
1307   if(isNaN(A[i])) return 1;
1308   if(isNaN(A[j])) return -1;
1309   return A[i] - A[j];
1310 });
1311
1312 // Map the sorted indices to the sorted values
1313 const sorted_scores = indices.map(i => A[i]);
1314 return [sorted_scores , indices];
1315 }

1316 /**
1317 * Computes the weighted sum of two vectors and stores the result in the ‘
1318 * ret ‘ array .
1319 * @param {number[]} ret - The array to store the result .
1320 * @param {number} w1 - Weight for the first vector .
1321 * @param {number[]} v1 - The first vector .
1322 * @param {number} w2 - Weight for the second vector .
1323 * @param {number[]} v2 - The second vector .
1324 */
1325 weightedSum(ret , w1, v1, w2, v2) {
1326   for(let j = 0; j < ret.length; ++j) {
1327     ret[j] = w1 * v1[j] + w2 * v2[j];
1328   }
1329 }

1330 /**
1331 * Computes the consecutive differences of elements in an array .
1332 * @param {number[]} A - The input array of numbers .
1333 * @returns {number[]} An array containing the differences between
1334 * consecutive elements of the input array .
1335 */
1336 condiff(A) {
1337   return A.slice(1).map((num, i) => num - A[i]);
1338 }

1339 /**
1340 * Generates an array with random floating-point numbers within a specified

```

```

        range.
1342  * @param {number} l - The lower bound of the range.
1343  * @param {number} u - The upper bound of the range.
1344  * @param {number} rows - The number of rows.
1345  * @param {number} cols - The number of columns.
1346  * @param {Function} [randFun=Math.random] - The random function to use.
1347  * @returns {Array<number>} An array filled with random numbers within the
     specified range.
1348  */
1349 arrayRand(l, u, rows, cols, randFun) {
1350   if(!this.jsl.format.isFunction(randFun)) {
1351     randFun = Math.random;
1352   }
1353   return this.plus(this.repCol(l, cols), this.multiplyEl(repCol(minus(u, 1),
     cols), Array.from({length: rows * cols}, () => randFun())));
1354 }
1355
1356 /**
1357 * Generates an array with random integer numbers within a specified range.
1358 * @param {number[]} lu - An array containing the lower and upper bounds [
     lower, upper].
1359 * @param {number} rows - The number of rows.
1360 * @param {number} cols - The number of columns.
1361 * @param {Function} [randFun=Math.random] - The random function to use.
1362 * @returns {Array<number>} An array filled with random integers within the
     specified range.
1363 */
1364 arrayRandi(lu, rows, cols, randFun) {
1365   if(!this.jsl.format.isFunction(randFun)) {
1366     randFun = Math.random;
1367   }
1368   return Array.from({length: rows * cols}, () => Math.floor(randFun() * (
     lu[1] - lu[0] + 1) + lu[0]));
1369 }
1370
1371 /**
1372 * Normalizes a 3D vector.
1373 * @param {number[]} v - The vector to normalize.
1374 * @returns {number[]} The normalized vector.
1375 */
1376 normalizeVector(v) {
1377   var len = this.jsl.env.math.norm(v);
1378   if(len === 0) return [0, 0, 0];
1379   return [v[0]/len, v[1]/len, v[2]/len];
1380 }
1381
1382 /**
1383 * Computes the dot product of two 3D vectors.
1384 * @param {number[]} a - The first vector.
1385 * @param {number[]} b - The second vector.
1386 * @returns {number} The dot product of the vectors.
1387 */
1388 dotVector(a, b) {
1389   return a[0]*b[0] + a[1]*b[1] + a[2]*b[2];
1390 }

```

```

1391
1392 /**
1393 * Calculates the angle between two vectors .
1394 * @param {number[]} a - The first vector .
1395 * @param {number[]} b - The second vector .
1396 * @returns {number} The angle in radians between vectors a and b .
1397 */
1398 angleVectors(a, b) {
1399   return acos(dotVector(a, b)/(norm(a)*norm(b)));
1400 }
1401
1402 /**
1403 * Creates a skew-symmetric matrix from a 3D vector .
1404 * @param {number[]} v - The vector to skew .
1405 * @returns {number[]} The skew-symmetric matrix as a 1D array .
1406 */
1407 skewVector(v) {
1408   return [
1409     0, v[2], -v[1],
1410     -v[2], 0, v[0],
1411     v[1], -v[0], 0
1412   ];
1413 }
1414
1415 /**
1416 * Generates coordinate matrices from coordinate vectors for N dimensions .
1417 * @param {...number[]} args - The coordinate vectors .
1418 * @returns {Array} The coordinate grids for each dimension .
1419 */
1420 meshgrid(...args) {
1421   var obj = this;
1422   const ndim = args.length;
1423   const sizes = args.map(a => a.length);
1424   const grids = [];
1425
1426   // Initialize the grids with N-dimensional arrays
1427   for(let k = 0; k < ndim; k++) {
1428     grids[k] = this.createArray(...sizes);
1429   }
1430
1431   // Recursive function to fill in the grids
1432   function fillGrid(indices, dim) {
1433     if(dim === ndim) {
1434       for(let k = 0; k < ndim; k++) {
1435         obj.setValueAt(grids[k], indices, args[k][indices[(k + 1) % ndim]]);
1436       }
1437     } else {
1438       for(let i = 0; i < sizes[dim]; i++) {
1439         indices[dim] = i;
1440         fillGrid(indices, dim + 1);
1441       }
1442     }
1443   }
1444
1445   fillGrid(new Array(ndim), 0);

```

```

1446
1447     return grids;
1448 }
1449
1450 /**
1451 * Determines if two arrays are equal by comparing each element.
1452 * @param {Array} A1 - The first array to compare.
1453 * @param {Array} A2 - The second array to compare.
1454 * @returns {boolean} True if the arrays are equal, otherwise false.
1455 */
1456 isEqual(A1, A2) {
1457   let n;
1458   if((n = A1.length) != A2.length) return false;
1459   for(let i = 0; i < n; i++) if(A1[i] !== A2[i]) return false;
1460   return true;
1461 }
1462
1463 /**
1464 * Displays a matrix with a variable name.
1465 * @param {string} varname - The name of the variable.
1466 * @param {number[]} A - The matrix array.
1467 * @param {number} rows - The number of rows in the matrix.
1468 * @param {number} cols - The number of columns in the matrix.
1469 * @returns {string} The string representation of the matrix.
1470 */
1471 dispMatrix(varname, A, rows, cols) {
1472   var str = ' ' + varname + ' = [\n';
1473   for(var i = 0; i < rows; i++) {
1474     str += ' ';
1475     for(var j = 0; j < cols; j++) {
1476       str += num2str(A[j * rows + i], 8) + ' ';
1477     }
1478     str += '\n';
1479   }
1480   str += ']';
1481   this.jsl.env.disp(str);
1482   return str + '\n';
1483 }
1484
1485 /**
1486 * Displays a row vector with a variable name.
1487 * @param {string} varname - The name of the variable.
1488 * @param {number[]} A - The row vector array.
1489 * @param {number} length - The length of the row vector.
1490 * @returns {string} The string representation of the row vector.
1491 */
1492 dispRowVector(varname, A, length) {
1493   var str = ' ' + varname + ' = [ ';
1494   for(var i = 0; i < length; i++) {
1495     str += num2str(A[i], 8) + ' ';
1496   }
1497   str += ']';
1498   this.jsl.env.disp(str);
1499   return str + '\n';
1500 }
```

```

1501
1502 /**
1503 * Displays a column vector with a variable name.
1504 * @param {string} varname - The name of the variable.
1505 * @param {number[]} A - The column vector array.
1506 * @param {number} length - The length of the column vector.
1507 * @returns {string} The string representation of the column vector.
1508 */
1509 dispColumnVector(varname, A, length) {
1510   var str = ' ' + varname + ' = [\n';
1511   for(var j = 0; j < length; j++) {
1512     str += ' ' + num2str(A[j], 8) + '\n';
1513   }
1514   str += ']';
1515   this.jsl.env.disp(str);
1516   return str+'\n';
1517 }
1518 }
1519
1520 exports.PRDC_JSLAB_LIB_ARRAY = PRDC_JSLAB_LIB_ARRAY;

```

Listing 71 - array.js

```

1 /**
2  * @file JSLAB library basic submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 /**
9  * Class for JSLAB basic submodule.
10 */
11 class PRDC_JSLAB_LIB_BASIC {
12
13 /**
14  * Initializes a new instance of the PRDC_JSLAB_LIB_BASIC class.
15  * @param {Object} js1 The JSLAB application instance this submodule is part
16  * of.
17 */
18 constructor(js1) {
19   var obj = this;
20   this.jsl = js1;
21
22   this._docs = JSON.parse(this.jsl.env.readFileSync(app_path + '/docs/
23   documentation.json', 'utf8'));
24
25 /**
26  * Stores the result of the last evaluated expression.
27  * @type {any}
28  */
29 this.ans;
30
31 /**
32  * Clears all defined variables in the current context.
33  * @name clear

```

```
32      * @kind function
33      * @memberof PRDC_JSLAB_LIB_BASIC
34      */
35  Object.defineProperty(this.jsl.context, 'clear', { configurable: true, get
36      : this._clear });
37
38  /**
39   * Clears the console screen.
40   * @name clc
41   * @kind function
42   * @memberof PRDC_JSLAB_LIB_BASIC
43   */
44  Object.defineProperty(this.jsl.context, 'clc', { configurable: true, get :
45      this._clc });
46
47  /**
48   * Clears the console screen. Alias for 'clc'.
49   * @name cls
50   * @kind function
51   * @memberof PRDC_JSLAB_LIB_BASIC
52   */
53  Object.defineProperty(this.jsl.context, 'cls', { configurable: true, get :
54      this._clc });
55
56  /**
57   * Returns the current version of the JSLAB.
58   * @name version
59   * @kind member
60   * @memberof PRDC_JSLAB_LIB_BASIC
61   */
62  Object.defineProperty(this.jsl.context, 'version', { configurable: true,
63      get: this._version });
64
65  /**
66   * Returns the platform on which JSLAB is running.
67   * @name platform
68   * @kind member
69   * @memberof PRDC_JSLAB_LIB_BASIC
70   */
71  Object.defineProperty(this.jsl.context, 'platform', { configurable: true,
72      get: this._platform });
73
74  /**
75   * Returns the file name of the current JSLAB script.
76   * @name jsl_file_name
77   * @kind member
78   * @memberof PRDC_JSLAB_LIB_BASIC
79   */
80  Object.defineProperty(this.jsl.context, 'jsl_file_name', { configurable:
81      true, get: this.jslFileName });
82
83  /**
84   * Provides information about the current environment.
85   * @name info
86   * @kind function
```

```
81      * @memberof PRDC_JSLAB_LIB_BASIC
82      */
83  Object.defineProperty(this.jsl.context, 'info', { configurable: true, get:
84      this._info });
85
86      /**
87      * Accesses or modifies the user settings for JSLAB.
88      * @name settings
89      * @kind function
90      * @memberof PRDC_JSLAB_LIB_BASIC
91      */
92  Object.defineProperty(this.jsl.context, 'settings', { configurable: true,
93      get: this._settings });
94
95      /**
96      * Provides help information for JSLAB commands.
97      * @name cmd_help
98      * @kind function
99      * @memberof PRDC_JSLAB_LIB_BASIC
100     */
101 Object.defineProperty(this.jsl.context, 'cmd_help', { configurable: true,
102     get: this._cmd_help });
103
104     /**
105     * Accesses the code editor interface within JSLAB.
106     * @name editor
107     * @kind function
108     * @memberof PRDC_JSLAB_LIB_BASIC
109     */
110 Object.defineProperty(this.jsl.context, 'editor', { configurable: true,
111     get: this._editor });
112
113     /**
114     * Returns the current working directory.
115     * @name pwd
116     * @kind member
117     * @memberof PRDC_JSLAB_LIB_BASIC
118     */
119 Object.defineProperty(this.jsl.context, 'pwd', { configurable: true, get:
120     this._pwd });
121
122     /**
123     * Sets a breakpoint in the code for debugging.
124     * @name breakpoint
125     * @kind function
126     * @memberof PRDC_JSLAB_LIB_BASIC
127     */
128 Object.defineProperty(this.jsl.context, 'breakpoint', { configurable: true
129     , get: this._breakpoint });
130
131     /**
132     * Returns the current debug flag status.
133     * @name debug_flag
134     * @kind member
135     * @memberof PRDC_JSLAB_LIB_BASIC
```

```
130 */  
131 Object.defineProperty(this.jsl.context, 'debug_flag', { configurable: true  
132     , get: this._debug_flag});  
133 /**  
134 * Enables or disables debug mode.  
135 * @name debug  
136 * @kind member  
137 * @memberof PRDC_JSLAB_LIB_BASIC  
138 */  
139 Object.defineProperty(this.jsl.context, 'debug', { configurable: true, get  
140     : this._debug});  
141 /**  
142 * Pauses the execution of the current script.  
143 * @name pause  
144 * @kind function  
145 * @memberof PRDC_JSLAB_LIB_BASIC  
146 */  
147 Object.defineProperty(this.jsl.context, 'pause', { configurable: true, get  
148     : this._pause});  
149 /**  
150 * Sets a stop point in the script execution.  
151 * @name stoppoint  
152 * @kind function  
153 * @memberof PRDC_JSLAB_LIB_BASIC  
154 */  
155 Object.defineProperty(this.jsl.context, 'stoppoint', { configurable: true,  
156     get: this._stoppoint});  
157 /**  
158 * Sets a log point to record information during execution.  
159 * @name logpoint  
160 * @kind function  
161 * @memberof PRDC_JSLAB_LIB_BASIC  
162 */  
163 Object.defineProperty(this.jsl.context, 'logpoint', { configurable: true,  
164     get: this._logpoint});  
165 /**  
166 * Updates specific points in the script during execution.  
167 * @name updatepoint  
168 * @kind function  
169 * @memberof PRDC_JSLAB_LIB_BASIC  
170 */  
171 Object.defineProperty(this.jsl.context, 'updatepoint', { configurable:  
172     true, get: this._updatepoint});  
173 /**  
174 * Checks if the execution should stop based on conditions.  
175 * @name checkStop  
176 * @kind function  
177 * @memberof PRDC_JSLAB_LIB_BASIC  
178 */
```

```

179   Object.defineProperty(this.jsl.context, 'checkStop', { configurable: true,
180     get: this._checkStop });
181
182   /**
183    * Marks the endpoint of a script or process.
184    * @name endPoint
185    * @kind function
186    * @memberof PRDC_JSLAB_LIB_BASIC
187   */
188   Object.defineProperty(this.jsl.context, 'endPoint', { configurable: true,
189     get: this._endPoint });
190
191   /**
192    * Runs a script from a specified path, optionally focusing on specific
193    * lines and controlling output visibility.
194    * @param {string} script_path The path to the script to run.
195    * @param {Array<number>} lines An array of line numbers to run or focus on
196    * within the script.
197    * @param {boolean} [silent=false] Whether to suppress output from the
198    * script execution.
199    * @param {Boolean} [force_run=false] If true, forces the script to run even
200    * if stop conditions are met.
201   */
202   async run(script_path, lines, silent = false, force_run = false) {
203     if(this.jsl.env.pathIsAbsolute(script_path)) {
204       this.jsl.last_script_path = script_path;
205       this.jsl.last_script_lines = lines;
206       this.jsl.last_script_silent = silent;
207       if(!force_run && this.jsl.env.checkScriptDir(script_path)) {
208         return;
209       }
210     }
211     return await this.jsl.eval.runScript(script_path, lines, silent, force_run
212       );
213   }
214
215   /**
216    * Retrieves documentation in JSON format based on the provided name and
217    * type.
218    * @param {string} [name] - The name of the documentation item.
219    * @param {string} [type] - The type of the documentation (e.g., 'category')
220    *
221    * @returns {string|undefined} The JSON string of the documentation or
222    * undefined if not found.
223   */
224   helpToJSON(name, type) {
225     if(!name) {
226       return this.jsl.setDepthSafeStringify(this._docs, 4);
227     }
228
229     const parts = name.split('.');
230     if(parts.length === 2) {
231       const [libName, itemName] = parts;
232       const data = this._docs.lib[libName];
233     }
234   }

```

```

224     if (data.hasOwnProperty(itemName)) {
225         const result = data[itemName];
226         if (result) {
227             result.type = 'lib';
228             result.category = libName;
229             return this.jsl.setDepthSafeStringify(result, Infinity);
230         }
231     } else {
232         if (type === 'category') {
233             for (const category in this._docs) {
234                 const categoryData = this._docs[category];
235                 if (categoryData.hasOwnProperty(name)) {
236                     return this.jsl.setDepthSafeStringify(categoryData[name], 4);
237                 }
238             }
239         } else {
240             for (const category in this._docs.global) {
241                 const categoryData = this._docs.global[category];
242                 if (categoryData.hasOwnProperty(name)) {
243                     const result = categoryData[name];
244                     if (result) {
245                         result.type = 'global';
246                         result.category = category;
247                         return this.jsl.setDepthSafeStringify(result, Infinity);
248                     }
249                 }
250             }
251         }
252     }
253     this.jsl.env.error('@help: ' + language.string(218) + name);
254 }
255
256 /**
257 * Retrieves documentation based on the provided name and type.
258 * @param {string} name - The name of the documentation item.
259 * @param {string} type - The type of the documentation.
260 * @returns {string|undefined} The JSON string of the documentation or
261 *         undefined if not found.
262 */
263 help(name, type) {
264     return this.helpToJSON(name, type);
265 }
266
267 /**
268 * Retrieves documentation based on the provided name and type.
269 * @param {string} name - The name of the documentation item.
270 * @param {string} type - The type of the documentation.
271 * @returns {string|undefined} The JSON string of the documentation or
272 *         undefined if not found.
273 */
274 doc(name, type) {
275     return this.help(name, type);
276 }
```

```

277  /**
278   * Retrieves documentation based on the provided name and type.
279   * @param {string} name - The name of the documentation item.
280   * @param {string} type - The type of the documentation.
281   * @returns {string|undefined} The JSON string of the documentation or
282   *         undefined if not found.
283   */
284 documentation(name, type) {
285   return this.help(name, type);
286 }
287 /**
288  * Searches the documentation for methods that match all words in the given
289  * query, regardless of order.
290  * @param {string} query - The search query containing keywords to match
291  *                       within the documentation.
292  * @returns {Array<Object>} Array of matching documentation entries, each
293  *                       entry containing 'type' and 'category' properties.
294  */
295 helpSearch(query) {
296   var obj = this;
297   var query_words = query.toLowerCase().split(' ');
298   var results = {};
299   Object.keys(this._docs).forEach(function(type) {
300     Object.keys(obj._docs[type]).forEach(function(category) {
301       Object.keys(obj._docs[type][category]).forEach(function(member) {
302         var member_obj = obj._docs[type][category][member];
303         var str = JSON.stringify(member_obj).toLowerCase();
304         var match = query_words.every((word) => str.includes(word));
305         if (match) {
306           member_obj.type = type;
307           member_obj.category = category;
308           results[member] = member_obj;
309         }
310       });
311     });
312   });
313 /**
314  * Searches the documentation for methods that match all words in the given
315  * query, regardless of order.
316  * @param {string} query - The search query containing keywords to match
317  *                       within the documentation.
318  * @returns {Array<Object>} Array of matching documentation entries, each
319  *                       entry containing 'type' and 'category' properties.
320  */
321 docSearch(query) {
322   return this.helpSearch(query);
323 }
324 /**
325  * Searches the documentation for methods that match all words in the given
326  * query, regardless of order.
327 */

```

```

324     * @param {string} query - The search query containing keywords to match
325     * within the documentation.
326     * @returns {Array<Object>} Array of matching documentation entries , each
327     * entry containing 'type' and 'category' properties.
328     */
329   documentationSearch(query) {
330     return this.helpSearch(query);
331   }
332
333   /**
334    * Opens the source file and navigates to the specified line based on the
335    * provided name.
336    * @param {string} name - The name of the source to locate.
337    */
338   source(name) {
339     const parts = name.split('.');
340     if(parts.length === 2) {
341       const [libName, itemName] = parts;
342       const data = this._docs.lib[libName];
343       if(data.hasOwnProperty(itemName)) {
344         const result = data[itemName];
345         if(result) {
346           this.jsl.env.editor(app_path + '/js/sandbox/' + result.
347             source_filename, result.source_lineno);
348           return;
349         }
350       }
351     } else {
352       for(const category in this._docs.global) {
353         const categoryData = this._docs.global[category];
354         if(categoryData.hasOwnProperty(name)) {
355           const result = categoryData[name];
356           if(result) {
357             this.jsl.env.editor(app_path + '/js/sandbox/' + result.
358               source_filename, result.source_lineno);
359             return;
360           }
361         }
362       }
363     }
364     this.jsl.env.error('@source: ' + language.string(220) + name);
365   }
366
367   /**
368    * Retrieves the file name of the currently active JSL script.
369    * @returns {string} The file name of the JSL script.
370    */
371   jslFileName() {
372     return this.jsl.jsl_file_name;
373   }
374
375   /**
376    * Clears the application's local storage.
377    */
378   clearStorage() {

```

```

374     return this.jsl.env.clearStorage();
375 }
376
377 /**
378 * Saves a path to the application's list of saved paths.
379 * @param {string} new_path The path to save.
380 */
381 savePath(new_path) {
382     new_path = this.jsl.env.addPathSep(new_path);
383     var i = this.jsl.saved_paths.indexOf(new_path);
384     if(i < 0) {
385         this.jsl.saved_paths.push(new_path);
386     }
387     this.jsl.env.savePath(new_path);
388 }
389
390 /**
391 * Removes a previously saved path from the application's list of saved
392 * paths.
393 * @param {string} saved_path The path to remove.
394 */
395 removePath(saved_path) {
396     saved_path = this.jsl.env.addPathSep(saved_path);
397     var i = this.jsl.saved_paths.indexOf(saved_path);
398     if(i >= 0) {
399         this.jsl.saved_paths.splice(i, 1);
400     }
401     this.jsl.env.removePath(saved_path);
402 }
403
404 /**
405 * Opens the help documentation for CMD window.
406 */
407 _cmd_help() {
408     this.jsl.env.cmd_help();
409     this.jsl.no_ans = true;
410     this.jsl.ignore_output = true;
411 }
412
413 /**
414 * Displays application info.
415 */
416 _info() {
417     this.jsl.env.info();
418     this.jsl.no_ans = true;
419     this.jsl.ignore_output = true;
420 }
421
422 /**
423 * Opens the settings menu.
424 */
425 _settings() {
426     this.jsl.env.settings();
427     this.jsl.no_ans = true;
        this.jsl.ignore_output = true;

```

```

428     }
429
430     /**
431      * Retrieves the current working directory.
432      * @returns {string} The current working directory.
433      */
434     _pwd() {
435       return this.jsl.current_path;
436     }
437
438     /**
439      * Changes the current working directory to the specified path.
440      * @param {string} new_path The new path to set as the current working
441      *   directory.
442      */
443     cd(new_path) {
444       this.jsl.setPath(new_path);
445       this.jsl.env.cd(new_path);
446       this.jsl.no_ans = true;
447       this.jsl.ignore_output = true;
448     }
449     /**
450      * Lists the contents of the current directory.
451      * @returns {Array<string>} An array of filenames in the current directory.
452      */
453     _ls() {
454       return this.jsl.env.listFolderContents();
455     }
456
457     /**
458      * Retrieves the application version.
459      * @returns {string} The application version.
460      */
461     _version() {
462       return this.jsl.env.version;
463     }
464
465     /**
466      * Retrieves the operating system platform.
467      * @returns {string} The OS platform.
468      */
469     _platform() {
470       return this.jsl.env.platform;
471     }
472
473     /**
474      * Retrieves if the JSLAB application is currently in debug mode.
475      * @returns {boolean} True if the application is in debug mode; otherwise,
476      *   false.
477      */
478     _debug_flag() {
479       return this.jsl.env.debug;
500     }

```

```
481  /**
482   * Retrieves the current workspace.
483   * @returns {Object} The current workspace object.
484   */
485  workspace() {
486    return this.jsl.getWorkspace();
487  }
488
489  /**
490   * Updates the workspace display based on the current state.
491   */
492  updateWorkspace() {
493    this.jsl.env.updateWorkspace();
494    this.jsl.no_ans = true;
495    this.jsl.ignore_output = true;
496  }
497
498  /**
499   * Updates the file browser display based on the current state.
500   */
501  updateFileBrowser() {
502    this.jsl.env.updateFileBrowser();
503    this.jsl.no_ans = true;
504    this.jsl.ignore_output = true;
505  }
506
507  /**
508   * Clears the workspace.
509   */
510  clear() {
511    this.jsl.clear();
512  }
513
514  /**
515   * Clears the command window.
516   */
517  clc() {
518    this.jsl.env.clc();
519    this.jsl.no_ans = true;
520    this.jsl.ignore_output = true;
521  }
522
523  /**
524   * Displays an error message.
525   * @param {string} msg The error message to display.
526   */
527  error(msg) {
528    this.jsl.env.error(msg);
529    this.jsl.no_ans = true;
530    this.jsl.ignore_output = true;
531  }
532
533  /**
534   * Displays a general message.
535   * @param {string} msg The message to display.
536  }
```

```

536     */
537     disp(msg) {
538         this.jsl.env.disp(msg);
539         this.jsl.no_ans = true;
540         this.jsl.ignore_output = true;
541         return msg + "\n";
542     }
543
544 /**
545 * Displays a general message with monospaced font.
546 * @param {string} msg The message to display.
547 */
548 dispMonospaced(msg) {
549     this.jsl.env.dispMonospaced(msg);
550     this.jsl.no_ans = true;
551     this.jsl.ignore_output = true;
552     return msg + "\n";
553 }
554
555 /**
556 * Displays a warning message.
557 * @param {string} msg The warning message to display.
558 */
559 warn(msg) {
560     this.jsl.env.warn(msg);
561     this.jsl.no_ans = true;
562     this.jsl.ignore_output = true;
563 }
564
565 /**
566 * Opens a specified file in the editor or just opens the editor if no file
567 * is specified.
568 * @param {string} [filepath] The path to the file to open in the editor.
569 */
570 _editor(filepath) {
571     this.jsl.env.editor(filepath);
572     this.jsl.no_ans = true;
573     this.jsl.ignore_output = true;
574 }
575
576 /**
577 * Logs a point in the script for debugging purposes.
578 */
579 _logpoint() {
580     this.jsl.env.setWorkspace();
581     this.checkStopLoop();
582     this.jsl.onStopLoop();
583 }
584
585 /**
586 * Updates the workspace or environment at a specific point during script
587 * execution.
588 */
589 _updatepoint() {
590     this._logpoint();

```

```
589     }
590
591     /**
592      * Checks whether script execution should be stopped at the current point.
593      */
594     _checkStop() {
595       this._logpoint();
596     }
597
598     /**
599      * Introduces a breakpoint in the script, pausing execution and potentially
600      * allowing the user to continue based on their input.
601      */
602     _breakpoint() {
603       this._logpoint();
604
605       var [line, column, script] = this.jsl.eval.getExpressionPosition();
606       var ret = this.jsl.env.showMessageBox({ title: language.currentString(15),
607         message: language.currentString(216)+': '+line+', '+language.
608         currentString(113)+': '+column+' ('+script+') . '+language.
609         currentString(213), buttons: [language.currentString(214), language.
610         currentString(215)], cancelId: 0});
611
612       if(ret == 1) {
613         this.jsl.stop_loop = true;
614         this.jsl.onStopLoop();
615       }
616
617     /**
618      * Introduces a breakpoint in the script, pausing execution and potentially
619      * allowing the user to continue based on their input.
620      */
621     _debug() {
622       this._breakpoint();
623     }
624
625     /**
626      * Pauses script execution, providing an opportunity to inspect the current
627      * state or continue execution based on user input.
628      */
629     _pause() {
630       this._logpoint();
631
632       var [line, column, script] = this.jsl.eval.getExpressionPosition();
633       var ret = this.jsl.env.showMessageBox({ title: language.currentString(15),
634         message: language.currentString(217)+': '+line+', '+language.
635         currentString(113)+': '+column+' ('+script+') . '+language.
636         currentString(213), buttons: [language.currentString(214), language.
637         currentString(215)], cancelId: 0});
638
639       if(ret == 1) {
640         this.jsl.stop_loop = true;
641         this.jsl.onStopLoop();
642       }
643     }
644   }
```

```

633
634     /**
635      * Forces the script execution to stop at a designated point.
636      */
637     _stoppoint() {
638         this.jsl.stop_loop = true;
639         this.jsl.onStopLoop();
640     }
641
642     /**
643      * Marks an endpoint in the script, throwing an error to signify a critical
644      * stop or exit point in execution.
645      */
646     _endPoint() {
647         var [line, column, script] = this.jsl.eval.getExpressionPosition();
648         throw {name: 'JslabError', message: language.string(116) + ': ' + line + ', ' +
649               language.string(113) + ': ' + column + ' (' + script + ') .'};
650     }
651     /**
652      * Verifies if a loop within the script execution should be terminated,
653      * typically used to avoid infinite or lengthy unnecessary execution.
654      */
655     checkStopLoop() {
656         if(!this.jsl.stop_loop) {
657             this.jsl.stop_loop = this.jsl.env.checkStopLoop();
658         }
659         return this.jsl.stop_loop;
660     }
661     /**
662      * Opens a specified file in an editor or opens the editor to a default or
663      * previously specified file.
664      * @param {string} [filepath] - Path to the file to be opened in the editor.
665      */
666     edit(filepath) {
667         _editor(filepath);
668     }
669     /**
670      * Returns a list of all example scripts available within a predefined
671      * directory.
672      * @return {Array<string>} An array of paths to the example scripts.
673      */
674     getExamples() {
675         var obj = this;
676         return this.jsl.env.readdirSync(app_path + '/examples')
677             .filter(function(file) { return file.match(new RegExp('.\\jsl')) });
678         % -----
679         % End of document
680         % -----
681         \end{document}); }).map(function(i) { return folder + '\\\\' + i; });
682     }

```

```

683  /**
684   * Opens a specified example script in the editor window.
685   * @param {string} filename - Name of the example file to open.
686   */
687  openExample(filename) {
688    if (!this.jsl.env.pathIsAbsolute(filename)) {
689      filename = app_path + '\\examples\\' + filename;
690    }
691    this.edit(filename);
692  }

693 /**
694  * Opens examples folder in File Explorer
695  */
696  openExamplesFolder() {
697    this.jsl.env.openFolder(app_path + '\\examples');
698  }

700 /**
701  * Opens examples folder in File Explorer
702  */
703  goToExamplesFolder() {
704    this.jsl.env.cd(app_path + '\\examples');
705  }

706 /**
707  * Displays a synchronous message box to the user and waits for their
708  * response.
709  * @param {Object} options - Configuration options for the message box.
710  * @return {number} The index of the button clicked by the user.
711  */
712  showMessageBox(options) {
713    return this.jsl.env.showMessageBox(options);
714  }

715 /**
716  * Saves specified variables to a JSON file.
717  * @param {string} file_path - Path where the JSON file will be saved.
718  * @param {...string} args - Variables to save. If 'all' is specified, saves
719  *   all available variables.
720  */
721  save(file_path, ...args) {
722    var obj = this;
723    var vars = {};
724    if (args.length == 0 || (args.length == 1 && args[0] == 'all')) {
725      var properties = this.jsl.getWorkspaceProperties();
726      properties.forEach(function(property) {
727        vars[property] = obj.jsl.context[property];
728      });
729    } else {
730      args.forEach(function(property) {
731        if (obj.jsl.context.hasOwnProperty(property)) {
732          vars[property] = obj.jsl.context[property];
733        }
734      });
735    }
  }

```

```
736     }
737     if (!this.jsl.env.pathIsAbsolute(file_path)) {
738         file_path = this.jsl.env.pathJoin(this.jsl.current_path, file_path);
739     }
740     var flag = this.jsl.env.writeFileSync(file_path, JSON.stringify(vars));
741     if(flag === false) {
742         this.jsl.env.error('@save: '+language.string(117)+': ' + file_path);
743     }
744 }
745
746 /**
747 * Loads variables from a specified JSON file into the specified scope or
748 * the default script context.
749 * If an error occurs during file reading or parsing, it logs an error
750 * message.
751 * @param {...*} args - A single filename or a scope and filename to specify
752 * where to load the variables.
753 */
754 load(...args) {
755     var obj = this;
756     var scope = obj.jsl.context;
757     var file_path;
758     if(args.length > 1) {
759         scope = args[0];
760         file_path = args[1];
761     } else {
762         file_path = args[0];
763     }
764
765     file_path = this.jsl.pathResolve(file_path);
766     if(file_path) {
767         try {
768             var vars = JSON.parse(readFile(file_path));
769             Object.keys(vars).forEach(function(property) {
770                 scope[property] = vars[property];
771             });
772         } catch(err) {
773             this.jsl.env.error('@load: '+language.string(118)+'.');
774         }
775     }
776
777 /**
778 * Executes a system shell command.
779 * @param {...*} arg - The command and its arguments to be executed.
780 * @return {string} The output of the executed command.
781 */
782 system(...arg) {
783     try {
784         return this.jsl.env.execSync(...arg).toString();
785     } catch(err) {
786         return err.message + ', command output: \n' + err.stdout.toString();
787     }
788 }
```

```

788  /**
789   * Retrieves completion suggestions based on the current context and input .
790   * @param {Array} data - Data containing the start of the string to complete
791   * , context , and keywords.
792   * @return {Array<string>} An array of completion suggestions .
793   */
794   getCompletions(data) {
795     var start = data[0];
796     var context = data[1];
797     var keywords = data[2];
798     var found = [] , global = window;
799
800     function maybeAdd(str) {
801       if(str.lastIndexOf(start , 0) == 0 && !arrayContains(found , str)) found .
802       push(str);
803     }
804     function gatherCompletions(obj) {
805       if(typeof obj == "string") forEach(("charAt charCodeAt indexOf
806       lastIndexOf substring substr slice trim trimLeft trimRight " +
807         "toUpperCase toLowerCase split concat match replace
808         "search").split(" ") , maybeAdd);
809       else if(obj instanceof Array) forEach("length concat join splice push
810         pop shift unshift slice reverse sort indexOf " +
811         "lastIndexOf every some filter forEach map reduce
812         "reduceRight").split(" ") , maybeAdd);
813       else if(obj instanceof Function) forEach("prototype apply call bind".
814         "split(" ") , maybeAdd);
815       if(!Object .getOwnPropertyNames || !Object .getPrototypeOf) {
816         for(var name in obj) maybeAdd(name);
817       } else {
818         for(var o = obj; o; o = Object .getPrototypeOf(o))
819           Object .getOwnPropertyNames(o).forEach(maybeAdd);
820       }
821     }
822
823     if(context && context.length) {
824       context = JSON .parse(context);
825       // If this is a property , see if it belongs to some object we can
826       // find in the current environment .
827       var obj = context.pop() , base;
828       if(obj .type && obj .type.indexOf("variable") == 0) {
829         base = base || global [obj .string];
830       } else if(obj .type == "string") {
831         base = "";
832       } else if(obj .type == "atom") {
833         base = 1;
834       } else if(obj .type == "function") {
835         if(global .jQuery != null && (obj .string == '
836
837 % -----
838 % End of document
839 % -----
840 \end{document} || obj .string == 'jQuery') &&
841         (typeof global .jQuery == 'function')) {
842           base = global .jQuery();
843

```

```

836         } else if(global._ != null && (obj.string == '_') && (typeof global.
837             _ == 'function')) {
838             base = global._();
839         }
840         while(base != null && context.length)
841             base = base[context.pop().string]; // switch base to variable
842         if(base != null) gatherCompletions(base);
843     } else {
844         gatherCompletions(global);
845         forEach(keywords, maybeAdd);
846     }
847     if(found.length) {
848         found.sort(function(a, b) {
849             return a.length - b.length || a.localeCompare(b);
850         });
851     }
852     return found;
853 }
854
855 /**
856 * Retrieves an object by matching a specific property value.
857 * @param {Object} obj - The object to search through.
858 * @param {string} prop - The property name to match.
859 * @param {*} val - The value to match against the property.
860 * @returns {Object|null} The found object with key and value, or null if
861 * not found.
862 */
863 getObjectByProp(obj, prop, val) {
864     const key = Object.keys(obj).find(function(key) { return obj[key][prop]
865         === val; });
866     return key ? { key, value: obj[key] } : null;
867 }
868
869 /**
870 * Retrieves multiple objects from a parent object by matching a specific
871 * property value.
872 * @param {Object} obj - The parent object to search through.
873 * @param {string} prop - The property name to match.
874 * @param {*} val - The value to match against the property.
875 * @returns {Object} An object containing all matched key-value pairs.
876 */
877 getObjectsByProp(obj, prop, val) {
878     return Object.fromEntries(Object.entries(obj)
879         .filter(function([key, value]) { return value[prop] === val; }));
880 }
881
882 /**
883 * Compares two strings lexicographically.
884 * @param {string} x - The first string.
885 * @param {string} y - The second string.
886 * @return {number} The result of the comparison.
887 */
888 strcmp(x, y) {
889     return this.jsl.env.math.compareText(x, y);

```

```

887     }
888
889     /**
890      * Unloads a previously required module from the cache.
891      * @param {string} module - The module to unrequire.
892     */
893     unrequire(module) {
894       module = this.jsl.pathResolve(module);
895       if(this.jsl.required_modules.includes(module)) {
896         var name = require.resolve(module);
897         if(name) {
898           delete require.cache[name];
899         }
900         this.jsl.array.removeElementByValue(this.jsl.required_modules, module);
901       }
902     }
903
904     /**
905      * Resets app.
906     */
907     resetApp() {
908       this.jsl.env.resetApp();
909     }
910
911     /**
912      * Resets the sandbox environment to its initial state.
913     */
914     resetSandbox() {
915       this.jsl.env.resetSandbox();
916     }
917
918     /**
919      * Opens the developer tools for the sandbox environment in the current
920      * context.
921      * @returns {void}
922     */
923     openDevTools() {
924       this.jsl.env.openSandboxDevTools();
925     }
926
927     /**
928      * Compiles a N-API module located at the specified path.
929      * @param {string} path - The path to the N-API module.
930      * @param {boolean} [show_output=true] - Whether to show output in the
931      * command window.
932      * @return {Array} An array containing the result of the compilation and
933      * targets.
934     */
935     compileNapi(path, show_output = false) {
936       var result = false;
937       var obj = this;
938       if(typeof path == 'string') {
939         path = this.jsl.pathResolve(path);
940       }
941       if(!path) {

```

```

939   var options = {
940     title: language.currentString(141),
941     buttonLabel: language.currentString(142),
942     properties: [ 'openDirectory' ],
943   };
944   path = this.jsl.env.showOpenDialogSync(options);
945   if(path === undefined) {
946     this.jsl.env.error('@compileNapi: '+language.string(119)+'.');
947     return false;
948   } else {
949     path = path[0];
950   }
951 }
952 path = this.jsl.env.addPathSep(path);
953
954 if(this.jsl.env.rmSync(path+'build/Release/', false) === false) {
955   this.jsl.env.error('@compileNapi: '+language.string(171));
956 }
957
958 var binding_file_path = path + 'binding.gyp';
959 if(this.jsl.env.checkFile(binding_file_path)) {
960   var targets = [];
961   try {
962     var binding_file_data = JSON.parse(this.jsl.env.readFileSync(
963       binding_file_path).toString());
964   } catch(err) {
965     this.jsl.env.error('@compileNapi: '+language.string(120)+'.');
966     return false;
967   }
968   binding_file_data.targets.forEach(function(target) {
969     targets.push(path + 'build/Release/' + target.target_name + '.node');
970   });
971   if(targets.length > 0) {
972     var exe = this.jsl.env.exe_path;
973     var node_gyp_path = app_path + '/node_modules/node-gyp/bin/node-gyp.js';
974     var npm_path = this.jsl.env.pathJoin(app_path, 'node_modules', 'npm',
975       'bin', 'npm-cli.js');
976     var msg = this.system('set ELECTRON_RUN_AS_NODE=1 & "' + exe + '" "' +
977       npm_path + '" cache clean --force & "' + exe + '" "' + npm_path +
978       '" install --build-from-source=false & "' + exe + '" "' +
979       node_gyp_path + '" rebuild --target='+process.version+' 2>&1', {
980         cwd: path, shell: false });
981
982     if(msg.endsWith('gyp info ok \n')) {
983       if(show_output) {
984         this.jsl.env.disp(msg.replaceAll('\n', '<br>'));
985       }
986       result = true;
987     } else if (!msg.endsWith('gyp ERR! not ok \n')) {
988       this.jsl.env.error('@compileNapi: '+language.string(121)+'. '+msg.
989        replaceAll('\n', '<br>'));
990     } else {
991       this.jsl.env.error('@compileNapi: '+language.string(170)+'. '+msg.
992        .replaceAll('\n', '<br>'));

```

```

985     }
986
987     if( result ) {
988         return [ result , targets ];
989     } else {
990         return [ result , undefined ];
991     }
992 } else {
993     this . jsl . env . error ( '@compileNapi: '+language . string (122)+'. '+msg .
994     replaceAll ( '\n' , '<br>' ) );
995     return [ result , undefined ];
996 }
997 } else {
998     this . jsl . env . error ( '@compileNapi: '+language . string (123)+'. '+msg .
999     replaceAll ( '\n' , '<br>' ) );
1000     return [ result , undefined ];
1001 }
1002 /**
1003 * Installs a module located at the specified path.
1004 * @param {string} path - The path to the module.
1005 * @param {boolean} [show_output=true] - Whether to show output in the
1006 * command window.
1007 */
1008 installModule (path , show_output = false) {
1009     if (typeof path == 'string') {
1010         path = this . jsl . pathResolve (path);
1011     }
1012     if (!path) {
1013         var options = {
1014             title: language . currentString (141) ,
1015             buttonLabel: language . currentString (142) ,
1016             properties: [ 'openDirectory' ],
1017         };
1018         path = this . jsl . env . showOpenDialogSync (options);
1019         if (path == undefined) {
1020             this . jsl . env . error ( '@installModule: '+language . string (119)+'. ' );
1021         } else {
1022             path = path [0];
1023         }
1024     }
1025     path = this . jsl . env . addPathSep (path);
1026
1027     var exe = this . jsl . env . exe_path;
1028     var npm_path = this . jsl . env . pathJoin (app_path , 'node_modules' , 'npm' , 'bin'
1029     , 'npm- cli . js' );
1030     var msg = this . system ( 'set ELECTRON_RUN_AS_NODE=1 & "' + exe + '" "' +
1031     npm_path + '" install 2>&1' , { cwd: path , shell: false } );
1032
1033     if (!msg . includes ( '\nnpm error' )) {
1034         if (show_output) {
1035             this . jsl . env . disp (msg);
1036         }
1037     } else {
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2497
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2598
2599
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2698
2699
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2798
2799
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2898
2899
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2959
2960
2961
296
```

```

1035     this.jsl.env.error('@installModule: '+msg.replaceAll('\n', '<br>'));
1036   }
1037 }
1038 /**
1039 * Registers an object for cleanup with a specified cleanup function.
1040 * @param {Object} obj - The object to be registered for cleanup.
1041 * @param {Function} fun - The function to execute during cleanup.
1042 */
1043 addForCleanup(...args) {
1044   this.jsl.addForCleanup(...args);
1045 }
1046 }
1047 }
1048
1049 exports.PRDC_JSLAB_LIB_BASIC = PRDC_JSLAB_LIB_BASIC;

```

Listing 72 - basic.js

```

1 /**
2  * @file JSLAB library color submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 /**
9  * Class for JSLAB color submodule.
10 */
11 class PRDC_JSLAB_LIB_COLOR {
12
13 /**
14  * Constructs a color submodule object with access to JSLAB's color
15  * functions.
16  * @constructor
17  * @param {Object} jsl - Reference to the main JSLAB object.
18 */
19 constructor(jsl) {
20   var obj = this;
21   this.jsl = jsl;
22
23 /**
24  * Color order
25  * @type {Array}
26 */
27 this.colororder = ['#0072BD', '#D95319', '#EDB120', '#7E2F8E', '#77AC30',
28 '#4DBEEE', '#A2142F'];
29
30 var colors = {
31   aliceblue: 'rgb(240, 248, 255)',
32   antiquewhite: 'rgb(250, 235, 215)',
33   aqua: 'rgb(0, 255, 255)',
34   aquamarine: 'rgb(127, 255, 212)',
35   azure: 'rgb(240, 255, 255)',
36   beige: 'rgb(245, 245, 220)',
37   bisque: 'rgb(255, 228, 196)',
38   black: 'rgb(0, 0, 0)',
39   blanchedalmond: 'rgb(255, 235, 205)'
40 };
41
42 /**
43  * Color names
44  * @type {Object}
45 */
46 this.colors = colors;
47
48 /**
49  * Color name to hex conversion
50  * @param {String} name - Color name
51  * @return {String} Hex color code
52 */
53 this.nameToHex = function(name) {
54   return this.colors[name];
55 }
56
57 /**
58  * Color hex to name conversion
59  * @param {String} hex - Hex color code
60  * @return {String} Color name
61 */
62 this.hexToName = function(hex) {
63   for (var name in this.colors) {
64     if (this.colors[name] === hex) {
65       return name;
66     }
67   }
68   return '';
69 }
70
71 /**
72  * Color name to RGB conversion
73  * @param {String} name - Color name
74  * @return {Object} RGB object
75 */
76 this.nameToRGB = function(name) {
77   var color = this.colors[name];
78   var r = color[0];
79   var g = color[1];
80   var b = color[2];
81   return {r: r, g: g, b: b};
82 }
83
84 /**
85  * Color RGB to name conversion
86  * @param {Object} rgb - RGB object
87  * @return {String} Color name
88 */
89 this.rgbToName = function(rgb) {
90   for (var name in this.colors) {
91     if (this.colors[name].r === rgb.r && this.colors[name].g === rgb.g && this.colors[name].b === rgb.b) {
92       return name;
93     }
94   }
95   return '';
96 }
97
98 /**
99  * Color name to HSL conversion
100 * @param {String} name - Color name
101 * @return {Object} HSL object
102 */
103 this.nameToHSL = function(name) {
104   var color = this.colors[name];
105   var r = color[0];
106   var g = color[1];
107   var b = color[2];
108   var h = 0;
109   var s = 0;
110   var l = 0;
111   var max = Math.max(r, g, b);
112   var min = Math.min(r, g, b);
113   var d = max - min;
114   if (d === 0) {
115     h = 0;
116     s = 0;
117     l = (max + min) / 2;
118   } else {
119     s = d / max;
120     l = (max + min) / 2;
121     if (max === r) {
122       h = (g - b) / d;
123     } else if (max === g) {
124       h = 2 + (b - r) / d;
125     } else if (max === b) {
126       h = 4 + (r - g) / d;
127     }
128     h = h * 60;
129   }
130   return {h: h, s: s, l: l};
131 }
132
133 /**
134  * Color HSL to name conversion
135  * @param {Object} hsl - HSL object
136  * @return {String} Color name
137 */
138 this.hslToName = function(hsl) {
139   for (var name in this.colors) {
140     if (this.colors[name].h === hsl.h && this.colors[name].s === hsl.s && this.colors[name].l === hsl.l) {
141       return name;
142     }
143   }
144   return '';
145 }
146
147 /**
148  * Color name to CMYK conversion
149  * @param {String} name - Color name
150  * @return {Object} CMYK object
151 */
152 this.nameToCMYK = function(name) {
153   var color = this.colors[name];
154   var r = color[0];
155   var g = color[1];
156   var b = color[2];
157   var c = 1 - r / 255;
158   var m = 1 - g / 255;
159   var y = 1 - b / 255;
160   var k = Math.min(c, m, y);
161   c = (c - k) / (1 - k);
162   m = (m - k) / (1 - k);
163   y = (y - k) / (1 - k);
164   return {c: c, m: m, y: y, k: k};
165 }
166
167 /**
168  * Color CMYK to name conversion
169  * @param {Object} cmyk - CMYK object
170  * @return {String} Color name
171 */
172 this.cmykToName = function(cmyk) {
173   for (var name in this.colors) {
174     if (this.colors[name].c === cmyk.c && this.colors[name].m === cmyk.m && this.colors[name].y === cmyk.y && this.colors[name].k === cmyk.k) {
175       return name;
176     }
177   }
178   return '';
179 }
180
181 /**
182  * Color name to LAB conversion
183  * @param {String} name - Color name
184  * @return {Object} LAB object
185 */
186 this.nameToLAB = function(name) {
187   var color = this.colors[name];
188   var r = color[0];
189   var g = color[1];
190   var b = color[2];
191   var l = (r + g + b) / 3;
192   var a = ((r - l) * 4.5) / 255;
193   var b = ((g - l) * 4.5) / 255;
194   return {l: l, a: a, b: b};
195 }
196
197 /**
198  * Color LAB to name conversion
199  * @param {Object} lab - LAB object
200  * @return {String} Color name
201 */
202 this.labToName = function(lab) {
203   for (var name in this.colors) {
204     if (this.colors[name].l === lab.l && this.colors[name].a === lab.a && this.colors[name].b === lab.b) {
205       return name;
206     }
207   }
208   return '';
209 }
210
211 /**
212  * Color name to HSI conversion
213  * @param {String} name - Color name
214  * @return {Object} HSI object
215 */
216 this.nameToHSI = function(name) {
217   var color = this.colors[name];
218   var r = color[0];
219   var g = color[1];
220   var b = color[2];
221   var h = 0;
222   var s = 0;
223   var i = 0;
224   var max = Math.max(r, g, b);
225   var min = Math.min(r, g, b);
226   var d = max - min;
227   if (d === 0) {
228     h = 0;
229     s = 0;
230     i = (max + min) / 2;
231   } else {
232     s = d / max;
233     i = (max + min) / 2;
234     if (max === r) {
235       h = (g - b) / d;
236     } else if (max === g) {
237       h = 2 + (b - r) / d;
238     } else if (max === b) {
239       h = 4 + (r - g) / d;
240     }
241     h = h * 60;
242   }
243   return {h: h, s: s, i: i};
244 }
245
246 /**
247  * Color HSI to name conversion
248  * @param {Object} hsi - HSI object
249  * @return {String} Color name
250 */
251 this.hsiToName = function(hsi) {
252   for (var name in this.colors) {
253     if (this.colors[name].h === hsi.h && this.colors[name].s === hsi.s && this.colors[name].i === hsi.i) {
254       return name;
255     }
256   }
257   return '';
258 }
259
260 /**
261  * Color name to XYZ conversion
262  * @param {String} name - Color name
263  * @return {Object} XYZ object
264 */
265 this.nameToXYZ = function(name) {
266   var color = this.colors[name];
267   var r = color[0];
268   var g = color[1];
269   var b = color[2];
270   var x = (r * 0.4124) + (g * 0.3586) + (b * 0.1896);
271   var y = (r * 0.2126) + (g * 0.7169) + (b * 0.0721);
272   var z = (r * 0.0193) + (g * 0.1192) + (b * 0.9505);
273   return {x: x, y: y, z: z};
274 }
275
276 /**
277  * Color XYZ to name conversion
278  * @param {Object} xyz - XYZ object
279  * @return {String} Color name
280 */
281 this.xyzToName = function(xyz) {
282   for (var name in this.colors) {
283     if (this.colors[name].x === xyz.x && this.colors[name].y === xyz.y && this.colors[name].z === xyz.z) {
284       return name;
285     }
286   }
287   return '';
288 }
289
290 /**
291  * Color name to L*a*b* conversion
292  * @param {String} name - Color name
293  * @return {Object} L*a*b* object
294 */
295 this.nameToLab = function(name) {
296   var color = this.colors[name];
297   var r = color[0];
298   var g = color[1];
299   var b = color[2];
300   var l = (r * 0.4124) + (g * 0.3586) + (b * 0.1896);
301   var a = (r * 0.2126) - (g * 0.7169) - (b * 0.0721);
302   var b = (r * 0.0193) - (g * 0.1192) + (b * 0.9505);
303   return {l: l, a: a, b: b};
304 }
305
306 /**
307  * Color L*a*b* to name conversion
308  * @param {Object} lab - L*a*b* object
309  * @return {String} Color name
310 */
311 this.labToName = function(lab) {
312   for (var name in this.colors) {
313     if (this.colors[name].l === lab.l && this.colors[name].a === lab.a && this.colors[name].b === lab.b) {
314       return name;
315     }
316   }
317   return '';
318 }
319
320 /**
321  * Color name to Luv conversion
322  * @param {String} name - Color name
323  * @return {Object} Luv object
324 */
325 this.nameToLuv = function(name) {
326   var color = this.colors[name];
327   var r = color[0];
328   var g = color[1];
329   var b = color[2];
330   var l = (r * 0.4124) + (g * 0.3586) + (b * 0.1896);
331   var u = (r * 0.2126) - (g * 0.7169) - (b * 0.0721);
332   var v = (r * 0.0193) - (g * 0.1192) + (b * 0.9505);
333   return {l: l, u: u, v: v};
334 }
335
336 /**
337  * Color Luv to name conversion
338  * @param {Object} luv - Luv object
339  * @return {String} Color name
340 */
341 this.luvToName = function(luv) {
342   for (var name in this.colors) {
343     if (this.colors[name].l === luv.l && this.colors[name].u === luv.u && this.colors[name].v === luv.v) {
344       return name;
345     }
346   }
347   return '';
348 }
349
350 /**
351  * Color name to HCL conversion
352  * @param {String} name - Color name
353  * @return {Object} HCL object
354 */
355 this.nameToHcl = function(name) {
356   var color = this.colors[name];
357   var r = color[0];
358   var g = color[1];
359   var b = color[2];
360   var h = 0;
361   var c = 0;
362   var l = 0;
363   var max = Math.max(r, g, b);
364   var min = Math.min(r, g, b);
365   var d = max - min;
366   if (d === 0) {
367     h = 0;
368     c = 0;
369     l = (max + min) / 2;
370   } else {
371     c = d / max;
372     l = (max + min) / 2;
373     if (max === r) {
374       h = (g - b) / d;
375     } else if (max === g) {
376       h = 2 + (b - r) / d;
377     } else if (max === b) {
378       h = 4 + (r - g) / d;
379     }
380     h = h * 60;
381   }
382   return {h: h, c: c, l: l};
383 }
384
385 /**
386  * Color HCL to name conversion
387  * @param {Object} hcl - HCL object
388  * @return {String} Color name
389 */
390 this.hclToName = function(hcl) {
391   for (var name in this.colors) {
392     if (this.colors[name].h === hcl.h && this.colors[name].c === hcl.c && this.colors[name].l === hcl.l) {
393       return name;
394     }
395   }
396   return '';
397 }
398
399 /**
400  * Color name to HSI conversion
401  * @param {String} name - Color name
402  * @return {Object} HSI object
403 */
404 this.nameToHSI = function(name) {
405   var color = this.colors[name];
406   var r = color[0];
407   var g = color[1];
408   var b = color[2];
409   var h = 0;
410   var s = 0;
411   var i = 0;
412   var max = Math.max(r, g, b);
413   var min = Math.min(r, g, b);
414   var d = max - min;
415   if (d === 0) {
416     h = 0;
417     s = 0;
418     i = (max + min) / 2;
419   } else {
420     s = d / max;
421     i = (max + min) / 2;
422     if (max === r) {
423       h = (g - b) / d;
424     } else if (max === g) {
425       h = 2 + (b - r) / d;
426     } else if (max === b) {
427       h = 4 + (r - g) / d;
428     }
429     h = h * 60;
430   }
431   return {h: h, s: s, i: i};
432 }
433
434 /**
435  * Color HSI to name conversion
436  * @param {Object} hsi - HSI object
437  * @return {String} Color name
438 */
439 this.hsiToName = function(hsi) {
440   for (var name in this.colors) {
441     if (this.colors[name].h === hsi.h && this.colors[name].s === hsi.s && this.colors[name].i === hsi.i) {
442       return name;
443     }
444   }
445   return '';
446 }
447
448 /**
449  * Color name to HSL conversion
450  * @param {String} name - Color name
451  * @return {Object} HSL object
452 */
453 this.nameToHsl = function(name) {
454   var color = this.colors[name];
455   var r = color[0];
456   var g = color[1];
457   var b = color[2];
458   var h = 0;
459   var s = 0;
460   var l = 0;
461   var max = Math.max(r, g, b);
462   var min = Math.min(r, g, b);
463   var d = max - min;
464   if (d === 0) {
465     h = 0;
466     s = 0;
467     l = (max + min) / 2;
468   } else {
469     s = d / max;
470     l = (max + min) / 2;
471     if (max === r) {
472       h = (g - b) / d;
473     } else if (max === g) {
474       h = 2 + (b - r) / d;
475     } else if (max === b) {
476       h = 4 + (r - g) / d;
477     }
478     h = h * 60;
479   }
480   return {h: h, s: s, l: l};
481 }
482
483 /**
484  * Color HSL to name conversion
485  * @param {Object} hsl - HSL object
486  * @return {String} Color name
487 */
488 this.hslToName = function(hsl) {
489   for (var name in this.colors) {
490     if (this.colors[name].h === hsl.h && this.colors[name].s === hsl.s && this.colors[name].l === hsl.l) {
491       return name;
492     }
493   }
494   return '';
495 }
496
497 /**
498  * Color name to HCH conversion
499  * @param {String} name - Color name
500  * @return {Object} HCH object
501 */
502 this.nameToHch = function(name) {
503   var color = this.colors[name];
504   var r = color[0];
505   var g = color[1];
506   var b = color[2];
507   var h = 0;
508   var c = 0;
509   var hch = 0;
510   var max = Math.max(r, g, b);
511   var min = Math.min(r, g, b);
512   var d = max - min;
513   if (d === 0) {
514     h = 0;
515     c = 0;
516     hch = (max + min) / 2;
517   } else {
518     c = d / max;
519     hch = (max + min) / 2;
520     if (max === r) {
521       h = (g - b) / d;
522     } else if (max === g) {
523       h = 2 + (b - r) / d;
524     } else if (max === b) {
525       h = 4 + (r - g) / d;
526     }
527     h = h * 60;
528   }
529   return {h: h, c: c, hch: hch};
530 }
531
532 /**
533  * Color HCH to name conversion
534  * @param {Object} hch - HCH object
535  * @return {String} Color name
536 */
537 this.hchToName = function(hch) {
538   for (var name in this.colors) {
539     if (this.colors[name].h === hch.h && this.colors[name].c === hch.c && this.colors[name].hch === hch.hch) {
540       return name;
541     }
542   }
543   return '';
544 }
545
546 /**
547  * Color name to HCL conversion
548  * @param {String} name - Color name
549  * @return {Object} HCL object
550 */
551 this.nameToHcl = function(name) {
552   var color = this.colors[name];
553   var r = color[0];
554   var g = color[1];
555   var b = color[2];
556   var h = 0;
557   var c = 0;
558   var l = 0;
559   var max = Math.max(r, g, b);
560   var min = Math.min(r, g, b);
561   var d = max - min;
562   if (d === 0) {
563     h = 0;
564     c = 0;
565     l = (max + min) / 2;
566   } else {
567     c = d / max;
568     l = (max + min) / 2;
569     if (max === r) {
570       h = (g - b) / d;
571     } else if (max === g) {
572       h = 2 + (b - r) / d;
573     } else if (max === b) {
574       h = 4 + (r - g) / d;
575     }
576     h = h * 60;
577   }
578   return {h: h, c: c, l: l};
579 }
580
581 /**
582  * Color HCL to name conversion
583  * @param {Object} hcl - HCL object
584  * @return {String} Color name
585 */
586 this.hclToName = function(hcl) {
587   for (var name in this.colors) {
588     if (this.colors[name].h === hcl.h && this.colors[name].c === hcl.c && this.colors[name].l === hcl.l) {
589       return name;
590     }
591   }
592   return '';
593 }
594
595 /**
596  * Color name to HSI conversion
597  * @param {String} name - Color name
598  * @return {Object} HSI object
599 */
600 this.nameToHSI = function(name) {
601   var color = this.colors[name];
602   var r = color[0];
603   var g = color[1];
604   var b = color[2];
605   var h = 0;
606   var s = 0;
607   var i = 0;
608   var max = Math.max(r, g, b);
609   var min = Math.min(r, g, b);
610   var d = max - min;
611   if (d === 0) {
612     h = 0;
613     s = 0;
614     i = (max + min) / 2;
615   } else {
616     s = d / max;
617     i = (max + min) / 2;
618     if (max === r) {
619       h = (g - b) / d;
620     } else if (max === g) {
621       h = 2 + (b - r) / d;
622     } else if (max === b) {
623       h = 4 + (r - g) / d;
624     }
625     h = h * 60;
626   }
627   return {h: h, s: s, i: i};
628 }
629
630 /**
631  * Color HSI to name conversion
632  * @param {Object} hsi - HSI object
633  * @return {String} Color name
634 */
635 this.hsiToName = function(hsi) {
636   for (var name in this.colors) {
637     if (this.colors[name].h === hsi.h && this.colors[name].s === hsi.s && this.colors[name].i === hsi.i) {
638       return name;
639     }
640   }
641   return '';
642 }
643
644 /**
645  * Color name to HSL conversion
646  * @param {String} name - Color name
647  * @return {Object} HSL object
648 */
649 this.nameToHsl = function(name) {
650   var color = this.colors[name];
651   var r = color[0];
652   var g = color[1];
653   var b = color[2];
654   var h = 0;
655   var s = 0;
656   var l = 0;
657   var max = Math.max(r, g, b);
658   var min = Math.min(r, g, b);
659   var d = max - min;
660   if (d === 0) {
661     h = 0;
662     s = 0;
663     l = (max + min) / 2;
664   } else {
665     s = d / max;
666     l = (max + min) / 2;
667     if (max === r) {
668       h = (g - b) / d;
669     } else if (max === g) {
670       h = 2 + (b - r) / d;
671     } else if (max === b) {
672       h = 4 + (r - g) / d;
673     }
674     h = h * 60;
675   }
676   return {h: h, s: s, l: l};
677 }
678
679 /**
680  * Color HSL to name conversion
681  * @param {Object} hsl - HSL object
682  * @return {String} Color name
683 */
684 this.hslToName = function(hsl) {
685   for (var name in this.colors) {
686     if (this.colors[name].h === hsl.h && this.colors[name].s === hsl.s && this.colors[name].l === hsl.l) {
687       return name;
688     }
689   }
690   return '';
691 }
692
693 /**
694  * Color name to HCH conversion
695  * @param {String} name - Color name
696  * @return {Object} HCH object
697 */
698 this.nameToHch = function(name) {
699   var color = this.colors[name];
700   var r = color[0];
701   var g = color[1];
702   var b = color[2];
703   var h = 0;
704   var c = 0;
705   var hch = 0;
706   var max = Math.max(r, g, b);
707   var min = Math.min(r, g, b);
708   var d = max - min;
709   if (d === 0) {
710     h = 0;
711     c = 0;
712     hch = (max + min) / 2;
713   } else {
714     c = d / max;
715     hch = (max + min) / 2;
716     if (max === r) {
717       h = (g - b) / d;
718     } else if (max === g) {
719       h = 2 + (b - r) / d;
720     } else if (max === b) {
721       h = 4 + (r - g) / d;
722     }
723     h = h * 60;
724   }
725   return {h: h, c: c, hch: hch};
726 }
727
728 /**
729  * Color HCH to name conversion
730  * @param {Object} hch - HCH object
731  * @return {String} Color name
732 */
733 this.hchToName = function(hch) {
734   for (var name in this.colors) {
735     if (this.colors[name].h === hch.h && this.colors[name].c === hch.c && this.colors[name].hch === hch.hch) {
736       return name;
737     }
738   }
739   return '';
740 }
741
742 /**
743  * Color name to HCL conversion
744  * @param {String} name - Color name
745  * @return {Object} HCL object
746 */
747 this.nameToHcl = function(name) {
748   var color = this.colors[name];
749   var r = color[0];
750   var g = color[1];
751   var b = color[2];
752   var h = 0;
753   var c = 0;
754   var l = 0;
755   var max = Math.max(r, g, b);
756   var min = Math.min(r, g, b);
757   var d = max - min;
758   if (d === 0) {
759     h = 0;
760     c = 0;
761     l = (max + min) / 2;
762   } else {
763     c = d / max;
764     l = (max + min) / 2;
765     if (max === r) {
766       h = (g - b) / d;
767     } else if (max === g) {
768       h = 2 + (b - r) / d;
769     } else if (max === b) {
770       h = 4 + (r - g) / d;
771     }
772     h = h * 60;
773   }
774   return {h: h, c: c, l: l};
775 }
776
777 /**
778  * Color HCL to name conversion
779  * @param {Object} hcl - HCL object
780  * @return {String} Color name
781 */
782 this.hclToName = function(hcl) {
783   for (var name in this.colors) {
784     if (this.colors[name].h === hcl.h && this.colors[name].c === hcl.c && this.colors[name].l === hcl.l) {
785       return name;
786     }
787   }
788   return '';
789 }
790
791 /**
792  * Color name to HSI conversion
793  * @param {String} name - Color name
794  * @return {Object} HSI object
795 */
796 this.nameToHSI = function(name) {
797   var color = this.colors[name];
798   var r = color[0];
799   var g = color[1];
800   var b = color[2];
801   var h = 0;
802   var s = 0;
803   var i = 0;
804   var max = Math.max(r, g, b);
805   var min = Math.min(r, g, b);
806   var d = max - min;
807   if (d === 0) {
808     h = 0;
809     s = 0;
810     i = (max + min) / 2;
811   } else {
812     s = d / max;
813     i = (max + min) / 2;
814     if (max === r) {
815       h = (g - b) / d;
816     } else if (max === g) {
817       h = 2 + (b - r) / d;
818     } else if (max === b) {
819       h = 4 + (r - g) / d;
820     }
821     h = h * 60;
822   }
823   return {h: h, s: s, i: i};
824 }
825
826 /**
827  * Color HSI to name conversion
828  * @param {Object} hsi - HSI object
829  * @return {String} Color name
830 */
831 this.hsiToName = function(hsi) {
832   for (var name in this.colors) {
833     if (this.colors[name].h === hsi.h && this.colors[name].s === hsi.s && this.colors[name].i === hsi.i) {
834       return name;
835     }
836   }
837   return '';
838 }
839
840 /**
841  * Color name to HSL conversion
842  * @param {String} name - Color name
843  * @return {Object} HSL object
844 */
845 this.nameToHsl = function(name) {
846   var color = this.colors[name];
847   var r = color[0];
848   var g = color[1];
849   var b = color[2];
850   var h = 0;
851   var s = 0;
852   var l = 0;
853   var max = Math.max(r, g, b);
854   var min = Math.min(r, g, b);
855   var d = max - min;
856   if (d === 0) {
857     h = 0;
858     s = 0;
859     l = (max + min) / 2;
860   } else {
861     s = d / max;
862     l = (max + min) / 2;
863     if (max === r) {
864       h = (g - b) / d;
865     } else if (max === g) {
866       h = 2 + (b - r) / d;
867     } else if (max === b) {
868       h = 4 + (r - g) / d;
869     }
870     h = h * 60;
871   }
872   return {h: h, s: s, l: l};
873 }
874
875 /**
876  * Color HSL to name conversion
877  * @param {Object} hsl - HSL object
878  * @return {String} Color name
879 */
880 this.hslToName = function(hsl) {
881   for (var name in this.colors) {
882     if (this.colors[name].h === hsl.h && this.colors[name].s === hsl.s && this.colors[name].l === hsl.l) {
883       return name;
884     }
885   }
886   return '';
887 }
888
889 /**
890  * Color name to HCH conversion
891  * @param {String} name - Color name
892  * @return {Object} HCH object
893 */
894 this.nameToHch = function(name) {
895   var color = this.colors[name];
896   var r = color[0];
897   var g = color[1];
898   var b = color[2];
899   var h = 0;
900   var c = 0;
901   var hch = 0;
902   var max = Math.max(r, g, b);
903   var min = Math.min(r, g, b);
904   var d = max - min;
905   if (d === 0) {
906     h = 0;
907     c = 0;
908     hch = (max + min) / 2;
909   } else {
910     c = d / max;
911     hch = (max + min) / 2;
912     if (max === r) {
913       h = (g - b) / d;
914     } else if (max === g) {
915       h = 2 + (b - r) / d;
916     } else if (max === b) {
917       h = 4 + (r - g) / d;
918     }
919     h = h * 60;
920   }
921   return {h: h, c: c, hch: hch};
922 }
923
924 /**
925  * Color HCH to name conversion
926  * @param {Object} hch - HCH object
927  * @return {String} Color name
928 */
929 this.hchToName = function(hch) {
930   for (var name in this.colors) {
931     if (this.colors[name].h === hch.h && this.colors[name].c === hch.c && this.colors[name].hch === hch.hch) {
932       return name;
933     }
934   }
935   return '';
936 }
937
938 /**
939  * Color name to HCL conversion
940  * @param {String} name - Color name
941  * @return {Object} HCL object
942 */
943 this.nameToHcl = function(name) {
944   var color = this.colors[name];
945   var r = color[0];
946   var g = color[1];
947   var b = color[2];
948   var h = 0;
949   var c = 0;
950   var l = 0;
951   var max = Math.max(r, g, b);
952   var min = Math.min(r, g, b);
953   var d = max - min;
954   if (d === 0) {
955     h = 0;
956     c = 0;
957     l = (max + min) / 2;

```

```
37 blanchedalmond: 'rgb(255, 235, 205)',  
38 blue: 'rgb(0, 0, 255)',  
39 blueviolet: 'rgb(138, 43, 226)',  
40 brown: 'rgb(165, 42, 42)',  
41 burlywood: 'rgb(222, 184, 135)',  
42 cadetblue: 'rgb(95, 158, 160)',  
43 chartreuse: 'rgb(127, 255, 0)',  
44 chocolate: 'rgb(210, 105, 30)',  
45 coral: 'rgb(255, 127, 80)',  
46 cornflowerblue: 'rgb(100, 149, 237)',  
47 cornsilk: 'rgb(255, 248, 220)',  
48 crimson: 'rgb(220, 20, 60)',  
49 cyan: 'rgb(0, 255, 255)',  
50 darkblue: 'rgb(0, 0, 139)',  
51 darkcyan: 'rgb(0, 139, 139)',  
52 darkgoldenrod: 'rgb(184, 134, 11)',  
53 darkgray: 'rgb(169, 169, 169)',  
54 darkgreen: 'rgb(0, 100, 0)',  
55 darkgrey: 'rgb(169, 169, 169)',  
56 darkkhaki: 'rgb(189, 183, 107)',  
57 darkmagenta: 'rgb(139, 0, 139)',  
58 darkolivegreen: 'rgb(85, 107, 47)',  
59 darkorange: 'rgb(255, 140, 0)',  
60 darkorchid: 'rgb(153, 50, 204)',  
61 darkred: 'rgb(139, 0, 0)',  
62 darksalmon: 'rgb(233, 150, 122)',  
63 darkseagreen: 'rgb(143, 188, 143)',  
64 darkslateblue: 'rgb(72, 61, 139)',  
65 darkslategray: 'rgb(47, 79, 79)',  
66 darkslategrey: 'rgb(47, 79, 79)',  
67 darkturquoise: 'rgb(0, 206, 209)',  
68 darkviolet: 'rgb(148, 0, 211)',  
69 deeppink: 'rgb(255, 20, 147)',  
70 deepskyblue: 'rgb(0, 191, 255)',  
71 dimgray: 'rgb(105, 105, 105)',  
72 dimgrey: 'rgb(105, 105, 105)',  
73 dodgerblue: 'rgb(30, 144, 255)',  
74 firebrick: 'rgb(178, 34, 34)',  
75 floralwhite: 'rgb(255, 250, 240)',  
76 forestgreen: 'rgb(34, 139, 34)',  
77 fuchsia: 'rgb(255, 0, 255)',  
78 gainsboro: 'rgb(220, 220, 220)',  
79 ghostwhite: 'rgb(248, 248, 255)',  
80 gold: 'rgb(255, 215, 0)',  
81 goldenrod: 'rgb(218, 165, 32)',  
82 gray: 'rgb(128, 128, 128)',  
83 green: 'rgb(0, 128, 0)',  
84 greenyellow: 'rgb(173, 255, 47)',  
85 grey: 'rgb(128, 128, 128)',  
86 honeydew: 'rgb(240, 255, 240)',  
87 hotpink: 'rgb(255, 105, 180)',  
88 indianred: 'rgb(205, 92, 92)',  
89 indigo: 'rgb(75, 0, 130)',  
90 ivory: 'rgb(255, 255, 240)',  
91 khaki: 'rgb(240, 230, 140)',
```

```
92     lavender: 'rgb(230, 230, 250)',  
93     lavenderblush: 'rgb(255, 240, 245)',  
94     lawngreen: 'rgb(124, 252, 0)',  
95     lemonchiffon: 'rgb(255, 250, 205)',  
96     lightblue: 'rgb(173, 216, 230)',  
97     lightcoral: 'rgb(240, 128, 128)',  
98     lightcyan: 'rgb(224, 255, 255)',  
99     lightgoldenrodyellow: 'rgb(250, 250, 210)',  
100    lightgray: 'rgb(211, 211, 211)',  
101    lightgreen: 'rgb(144, 238, 144)',  
102    lightgrey: 'rgb(211, 211, 211)',  
103    lightpink: 'rgb(255, 182, 193)',  
104    lightsalmon: 'rgb(255, 160, 122)',  
105    lightseagreen: 'rgb(32, 178, 170)',  
106    lightskyblue: 'rgb(135, 206, 250)',  
107    lightslategray: 'rgb(119, 136, 153)',  
108    lightslategrey: 'rgb(119, 136, 153)',  
109    lightsteelblue: 'rgb(176, 196, 222)',  
110    lightyellow: 'rgb(255, 255, 224)',  
111    lime: 'rgb(0, 255, 0)',  
112    limegreen: 'rgb(50, 205, 50)',  
113    linen: 'rgb(250, 240, 230)',  
114    magenta: 'rgb(255, 0, 255)',  
115    maroon: 'rgb(128, 0, 0)',  
116    mediumaquamarine: 'rgb(102, 205, 170)',  
117    mediumblue: 'rgb(0, 0, 205)',  
118    mediumorchid: 'rgb(186, 85, 211)',  
119    mediumpurple: 'rgb(147, 112, 219)',  
120    mediumseagreen: 'rgb(60, 179, 113)',  
121    mediumslateblue: 'rgb(123, 104, 238)',  
122    mediumspringgreen: 'rgb(0, 250, 154)',  
123    mediumturquoise: 'rgb(72, 209, 204)',  
124    mediumvioletred: 'rgb(199, 21, 133)',  
125    midnightblue: 'rgb(25, 25, 112)',  
126    mintcream: 'rgb(245, 255, 250)',  
127    mistyrose: 'rgb(255, 228, 225)',  
128    moccasin: 'rgb(255, 228, 181)',  
129    navajowhite: 'rgb(255, 222, 173)',  
130    navy: 'rgb(0, 0, 128)',  
131    oldlace: 'rgb(253, 245, 230)',  
132    olive: 'rgb(128, 128, 0)',  
133    olivedrab: 'rgb(107, 142, 35)',  
134    orange: 'rgb(255, 165, 0)',  
135    orangered: 'rgb(255, 69, 0)',  
136    orchid: 'rgb(218, 112, 214)',  
137    palegoldenrod: 'rgb(238, 232, 170)',  
138    palegreen: 'rgb(152, 251, 152)',  
139    paleturquoise: 'rgb(175, 238, 238)',  
140    palevioletred: 'rgb(219, 112, 147)',  
141    papayawhip: 'rgb(255, 239, 213)',  
142    peachpuff: 'rgb(255, 218, 185)',  
143    peru: 'rgb(205, 133, 63)',  
144    pink: 'rgb(255, 192, 203)',  
145    plum: 'rgb(221, 160, 221)',  
146    powderblue: 'rgb(176, 224, 230)',
```

```

147   purple: 'rgb(128, 0, 128)',  

148   red: 'rgb(255, 0, 0)',  

149   rosybrown: 'rgb(188, 143, 143)',  

150   royalblue: 'rgb(65, 105, 225)',  

151   saddlebrown: 'rgb(139, 69, 19)',  

152   salmon: 'rgb(250, 128, 114)',  

153   sandybrown: 'rgb(244, 164, 96)',  

154   seagreen: 'rgb(46, 139, 87)',  

155   seashell: 'rgb(255, 245, 238)',  

156   sienna: 'rgb(160, 82, 45)',  

157   silver: 'rgb(192, 192, 192)',  

158   skyblue: 'rgb(135, 206, 235)',  

159   slateblue: 'rgb(106, 90, 205)',  

160   slategray: 'rgb(112, 128, 144)',  

161   slategrey: 'rgb(112, 128, 144)',  

162   snow: 'rgb(255, 250, 250)',  

163   springgreen: 'rgb(0, 255, 127)',  

164   steelblue: 'rgb(70, 130, 180)',  

165   tan: 'rgb(210, 180, 140)',  

166   teal: 'rgb(0, 128, 128)',  

167   thistle: 'rgb(216, 191, 216)',  

168   tomato: 'rgb(255, 99, 71)',  

169   turquoise: 'rgb(64, 224, 208)',  

170   violet: 'rgb(238, 130, 238)',  

171   wheat: 'rgb(245, 222, 179)',  

172   white: 'rgb(255, 255, 255)',  

173   whitesmoke: 'rgb(245, 245, 245)',  

174   yellow: 'rgb(255, 255, 0)',  

175   yellowgreen: 'rgb(154, 205, 50)'  

176 };
177  

178 this.RE_RGB = /^rgb\((\s*(\d+)\s*,\s*(\d+)\s*,\s*(\d+)\s*)\)$/;  

179 this.RE_RGBA = /^rgba\((\s*(\d+)\s*,\s*(\d+)\s*,\s*(\d+)\s*,\s*([\d\.]+)\s*\s*)\$/;  

180 this.RE_HSL = /^hsl\((\s*([\d\.]+)\s*,\s*([\d\.]+)%\s*,\s*([\d\.]+)%\s*)\$/;  

181 this.RE_HSLA = /^hsla\((\s*([\d\.]+)\s*,\s*([\d\.]+)%\s*,\s*([\d\.]+)%\s*,\s*([\d\.]+)\s*)\$/;  

182 this.RE_HEX = /^([0-9a-fA-F]{2})([0-9a-fA-F]{2})([0-9a-fA-F]{2})$/; // 6 digit  

183  

184 // Global object
185 var Color = {
186   create: function(str) {
187     str = str.replace(/\s*#/|\s*$/g, '');
188     str = str.toLowerCase();
189     if (KEYWORDS[str]) str = KEYWORDS[str];
190
191     var match;
192
193     // RGB(A)
194     if ((match = str.match(RE_RGB) || str.match(RE_RGBA))) {
195       return new Color.RGBA(
196         parseInt(match[1], 10),
197         parseInt(match[2], 10),

```

```

198         parseInt(match[3], 10),
199         parseFloat(match.length === 4 ? 1 : match[4])
200     );
201 }
202
203 // HSL(A)
204 if((match = str.match(RE_HSL) || str.match(RE_HSLA))) {
205     return new Color.HSLA(
206         parseFloat(match[1]),
207         parseFloat(match[2]),
208         parseFloat(match[3]),
209         parseFloat(match.length === 4 ? 1 : match[4])
210     );
211 }
212
213 // Hex
214 if(str.length === 3) {
215     // Hex 3 digit -> 6 digit
216     str = str.replace(/(.) /g, '$1$1$1');
217 }
218 if((match = str.match(RE_HEX))) {
219     return new Color.RGBA(
220         parseInt(match[1], 16),
221         parseInt(match[2], 16),
222         parseInt(match[3], 16),
223         1
224     );
225 }
226
227     return null;
228 },
229
230     luminance: function(color) {
231         if(color instanceof Color.HSLA) color = color.toRGB();
232         return 0.298912 * color.r + 0.586611 * color.g + 0.114478 * color.b;
233     },
234
235     greyscale: function(color) {
236         var lum = Color.luminance(color);
237         return new Color.RGBA(lum, lum, lum, this.a);
238     },
239
240     negate: function(color) {
241         if(color instanceof Color.HSLA) color = color.toRGB();
242         return new Color.RGBA(255 - color.r, 255 - color.g, 255 - color.b,
243             color.a);
244     },
245
246     /**
247      * @see http://sass-lang.com/docs/yardoc/Sass/Script/Functions.html#mix-
248      * instance_method
249     */
250     mix: function(color1, color2, weight) {
251         if(color1 instanceof Color.HSLA) color1 = color1.toRGB();
252         if(color2 instanceof Color.HSLA) color2 = color2.toRGB();

```

```

251     if (isNull(weight) || obj.jsl.format.isUndefined(weight)) weight = 0.5;
252
253     var w0 = 1 - weight;
254     var w = w0 * 2 - 1;
255     var a = color1.a - color2.a;
256     var w1 = ((w * a === -1 ? w : (w + a) / (1 + w * a)) + 1) / 2;
257     var w2 = 1 - w1;
258
259     return new Color.RGBA(
260       Math.round(color1.r * w1 + color2.r * w2),
261       Math.round(color1.g * w1 + color2.g * w2),
262       Math.round(color1.b * w1 + color2.b * w2),
263       Math.round(color1.a * w0 + color2.a * weight)
264     );
265   };
266 }
267
268 /**
269 * Color.RGBA
270 */
271 Color.RGBA = function(r, g, b, a) {
272   if.isArray(r)) {
273     g = r[1];
274     b = r[2];
275     a = r[3];
276     r = r[0];
277   } else if(isObject(r)) {
278     g = r.g;
279     b = r.b;
280     a = r.a;
281     r = r.r;
282   }
283
284   this.r = r || 0;
285   this.g = g || 0;
286   this.b = b || 0;
287   this.a = !isNumber(a) ? 1 : a;
288 };
289
290 Color.RGBA.prototype = {
291   toHSLA: function() {
292     var hsl = rgbToHsl(Math.round(this.r), Math.round(this.g), Math.round(
293       this.b));
294     return new Hsla(hsl[0], hsl[1], hsl[2], this.a);
295   },
296   toArray: function() {
297     return [Math.round(this.r), Math.round(this.g), Math.round(this.b),
298       this.a];
299   },
300   clone: function() {
301     return new Color.RGBA(this);
302   },
303 }
```

```

304     toString: function() {
305       return 'rgba(' + Math.round(this.r) + ', ' + Math.round(this.g) + ', '
306         + Math.round(this.b) + ', ' + this.a + ')';
307     }
308   };
309
310   /**
311    * Color.HSLA
312    */
313   Color.HSLA = function(h, s, l, a) {
314     if(isArray(h)) {
315       s = h[1];
316       l = h[2];
317       a = h[3];
318       h = h[0];
319     } else if(isObject(h)) {
320       s = h.s;
321       l = h.l;
322       a = h.a;
323       h = h.h;
324     }
325
326     this.h = h || 0;
327     this.s = s || 0;
328     this.l = l || 0;
329     this.a = !isNumber(a) ? 1 : a;
330   };
331
332   Color.HSLA.prototype = {
333     toRGB: function() {
334       var rgb = this.hslToRgb(this.h, this.s, this.l);
335       return new Rgba(rgb[0], rgb[1], rgb[2], this.a);
336     },
337
338     toArray: function() {
339       return [this.h, this.s, this.l, this.a];
340     },
341
342     clone: function() {
343       return new Color.HSLA(this);
344     },
345
346     toString: function() {
347       return 'hsla(' + this.h + ', ' + this.s + '%, ' + this.l + '%, ' +
348           this.a + ')';
349     }
350   };
351
352   /**
353    * Returns a color code based on the provided identifier.
354    * @param {number|string|Array<number>} id - Identifier for the color. Can
355    * be a numeric index, a predefined color name, or an RGB array.
356    * @return {string} The hex code of the color.

```

```

356     */
357     color(id) {
358         var c = '#0072BD';
359         if(typeof id == 'number') {
360             c = colororder[id % 7];
361         } else if(Array.isArray(id)) {
362             if(id.length == 3) {
363                 c = rgb2hex(id);
364             }
365         } else {
366             switch(id) {
367                 case 'y':
368                 case 'yellow':
369                     c = '#FFFF00';
370                     break;
371                 case 'm':
372                 case 'magenta':
373                     c = '#FF00FF';
374                     break;
375                 case 'c':
376                 case 'cyan':
377                     c = '#00FFFF';
378                     break;
379                 case 'r':
380                 case 'red':
381                     c = '#FF0000';
382                     break;
383                 case 'g':
384                 case 'green':
385                     c = '#00FF00';
386                     break;
387                 case 'b':
388                 case 'blue':
389                     c = '#0000FF';
390                     break;
391                 case 'w':
392                 case 'white':
393                     c = '#FFFFFF';
394                     break;
395                 case 'k':
396                 case 'black':
397                     c = '#000000';
398                     break;
399             }
400         }
401         return c;
402     }
403
404 /**
405 * Calculates a color on a gradient from green to red based on a value.
406 * @param {number} value - A number between 0 and 1 indicating position on
407 *   the gradient.
408 * @param {number} [k=0.3] - A scaling factor to adjust the gradient effect.
409 * @return {string} The hsl color string.
410 */

```



```
410 getColorG2R ( value , k = 0.3 ) {
411     if ( value < 0 ) {
412         value = 0;
413     } else if ( value < k ) {
414         value = value / k;
415     } else {
416         value = 1;
417     }
418     var hue = ( 120 * value ). toString ( 10 );
419     return [ " hsl ( " , hue , " , 100 % , 50 % ) " ]. join ( " " );
420 }
421
422 /**
423 * Calculates the gradient color between two colors based on a percentage.
424 * @param {number} p - The percentage (0 to 1) between the two colors.
425 * @param {Array<number>} rgb_beginning - The RGB values of the start color.
426 * @param {Array<number>} rgb_end - The RGB values of the end color.
427 * @return {Array<number>} The RGB values of the calculated gradient color.
428 */
429 colourGradientor ( p , rgb_beginning , rgb_end ) {
430     var w = p * 2 - 1;
431     var w1 = ( w + 1 ) / 2.0;
432     var w2 = 1 - w1;
433
434     var rgb = [ parseInt ( rgb_beginning [ 0 ] * w1 + rgb_end [ 0 ] * w2 ) ,
435                 parseInt ( rgb_beginning [ 1 ] * w1 + rgb_end [ 1 ] * w2 ) ,
436                 parseInt ( rgb_beginning [ 2 ] * w1 + rgb_end [ 2 ] * w2 ) ];
437     return rgb ;
438 }
439
440 /**
441 * Converts RGB color values to HEX.
442 * @param {number} r - The red color value (0-255).
443 * @param {number} g - The green color value (0-255).
444 * @param {number} b - The blue color value (0-255).
445 * @return {Array<number>} The HEX representation of the color.
446 */
447 rgb2hex ( r , g , b ) {
448     // If the first argument is an array , unpack its values
449     if ( Array . isArray ( r ) ) {
450         [ r , g , b ] = r ;
451     }
452
453     function toHex ( c ) {
454         // Scale to 0-255 if the value is between 0 and 1
455         c = ( c <= 1 ) ? Math . round ( c * 255 ) : Math . round ( c );
456         // Clamp the value between 0 and 255
457         c = Math . min ( 255 , Math . max ( 0 , c ) );
458         // Convert to hexadecimal and pad with zeros if necessary
459         return c . toString ( 16 ) . padStart ( 2 , ' 0 ' ) . toUpperCase ();
460     }
461     return '#' + toHex ( r ) + toHex ( g ) + toHex ( b );
462 }
463
464 /**
```



```
465 * Converts RGB color values to HSL (Hue, Saturation, Lightness).  
466 * @param {number} r - The red color value (0-255).  
467 * @param {number} g - The green color value (0-255).  
468 * @param {number} b - The blue color value (0-255).  
469 * @return {Array<number>} The HSL representation of the color.  
470 */  
471 rgbToHsl(r, g, b) {  
472     r /= 255;  
473     g /= 255;  
474     b /= 255;  
475  
476     var max = Math.max(r, g, b);  
477     var min = Math.min(r, g, b);  
478     var h, s, l;  
479  
480     l = (max + min) / 2;  
481  
482     if(max === min) {  
483         h = s = 0;  
484     } else {  
485         var d = max - min;  
486         switch (max) {  
487             case r: h = ((g - b) / d * 60 + 360) % 360; break;  
488             case g: h = (b - r) / d * 60 + 120; break;  
489             case b: h = (r - g) / d * 60 + 240; break;  
490         }  
491         s = l <= 0.5 ? d / (max + min) : d / (2 - max - min);  
492     }  
493  
494     return [h, s * 100, l * 100];  
495 }  
496  
497 /**
498 * Converts HSL color values to RGB.
499 * @param {number} h - The hue value (0-360).
500 * @param {number} s - The saturation value (0-100).
501 * @param {number} l - The lightness value (0-100).
502 * @return {Array<number>} The RGB representation of the color.
503 */
504 hslToRgb(h, s, l) {
505     s /= 100;
506     l /= 100;
507  
508     var r, g, b;
509  
510     if(s === 0){
511         r = g = b = l * 255;
512     } else {
513         var v2 = l < 0.5 ? l * (1 + s) : l + s - l * s;
514         var v1 = 2 * l - v2;
515         r = Math.round(this.hueToRgb(v1, v2, h + 120) * 255);
516         g = Math.round(this.hueToRgb(v1, v2, h) * 255);
517         b = Math.round(this.hueToRgb(v1, v2, h - 120) * 255);
518     }
519 }
```

```

520     return [ r , g , b ];
521 }
522
523 /**
524 * Helper function for converting a hue to RGB.
525 * @param {number} v1 - Helper value 1.
526 * @param {number} v2 - Helper value 2.
527 * @param {number} vh - The hue value to convert.
528 * @return {number} The RGB value for the hue.
529 */
530 hueToRgb(v1 , v2 , vh) {
531   vh /= 360;
532   if(vh < 0) vh++;
533   if(vh > 1) vh--;
534   if(vh < 1 / 6) return v1 + (v2 - v1) * 6 * vh;
535   if(vh < 1 / 2) return v2;
536   if(vh < 2 / 3) return v1 + (v2 - v1) * (2 / 3 - vh) * 6;
537   return v1;
538 }
539 }
540
541 exports.PRDC_JSLAB_LIB_COLOR = PRDC_JSLAB_LIB_COLOR;

```

Listing 73 - color.js

```

1 /**
2 * @file JSLAB library control submodule
3 * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4 * PR-DC, Republic of Serbia
5 * info@pr-dc.com
6 */
7
8 /**
9 * Class for JSLAB control submodule.
10 */
11 class PRDC_JSLAB_LIB_CONTROL {
12
13 /**
14 * Initializes the control submodule.
15 * @param {Object} js1 Reference to the main JSLAB object.
16 */
17 constructor(js1) {
18   var obj = this;
19   this.js1 = js1;
20 }
21
22 /**
23 * Create a transfer function representation.
24 * @param {number[]} num - Numerator coefficients of the transfer function.
25 * @param {number[]} den - Denominator coefficients of the transfer function
26
27 * @param {number} [Ts=0] - Sampling time, defaults to 0 for continuous-time
28 * systems.
29 * @returns {object} An object representing the transfer function { num, den
30   , Ts }.
31 */
32

```

```

29  tf (num, den, Ts = 0) {
30    return { num, den, Ts };
31  }
32
33  /**
34   * Create a state-space representation.
35   * @param {number[][]} A - System matrix.
36   * @param {number[][]} B - Input matrix.
37   * @param {number[][]} C - Output matrix.
38   * @param {number[][]} D - Feedthrough matrix.
39   * @param {number} [Ts=0] - Sampling time, defaults to 0 for continuous-time
40   *                         systems.
41   * @returns {object} An object representing the state-space system { A, B, C
42   *                   , D, Ts }.
43   */
44  ss(A, B, C, D, Ts = 0) {
45    return { A, B, C, D, Ts };
46  }
47
48  /**
49   * Convert a transfer function to a state-space representation.
50   * @param {number[]} num - Numerator coefficients of the transfer function.
51   * @param {number[]} den - Denominator coefficients of the transfer function
52   *
53   * @returns {object} State-space representation of the system.
54   */
55  tf2ss(num, den) {
56    const n = den.length;
57    num = [...zeros(n - num.length), ...num];
58
59    let A = zeros(n-1).map(() => zeros(n-1));
60    for(let i = 1; i < n - 1; i++) A[i-1][i] = 1;
61    A[n - 2] = fliplr(den.slice(1).map(d => -d));
62
63    const C = fliplr(num.map((c, i) => c - den[i] * num[0]).slice(1));
64    const B = zeros(n - 2).concat([1]);
65    const D = num[0];
66
67    return this.ss(A, B, C, D);
68  }
69
70  /**
71   * Convert a state-space representation to a transfer function.
72   * @param {object} sys - State-space system { A, B, C, D }.
73   * @returns {object} Transfer function representation { num, den }.
74   */
75  ss2tf(sys) {
76    const { A, B, C, D } = sys;
77
78    const den = charpoly(A);
79    const num = plus(charpoly(minus(A, this.jsl.env.math.multiply(B, [C]))),
80                     scale(den, D-1));
81
82    var p = den.length - num.length;
83    if(p > 0) {

```

```

81     num = [ ... zeros(p) , ... num ] ;
82   }
83   return { num, den } ;
84 }
85
86 /**
87 * Convert a continuous-time transfer function to discrete-time .
88 * @param {number[]} numc - Continuous-time numerator coefficients .
89 * @param {number[]} denc - Continuous-time denominator coefficients .
90 * @param {number} Ts - Sampling time .
91 * @returns {object} Discrete-time transfer function representation { num,
92           den } .
93 */
94 c2d(numc, denc, Ts) {
95   const sysc = this.tf2ss(numc, denc);
96   const sysd = this._c2dZOH(sysc, Ts);
97   return this.ss2tf(sysd);
98 }
99
100 /**
101 * Convert a continuous-time state-space system to discrete-time using Zero-
102 Order Hold .
103 * @param {object} sysc - Continuous-time state-space system { A, B, C, D } .
104 * @param {number} Ts - Sampling time .
105 * @returns {object} Discrete-time state-space system { A, B, C, D, Ts } .
106 */
107 _c2dZOH(sysc, Ts) {
108   const { A, B, C, D } = sysc;
109   const n = A.length;
110
111   const M = A.map((row, i) => [ ... row, B[i] ]) ;
112   M.push(zeros(n + 1));
113   const M_exp = js1.env.math.expm(js1.env.math.multiply(M, Ts)) . _data . slice
114     (0, n);
115
116   const Ad = M_exp.map(row => row.slice(0, n));
117   const Bd = M_exp.map(row => row.slice(n));
118
119   return this.ss(Ad, Bd, C, D, Ts);
120 }
121
122 /**
123 * Simulate the time response of a discrete-time linear system .
124 * @param {number[]} sys - transfer function of system .
125 * @param {number[]} u - Input signal array .
126 * @param {number[]} t - Time vector array .
127 * @returns {object} An object containing the response :
128 *           - y: Output signal array .
129 *           - t: Time vector array (same as input) .
130 */
131 lsim(sys, u, t, Ts) {
132   var sysd = sys;
133   if(!sys.Ts && Ts) {
134     sysd = this.c2d(sys.num, sys.den, Ts);
135   }

```

```

133     var num = sysd.num;
134     var den = sysd.den;
135
136     // Normalize the denominator coefficients so that den[0] = 1
137     if(den[0] !== 1) {
138         const den0 = den[0];
139         for(let i = 0; i < den.length; i++) {
140             den[i] /= den0;
141         }
142         for(let i = 0; i < num.length; i++) {
143             num[i] /= den0;
144         }
145     }
146
147     const N = u.length;
148     var y = zeros(N); // Initialize output array with zeros
149
150     // Iterate over each time step starting from n=1
151     for(let n = 0; n < N; n++) {
152         // Compute feedforward terms: sum(num[k] * u[n - k]) for k =0 to M
153         for(let k = 0; k < num.length; k++) {
154             const idx = n - k;
155             if(idx >= 0) {
156                 y[n] += num[k] * u[idx];
157             }
158         }
159
160         // Compute feedback terms: sum(den[k] * y[n - k]) for k =1 to N
161         for(let k = 1; k < den.length; k++) {
162             const idx = n - k;
163             if(idx >= 0) {
164                 y[n] -= den[k] * y[idx];
165             }
166         }
167     }
168
169     if(num.length < den.length) {
170         var n = den.length - num.length;
171         y = [...zeros(n), ...y.slice(0, y.length - n)];
172     }
173
174     return { y, t };
175 }
176
177 /**
178 * Simulates the system response over a specified time period.
179 * @param {Object} sys - The system to simulate.
180 * @param {number} Tfinal - The final time for the simulation.
181 * @returns {Object} The simulation result.
182 */
183 step(sys, Tfinal) {
184     var u = ones(101);
185     var t = linspace(0, Tfinal, 101);
186     var Ts = Tfinal/100;
187     return this.lsim(sys, u, t, Ts);

```

```

188 }
189
190 /**
191 * Estimates a transfer function model using numerical optimization.
192 * @param {Array<number>} t - Time vector.
193 * @param {Array<number>} u - Input signal vector.
194 * @param {Array<number>} y - Output signal vector.
195 * @param {number} np - Number of poles.
196 * @param {number} nz - Number of zeros.
197 * @param {string} [method='NelderMead'] - Optimization method ('NelderMead'
198   or 'Powell').
199 * @returns {{sys: object, error: number}} Estimated transfer function and
200   mean squared error.
201 */
202 tfest(t, u, y, np, nz, method = 'NelderMead') {
203   var obj = this;
204   var Ts = mean(diff(t));
205   var N = np+nz+1;
206   var f = (x) => { // funkcija prilagodjenosti
207     try {
208       var den = [1, ...x.slice(0, np)];
209       var num = x.slice(np);
210       var sys = tf(num, den);
211       var r = obj.lsim(sys, u, t, Ts);
212       var mse = obj.jsl.math.mse(y, r.y);
213       return mse;
214     } catch(err) {
215       return Infinity;
216     }
217   };
218   var r;
219   if(method === 'NelderMead') {
220     r = this.jsl.optim.optimNelderMead(f, zeros(N));
221   } else if(method === 'Powell') {
222     r = this.jsl.optim.optimPowell(f, zeros(N));
223   }
224   var den = [1, ...r.x.slice(0, np)];
225   var num = r.x.slice(np);
226   return {sys: tf(num, den), error: r.fx};
227 }
228 exports.PRDC_JSLAB_LIB_CONTROL = PRDC_JSLAB_LIB_CONTROL;

```

Listing 74 - control.js

```

1 /**
2  * @file JSLAB library conversion submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 /**
9  * Class for JSLAB conversion submodule.
10 */

```



```
11 class PRDC_JSLAB_LIB_CONVERSION {
12
13     /**
14      * Constructs a conversion submodule object with access to JSLAB's
15      * conversion functions.
16      * @constructor
17      * @param {Object} js1 - Reference to the main JSLAB object.
18      */
19     constructor(js1) {
20         var obj = this;
21         this.js1 = js1;
22
23         this.speech = this.js1.env.speech;
24
25         /**
26          * Serializes BigInt values to JSON by converting them to strings.
27          */
28         BigInt.prototype.toJSON = function() { return this.toString(); };
29
30         /**
31          * Converts text to speech using the Web Speech API.
32          * @param {string} msg - The text message to be spoken.
33          */
34         speak(msg) {
35             if(!global.is_worker) {
36                 this.speech.text = msg;
37                 speechSynthesis.speak(this.speech);
38             }
39         }
40
41         /**
42          * Converts a number to a string with specified precision.
43          * @param {number} num - The number to convert.
44          * @param {number} precision - The number of digits after the decimal point.
45          * @returns {string} The formatted string.
46          */
47         num2str(num, precision = 2) {
48             if(isNumber(precision)) {
49                 return num.toFixed(precision);
50             } else {
51                 return sprintf(precision, num);
52             }
53         }
54
55         /**
56          * Changes the extension of a file path.
57          * @param {string} path - The original file path.
58          * @param {string} ext_new - The new extension without the dot.
59          * @returns {string} The file path with the new extension.
60          */
61         changeExtension(path, ext_new) {
62             var pos = path.lastIndexOf('.');
63             return path.substr(0, pos < 0 ? path.length : pos) + '.' + ext_new;
64         }
65     }
66 }
```

```

65
66 /**
67 * Remove the extension of a file path.
68 * @param {string} path - The original file path.
69 * @returns {string} The file path without extension.
70 */
71 removeExtension(path) {
72     var pos = path.lastIndexOf('.');
73     return path.substr(0, pos < 0 ? path.length : pos);
74 }
75
76 /**
77 * Transforms coordinates from NED (North, East, Down) frame to RPY (Roll,
78 * Pitch, Yaw) frame.
79 * @param {number} roll - The roll angle in radians.
80 * @param {number} pitch - The pitch angle in radians.
81 * @param {number} yaw - The yaw angle in radians.
82 * @param {Array<number>} [A] - Optional matrix to apply transformation to.
83 * @returns {Array<Array<number>>} The transformation matrix.
84 */
85 ned2rpy(roll, pitch, yaw, A) {
86     T = [
87         [Math.cos(yaw)*Math.cos(pitch),
88          Math.sin(yaw)*Math.cos(pitch),
89          -Math.sin(pitch)],
90         [Math.cos(yaw)*Math.sin(pitch)*Math.sin(roll)-Math.sin(yaw)*Math.cos(roll),
91          Math.sin(yaw)*Math.sin(pitch)*Math.sin(roll)+Math.cos(yaw)*Math.cos(roll),
92          Math.cos(pitch)*Math.sin(roll)],
93         [Math.cos(yaw)*Math.sin(pitch)*Math.cos(roll)+Math.sin(yaw)*Math.sin(roll),
94          Math.sin(yaw)*Math.sin(pitch)*Math.cos(roll)-Math.cos(yaw)*Math.sin(roll),
95          Math.cos(pitch)*Math.cos(roll)]]
96     ];
97
98     if(A != undefined) {
99         return this.jsl.env.math.multiply(T, A);
100    } else {
101        return T;
102    }
103 }
104 /**
105 * Converts an array of uint8 numbers to a string.
106 * @param {Uint8Array} data - The array of uint8 numbers.
107 * @returns {string} The converted string.
108 */
109 uint8ToString(data) {
110     return String.fromCharCode(...data);
111 }
112
113 /**
114 * Converts an array of hexadecimal strings to an array of decimal numbers.

```

```

115  * @param {Array<string>} hex - The array of hexadecimal strings .
116  * @returns {Array<number>} The array of decimal numbers .
117  */
118 hex2dec(hex) {
119   return hex.map(function(e) {
120     return parseInt(e, 16);
121   });
122 }
123
124 /**
125  * Converts a number to its ASCII character equivalent .
126  * @param {number} num - The number to convert .
127  * @returns {string} The ASCII character .
128  */
129 numToASCII(num) {
130   return ('' + num).charCodeAt(0);
131 }
132
133 /**
134  * Converts a number to a hexadecimal string with a fixed number of digits .
135  * @param {number} num - The number to convert .
136  * @param {number} dig - The number of digits in the resulting string .
137  * @param {boolean} prefix - Whether to add a '0x' prefix .
138  * @returns {string} The hexadecimal string .
139  */
140 numToHexStr(num, dig = 4, prefix = false) {
141   var str = ('0000'.repeat(dig) + num.toString(16).toUpperCase()).slice(-dig);
142   if(prefix) {
143     return '0x' + str;
144   }
145   return str;
146 }
147
148 /**
149  * Converts an int8 number to two ASCII characters .
150  * @param {number} num - The int8 number .
151  * @returns {string} The two ASCII characters .
152  */
153 int8To2ASCII(num) {
154   var data = ("0" + (Uint8Array.from([num])[0]).toString(16)).substr(-2);
155   return [data[0].charCodeAt(0), data[1].charCodeAt(0)];
156 }
157
158 /**
159  * Converts an int16 number to four ASCII characters representing its
160  * hexadecimal value .
161  * @param {number} num - The int16 number to convert .
162  * @returns {string} A string of four ASCII characters .
163  */
164 int16To4ASCII(num) {
165   var data = ("000" + (Uint16Array.from([num])[0]).toString(16).toUpperCase()
166   ()).substr(-4);
167   return [data[0].charCodeAt(0), data[1].charCodeAt(0), data[2].charCodeAt(
168   0), data[3].charCodeAt(0)];

```

```

166     }
167
168     /**
169      * Combines two uint8 values into an int16 value.
170      * @param {number} part1 - The first uint8 value.
171      * @param {number} part2 - The second uint8 value.
172      * @returns {number} The combined int16 value.
173      */
174     uint8sToInt16(part1, part2) {
175         part1 &= 0xFF;
176         part2 &= 0xFF;
177
178         let result = (part1 << 8) | part2;
179         return (result & 0x8000) ? -((result ^ 0xFFFF) + 1) : result;
180     }
181
182     /**
183      * Combines four uint8 values into an int32 value.
184      * @param {number} part1 - The first uint8 value.
185      * @param {number} part2 - The second uint8 value.
186      * @param {number} part3 - The third uint8 value.
187      * @param {number} part4 - The fourth uint8 value.
188      * @returns {number} The combined int32 value.
189      */
190     uint8sToInt32(part1, part2, part3, part4) {
191         part1 &= 0xFF;
192         part2 &= 0xFF;
193         part3 &= 0xFF;
194         part4 &= 0xFF;
195
196         let result = (part1 << 24) | (part2 << 16) | (part3 << 8) | part4;
197         return (result & 0x80000000) ? -((result ^ 0xFFFFFFFF) + 1) : result;
198     }
199
200     /**
201      * Converts four uint8 values into a floating-point number.
202      * @param {number} part1 - The first uint8 value.
203      * @param {number} part2 - The second uint8 value.
204      * @param {number} part3 - The third uint8 value.
205      * @param {number} part4 - The fourth uint8 value.
206      * @returns {number} The floating-point number.
207      */
208     uint8sToFloat(part1, part2, part3, part4) {
209         // Combine the Uint8 values into a 32-bit integer
210         let combinedValue = (part4 << 24) | (part3 << 16) | (part2 << 8) | part1;
211
212         // Interpret the integer as a float without using Math.pow
213         let sign = (combinedValue & 0x80000000) ? -1 : 1;
214         let exponent = ((combinedValue >> 23) & 0xFF) - 127;
215         let mantissa = (combinedValue & 0x7FFFFFFF | 0x8000000) / (1 << 23);
216
217         return sign * mantissa * (2 ** exponent);
218     }
219
220     /**

```



```
221 * Converts a uint16 value to an int16 value.  
222 * @param {number} num - The uint16 value to convert.  
223 * @returns {number} The converted int16 value.  
224 */  
225 uint16ToInt16(num) {  
226     return (num & 0x8000) ? -((num ^ 0xFFFF) + 1) : num;  
227 }  
228  
229 /**
230 * Converts a uint8 number to two ASCII characters.  
231 * @param {number} num - The uint8 number to convert.  
232 * @returns {string} A string containing two ASCII characters representing  
the hexadecimal value.  
233 */  
234 uint8To2ASCII(num) {  
235     var data = ("0" + (num % 256).toString(16).toUpperCase()).substr(-2);  
236     return [data[0].charCodeAt(0), data[1].charCodeAt(0)];  
237 }  
238  
239 /**
240 * Converts milliseconds to a time string in mm:ss format.  
241 * @param {number} ms - The time in milliseconds.  
242 * @returns {string} The time string.  
243 */  
244 ms2time(ms) {  
245     min = Math.floor((ms/1000/60) << 0),  
246     sec = Math.floor((ms/1000) % 60);  
247     return ('0' + min).slice(-2) + ':' + ('0' + sec).slice(-2);  
248 }  
249  
250 /**
251 * Converts a decimal number to a binary string.  
252 * @param {number} x - The decimal number.  
253 * @returns {string} The binary string.  
254 */  
255 dec2bin(x) {  
256     return this.jsl.env.math.bin(x);  
257 }  
258  
259 /**
260 * Converts a decimal number to a hexadecimal string.  
261 * @param {number} x - The decimal number.  
262 * @returns {string} The hexadecimal string.  
263 */  
264 dec2hex(x) {  
265     return this.jsl.env.math.hex(x);  
266 }  
267  
268 /**
269 * Converts a decimal number to an octal string.  
270 * @param {number} x - The decimal number.  
271 * @returns {string} The octal string.  
272 */  
273 dec2oct(x) {  
274     return this.jsl.env.math.oct(x);
```

```

275 }
276
277 /**
278 * Rounds a number to a specified number of decimal places.
279 * @param {number} number - The number to round.
280 * @param {number} [decimals=2] - The number of decimal places.
281 * @param {boolean} [string=false] - Whether to return the result as a
282 *     string.
283 * @returns {number|string} The rounded number.
284 */
285 round(number, decimals = 2, string = false) {
286   var result;
287   if(typeof value === 'number') {
288     result = number.toFixed(decimals);
289   } else {
290     result = Number(number).toFixed(decimals);
291   }
292   if(result === 0) {
293     result = '0'; // removes -0
294   }
295   if(string) {
296     return result;
297   } else {
298     return Number(result);
299   }
300 }
301 /**
302 * Rounds a number to a fixed number of decimal places if it is a number.
303 * @param {number} value - The value to round.
304 * @param {number} p - The number of digits after the decimal point.
305 * @returns {number} The rounded number with a fixed number of decimal
306 *     places, or the original value if it is not a number.
307 */
308 roundIf(value, p) {
309   if(typeof value === 'number') {
310     return Number(value.toFixed(p));
311   } else {
312     return value;
313   }
314 }
315 /**
316 * Rounds a number to a fixed number of decimal places if it is a number.
317 * @param {number} value - The value to round.
318 * @param {number} p - The number of digits after the decimal point.
319 * @returns {number} The rounded number with a fixed number of decimal
320 *     places, or the original value if it is not a number.
321 */
322 roundIfPrec(value, p) {
323   if(typeof p === 'number') {
324     return round(value, p);
325   } else {
326     return value;
327   }
328 }
329 
```

```

327 }
328
329 /**
330 * Converts a uint8_t number to a bit string.
331 * @param {number} n - The number to convert.
332 * @returns {string} A bit string representing the number.
333 */
334 bitString(n) {
335   return ("00000000" + n.toString(2)).substr(-8);
336 }
337
338 /**
339 * Generates a set of flags from a bitfield based on a mapping.
340 * @param {Object} map - The mapping of bit positions to flag names.
341 * @param {string} name_column - The column name in the mapping that
342   contains the flag names.
343 * @param {number} val - The bitfield value.
344 * @returns {Object} An object with keys as flag names and values indicating
345   the presence (1) or absence (0) of each flag.
346 */
347 getBitFlags(map, name_column, val) {
348   var flags = {};
349   var keys = Object.keys(map);
350   var bits = [...Array(keys.length)].map(function(x, i) {
351     return val >> i & 1;
352   });
353   keys.forEach(function(key) {
354     flags[map[key][name_column]] = bits[key];
355   });
356   return flags;
357 }
358 /**
359 * Retrieves the enumeration value based on a property match.
360 * @param {Object} enum_object - The enumeration object to search.
361 * @param {string} prop - The property name to match.
362 * @param {*} val - The property value to match.
363 * @returns {number} The enumeration key as a number, or the index if not
364   found.
365 */
366 getEnumVal(enum_object, prop, val) {
367   var entries = Object.entries(enum_object);
368   var idx = entries.findIndex(function(f) {
369     return f[1][prop] === val;
370   });
371   if(idx > 0) {
372     return Number(entries[idx][0]);
373   }
374   return idx;
375 }
376 /**
377 * Inverts an enumeration, swapping keys and values, optionally based on a
378   specific property of the enumeration values.
379 * @param {Object} enum_object - The enumeration object to invert.

```

```

378   * @param {string} [prop] - An optional property name to use from the
379   * enumeration values.
380   */
381 invertEnum(enum_object, prop) {
382   var enum_object_inv = {};
383   Object.entries(enum_object).forEach(function([key, value]) {
384     var ind = value;
385     if(prop) {
386       ind = value[prop];
387     }
388     var num_key = Number(key);
389     if(num_key == key) {
390       enum_object_inv[ind] = num_key;
391     } else {
392       enum_object_inv[ind] = key;
393     }
394   });
395   return enum_object_inv;
396 }
397
398 /**
399 * Converts an array of numbers to a string of hexadecimal values,
400 * optionally prefixed with "0x".
401 * @param {Array<number>} A - The array to convert.
402 * @param {boolean} [prefix=true] - Whether to add a "0x" prefix to each hex
403 *       value.
404 * @returns {string} A string of hexadecimal values.
405 */
406 arrayToHexStr(A, prefix = true) {
407   var A_uint8 = new Uint8Array(A);
408   var len = A.length;
409   var A_hex = Array(len);
410   for(let i = 0; i < len; i++) {
411     A_hex[i] = A_uint8[i].toString(16).toUpperCase().padStart(2, '0');
412   }
413   if(prefix) {
414     return '0x' + A_hex.join(' 0x');
415   } else {
416     return A_hex.join(' ');
417   }
418 }
419 /**
420 * Converts an array of numbers to an ASCII string.
421 * @param {Array<number>} array - The array of numbers to convert.
422 * @returns {string} The ASCII string representation of the array.
423 */
424 arrayToASCII(A) {
425   return String.fromCharCode(...A);
426 }
427 /**
428 * Extends an object with properties from additional objects.
429 * @param {...Object} objects - The objects to merge into the target object.

```

```
430     * @returns { Object } The extended object.
431     */
432     extend () {
433         var target = arguments[0] || {}, o, p;
434
435         for (var i = 1, len = arguments.length; i < len; i++) {
436             o = arguments[i];
437
438             if (!thisisObject(o)) continue;
439
440             for (p in o) {
441                 target[p] = o[p];
442             }
443         }
444
445         return target;
446     }
447
448 /**
449 * Normalizes the value of a radio control (RC) input.
450 * @param {number} rc - The RC input value.
451 * @param {number} [deadzone=0] - The deadzone value below which the output
452 *     is set to zero.
453 * @returns {number} The normalized value.
454 */
455 normalizeRC(rc, deadzone = 0) {
456     var val = rc - 1500;
457     if (Math.abs(val) < deadzone) {
458         return 0;
459     }
460     return (val - Math.sign(val) * deadzone) / (500 - deadzone);
461 }
462 /**
463 * Checks if a value has been updated and updates the last value if it has.
464 * @param {Object} data - An object containing the current value and the
465 *     last value.
466 * @returns {boolean} True if the value has been updated; false otherwise.
467 */
468 checkValueUpdate(data) {
469     if (data.value !== data.last_value) {
470         data.last_value = data.value;
471         return true;
472     } else {
473         return false;
474     }
475 }
476 /**
477 * Resets the value and last value properties of an object.
478 * @param {Object} data - The object whose value and last value properties
479 *     will be reset.
480 */
481 resetValue(data) {
482     data.last_value = undefined;
```

```

482     data.value = undefined;
483 }
484
485 /**
486 * Converts an ADC count to force in Newtons.
487 * @param {number} adc_count - The raw ADC count value.
488 * @param {number} load_cell_capacity - The capacity of the load cell in
489 * Newtons.
490 * @param {number} [adc_resolution=24] - The ADC resolution in bits.
491 * @param {number} [adc_gain=128] - The gain applied to the ADC.
492 * @param {number} [adc_sensitivity=2] - The ADC sensitivity in mV/V.
493 * @param {boolean} [adc_bipolar=true] - Indicates if the ADC is bipolar.
494 * @returns {number} The calculated force in Newtons.
495 */
496 adcToNewtons(adc_count, load_cell_capacity,
497   adc_resolution = 24, adc_gain = 128,
498   adc_sensitivity = 2, adc_bipolar = true) {
499   if(adc_bipolar) {
500     adc_resolution = adc_resolution - 1;
501   }
502   adc_sensitivity = adc_sensitivity / 1000;
503   const max_adc_count = Math.pow(2, adc_resolution) - 1;
504   const measured_adc = (adc_count / max_adc_count) / adc_gain;
505   const force = (measured_adc / adc_sensitivity) * load_cell_capacity;
506   return force;
507 }
508 /**
509 * Converts file data from a given path to a blob URL.
510 * @param {string} path - Relative path from the application's base path to
511 * the file.
512 * @returns {string} A blob URL representing the file's data.
513 */
514 data2blobUrl(path) {
515   return URL.createObjectURL(new Blob([fs.readFileSync(app_path+'\\"+path)]);
516 }
517 /**
518 * Converts top, right, bottom, and left margins into x and y coordinates.
519 * @param {number} cont_width - Container width.
520 * @param {number} cont_height - Container height.
521 * @param {number} width - Element width.
522 * @param {number} height - Element height.
523 * @param {number} top - Top margin.
524 * @param {number} right - Right margin.
525 * @param {number} bottom - Bottom margin.
526 * @param {number} left - Left margin.
527 * @returns {Array} Array containing x and y coordinates.
528 */
529 trbl2xy(cont_width, cont_height, width, height, top, right, bottom, left) {
530   var x = 0;
531   var y = 0;
532
533   if (!isUndefined(left)) {

```

```

534         x = left ;
535     } else if (!isUndefined(right)) {
536         x = cont_width - width - right ;
537     }
538     if (!isUndefined(top)) {
539         y = top ;
540     } else if (!isUndefined(bottom)) {
541         y = cont_height - height - bottom ;
542     }
543     return [x, y];
544 }
545
546 /**
547 * Generates a CSV string from an simple object containing arrays of values .
548 * Each key in the object represents a column in the CSV. This function
549 * handles uneven array lengths by filling missing values with an empty
550 * string .
551 * @param {Object} data - The object containing arrays of data. Each key
552 * will be a column header .
553 * @param {string} [delimiter=','] - The delimiter to use for separating
554 * entries in the CSV (defaults to a comma) .
555 * @returns {string} The generated CSV as a string , with each row
556 * representing an entry and each column representing data from the
557 * corresponding key in the input object .
558 */
559 simpleObj2Csv(data, delimiter = ',') {
560     const keys = Object.keys(data);
561     let csv_content = keys.join(delimiter) + '\n';
562
563     const maxEntries = Math.max(...keys.map(function(key) {return data[key].length;}));
564     for(let i = 0; i < maxEntries; i++) {
565         let row = keys.map(function(key) { return (i < data[key].length) ? data[key][i] : '' ; }).join(delimiter);
566         csv_content += row + '\n';
567     }
568
569     return csv_content;
570 }
571
572 /**
573 * Converts a 2D array into a CSV string format .
574 * @param {Array} data - An array of arrays to be converted into CSV format .
575 * @param {string} [delimiter=','] - The delimiter to separate the values in
576 * the CSV .
577 * @returns {string} The formatted CSV string .
578 */
579 simpleArray2Csv(data, delimiter = ',') {
580     let csv_content = '';
581     const maxEntries = Math.max(...data.map(function(e) {return e.length;}));
582     for(let i = 0; i < maxEntries; i++) {
583         let row = data.map(function(e) { return (i < e.length) ? e[i] : '' ; }).join(delimiter);
584         csv_content += row + '\n';
585     }

```

```

579         return csv_content;
580     }
581 }
582 }
583
584 exports.PRDC_JSLAB_LIB_CONVERSION = PRDC_JSLAB_LIB_CONVERSION;

```

Listing 75 - conversion.js

```

1  /**
2  * @file JSLAB library device gamepad submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 /**
9 * Class for gamepad.
10 */
11 class PRDC_JSLAB_DEVICE_GAMEPAD {
12
13 /**
14 * Initializes the gamepad device instance.
15 * @param {Object} js1 - Reference to the main JSLAB object.
16 * @param {string} id - Unique identifier for the gamepad.
17 * @param {number} [dt=10] - Data reading interval in milliseconds.
18 */
19 constructor(js1, id, dt = 10) {
20     var obj = this;
21     this.js1 = js1;
22     this.id = id;
23     this.active = false;
24     this.data;
25
26     this.read_gamepad_loop;
27     this.read_gamepad_dt = dt;
28
29     this._checkGamepadFun = function() {
30         obj._checkGamepad();
31     };
32     this.js1.context.addEventListener("gamepadconnected", obj._checkGamepadFun
33         );
34
35     this.js1.addForCleanup(this, function() {
36         obj.close();
37     });
38
39     this._checkGamepad();
40 }
41 /**
42 * Checks if the gamepad is connected and updates its state.
43 */
44 _checkGamepad() {
45     var gamepad = this._getGamepad();
46     if(gamepad) {

```

```
47      if (!this.active) {
48        this._onConnect();
49      }
50    }
51  }
52
53
54 /**
55 * Retrieves the gamepad object if available.
56 * @returns {Gamepad|boolean} The gamepad object if found, otherwise false.
57 */
58 _getGamepad() {
59   var gamepads = this.jsl.env.navigator.getGamepads();
60   for(let i = 0; i < gamepads.length; i++) {
61     var gamepad = gamepads[i];
62     if(gamepad != null) {
63       if(gamepad.id == this.id) {
64         return gamepad.toJSON();
65       }
66     }
67   }
68   return false;
69 }
70
71 /**
72 * Handles gamepad connection events.
73 */
74 _onConnect() {
75   var obj = this;
76
77   // Read loop
78   this.detect_gamepad_loop = clearIntervalIf(this.detect_gamepad_loop);
79   this.active = true;
80   clearIntervalIf(this.read_gamepad_loop);
81   this.read_gamepad_loop = setInterval(function() {
82     var gamepad = obj._getGamepad();
83     if(gamepad) {
84       if(gamepad.connected) {
85         obj._onData(gamepad);
86       } else {
87         obj._onDisconnect();
88       }
89     } else {
90       obj._onDisconnect();
91     }
92   }, this.read_gamepad_dt);
93
94   if(this.jsl.format.isFunction(this.onConnectCallback)) {
95     this.onConnectCallback();
96   }
97 }
98
99 /**
100 * Handles gamepad disconnection events.
101 */
```

```

102     _onDisconnect() {
103         this.read_gamepad_loop = clearIntervalIf(this.read_gamepad_loop);
104         this.active = false;
105         this._checkGamepad();
106         if(this.jsl.format.isFunction(this.onDisconnectCallback)) {
107             this.onDisconnectCallback();
108         }
109     }
110
111     /**
112      * Handles incoming gamepad data.
113      * @param {Gamepad} gamepad - The connected gamepad object.
114      */
115     _onData(gamepad) {
116         this.data = gamepad;
117         if(this.jsl.format.isFunction(this.onDataCallback)) {
118             this.onDataCallback(gamepad);
119         }
120     }
121
122     /**
123      * Sets the callback function to handle incoming gamepad data.
124      * @param {Function} callback - Function to execute when data is received.
125      */
126     setOnData(callback) {
127         if(this.jsl.format.isFunction(callback)) {
128             this.onDataCallback = callback;
129         }
130     }
131
132     /**
133      * Sets the callback function for gamepad connection events.
134      * @param {Function} callback - Function to execute on connection.
135      */
136     setOnConnect(callback) {
137         if(this.jsl.format.isFunction(callback)) {
138             this.onConnectCallback = callback;
139             if(this.active) {
140                 this.onConnectCallback();
141             }
142         }
143     }
144
145     /**
146      * Sets the callback function for gamepad disconnection events.
147      * @param {Function} callback - Function to execute on disconnection.
148      */
149     setOnDisconnect(callback) {
150         if(this.jsl.format.isFunction(callback)) {
151             this.onDisconnectCallback = callback;
152         }
153     }
154
155     /**
156      * Cleans up the gamepad instance and stops data reading.

```

```

157   */
158   close() {
159     this.active = false;
160     this.read_gamepad_loop = clearIntervalIf(this.read_gamepad_loop);
161     this.jsl.context.removeEventListener("gamepadconnected", this.
162       _checkGamepadFun);
163   }
164
165 exports.PRDC_JSLAB_DEVICE_GAMEPAD = PRDC_JSLAB_DEVICE_GAMEPAD;

```

Listing 76 - device-gamepad.js

```

1 /**
2  * @file JSLAB library device submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8
9 var { PRDC_JSLAB_DEVICE_GAMEPAD } = require('../device-gamepad');
10
11 /**
12  * Class for JSLAB device submodule.
13  */
14 class PRDC_JSLAB_LIB_DEVICE {
15
16 /**
17  * Constructs a device submodule object with access to JSLAB's device
18  * functions.
19  * @constructor
20  * @param {Object} jsl - Reference to the main JSLAB object.
21  */
22 constructor(jsl) {
23   var obj = this;
24   this.jsl = jsl;
25
26   this._camera_resolutions = [
27     { "label": "4K (UHD)", "width": 3840, "height": 2160, "ratio": "16:9" },
28     { "label": "1080p (FHD)", "width": 1920, "height": 1080, "ratio": "16:9" },
29     { "label": "720p (HD)", "width": 1280, "height": 720, "ratio": "16:9" },
30     { "label": "480p (VGA)", "width": 640, "height": 480, "ratio": "4:3" },
31     { "label": "360p (nHD)", "width": 640, "height": 360, "ratio": "16:9" },
32     { "label": "240p (QVGA)", "width": 320, "height": 240, "ratio": "4:3" },
33     { "label": "144p (QCIF)", "width": 176, "height": 144, "ratio": "4:3" },
34   ];
35
36 /**
37  * Checks if there is a USB device connected with the specified Vendor ID
38  * and Product ID.
39  * @param {string} VID - Vendor ID of the USB device.
40  * @param {string} PID - Product ID of the USB device.

```

```

40   * @returns {boolean} True if the device is found, false otherwise.
41   */
42 checkDeviceUSB(VID, PID) {
43   var val = this.jsl.env.execSync('wmic path win32_pnpsigneddriver get
44     deviceid, driverversion | find "USB\\VID_' + VID + '&PID_' + PID + '"';
45   );
46   if(val.state == 'success') {
47     if(val.data.length) {
48       if(this.jsl.debug) {
49         this.jsl.env disp('@checkDeviceUSB: ' + val.data);
50       }
51       return true;
52     } else {
53       return false;
54     }
55   } else {
56     if(this.jsl.debug) {
57       this.jsl.env.error('@checkDeviceUSB: ' + val.data);
58     }
59     return false;
60   }
61 /**
62  * Checks for a connected USB device by STM and an optional Product ID.
63  * @param {string} [PID='5740'] - Product ID of the USB device, default is
64  *   for Virtual COM Port.
65  * @returns {boolean} True if the device is found, false otherwise.
66  */
67 checkDeviceSTM(PID = '5740') {
68   return this.checkDeviceUSB('0483', PID);
69 }
70 /**
71  * Checks if there is a USB device connected using a CH340 chip.
72  * @returns {boolean} True if the device is found, false otherwise.
73  */
74 checkDeviceCH340() {
75   return this.checkDeviceUSB('1A86', '7523');
76 }
77 /**
78  * Checks if a specific driver is installed on the system.
79  * @param {string} driver_name - Name of the driver to check.
80  * @returns {boolean} True if the driver is found, false otherwise.
81  */
82 checkDriver(driver_name) {
83   var val = this.jsl.env.execSync('driverquery');
84   if(val.state == 'success') {
85     var output = val.data.split(/[\r\n]+/);
86     var output_lc = output.map(function(x) { return x.toLowerCase(); });
87     if(!Array.isArray(driver_name)) {
88       driver_name = [driver_name];
89     }
90     driver_name = driver_name.map(function(x) { return x.toLowerCase(); });

```

```

92     var data_out = [];
93     for(var i = 0; i < driver_name.length; i++) {
94         var idx = output_lc.map(function(x) { return x.startsWith(driver_name[
95             i]); }).findIndex(function(x) { return x == true; });
96         if(idx >= 0) {
97             data_out[i] = output[idx];
98         } else {
99             data_out[i] = '';
100        }
101    }
102    if(data_out.every(function(x) { return x.length; })) {
103        if(this.jsl.debug) {
104            this.jsl.env.disp('@checkDriver: ' + data_out);
105        }
106        return true;
107    } else {
108        return false;
109    }
110    if(this.jsl.debug) {
111        this.jsl.env.error('@checkDriver: ' + val.data);
112    }
113    return false;
114 }
115 }
116 /**
117 * Checks if the drivers for FTDI devices are installed.
118 * @returns {boolean} True if the drivers are found, false otherwise.
119 */
120 checkDriverFTDI() {
121     return this.checkDriver(['FTDIBUS', 'FTSER2K']);
122 }
123
124 /**
125 * Checks if the drivers for Silicon Labs CP210x USB to UART bridge are
126 * installed.
127 * @returns {boolean} True if the drivers are found, false otherwise.
128 */
129 checkDriverCP210x() {
130     return this.checkDriver('silabser');
131 }
132
133 /**
134 * Checks if the drivers for CH340 USB to serial converter are installed.
135 * @returns {boolean} True if the drivers are found, false otherwise.
136 */
137 checkDriverCH340() {
138     return this.checkDriver('CH341SER_A64');
139 }
140
141 /**
142 * Check if Arduino CLI is available.
143 * @returns {boolean} True if available.
144 */

```

```

145  checkArduino() {
146    try {
147      var output = this.jsl.env.execSync('arduino-cli -h', { stdio: 'pipe' });
148      return true;
149    } catch(err) {
150      this.jsl.env.error('@checkArduino: '+language.string(224));
151    }
152  }
153
154 /**
155 * Compile Arduino project.
156 * @param {string} dir - Project directory.
157 * @returns {Object|boolean} Compilation result or false on error.
158 */
159 compileArduino(dir) {
160   var config_file = this.jsl.env.pathJoin(dir, 'config.json');
161   if(!this.jsl.env.checkFile(config_file)) {
162     this.jsl.env.error('@compileArduino: '+language.string(225));
163     return false;
164   }
165
166   try {
167     var config = JSON.parse(this.jsl.env.readFileSync(config_file))
168     var build_property_str = '';
169     if(config.build_property) {
170       build_property_str = '--build-property "${config.build_property}"';
171     }
172     var output = this.jsl.env.spawnSync('arduino-cli compile --json -b "${{
173       config.b}" ${build_property_str} "${{dir}}", {
174       shell: true,
175       encoding: 'utf8',
176       stdio: 'pipe'
177     });
178     if(output.stdout) {
179       return JSON.parse(output.stdout);
180     } else if(output.stderr) {
181       return JSON.parse(output.stderr);
182     }
183   } catch(err) {
184     this.jsl.env.error('@compileArduino: Error: '+err.toString());
185   }
186
187 /**
188 * Upload Arduino project.
189 * @param {string} dir - Project directory.
190 * @param {string} [port] - Optional port.
191 * @returns {Object|boolean} Upload result or false on error.
192 */
193 uploadArduino(dir, port) {
194   var config_file = this.jsl.env.pathJoin(dir, 'config.json');
195   if(!this.jsl.env.checkFile(config_file)) {
196     this.jsl.env.error('@uploadArduino: '+language.string(225));
197     return false;
198   }

```

```

199
200   try {
201     var config = JSON.parse(this.jsl.env.readFileSync(config_file));
202     var build_property_str = '';
203     if(config.build_property) {
204       build_property_str = '--build-property ${config.build_property}';
205     }
206     var port_str = '';
207     if(port) {
208       port_str = '-p ${port}';
209     }
210     var output = this.jsl.env.spawnSync(`arduino-cli compile --json -u -t -b
211       ${config.b} ${build_property_str} ${port_str} ${dir}`,
212       shell: true,
213       encoding: 'utf8',
214       stdio: 'pipe'
215     );
216     if(output.stdout) {
217       return JSON.parse(output.stdout);
218     } else if(output.stderr) {
219       return JSON.parse(output.stderr);
220     }
221   } catch(err) {
222     this.jsl.env.error('@uploadArduino: Error: '+err.toString());
223   }
224
225 /**
226 * Retrieves all available serial ports.
227 * @returns {Promise<Array>} Resolves with an array of serial port info.
228 */
229 async listSerialPorts() {
230   return await this.jsl.env.SerialPort.list();
231 }
232
233 /**
234 * Opens a serial port.
235 * @param {string} port - Port path.
236 * @param {number} [baudrate=9600] - Baud rate.
237 * @param {object} [opts={}] - Additional options.
238 * @returns {SerialPort} The opened SerialPort instance.
239 */
240 async connectSerialPorts(port, baudrate = 9600, opts_in = {}) {
241   var opts = {
242     dataBits: 8,
243     parity: 'none',
244     stopBits: 1,
245     flowControl: false,
246     ...opts_in
247   }
248   var sp = new this.jsl.env.SerialPort({
249     path: port,
250     baudRate: baudrate,
251     ...opts
252   });

```

```

253     sp.on('open', function() {
254         try {
255             sp.set({
256                 dtr: true,
257                 rts: false
258             });
259         } catch(err) {}
260     });
261     return sp;
262 }
263 /**
264 * Retrieves the current state of all connected gamepads.
265 * @returns {Object[]} An array of connected gamepad objects.
266 */
267 getGamepads() {
268     return this.jsl.env.navigator.getGamepads()
269         .filter((g) => !isNull(g))
270         .map(g => g.toJSON());
271 }
272 /**
273 * Registers a callback function to be called when a gamepad is connected.
274 * @param {Function} callback - The function to execute when a gamepad
275 * connects.
276 */
277 onGamepadConnected(callback) {
278     var obj = this;
279     var listener = function(e) {
280         if(e.gamepad) {
281             e.gamepad = e.gamepad.map(g => g.toJSON());
282         }
283         callback(e);
284     };
285     this.jsl.env.context.addEventListener("gamepadconnected", listener);
286     this.jsl.addForCleanup(this, function() {
287         obj.jsl.env.context.removeEventListener("gamepadconnected", listener);
288     });
289 }
290 /**
291 * Registers a callback function to be called when a gamepad is disconnected
292 * @param {Function} callback - The function to execute when a gamepad
293 * disconnects.
294 */
295 onGamepadDisconnected(callback) {
296     var obj = this;
297     var listener = function(e) {
298         if(e.gamepad) {
299             e.gamepad = e.gamepad.map(g => g.toJSON());
300         }
301         callback(e);
302     };
303     this.jsl.env.context.addEventListener("gamepaddisconnected", listener);

```



```
305     this.jsl.addForCleanup(this, function() {
306         obj.jsl.env.context.removeEventListener("gamepaddirconnected", listener)
307         ;
308     });
309 }
310 /**
311 * Retrieves a specific gamepad by its ID.
312 * @param {number} id - The index of the gamepad to retrieve.
313 * @param {number} dt - Data reading interval in milliseconds.
314 * @returns {PRDC_JSLAB_DEVICE_GAMEPAD} The corresponding gamepad object.
315 */
316 getGamepad(id, dt) {
317     return new PRDC_JSLAB_DEVICE_GAMEPAD(this.jsl, id, dt);
318 }
319 /**
320 * Retrieves a list of available webcam (video input) devices.
321 * @returns {Object[]} A promise that resolves to an array of video input
322 devices.
323 */
324 async getWebcams() {
325     var devices = await this.jsl.env.navigator.mediaDevices.enumerateDevices()
326         ;
327     return devices
328         .filter(device => device.kind === 'videoinput')
329         .map(device => device.toJSON());
330 }
331 /**
332 * Retrieves a list of available microphone (audio input) devices.
333 * @returns {Object[]} A promise that resolves to an array of audio input
334 devices.
335 */
336 async getMicrophones() {
337     var devices = await this.jsl.env.navigator.mediaDevices.enumerateDevices()
338         ;
339     return devices
340         .filter(device => device.kind === 'audioinput')
341         .map(device => device.toJSON());
342 }
343 /**
344 * Retrieves a list of available audio output devices.
345 * @returns {Object[]} A promise that resolves to an array of audio output
346 devices.
347 */
348 async getAudioOutputs() {
349     var devices = await this.jsl.env.navigator.mediaDevices.enumerateDevices()
350         ;
351     return devices
352         .filter(device => device.kind === 'audiooutput')
353         .map(device => device.toJSON());
354 }
```

```

353  /**
354   * Opens a new window to display the webcam feed from the specified device.
355   * @param {string} device_id - The unique identifier of the webcam device to
356   *     use.
357   * @returns {Promise<WebcamResult>} An object containing the window instance
358   *     , video element, and media stream.
359   */
360   async webcam(device_id) {
361     var win = await openWindowBlank();
362     win.setTitle('Webcam');
363     win.document.body.innerHTML += '<video id="video"></video>';
364     var dom = win.document.getElementById('video');
365     Object.assign(dom.style, {
366       position: 'absolute',
367       top: '50%',
368       left: '50%',
369       transform: 'translate(-50%, -50%)',
370       width: '100%',
371       height: '100%',
372       objectFit: 'contain'
373     });
374     try {
375       var constraints = {
376         video: { deviceId: { exact: device_id } },
377         audio: false
378       };
379       var stream = await this.jsl.env.navigator.mediaDevices.getUserMedia(
380         constraints);
381     } catch(err) {
382       this.jsl._console.log(err, constraints);
383       this.jsl.env.error('@capture: '+language.string(222));
384     }
385     dom.srcObject = stream;
386     dom.play();
387     this.jsl.addForCleanup(this, function() {
388       stream.getTracks().forEach(track => track.stop());
389       dom.srcObject = null;
390     });
391     return { win, dom, stream };
392   }

393 /**
394  * Initiates webcam video capture.
395  * @param {Object} opts - Configuration options for webcam capture.
396  * @param {function} frameCallback - Callback invoked with each frame's
397  *     image data buffer.
398  * @param {function} [editCallback] - Optional callback to edit each frame
399  *     before processing.
400  */
401 webcamCapture(opts, frameCallback, editCallback) {
402   opts.type = 'webcam';
403   this.capture(opts, frameCallback, editCallback);
404 }

```

```

403  /**
404   * Retrieves desktop sources from the current environment.
405   * @returns {DesktopSource[]} An array of desktop sources.
406   */
407  getDesktopSources() {
408      return this.jsl.env.getDesktopSources();
409  }
410
411 /**
412  * Displays the available desktop sources by generating and injecting HTML
413  * elements for each source.
414  * @returns {void}
415  */
416  showDesktopSources() {
417      var html = '';
418      var sources = this.jsl.env.getDesktopSources();
419      for(source of sources) {
420          var { width, height } = source.thumbnail.getSize();
421          html += '<div style="padding:10px; margin: 10px; border: #ccc 1px solid;
422              border-radius: 5px;"><div><b>Name:</b> ' + source.name + '</div><div><b>Id:</b> ' + source.id + '</div><div><b>DisplayId:</b> ' + source.
423              display_id + '</div></img>
425          </div>';
426      }
427      this.jsl.env.disp(html);
428  }
429
430 /**
431  * Initiates desktop screen capture.
432  * @param {Object} opts - Configuration options for desktop capture.
433  * @param {function} frameCallback - Callback invoked with each frame's
434  *     image data buffer.
435  * @param {function} [editCallback] - Optional callback to edit each frame
436  *     before processing.
437  */
438  desktopCapture(opts, frameCallback, editCallback) {
439      opts.type = 'desktop';
440      this.capture(opts, frameCallback, editCallback);
441  }
442
443 /**
444  * Captures media frames based on the provided options.
445  * @param {Object} opts - Configuration options for capturing.
446  * @param {string} opts.id - The ID of the media source.
447  * @param {string} opts.type - Type of capture ('webcam' or 'desktop').
448  * @param {function} frameCallback - Callback invoked with each frame's
449  *     image data buffer.
450  * @param {function} [editCallback] - Optional callback to edit each frame
451  *     before processing.
452  * @returns {Object} An object containing control functions and resources
453  *     for the capture session.
454  * @returns {function} return.stop - Function to stop the capture.
455  * @returns {CanvasRenderingContext2D} return.ctx - The 2D rendering context
456  *     of the OffscreenCanvas.

```

```

446   * @returns {OffscreenCanvas} return.offscreenCanvas - The OffscreenCanvas
447   * used for rendering frames.
448   * @returns {MediaStreamTrack} return.videoTrack - The video track being
449   * captured.
450   * @returns {MediaStreamTrackProcessor} return.trackProcessor - The
451   * processor for the video track.
452   * @returns {ReadableStreamDefaultReader} return.reader - Reader for the
453   * media stream.
454   */
455   async capture(opts, frameCallback, editCallback) {
456     var { id, type, ...otherOpts } = opts;
457
458     var active = true;
459     var constraints;
460
461     // Define media constraints based on capture type
462     if(type === 'webcam') {
463       constraints = {
464         video: {
465           deviceId: { exact: id },
466           ...otherOpts
467         },
468         audio: false
469       };
470     } else if(type === 'desktop') {
471       constraints = {
472         video: {
473           mandatory: {
474             chromeMediaSource: 'desktop',
475             chromeMediaSourceId: id,
476             ...otherOpts
477           }
478         },
479         audio: false
480       };
481     }
482
483     try {
484       var stream = await this.jsl.env.navigator.mediaDevices.getUserMedia(
485         constraints);
486     } catch(err) {
487       this.jsl._console.log(err);
488       this.jsl._console.log(constraints);
489       this.jsl.env.error('@capture: '+language.string(222));
490     }
491
492     var videoTrack = stream.getVideoTracks()[0];
493     var settings = videoTrack.getSettings();
494
495     var width = settings.width || 1280;

```

```

496   var height = settings.height || 720;
497
498   // Initialize OffscreenCanvas with retrieved width and height
499   var offscreenCanvas = new OffscreenCanvas(width, height);
500   var ctx = offscreenCanvas.getContext('2d', {willReadFrequently: true});
501
502   /**
503    * Stops the frame capture by setting active to false.
504    */
505   function stop() {
506     active = false;
507   }
508
509   /**
510    * Continuously reads frames from the media stream, processes them, and
511    * invokes callbacks.
512    */
513   async function getFrames() {
514     while(active) {
515       var { value, done } = await reader.read();
516       if(done) {
517         break;
518       }
519
520       var frame = value;
521       ctx.drawImage(frame, 0, 0, width, height);
522
523       if(typeof editCallback === 'function') {
524         editCallback(frame, width, height);
525       }
526
527       var imageData = ctx.getImageData(0, 0, width, height);
528       frameCallback(imageData.data.buffer, width, height, frame);
529       frame.close();
530     }
531
532     // Cleanup after capturing frames
533     videoTrack.stop();
534     reader.releaseLock();
535     stream.getTracks().forEach(track => track.stop());
536   }
537
538   // Initialize MediaStreamTrackProcessor and reader
539   var trackProcessor = new MediaStreamTrackProcessor({ track: videoTrack });
540   var reader = trackProcessor.readable.getReader();
541
542   this.jsl.addForCleanup(this, stop);
543
544   // Start processing frames
545   getFrames();
546
547   // Return control functions and resources
548   return { stop, ctx, offscreenCanvas, videoTrack, trackProcessor, reader };
549 }
```

```

550  /**
551   * Gets supported camera resolutions for a specific device.
552   * @param {string} device_id - The camera device ID.
553   * @returns {Promise<Array<Object>>} Supported resolutions.
554   */
555  async getCameraResolutions(device_id) {
556    const resolutions = [];
557    for(const resolution of this._camera_resolutions) {
558      var constraints = {
559        audio: false,
560        video: {
561          deviceId: { exact: device_id },
562          width: { exact: resolution.width },
563          height: { exact: resolution.height }
564        }
565      };
566      try {
567        const stream = await this.jsl.env.navigator.mediaDevices.getUserMedia(
568          constraints);
569        stream.getTracks().forEach(track => track.stop());
570        resolutions.push(resolution);
571      } catch(err) {
572        this.jsl._console.log(err);
573      };
574    }
575    return resolutions;
576  }
577  /**
578   * Displays an audio waveform on the canvas.
579   * @param {string} device_id - The microphone device ID.
580   * @param {number} [fftSize=2048] - FFT size for analysis.
581   * @returns {Object} Controls to stop or reset the waveform.
582   */
583  async showAudioWaveform(device_id, fftSize = 2048) {
584    var obj = this;
585
586    var win = await openCanvas();
587    win.setTitle('Audio Waveform');
588    var draw_loop;
589    var audio_ctx = new AudioContext();
590    var analyser = audio_ctx.createAnalyser();
591    analyser.fftSize = fftSize;
592    var buffer_length = analyser.frequencyBinCount;
593    var data = new Uint8Array(buffer_length);
594
595    var canvas = win.canvas;
596    var canvas_ctx = canvas.getContext("2d");
597    var canvas_width = 1000;
598    var canvas_height = 500;
599    canvas.width = canvas_width;
600    canvas.height = canvas_height;
601    canvas.style.width = '100vw';
602    canvas.style.height = '100vh';
603

```

```
604     canvas.ctx.lineWidth = 1;
605     canvas.ctx.strokeStyle = "#000";
606     var sliceWidth = canvas.width / bufferLength;
607     reset();
608
609     var constraints = {
610       audio: {
611         deviceId: { exact: device_id }
612       }
613     };
614     var stream = await this.jsl.env.navigator.mediaDevices.getUserMedia(
615       constraints);
616     var source = audioCtx.createMediaStreamSource(stream);
617     source.connect(analyser);
618
619     _update();
620
621     function _update() {
622       drawLoop = obj.jsl.context.requestAnimationFrame(function() {
623         _update();
624       });
625
626       analyser.getByteTimeDomainData(data);
627
628       canvas.ctx.clearRect(0, 0, canvasWidth, canvasHeight);
629       canvas.ctx.beginPath();
630
631       let x = 0;
632       for(let i = 0; i < bufferLength; i++) {
633         const v = data[i] / 128.0;
634         const y = v * (canvasHeight / 2);
635
636         if(i === 0) {
637           canvas.ctx.moveTo(x, y);
638         } else {
639           canvas.ctx.lineTo(x, y);
640         }
641
642         x += sliceWidth;
643
644         canvas.ctx.lineTo(canvasWidth, canvasHeight / 2);
645         canvas.ctx.stroke();
646     }
647
648     function stop() {
649       obj.jsl.context.cancelAnimationFrame(drawLoop);
650       reset();
651     }
652
653     function reset() {
654       canvas.ctx.clearRect(0, 0, canvasWidth, canvasHeight);
655       canvas.ctx.beginPath();
656       canvas.ctx.moveTo(0, canvasHeight/2);
657       canvas.ctx.lineTo(canvasWidth, canvasHeight / 2);
```

```

658         canvas.ctx.stroke();
659     }
660
661     this.jsl.addForCleanup(this, stop);
662
663     return { win, stop, reset };
664   }
665 }
666
667 exports.PRDC_JSLAB_LIB_DEVICE = PRDC_JSLAB_LIB_DEVICE;

```

Listing 77 - device.js

```

1  /**
2   * @file JSLAB library figures submodule
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   */
7
8  /**
9   * Class for JSLAB figures submodule.
10  */
11 class PRDC_JSLAB_LIB FIGURES {
12
13 /**
14  * Initializes a new instance of the figures submodule.
15  * @param {Object} js1 Reference to the main JSLAB object.
16  */
17 constructor(js1) {
18   var obj = this;
19   this.jsl = js1;
20
21   this._fonts_registered = false;
22   this._fonts = [];
23   this._fid = 0;
24   this._pid = 0;
25   this._html_figure = this.jsl.env.readFileSync(app_path+/html/html_figure.
26       html').toString();
27
28 /**
29  * Array of open figures.
30  * @type {Array}
31  */
32   this.open_figures = {};
33
34 /**
35  * Current active figure ID.
36  * @type {Number}
37  */
38   this.active_figure = -1;
39
40 /**
41  * Opens or updates a figure with specified options.
42  * @param {Number} id Identifier for the figure.

```

```

43   * @returns {Number} The identifier of the opened or updated figure.
44   */
45 figure(fid) {
46   if(!!(fid >= 0) || !this.open_figures.hasOwnProperty(fid)) {
47     if(!(fid >= 0)) {
48       this._fid += 1;
49       fid = this._fid;
50     }
51     this.open_figures[fid] = new PRDC_JSLAB_PICTURE(this.jsl, fid);
52     this.open_figures[fid].init();
53   } else {
54     this.open_figures[fid].focus();
55   }
56   this._ setActiveFigure(fid);
57   this.jsl.no_ans = true;
58   this.jsl.ignore_output = true;
59   return fid;
60 }
61 /**
62  * Retrieves the figure object associated with the specified figure ID.
63  * @param {string} fid - The identifier of the figure to retrieve.
64  * @returns {(Object|boolean)} The figure object if found, otherwise 'false'.
65  */
66 getFigure(fid) {
67   if(this.open_figures.hasOwnProperty(fid)) {
68     return this.open_figures[fid];
69   } else {
70     return false;
71   }
72 }
73 /**
74  * Retrieves the window of the figure object associated with the specified
75  * figure ID.
76  * @param {string} fid - The identifier of the figure to retrieve.
77  * @returns {(Object|boolean)} The figure object if found, otherwise 'false'.
78  */
79 getFigureWindow(fid) {
80   if(this.open_figures.hasOwnProperty(fid)) {
81     return this.open_figures[fid].win;
82   } else {
83     return false;
84   }
85 }
86 /**
87  * Retrieves current active figure object.
88  * @returns {(Object|boolean)} The figure object if found, otherwise 'false'.
89  */
90 getCurrentFigure() {
91   if(this.open_figures.hasOwnProperty(this.active_figure)) {

```

```
94         return this.open_figures[this.active_figure];
95     } else {
96         return false;
97     }
98 }
99
100 /**
101 * Retrieves current active figure object.
102 * @returns {Object|boolean} The figure object if found, otherwise 'false'.
103 */
104 gcf() {
105     return this.getCurrentFigure();
106 }
107
108 /**
109 * Retrieves the plot object for a specified figure ID.
110 * @param {string} fid - The identifier of the figure to retrieve the plot for.
111 * @returns {Object|boolean} The plot object if it exists, otherwise 'false'.
112 */
113 getPlot(fid) {
114     if(this.open_figures.hasOwnProperty(fid) && this.open_figures[fid].plot) {
115         return this.open_figures[fid].plot;
116     } else {
117         return false;
118     }
119 }
120
121 /**
122 * Retrieves the plot object for a specified figure ID.
123 * @param {string} fid - The identifier of the figure to retrieve the plot for.
124 * @returns {Object|boolean} The plot object if it exists, otherwise 'false'.
125 */
126 getAxes(fid) {
127     return this.getPlot(fid);
128 }
129
130 /**
131 * Retrieves plot from current active figure object.
132 * @returns {Object|boolean} The figure object if found, otherwise 'false'.
133 */
134 getCurrentPlot() {
135     if(this.open_figures.hasOwnProperty(this.active_figure)) {
136         return this.open_figures[this.active_figure].plot;
137     } else {
138         return false;
139     }
140 }
141
142 /**
```

```

143   * Retrieves plot from current active figure object.
144   * @returns {Object|boolean} The figure object if found, otherwise 'false'
145   *
146   gcp() {
147     return this.getCurrentPlot();
148   }
149
150 /**
151  * Retrieves plot from current active figure object.
152  * @returns {Object|boolean} The figure object if found, otherwise 'false'
153  *
154  getCurrentAxes() {
155    return this.getCurrentPlot();
156  }
157
158 /**
159  * Retrieves plot from current active figure object.
160  * @returns {Object|boolean} The figure object if found, otherwise 'false'
161  *
162  gca() {
163    return this.getCurrentPlot();
164  }
165
166 /**
167  * Brings the specified figure to the foreground.
168  * @param {number} fid - The ID of the figure to focus.
169  * @returns {boolean|undefined} - Returns false if the figure ID is invalid.
170  */
171 focusFigure(fid) {
172   if(this.open_figures.hasOwnProperty(fid)) {
173     return this.open_figures[fid].focus();
174   } else {
175     return false;
176   }
177 }
178
179 /**
180  * Sets the size of a specified figure.
181  * @param {number} fid - The ID of the figure.
182  * @param {number} width - The new width of the figure.
183  * @param {number} height - The new height of the figure.
184  * @returns {boolean|undefined} - Returns false if the figure ID is invalid.
185  */
186 setFigureSize(fid, width, height) {
187   if(this.open_figures.hasOwnProperty(fid)) {
188     return this.open_figures[fid].setSize(width, height);
189   } else {
190     return false;
191   }
192 }
193
194 /**

```

```
195 * Sets the position of a specified figure.
196 * @param {number} fid - The ID of the figure.
197 * @param {number} left - The new left position of the figure.
198 * @param {number} top - The new top position of the figure.
199 * @returns {boolean|undefined} - Returns false if the figure ID is invalid.
200 */
201 setFigurePos(fid, left, top) {
202     if(this.open_figures.hasOwnProperty(fid)) {
203         return this.open_figures[fid].setPos(left, top);
204     } else {
205         return false;
206     }
207 }
208 /**
209 * Sets the title of the specified figure.
210 * @param {string} fid - The figure ID.
211 * @param {string} title - The new title for the figure.
212 * @returns {boolean| *} The result of setting the title, or false if the
213     figure does not exist.
214 */
215 setFigureTitle(fid, title) {
216     if(this.open_figures.hasOwnProperty(fid)) {
217         return this.open_figures[fid].setTitle(title);
218     } else {
219         return false;
220     }
221 }
222 /**
223 * Retrieves the size of a specified figure.
224 * @param {number} fid - The ID of the figure.
225 * @returns {Array|boolean} - Returns an array [width, height] or false if
226     the figure ID is invalid.
227 */
228 getSize(fid) {
229     if(this.open_figures.hasOwnProperty(fid)) {
230         return this.open_figures[fid].getSize();
231     } else {
232         return false;
233     }
234 }
235 /**
236 * Retrieves the position of a specified figure.
237 * @param {number} fid - The ID of the figure.
238 * @returns {Array|boolean} - Returns an array [left, top] or false if the
239     figure ID is invalid.
240 */
241 getFigurePos(fid) {
242     if(this.open_figures.hasOwnProperty(fid)) {
243         return this.open_figures[fid].getPos();
244     } else {
245         return false;
246     }
}
```

```

247   }
248
249  /**
250   * Closes a specified figure.
251   * @param {number} fid - The ID of the figure to close.
252   * @returns {boolean|undefined} - Returns false if the figure ID is invalid.
253   */
254  closeFigure(fid) {
255    if(this.open_figures.hasOwnProperty(fid)) {
256      return this.open_figures[fid].close();
257    } else {
258      return false;
259    }
260  }
261
262  /**
263   * Closes a figure or window by its identifier.
264   * @param {number|string} id - The identifier of the figure or window to
265   *   close. Use "all" to close all.
266   * @param {string} [type='figure'] - The type of object to close ('figure'
267   *   or 'window').
268   */
269  close(id, type = 'figure') {
270    if(id === "all") {
271      this.jsl.env.closeWindow(id);
272    } else {
273      if(type === 'window') {
274        this.jsl.env.closeWindow(id);
275      } else if(type === 'figure') {
276        this.jsl.env.closeFigure(id);
277      }
278    }
279    this.jsl.no_ans = true;
280    this.jsl.ignore_output = true;
281  }
282
283  /**
284   * Opens a dialog for saving a figure in various formats.
285   * @param {String} fid - The figure identifier.
286   */
287  async saveFileDialog(fid) {
288    let options = {
289      title: language.currentString(143),
290      defaultPath: fid + '.svg',
291      buttonLabel: language.currentString(143),
292      filters:[
293        {name: 'svg', extensions: ['svg']},
294        {name: 'pdf', extensions: ['pdf']},
295        {name: 'png', extensions: ['png']},
296        {name: 'jpg', extensions: ['jpg', 'jpeg']},
297        {name: 'webp', extensions: ['webp']},
298        {name: 'html', extensions: ['html']}
299      ];
300    }
301    var figure_path = this.jsl.env.showSaveDialogSync(options);

```

```

300     if (figure_path) {
301         await this.saveFigure(fid, figure_path);
302     }
303 }
304
305 /**
306 * Saves a figure to a specified path in various formats.
307 * @param {String} fid - The figure identifier.
308 * @param {String} figure_path - The path where the figure should be saved.
309 * @param {Array} size - Optional dimensions [width, height] to use if
310   saving as a PDF.
311 */
312 async saveFigure(fid, figure_path, size) {
313     var pdf_flag = false;
314     var html_flag = false;
315     var ext = figure_path.split('.').pop();
316     if(['svg', 'pdf', 'png', 'jpg', 'jpeg', 'webp', 'html'].includes(ext)) {
317         if(ext === 'jpg') {
318             ext = 'jpeg';
319         } else if(ext === 'pdf') {
320             pdf_flag = true;
321             ext = 'svg';
322         } else if(ext === 'html') {
323             html_flag = true;
324             ext = 'svg';
325         }
326         var data_url = await this.open_figures[fid].context.plot.toImage(ext,
327             size);
328         var data;
329         if(ext === 'svg') {
330             if(html_flag) {
331                 data = this._makeFigureHTML(fid, data_url);
332             } else {
333                 data = decodeURIComponent(data_url.replace('data:image/svg+xml,', '')) ;
334             }
335             data = data.replace(/\bLatinModern\b/g, "LatinModernMath");
336         } else {
337             data_url = data_url.replace('data:image/png;base64,', '');
338             data_url = data_url.replace('data:image/jpeg;base64,', '');
339             data_url = data_url.replace('data:image/webp;base64,', '');
340             data = new Buffer(data_url, 'base64');
341         }
342         if(pdf_flag) {
343             var pdf_data = await this._svg2pdf(fid, data, size);
344             try {
345                 this.jsl.env.writeFileSync(figure_path, pdf_data);
346             } catch(err) {
347                 this.jsl.env.error('@saveFigure: '+err.stack);
348             }
349         } else {
350             try {
351                 this.jsl.env.writeFileSync(figure_path, data);
352             } catch(err) {

```

```
352         this.jsl.env.error('@saveFigure: '+err.stack);
353     }
354   } else {
355     this.jsl.env.error('@saveFigure: '+language.string(124));
356   }
357 }
358 */
359 /**
360 * Sets the label for the x-axis of the active figure.
361 * @param {String} label - The label for the x-axis.
362 */
363 legend(state) {
364   if(this.active_figure >= 0 && this.open_figures[this.active_figure].plot)
365   {
366     this.open_figures[this.active_figure].plot.legend(state);
367   }
368   this.jsl.no_ans = true;
369   this.jsl.ignore_output = true;
370 }
371 /**
372 * Sets the label for the x-axis of the active figure.
373 * @param {String} label - The label for the x-axis.
374 */
375 xlabel(label) {
376   if(this.active_figure >= 0 && this.open_figures[this.active_figure].plot)
377   {
378     this.open_figures[this.active_figure].plot.xlabel(label);
379   }
380   this.jsl.no_ans = true;
381   this.jsl.ignore_output = true;
382 }
383 /**
384 * Sets the label for the y-axis of the active figure.
385 * @param {String} label - The label for the y-axis.
386 */
387 ylabel(label) {
388   if(this.active_figure >= 0 && this.open_figures[this.active_figure].plot)
389   {
390     this.open_figures[this.active_figure].plot.ylabel(label);
391   }
392   this.jsl.no_ans = true;
393   this.jsl.ignore_output = true;
394 }
395 /**
396 * Sets the label for the z-axis of the active figure.
397 * @param {String} label - The label for the z-axis.
398 */
399 zlabel(label) {
400   if(this.active_figure >= 0 && this.open_figures[this.active_figure].plot)
401   {
402     this.open_figures[this.active_figure].plot.zlabel(label);
```

```
403     }
404     this.jsl.no_ans = true;
405     this.jsl.ignore_output = true;
406   }
407
408 /**
409 * Sets the title of the active figure.
410 * @param {String} label - The title text.
411 */
412 title(label) {
413   if(this.active_figure >= 0 && this.open_figures[this.active_figure].plot)
414   {
415     this.open_figures[this.active_figure].plot.title(label);
416   }
417   this.jsl.no_ans = true;
418   this.jsl.ignore_output = true;
419 }
420
421 /**
422 * Sets the xlim of the active figure.
423 * @param {String} lim - x limits.
424 */
425 xlim(lim) {
426   if(this.active_figure >= 0 && this.open_figures[this.active_figure].plot)
427   {
428     this.open_figures[this.active_figure].plot.xlim(lim);
429   }
430   this.jsl.no_ans = true;
431   this.jsl.ignore_output = true;
432 }
433
434 /**
435 * Sets the ylim of the active figure.
436 * @param {String} lim - y limits.
437 */
438 ylim(lim) {
439   if(this.active_figure >= 0 && this.open_figures[this.active_figure].plot)
440   {
441     this.open_figures[this.active_figure].plot.ylim(lim);
442   }
443
444 /**
445 * Sets the zlim of the active figure.
446 * @param {String} lim - z limits.
447 */
448 zlim(lim) {
449   if(this.active_figure >= 0 && this.open_figures[this.active_figure].plot)
450   {
451     this.open_figures[this.active_figure].plot.zlim(lim);
452   }
453   this.jsl.no_ans = true;
454   this.jsl.ignore_output = true;
```

```
454     }
455
456     /**
457      * Adjusts the view based on azimuth and elevation angles.
458      * @param {number} azimuth - The azimuth angle.
459      * @param {number} elevation - The elevation angle.
460     */
461     view(azimuth, elevation) {
462       if(this.active_figure >= 0 && this.open_figures[this.active_figure].plot)
463         {
464           this.open_figures[this.active_figure].plot.view(azimuth, elevation);
465         }
466       this.jsl.no_ans = true;
467       this.jsl.ignore_output = true;
468     }
469
470     /**
471      * Adjusts the zoom based on zoom factor.
472      * @param {number} factor - The zoom factor.
473     */
474     zoom(factor) {
475       if(this.active_figure >= 0 && this.open_figures[this.active_figure].plot)
476         {
477           this.open_figures[this.active_figure].plot.zoom(factor);
478         }
479       this.jsl.no_ans = true;
480       this.jsl.ignore_output = true;
481     }
482
483     /**
484      * Applies the specified style to the active figure's plot axis.
485      * @param {Object} style - The style configuration to apply to the axis.
486     */
487     axis(style) {
488       if(this.active_figure >= 0 && this.open_figures[this.active_figure].plot)
489         {
490           this.open_figures[this.active_figure].plot.axis(style);
491         }
492       this.jsl.no_ans = true;
493       this.jsl.ignore_output = true;
494     }
495
496     /**
497      * Prints the currently active figure to a file.
498      * @param {String} filename - The name of the file where the figure should
499      *   be printed.
500      * @param {Object} options - Printing options.
501     */
502     async printFigure(filename, options) {
503       if(this.active_figure >= 0 && this.open_figures[this.active_figure].plot)
504         {
505           await this.open_figures[this.active_figure].plot.print(filename, options
506             );
507         }
508       this.jsl.no_ans = true;
```

```

503     this.jsl.ignore_output = true;
504 }
505
506 /**
507 * Plots data on the active figure.
508 * @param {Array} traces - Data traces to plot.
509 * @param {Object} options - Configuration options for plotting.
510 * @returns {Number} The plot identifier.
511 */
512 plot(traces, options) {
513   var fid = this.active_figure;
514   if(options == undefined) {
515     options = {};
516   } else {
517     if(options.hasOwnProperty(fid)) {
518       fid = options.fid;
519     }
520   }
521   this._pid += 1;
522   var id = this._pid;
523
524   fid = this.figure(fid);
525   this.open_figures[fid]._newPlot(id, traces, options);
526
527   this.jsl.no_ans = true;
528   this.jsl.ignore_output = true;
529   return this.open_figures[fid].plot;
530 }
531
532 /**
533 * Updates the language of the text elements within all open figures.
534 */
535 _updateLanguage() {
536   Object.values(this.open_figures).forEach(function(figure) {
537     figure._updateLanguage(false);
538   });
539 }
540
541 /**
542 * Sets the specified figure as the active figure.
543 * @param {string} fid - The identifier of the figure to set as active.
544 */
545 _setActiveFigure(fid) {
546   if(this.open_figures.hasOwnProperty(fid)) {
547     this.active_figure = fid;
548   } else {
549     this.active_figure = -1;
550   }
551 }
552
553 /**
554 * Closes a figure identified by the given ID and updates the active figure
555 * if needed.
556 * @param {string} id - The identifier of the figure to close.
557 */

```

```

557   _closedFigure(fid) {
558     if(this.open_figures.hasOwnProperty(fid)) {
559       var new_fid = -1;
560       if(this.active_figure == fid) {
561         var fids = Object.keys(this.open_figures);
562         var N = fids.length;
563         if(N > 1) {
564           if(fids[N-1] !== fid) {
565             new_fid = fids[N-1];
566           } else {
567             new_fid = fids[N-2];
568           }
569         }
570         this._ setActiveFigure(new_fid);
571       }
572       delete this.open_figures[fid];
573     }
574   }
575
576 /**
577 * Reads and returns font data from a specified path.
578 * @param {String} font_path Path to the font file.
579 * @returns {Buffer} The font data.
580 */
581 _getFontData(font_path) {
582   try {
583     return this.jsl.env.readFileSync(font_path);
584   } catch(err) {
585     this.jsl.env.error('@getFontData: '+err.stack);
586   }
587   return false;
588 }
589 /**
590 * Registers fonts for use in figures.
591 */
592 _registerFonts() {
593   if(!this._fonts_registered) {
594     this._fonts.push(this._getFontData(
595       app_path+'/font/roboto-v20-latin-ext_latin_greek-ext_greek_cyrillic-
596       ext_cyrillic-regular.ttf',
597     ));
598     this._fonts.push(this._getFontData(
599       app_path+'/font/latinmodern-math.otf',
600     ));
601     this._fonts_registered = true;
602   }
603 }
604 /**
605 * Converts SVG data to PDF format.
606 * @param {String} fid - The figure identifier.
607 * @param {String} data - The SVG data to convert.
608 * @param {Array} size - The dimensions [width, height] to use for the PDF.
609 * @returns {Promise<Buffer>} A promise that resolves with the generated PDF
610   data.

```

```

610     */
611     _svg2pdf(fid, data, size) {
612         var obj = this;
613         this._registerFonts();
614         var plot_cont = this.open_figures[fid].dom.querySelector('#figure-content
615             .plot-cont');
616         var width = plot_cont.clientWidth;
617         var height = plot_cont.clientHeight;
618         if(typeof size != 'undefined') {
619             width = size[0];
620             height = size[1];
621         }
622
623         return new Promise(function(resolve) {
624             var doc = new obj.jsl.env.PDFDocument({
625                 size: [width, height]
626             });
627             doc.registerFont('Roboto', obj._fonts[0]);
628             doc.registerFont('LatinModernMath', obj._fonts[1]);
629             obj.jsl.env.SVGtoPDF(doc, data, 0, 0, {
630                 width: width,
631                 height: height,
632                 assumePt: true
633             });
634             var buf = [];
635             doc.on('data', buf.push.bind(buf));
636             doc.on('end', function() {
637                 var pdfData = Buffer.concat(buf);
638                 resolve(pdfData);
639             });
640             doc.end();
641         });
642
643     /**
644      * Generates HTML content for a figure.
645      * @param {String} fid - The figure identifier.
646      * @param {String} data - The data used to generate the HTML.
647      * @returns {String} Generated HTML content.
648     */
649     _makeFigureHTML(fid, data) {
650         var html = this._html_figure.replaceAll('%title%', this.open_figures[fid].
651             dom.title);
652         return html.replace('%image_source%', data);
653     }
654
655     exports.PRDC_JSLAB_LIB FIGURES = PRDC_JSLAB_LIB FIGURES;
656
657     /**
658      * Represents an individual figure within the JSLAB environment, providing
659      * detailed configuration and interaction capabilities.
660     */
661     class PRDC_JSLAB FIGURE {

```



```
662 #jsl;
663
664 /**
665 * Initializes a new instance of a JSLAB figure.
666 * @param {Object} jsl Reference to the main JSLAB object.
667 * @param {Number} id Identifier for the figure.
668 * @param {Object} options Configuration options for the figure.
669 */
670 constructor(jsl, fid) {
671     var obj = this;
672
673     this.#jsl = jsl;
674     this.fid = fid;
675
676     this.wid;
677
678     this.context;
679     this.dom;
680     this.name;
681     this.size;
682     this.position;
683     this.grid = 'on';
684     this.box = 'on';
685     this.hold = 'off';
686     this.zeroline = 'off';
687     this.fig_ready = false;
688     this.opened = false;
689
690     this.plot = undefined;
691
692     this.ready = new Promise((resolve) => {
693         obj._readyResolve = resolve;
694     });
695
696     this.wid = this.#jsl.windows.openWindow('figure.html');
697     this.#jsl.windows.open_windows[this.wid].onClosed = function() {
698         obj.#jsl.figures._closedFigure(obj.fid);
699     }
700 }
701
702 /**
703 * Initializes figure.
704 */
705 async init() {
706     if(!this.opened) {
707         await this.#jsl.windows.open_windows[this.wid].ready;
708         await this._onReady();
709         this.opened = true;
710     }
711 }
712
713 /**
714 * Brings the figure window to the foreground.
715 */
716 async focus() {
```

```
717     await this.#jsl.promiseOrStoped(this.ready);
718     return await this.#jsl.windows.open_windows[this.wid].focus();
719 }
720 /**
721 * Sets the size of the window.
722 * @param {number} width - The desired width of the window.
723 * @param {number} height - The desired height of the window.
724 * @returns {Promise} - Resolves when the window size is set.
725 */
726 async setSize(width, height) {
727     await this.#jsl.promiseOrStoped(this.ready);
728     return await this.#jsl.windows.open_windows[this.wid].setSize(width,
729         height);
730 }
731 /**
732 * Sets the position of the window.
733 * @param {number} left - The desired left position of the window.
734 * @param {number} top - The desired top position of the window.
735 * @returns {Promise} - Resolves when the window position is set.
736 */
737 async setPos(left, top) {
738     await this.#jsl.promiseOrStoped(this.ready);
739     return await this.#jsl.windows.open_windows[this.wid].setPos(left, top);
740 }
741 /**
742 * Sets the title of the current window.
743 * @param {string} title - The new title for the window.
744 * @returns {Promise<*>} A promise that resolves when the title is set.
745 */
746 async setTitle(title) {
747     await this.#jsl.promiseOrStoped(this.ready);
748     return await this.#jsl.windows.open_windows[this.wid].setTitle(title);
749 }
750 /**
751 * Retrieves the size of the window.
752 * @returns {Promise<Array>} - Resolves with an array [width, height].
753 */
754 async getSize() {
755     await this.#jsl.promiseOrStoped(this.ready);
756     return await this.#jsl.windows.open_windows[this.wid].getSize();
757 }
758 /**
759 * Retrieves the position of the window.
760 * @returns {Promise<Array>} - Resolves with an array [left, top].
761 */
762 async getPos() {
763     await this.#jsl.promiseOrStoped(this.ready);
764     return await this.#jsl.windows.open_windows[this.wid].getPos();
765 }
```

```

771 /**
772 * Closes the window.
773 * @returns {Promise} - Resolves when the window is closed.
774 */
775 async close() {
776     await this.#jsl.promiseOrStoped(this.ready);
777     return await this.#jsl.windows.open_windows[this.wid].close();
778 }
779
780 /**
781 * Creates a new plot in the figure.
782 * @param {Number} id Identifier for the new plot.
783 * @param {Array} traces Data traces for the plot.
784 * @param {Object} options Plot configuration options.
785 */
786 _newPlot(id, traces, options) {
787     if(this.plot) {
788         this.plot.remove();
789     }
790     this.plot = new PRDC_JSLAB_PLOT(this.#jsl, this.fid, id, traces, options);
791     if(this.fig_ready) {
792         this.plot._onFigureReady();
793     }
794 }
795
796 /**
797 * Method called when the figure is ready. Initializes interactive elements
798 * within the figure's DOM.
799 * @param {Element} dom The DOM element associated with the figure.
800 */
801 async _onReady() {
802     var obj = this;
803     this.fig_ready = true;
804     this.win = this.#jsl.windows.open_windows[this.wid];
805     this.context = this.win.context;
806
807     this.dom = this.context.document;
808
809     this.context.addEventListener("resize", function() {
810         if(obj.#jsl.figures.open_figures.hasOwnProperty(obj.fid)) {
811             obj._onResize();
812         }
813     });
814
815     this.dom.title = "Figure " + this.fid + " - JSLAB | PR-DC";
816
817     // Menu showing
818     var menu_button = this.dom.getElementById('figure-menu-button');
819     var menu = this.dom.getElementById('figure-menu-container');
820
821     menu_button.addEventListener('click', function (e) {
822         e.stopPropagation();
823         menu.classList.toggle('active');
824         menu_button.classList.toggle('active');
825     });

```

```

825
826     this.dom.addEventListener('click', function (e) {
827         if (!menu.contains(e.target) && !menu_button.contains(e.target)) {
828             menu.classList.remove('active');
829             menu_button.classList.remove('active');
830         }
831     });
832
833     var interval;
834     menu.addEventListener('mouseenter', function() {
835         clearInterval(interval);
836         menu.classList.add('hovered');
837         menu_button.classList.add('hovered');
838     });
839
840     menu.addEventListener('mouseleave', function() {
841         interval = setTimeout(function() {
842             menu.classList.remove('hovered');
843             menu_button.classList.remove('hovered');
844         }, 300);
845     });
846
847 // Menu buttons
848 this.dom.getElementById('save-as-menu')
849     .addEventListener('click', function() {
850         obj.#jsl.figures.saveFileDialog(obj.fid);
851     });
852 this.dom.getElementById('zoom-menu')
853     .addEventListener('click', function() {
854         var btn = obj.dom.querySelector('a[data-attr="dragmode"][data-val="zoom"]');
855         if(btn) {
856             btn.click();
857         }
858     });
859 this.dom.getElementById('zoom-in-menu')
860     .addEventListener('click', function() {
861         var btn = obj.dom.querySelector('a[data-attr="zoom"][data-val="in"]');
862         if(btn) {
863             btn.click();
864         }
865     });
866 this.dom.getElementById('zoom-out-menu')
867     .addEventListener('click', function() {
868         var btn = obj.dom.querySelector('a[data-attr="zoom"][data-val="out"]');
869         if(btn) {
870             btn.click();
871         }
872     });
873 this.dom.getElementById('pan-menu')
874     .addEventListener('click', function() {
875         var btn = obj.dom.querySelector('a[data-attr="dragmode"][data-val="pan"]');
876         if(btn) {
877             btn.click();

```

```

878     }
879   });
880   this.dom.getElementById('rotate-menu')
881     .addEventListener('click', function() {
882       var btn = obj.dom.querySelector('a[data-attr="dragmode"]的数据-val="orbit'
883         ']');
884       if(btn) {
885         btn.click();
886       }
887     });
888   this.dom.getElementById('fit-menu')
889     .addEventListener('click', function() {
890       var btn = obj.dom.querySelector('a[data-attr="zoom"]的数据-val="auto"]');
891       if(btn) {
892         btn.click();
893       }
894     });
895   this.dom.getElementById('reset-menu')
896     .addEventListener('click', function() {
897       obj.#jsl.ploter.updatePlotLayout(obj.plot.fid);
898     });
899   this.dom.getElementById('pan-menu-3d')
900     .addEventListener('click', function() {
901       var btn = obj.dom.querySelector('a[data-attr="scene.dragmode"]的数据-val
902         ="pan"]');
903       if(btn) {
904         btn.click();
905       }
906     });
907   this.dom.getElementById('rotate-menu-3d')
908     .addEventListener('click', function() {
909       var btn = obj.dom.querySelector('a[data-attr="scene.dragmode"]数据-val
910         ="orbit"]');
911       if(btn) {
912         btn.click();
913       }
914     });
915   this.dom.getElementById('fit-menu-3d')
916     .addEventListener('click', function() {
917       var btn = obj.dom.querySelector('a[data-attr="resetDefault"]');
918       if(btn) {
919         btn.click();
920       }
921     });
922   this.dom.getElementById('reset-menu-3d')
923     .addEventListener('click', function() {
924       obj.#jsl.ploter.updatePlotLayout(obj.plot.fid);
925     });
926   this._readyResolve(true);
927   if(this.plot) {
928     await this.plot._onFigureReady();
929   }

```

```

930  /**
931   * Handles figure resize events by updating the plot layout to fit the new
932   * dimensions.
933   */
934   _onResize() {
935     if(this.plot) {
936       this.plot._onResize();
937     }
938   }
939
940 /**
941  * Represents an individual plot within the JSLAB environment, holding
942  * configuration details and facilitating interaction with the plot.
943  */
944 class PRDC_JSLAB_PLOT {
945
946   #jsl;
947
948 /**
949  * Constructs a PRDC_JSLAB_PLOT instance with specified plot data and
950  * configuration.
951  * @param {Object} js1 Reference to the main JSLAB object.
952  * @param {Number} fid Identifier for the figure containing this plot.
953  * @param {Number} id Unique identifier for this plot.
954  * @param {Array} traces Data traces to be displayed in the plot.
955  * @param {Object} options Configuration options for the plot.
956  */
957 constructor(jsl, fid, id, traces, options) {
958   var obj = this;
959
960   this.#jsl = js1;
961   this.fid = fid;
962   this.id = id;
963   this.traces = traces;
964   this.options = options;
965
966   this.size;
967   this.position;
968
969   this.title_val;
970   this.xlabel_val;
971   this.ylabel_val;
972   this.zlabel_val;
973   this.xlim_val;
974   this.ylim_val;
975   this.zlim_val;
976   this.view_val = [37.5+180, 30];
977   this.zoom_val = 1;
978   this.axis_style_val;
979   this.legend_state;
980
981   this.plot_ready = false;
982   this.lim_update = false;

```

```
982     this.ready = new Promise((resolve) => {
983         obj._readyResolve = resolve;
984     });
985 }
986
987 /**
988 * Sets the label for the x-axis of the plot.
989 * @param {String} label Label text for the x-axis.
990 */
991 legend(state) {
992     this.legend_state = state;
993     if(this.plot_ready) {
994         this.#jsl.ploter.updatePlotLayout(this.fid);
995     }
996 }
997
998 /**
999 * Sets the label for the x-axis of the plot.
1000 * @param {String} label Label text for the x-axis.
1001 */
1002 xlabel(label) {
1003     this.xlabel_val = label;
1004     if(this.plot_ready) {
1005         this.#jsl.ploter.updatePlotLayout(this.fid);
1006     }
1007 }
1008
1009 /**
1010 * Sets the label for the y-axis of the plot.
1011 * @param {String} label Label text for the y-axis.
1012 */
1013 ylabel(label) {
1014     this.ylabel_val = label;
1015     if(this.plot_ready) {
1016         this.#jsl.ploter.updatePlotLayout(this.fid);
1017     }
1018 }
1019
1020 /**
1021 * Sets the label for the z-axis of the plot.
1022 * @param {String} label Label text for the z-axis.
1023 */
1024 zlabel(label) {
1025     this.zlabel_val = label;
1026     if(this.plot_ready) {
1027         this.#jsl.ploter.updatePlotLayout(this.fid);
1028     }
1029 }
1030
1031 /**
1032 * Sets the title of the plot.
1033 * @param {String} label Title text for the plot.
1034 */
1035 title(label) {
1036     this.title_val = label;
```

```
1037     if(this.plot_ready) {
1038         this.#jsl.ploter.updatePlotLayout(this.fid);
1039     }
1040 }
1041
1042 /**
1043 * Sets the limits for the x-axis of the plot.
1044 * @param {String} lim Limits for the x-axis.
1045 */
1046 xlim(lim) {
1047     this.lim_update = true;
1048     this.xlim_val = lim;
1049     if(this.plot_ready) {
1050         this.#jsl.ploter.updatePlotLayout(this.fid);
1051     }
1052 }
1053
1054 /**
1055 * Sets the limits for the y-axis of the plot.
1056 * @param {String} lim Limits for the y-axis.
1057 */
1058 ylim(lim) {
1059     this.lim_update = true;
1060     this.ylim_val = lim;
1061     if(this.plot_ready) {
1062         this.#jsl.ploter.updatePlotLayout(this.fid);
1063     }
1064 }
1065
1066 /**
1067 * Sets the limits for the z-axis of the plot.
1068 * @param {String} lim Limits for the z-axis.
1069 */
1070 zlim(lim) {
1071     this.lim_update = true;
1072     this.zlim_val = lim;
1073     if(this.plot_ready) {
1074         this.#jsl.ploter.updatePlotLayout(this.fid);
1075     }
1076 }
1077
1078 /**
1079 * Adjusts the view based on azimuth and elevation angles.
1080 * @param {number} azimuth - The azimuth angle.
1081 * @param {number} elevation - The elevation angle.
1082 */
1083 view(azimuth, elevation) {
1084     this.view_val = [azimuth, elevation];
1085     if(this.plot_ready) {
1086         this.#jsl.ploter.updatePlotLayout(this.fid);
1087     }
1088 }
1089
1090 /**
1091 * Adjusts the zoom based on factor.
```

```

1092     * @param {number} factor - The zoom factor .
1093     */
1094   zoom(factor) {
1095     this.zoom_val = factor;
1096     if(this.plot_ready) {
1097       this.#jsl.ploter.updatePlotLayout(this.fid);
1098     }
1099   }
1100
1101 /**
1102  * Sets the axis style value and updates the plot layout if the plot is
1103  * ready.
1104  * @param {Object} style - The style configuration to set for the axis.
1105  */
1106 axis(style) {
1107   this.axis_style_val = style;
1108   if(this.plot_ready) {
1109     this.#jsl.ploter.updatePlotLayout(this.fid);
1110   }
1111
1112 /**
1113  * Adds a print job to the queue and prints it if the system is ready.
1114  * @param {String} filename - The filename for the print job.
1115  * @param {Object} options - Options for the print job.
1116  */
1117 async print(filename, options) {
1118   await this.#jsl.promiseOrStoped(this.ready);
1119   var type = 'png';
1120   var size;
1121   if(options) {
1122     if(options.type) {
1123       type = options.type;
1124     }
1125     if(options.size) {
1126       size = options.size;
1127     }
1128   }
1129   await this.#jsl.figures.saveFigure(this.fid, filename+'.'+type, size);
1130 }
1131
1132 /**
1133  * Updates plot data by delegating to the 'updateData' method.
1134  * @param {Object} traces - The trace data to be updated in the plot.
1135  * @param {number} N - The data length or index for updating the plot.
1136  */
1137 update(traces, N) {
1138   this.#jsl.ploter.updateData(this.fid, traces, N);
1139 }
1140
1141 /**
1142  * Removes the plot from the figure, cleaning up any resources associated
1143  * with it.
1144  */
remove() {

```

```

1145     if(this.plot_ready) {
1146         this.#jsl.ploter.remove(this.fid);
1147     }
1148 }
1149 /**
1150 * Called when the figure containing this plot is ready, allowing for final
1151 * adjustments or updates before displaying.
1152 */
1153 async _onFigureReady() {
1154     await this.#jsl.ploter.plot(this.fid);
1155     this.plot_ready = true;
1156     this.#jsl.ploter.updatePlotLayout(this.fid);
1157     this._readyResolve(true);
1158 }
1159 /**
1160 * Handles plot resize events, updating the plot layout to accommodate new
1161 * dimensions.
1162 */
1163 _onResize() {
1164     this.#jsl.ploter.onResize(this.fid);
1165 }
1166 }
```

Listing 78 - figures.js

```

1 /**
2  * @file JSLAB library file system submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 /**
9  * Class for JSLAB file system submodule.
10 */
11 class PRDC_JSLAB_LIB_FILE_SYSTEM {
12
13 /**
14 * Initializes a new instance of the file system submodule.
15 * @param {Object} js1 Reference to the main JSLAB object.
16 */
17 constructor(js1) {
18     var obj = this;
19     this.js1 = js1;
20 }
21
22 /**
23 * Reads the content of a file at the specified path.
24 * @param {string} file_path Path to the file.
25 * @returns {((Buffer|string|false)} The content of the file or false in case
26 *          of an error.
27 */
28 readFile(...args) {
29     return this.js1.env.readFileSync(...args);
30 }
```

```
29     }
30
31     /**
32      * Writes data to a specified file synchronously. This method should
33      * overwrite the file if it already exists.
34      * @param {string} file_path The path to the file where data will be written
35      *
36      * @param {Buffer|string} data The data to write to the file.
37      * @returns {boolean} Returns true if the file was written successfully,
38      *                   false if an error occurred.
39      */
40     writeFile(... args) {
41       return this.jsl.env.writeFileSync(... args);
42     }
43
44     /**
45      * Deletes a specified file synchronously.
46      * @param {string} file_path The path to the file that should be deleted.
47      * @returns {boolean} Returns true if the file was deleted successfully,
48      *                   false if an error occurred.
49      */
50     deleteFile(file_path) {
51       return this.jsl.env.rmSync(file_path);
52     }
53
54     /**
55      * Deletes a specified file synchronously.
56      * @param {string} file_path The path to the file that should be deleted.
57      * @returns {boolean} Returns true if the file was deleted successfully,
58      *                   false if an error occurred.
59      */
60     deleteDir(file_path) {
61       return this.jsl.env.rmSync(file_path);
62     }
63
64     /**
65      * Moves a file from source to destination.
66      * @param {string} source - The path to the source file.
67      * @param {string} destination - The path to the destination file.
68      */
69     moveFile(source, destination) {
70       if(comparePaths(source, destination)) {
71         return true;
72       }
73       try {
74         this.jsl.env.writeFileSync(source, destination);
75         this.jsl.env.rmSync(source);
76         return true;
77       } catch(err) {
78         this.jsl.error(`@moveFile: ${err}`);
79       }
80     }
81
82     /**
83      * Lists files in a specified folder, optionally filtering by extension.
84      */
85     listFiles(folder, filter) {
86       const files = this.jsl.env.readdirSync(folder);
87       const filtered = files.filter(file => file.endsWith(filter));
88       return filtered;
89     }
90   }
91 }
```

```

79   * @param {string} folder Path to the folder.
80   * @param {string} ext File extension filter.
81   * @returns {string[]|void} Array of file paths matching the extension in
82   * the specified folder.
83   */
84   filesInFolder(folder, ext) {
85     var obj = this;
86     var files = this.jsl.env.readdirSync(folder);
87     if(Array.isArray(files)) {
88       return files
89         .filter(function(file) {
90           if(!ext) return file.includes('.');
91           return file.endsWith('.' + ext);
92         })
93         .map(function(file) { return obj.jsl.env.pathJoin(folder, file); });
94     } else {
95       this.jsl.env.error('@filesInFolder: '+language.string(128)+': ' + folder
96         );
97     }
98     return false;
99   }
100 /**
101  * Lists all files in a specified folder
102  * @param {string} folder Path to the folder.
103  * @returns {string[]|void} Array of file names.
104  */
105 allFilesInFolder(folder) {
106   return this.jsl.env.readdirSync(folder).reduce((acc, file) => {
107     const file_path = this.jsl.env.pathJoin(folder, file);
108     return this.jsl.env.isDirectory(file_path)
109       ? acc.concat(this.allFilesInFolder(file_path))
110       : acc.concat(file);
111   }, []);
112 }
113 /**
114  * Opens a dialog for the user to choose a file, synchronously.
115  * @param {Object} options Configuration options for the dialog.
116  * @returns {string|string[]} The selected file path(s) or an empty array if
117  * canceled.
118  */
119 chooseFile(options) {
120   var file_path = this.jsl.env.showOpenDialogSync(options);
121   if(file_path === undefined) {
122     this.jsl.env.error('@chooseFileSync: '+language.string(126));
123     return [];
124   }
125   return file_path;
126 }
127 /**
128  * Retrieves a default path based on a specified type.
129  * @param {string} type Type of the default path (e.g., 'root', 'documents')
130  *

```

```
130     * @returns {string} The default path for the specified type.
131     */
132     getDefaultPath(type) {
133         return this.jsl.env.getDefaultPath(type);
134     }
135
136     /**
137      * Opens the specified folder in the system's file manager.
138      * @param {string} filepath Path to the folder.
139      */
140     openFolder(filepath) {
141         this.jsl.env.openFolder(filepath);
142     }
143
144     /**
145      * Creates a directory at the specified path if it does not already exist.
146      * This method delegates the directory creation task to the environment's
147      * makeDirectory function.
148      * @param {string} directory - The path where the directory will be created.
149      * @returns {boolean} True if the directory was successfully created or
150      * already exists, false if an error occurred.
151      */
152     makeDirectory(directory) {
153         return this.jsl.env.makeDirectory(directory);
154     }
155
156     /**
157      * Alias for makeDirectory. Creates a directory at the specified path if it
158      * does not already exist.
159      * This method delegates the directory creation task to the environment's
160      * makeDirectory function.
161      * @param {string} directory - The path where the directory will be created.
162      * @returns {boolean} True if the directory was successfully created or
163      * already exists, false if an error occurred.
164      */
165     mkdir(directory) {
166         return this.jsl.env.makeDirectory(directory);
167     }
168
169     /**
170      * Opens the specified directory in the system's file manager. Alias for 'openFolder'.
171      * @param {string} filepath Path to the directory.
172      */
173     openDir(filepath) {
174         this.jsl.env.openDir(filepath);
175     }
176
177     /**
178      * Shows the specified folder in the system's file manager. Alias for 'openFolder'.
179      * @param {string} filepath Path to the folder.
180      */
181     showFolder(filepath) {
182         this.jsl.env.openFolder(filepath);
```

```

178 }
179
180 /**
181 * Shows the specified directory in the system's file manager. Alias for 'openDir'.
182 * @param {string} filepath Path to the directory.
183 */
184 showDir(filepath) {
185   this.jsl.env.openDir(filepath);
186 }
187
188 /**
189 * Opens the program's root folder in the system's file manager.
190 */
191 openProgramFolder() {
192   this.jsl.env.openFolder(this.jsl.env.getDefaultPath('root'));
193 }
194
195 /**
196 * Shows the specified file in its containing folder within the system's file manager.
197 * @param {string} filepath Path to the file.
198 */
199 showFileInFolder(filepath) {
200   this.jsl.env.showFileInFolder(filepath);
201 }
202
203 /**
204 * Shows the specified file in its containing directory within the system's file manager. Alias for 'showFileInFolder'.
205 * @param {string} filepath Path to the file.
206 */
207 showFileInDir(filepath) {
208   this.jsl.env.showFileInDir(filepath);
209 }
210
211 /**
212 * Reads a CSV file and returns a promise that resolves with the parsed data
213 * @param {string} filePath - Path to the CSV file.
214 * @param {string} delimiter - Delimiter used in the CSV file (e.g., ',', ',',
215 *                           ';', '\t').
216 * @returns {Array<Object>} - Parsed CSV data as an array of objects.
217 */
218 readcsv(filePath, delimiter = ',', hasHeader = false) {
219   var data = this.jsl.env.readFileSync(filePath, 'utf-8');
220   var lines = data.split('\n').filter(function(line) { return line.trim() !== '' });
221   var headers = lines[0].split(delimiter).map(function(header) { return header.trim() });
222   var result = [];
223
224   if(hasHeader) {
225     // If there is a header, parse as objects
226     var headers = lines[0].split(delimiter).map(header => header.trim());

```

```
226     for (let i = 1; i < lines.length; i++) {
227         var row = lines[i].split(delimiter).map(function(cell) { return cell.trim(); });
228         var row_object = {};
229         headers.forEach(function(header, index) {
230             row_object[header] = row[index];
231         });
232         result.push(row_object);
233     }
234 } else {
235     // If no header, parse as arrays
236     for(let i = 0; i < lines.length; i++) {
237         var row = lines[i].split(delimiter).map(function(cell) { return cell.trim(); });
238         result.push(row);
239     }
240 }
241
242     return result;
243 }
244
245 /**
246 * Checks if the specified file exists.
247 * @param {string} file - The path to the file to check.
248 */
249 checkFile(file) {
250     return this.jsl.env.checkFile(file);
251 }
252
253 /**
254 * Checks if the specified file exists.
255 * @param {string} file - The path to the file to check.
256 */
257 existFile(file) {
258     return this.checkFile(file);
259 }
260
261 /**
262 * Checks if the specified directory exists.
263 * @param {string} directory - The path to the directory to check.
264 */
265 checkDirectory(directory) {
266     return this.jsl.env.checkDirectory(directory);
267 }
268
269 /**
270 * Checks if the specified directory exists.
271 * @param {string} directory - The path to the directory to check.
272 */
273 existDirectory(directory) {
274     return this.checkDirectory(directory);
275 }
276
277 /**
278 * Recursively copies a directory from the source path to the destination
```

```

    path.
279  * @param {string} src - The source directory path.
280  * @param {string} dest - The destination directory path.
281  */
282 copyDir(src, dest) {
283   // Check if the source directory exists
284   if(!this.jsl.env.checkDirectory(src)) {
285     this.jsl.env.error('@copyDir: '+language.string(173));
286   }
287
288   // Create the destination directory if it doesn't exist
289   this.jsl.env.makeDirectory(dest);
290
291   // Read all the files and directories in the source directory
292   const entries = this.jsl.env.readdirSync(src, { withFileTypes: true });
293
294   // Iterate through each entry (file or directory)
295   for(const entry of entries) {
296     const src_path = this.jsl.env.pathJoin(src, entry.name);
297     const dest_path = this.jsl.env.pathJoin(dest, entry.name);
298
299     if(entry.isDirectory()) {
300       // Recursively copy directories
301       this.copyDir(src_path, dest_path);
302     } else {
303       // Copy files
304       this.jsl.env.writeFileSync(src_path, dest_path);
305     }
306   }
307 }
308
309 /**
310  * Copies a folder from the source path to the destination path.
311  * @param {string} src - The source folder path.
312  * @param {string} dest - The destination folder path.
313  */
314 copyFolder(src, dest) {
315   return this.copyDir(src, dest);
316 }
317
318 /**
319  * Copies a directory from the source path to the destination path.
320  * @param {string} src - The source directory path.
321  * @param {string} dest - The destination directory path.
322  */
323 cp(src, dest) {
324   return this.copyDir(src, dest);
325 }
326
327 /**
328  * Copies a 7z archive from the source path to the destination path,
329  * extracts it, and removes the archive.
330  * @param {string} src - The source 7z archive path.
331  * @param {string} dest - The destination directory path.
332  */

```

```

332     copyDir7z(src, dest) {
333         var name = this.jsl.path.pathFileName(src);
334         var ext = this.jsl.path.pathExtName(src);
335         var filePath = this.jsl.env.pathJoin(dest, name + ext);
336
337         this.jsl.env.copyFileSync(src, filePath);
338         this.jsl.env.execSync(`.${this.jsl.env.bin7zip} x "${filePath}" -o"${dest}"
339                         -y`);
340         this.jsl.env.rmSync(filePath);
341     }
342 }
343 exports.PRDC_JSLAB_LIB_FILE_SYSTEM = PRDC_JSLAB_LIB_FILE_SYSTEM;

```

Listing 79 - file-system.js

```

1  /**
2   * @file JSLAB library format submodule
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   */
7
8  /**
9   * Class for JSLAB format submodule.
10  */
11 class PRDC_JSLAB_LIB_FORMAT {
12
13  /**
14   * Initializes a new instance of the format submodule.
15   * @param {Object} jsl Reference to the main JSLAB object.
16   */
17 constructor(jsl) {
18     var obj = this;
19     this.jsl = jsl;
20 }
21
22 /**
23  * Retrieves the MIME type based on the file extension.
24  * @param {string} filePath - The path to the file.
25  * @returns {string} The corresponding MIME type.
26  */
27 getContentType(filePath) {
28     const mime_types = {
29         // Text files
30         '.html': 'text/html',
31         '.htm': 'text/html',
32         '.js': 'text/javascript',
33         '.mjs': 'text/javascript',
34         '.css': 'text/css',
35         '.json': 'application/json',
36         '.txt': 'text/plain',
37         '.xml': 'application/xml',
38
39         // Image files
40         '.png': 'image/png',

```

```

41   '.jpg': 'image/jpeg',
42   '.jpeg': 'image/jpeg',
43   '.gif': 'image/gif',
44   '.bmp': 'image/bmp',
45   '.webp': 'image/webp',
46   '.svg': 'image/svg+xml',
47   '.ico': 'image/x-icon',
48
49 // Audio files
50   '.mp3': 'audio/mpeg',
51   '.wav': 'audio/wav',
52   '.ogg': 'audio/ogg',
53   '.m4a': 'audio/mp4',
54
55 // Video files
56   '.mp4': 'video/mp4',
57   '.avi': 'video/x-msvideo',
58   '.mov': 'video/quicktime',
59   '.wmv': 'video/x-ms-wmv',
60   '.flv': 'video/x-flv',
61   '.webm': 'video/webm',
62   '.mkv': 'video/x-matroska',
63
64 // Application files
65   '.pdf': 'application/pdf',
66   '.zip': 'application/zip',
67   '.rar': 'application/vnd.rar',
68   '.7z': 'application/x-7z-compressed',
69   '.tar': 'application/x-tar',
70   '.gz': 'application/gzip',
71   '.exe': 'application/vnd.microsoft.portable-executable',
72   '.msi': 'application/x-msdownload',
73   '.doc': 'application/msword',
74   '.docx': 'application/vnd.openxmlformats-officedocument.wordprocessingml
              .document',
75   '.xls': 'application/vnd.ms-excel',
76   '.xlsx': 'application/vnd.openxmlformats-officedocument.spreadsheetml
              .sheet',
77   '.ppt': 'application/vnd.ms-powerpoint',
78   '.pptx': 'application/vnd.openxmlformats-officedocument.presentationml
              .presentation',
79   '.eot': 'application/vnd.ms-fontobject',
80   '.ttf': 'font/ttf',
81   '.woff': 'font/woff',
82   '.woff2': 'font/woff2',
83
84 // Model files
85   '.glb': 'model/gltf-binary',
86   '.gltf': 'model/gltf+json',
87   '.obj': 'application/octet-stream', // Common for OBJ, but can vary
88   '.fbx': 'application/octet-stream',
89
90 // Others
91   '.csv': 'text/csv',
92   '.md': 'text/markdown',

```

```

93     '.apk': 'application/vnd.android.package-archive',
94     '.iso': 'application/x-iso9660-image',
95     '.sh': 'application/x-sh',
96     '.bat': 'application/x-msdownload',
97     '.php': 'application/x-httpd-php',
98     '.asp': 'application/x-aspx',
99     '.aspx': 'application/x-aspx',
100    '.jsp': 'application/java-archive',
101    '.rb': 'application/x-ruby',
102    '.py': 'application/x-python-code',
103    '.swift': 'application/x-swift',
104    '.lua': 'application/x-lua',
105  };
106
107  const ext = String(this.jsl.env.pathExtName(file_path)).toLowerCase();
108  return mime_types[ext] || 'application/octet-stream';
109 }
110
111 /**
112 * Formats the given byte count into a readable string.
113 * @param {Number} bytes Number of bytes.
114 * @param {Number} [decimals=2] Number of decimal places to include in the
115 *   formatted string.
116 * @returns {String} Formatted bytes string with appropriate unit.
117 */
118 formatBytes(bytes, decimals = 2) {
119   if(!+bytes) return '0 Bytes';
120
121   const k = 1024;
122   const dm = decimals < 0 ? 0 : decimals;
123   const units = ['Bytes', 'KiB', 'MiB', 'GiB', 'TiB', 'PiB', 'EiB', 'ZiB', 'YiB'];
124
125   const i = Math.floor(Math.log(bytes) / Math.log(k));
126
127   return `${parseFloat((bytes / Math.pow(k, i)).toFixed(dm))} ${units[i]}`;
128 }
129 /**
130 * Formats the given bits per second (bps) into a readable string.
131 * @param {Number} bps Number of bits per second.
132 * @param {Number} [decimals=2] Number of decimal places to include in the
133 *   formatted string.
134 * @returns {String} Formatted bits per second string with appropriate unit.
135 */
136 formatBPS(bps, decimals = 2) {
137   if(!+bps) return '0 bps';
138
139   const k = 1000;
140   const dm = decimals < 0 ? 0 : decimals;
141   const units = ["bps", "kbps", "Mbps", "Gbps", "Tbps", "Pbps", "Ebps", "Zbps",
142     "", "Ybps"];
143
144   const i = Math.floor(Math.log(bps) / Math.log(k));

```

```

144     return `${parseFloat((bps / Math.pow(k, i)).toFixed(dm))} ${units[i]}`;
145   }
146
147   /**
148    * Formats a number with metric prefixes (k, M, G, etc.) based on its value.
149    * @param {Number} number The number to format.
150    * @param {Number} [decimals=2] Number of decimal places to include in the
151    * formatted string.
152    * @returns {String} Formatted number string with metric prefix.
153    */
154   formatPrefix(number, decimals = 2) {
155     if(!+number) return '0';
156
157     const k = 1000;
158     const dm = decimals < 0 ? 0 : decimals;
159     const units = [ "", "k", "M", "G", "T", "P", "E", "Z", "Y"];
160
161     const i = Math.floor(Math.log(number) / Math.log(k));
162
163     return `${parseFloat((number / Math.pow(k, i)).toFixed(dm))} ${units[i]}`;
164   }
165   /**
166    * Formats a number to a specified number of decimal places.
167    * @param {Number} number The number to format.
168    * @param {Number} [decimals=2] The number of decimal places.
169    * @returns {String} The formatted number as a string.
170    */
171   formatNum(number, decimals = 2) {
172     return Number(number).toFixed(decimals);
173   }
174
175   /**
176    * Formats a floating-point number to have a fixed number of significant
177    * digits.
178    * @param {number} number - The number to format.
179    * @param {number} [digits=1] - The number of significant digits.
180    * @returns {string} The formatted number as a string.
181    */
182   formatFloatDigits(number, digits = 1) {
183     var precision = digits-(number.toString().split("."))[0].length;
184     if(precision < 0) {
185       precision = 0;
186     }
187     return round(number, precision, true);
188   }
189   /**
190    * Clips a number to a specified value based on a condition.
191    * @param {number} number - The number to clip.
192    * @param {number} control_number - The reference number for the clipping
193    * condition.
194    * @param {number} clip_value - The value to clip to.
195    * @param {boolean} [direction=true] - The direction of clipping (true for
196    * max, false for min).
197    * @returns {number} The clipped number.

```

```
195  /*
196   * clip(number, control_number, clip_value, direction = true) {
197   *     if((direction && number >= control_number) || (!direction && number <=
198   *         control_number)) {
199   *         number = clip_value;
200   *     }
201   *     return number;
202   */
203 /**
204  * Clips a height value to a specified range.
205  * @param {number} value - The value to clip.
206  * @param {number} [min=0] - The minimum value of the range.
207  * @param {number} [max=100] - The maximum value of the range.
208  * @returns {number} The clipped height value.
209 */
210 clipHeight(value, min = 0, max = 100) {
211     value = Number(value);
212     if(isNaN(value)) {
213         value = min;
214     } else if(value > max) {
215         value = max;
216     } else if(value < min) {
217         value = min;
218     }
219     return value;
220 }
221 /**
222  * Retrieves the field names (keys) of the given object.
223  * @param {Object} obj - The object from which to extract keys.
224  * @returns {string[]} An array containing the keys of the object.
225  */
226 fieldnames(obj) {
227     return Object.keys(obj);
228 }
229 /**
230  * Replaces all occurrences of a specified substring within a string.
231  * @param {string} str - The original string.
232  * @param {string} old_string - The substring to be replaced.
233  * @param {string} new_string - The substring to replace with.
234  * @returns {string} The resulting string after replacements.
235  */
236 strrep(str, old_string, new_string) {
237     return str.replaceAll(old_string, new_string);
238 }
239 /**
240  * Clips a value to a specified range.
241  * @param {number} value - The value to clip.
242  * @param {number} [min=0] - The minimum value of the range.
243  * @param {number} [max=100] - The maximum value of the range.
244  * @returns {number} The clipped value.
245  */
246
```

```

249 clipVal(value, min = 0, max = 100) {
250   value = Number(value);
251   if(isNaN(value)) {
252     value = min;
253   } else if(value > max) {
254     value = max;
255   } else if(value < min) {
256     value = min;
257   }
258   return value;
259 }
260
261 /**
262 * Rounds a number to a fixed number of decimal places, but only if it's a
263 * floating point number.
264 * @param {Number} value The value to round.
265 * @param {Number} p The number of decimal places to round to.
266 * @returns {Number} The rounded number, or the original number if it wasn't
267 * a float.
268 */
269toFixedIf(value, p) {
270   return +parseFloat(value).toFixed(p);
271 }
272
273 /**
274 * Provides a replacer function for JSON.stringify() to prevent circular
275 * references.
276 * @returns {Function} A replacer function that can be used with JSON.
277 * stringify to handle circular references.
278 */
279getCircularReplacer() {
280   const seen = new WeakSet();
281   return function(key, value) {
282     if(typeof value === 'object' && value !== null) {
283       if(seen.has(value)) {
284         return;
285       }
286       seen.add(value);
287     }
288     return value;
289   };
290 }
291
292 /**
293 * Determines if the provided value is NaN.
294 * @param {*} value - The value to check.
295 * @returns {boolean} True if the value is NaN, false otherwise.
296 */
297isNaN(value) {
298   if.isArray(value)) {
299     return mathjs_isNaN(value);
300   } else {
301     return this.jsl._isNaN(value);
302   }
303 }
```

```
300
301  /**
302   * Determines if the provided value is an object.
303   * @param {*} value - The value to check.
304   * @param {boolean} [ignore_array=false] - Whether to consider arrays as not
305   * objects.
306   * @returns {boolean} True if the value is an object, false otherwise.
307   */
308  isObject(value, ignore_array) {
309    if(ignore_array && Array.isArray(value)) {
310      return false;
311    }
312    return typeof value === 'object' && value !== null;
313  }
314
315  /**
316   * Determines if the provided value is a number.
317   * @param {*} value - The value to check.
318   * @returns {boolean} True if the value is a number, false otherwise.
319   */
320  isNumber(value) {
321    return typeof value === 'number';
322  }
323
324  /**
325   * Determines if the provided value is a string.
326   * @param {*} value - The value to check.
327   * @returns {boolean} True if the value is a string, false otherwise.
328   */
329  isString(value) {
330    return typeof value === 'string';
331  }
332
333  /**
334   * Checks if a string is empty or contains only whitespace.
335   * @param {string} str - The string to check.
336   * @returns {boolean} - True if the string is empty or contains only
337   * whitespace, otherwise false.
338   */
339  isEmptyString(str) {
340    return typeof str === 'string' && str.trim().length === 0;
341  }
342
343  /**
344   * Determines if the provided value is a function.
345   * @param {*} value - The value to check
346   * @returns {boolean} True if the value is a function, false otherwise.
347   */
348  isFunction(value) {
349    return typeof value === 'function';
350  }
351
352  /**
353   * Determines if the provided value is an array.
354   * @param {*} value - The value to check.
355   */
```

```

353     * @returns {boolean} True if the value is an array , false otherwise.
354     */
355 isArray(value) {
356     return Object.prototype.toString.call(value) === '[object Array]';
357 }
358
359 /**
360  * Determines if the provided value is null.
361  * @param {*} value - The value to check.
362  * @returns {boolean} True if the value is null , false otherwise.
363  */
364 isNull(value) {
365     return value === null;
366 }
367
368 /**
369  * Checks if an array is empty.
370  * @param {Array} array - The array to check.
371  * @returns {boolean} True if the array is empty , false otherwise.
372  */
373 isEmpty(array) {
374     return !(typeof array === 'object' && array.length != 0);
375 }
376
377 /**
378  * Checks if the given value(s) are infinite.
379  * @param {number|number[]} value - The value or array of values to check.
380  * @returns {boolean|boolean[]} - Returns true if infinite , otherwise false.
381  *           Returns an array of booleans if input is an array.
382  */
383 isInfinity(value) {
384     if(Array.isArray(value)) {
385         return value.map((v) => v === Infinity || v === -Infinity);
386     } else {
387         return value === Infinity || value === -Infinity;
388     }
389 }
390
391 /**
392  * Checks if a variable is numeric.
393  * @param {*} variable - The variable to check.
394  * @returns {boolean} True if the variable is numeric , false otherwise.
395  */
396 isNumeric(variable) {
397     return !isNaN(parseFloat(variable)) && isFinite(variable);
398 }
399
400 /**
401  * Checks if the specified object has the given key.
402  * @param {Object} object - The object to check for the presence of the key .
403  * @param {string} key - The key to check for in the object .
404  * @returns {boolean} - True if the object has the key , false otherwise .
405  */
406 hasKey(object , key) {
407     return object.hasOwnProperty(key);

```

```

407     }
408
409     /**
410      * Determines if the provided value is undefined.
411      * @param {*} value - The value to check.
412      * @returns {boolean} True if the value is undefined, false otherwise.
413      */
414     isUndefined(value) {
415       return typeof value === 'undefined';
416     }
417
418     /**
419      * Checks if a string is a valid UUID.
420      * @param {string} str The string to check.
421      * @returns {boolean} True if the string is a valid UUID, false otherwise.
422      */
423     isUUID(str) {
424       var uuid_pattern = /^[0-9a-f]{8}-[0-9a-f]{4}-[0-5][0-9a-f]{3}-[089ab][0-9a-f]{3}-[0-9a-f]{12}$/i;
425       return uuid_pattern.test(str);
426     }
427
428     /**
429      * Normalizes an angle to the range of -180 to 180 degrees.
430      * @param {number} angle - The angle to normalize.
431      * @returns {number} The normalized angle.
432      */
433     normalizeAngle(angle) {
434       return ((angle + 180) % 360 + 360) % 360 - 180;
435     }
436
437     /**
438      * Normalizes an angle to the range of 0 to 360 degrees.
439      * @param {number} angle - The angle to normalize.
440      * @returns {number} The normalized angle.
441      */
442     normalizeAngle360(angle) {
443       return (angle % 360 + 360) % 360;
444     }
445
446     /**
447      * Normalizes latitude to the range of -90 to 90 degrees with specified
448      * precision.
449      * @param {number} latitude - The latitude to normalize.
450      * @param {number} precision - The precision of the normalization.
451      * @returns {number} The normalized latitude.
452      */
453     normalizeLat(latitude, precision) {
454       return round(Math.max(-90, Math.min(90, latitude)), precision);
455     }
456
457     /**
458      * Normalizes longitude to the range of -180 to 180 degrees with specified
459      * precision.
460      * @param {number} longitude - The longitude to normalize.

```

```

459   * @param {number} precision - The precision of the normalization.
460   * @returns {number} The normalized longitude.
461   */
462 normalizeLon(longitude, precision) {
463   return round(normalizeAngle(longitude), precision);
464 }
465
466 /**
467  * Checks a value for undefined and returns an empty string if it is
468  * undefined, otherwise returns the value.
469  * @param {*} val - The value to check.
470  * @returns {*} The original value if not undefined, otherwise an empty
471  * string.
472  */
473 checkUndefined(val) {
474   if(val === undefined) {
475     return '';
476   } else {
477     return val;
478   }
479 }
480 /**
481  * Pretty-prints data, converting it into a more readable format for display
482  * . Handles strings, objects, and other data types.
483  * @param {*} data The data to pretty-print.
484  * @returns {Array} An array containing the pretty-printed data as a string
485  * and a boolean indicating if the data was an object.
486  */
487 prettyPrint(data) {
488   if(Array.isArray(data)) {
489     return [this.safeStringify(data, 5), true];
490   } else if(typeof data === 'object' && typeof data.toPrettyString ===
491     'function') {
492     return [data.toPrettyString(), false];
493   } else if(typeof data === 'object' && typeof data.toSafeJSON === 'function')
494   {
495     return [this.safeStringify(data), true];
496   } else if(typeof data === 'string') {
497     return ["'" + data.replace(/\n/g, '<br>') + "'", false];
498   } else if(typeof data === 'object') {
499     return [this.safeStringify(data), true];
500   } else {
501     return [String(data), false];
502   }
503 }
504
505 /**
506  * Converts an object to a JSON string if it is an object, otherwise returns
507  * the object as is.
508  * @param {*} object - The object to stringify.
509  * @returns {string|*} The JSON string representation of the object or the
510  * object itself if not an object.
511  */

```

```

506   stringify(object) {
507     if(typeof object.toPrettyString === 'function') {
508       return object.toPrettyString();
509     }
510     return JSON.stringify(object);
511   }
512
513 /**
514  * Safely serializes an object into a JSON string, handling circular
515  * references and deep structures, with depth control.
516  * It also escapes strings to prevent HTML injection.
517  * @param {Object} data - The object to stringify.
518  * @param {number} [depth_limit=3] - The maximum depth to traverse in the
519  * object, beyond which the traversal is stopped.
520  * @returns {string} A JSON string representation of the object, with
521  * special handling for deep objects, circular references, and HTML
522  * escaping of strings.
523 */
524 safeStringify(data, depth_limit = 3) {
525   if(typeof data !== 'object') {
526     return data;
527   }
528   if(typeof data.toPrettyString === 'function') {
529     return data.toPrettyString();
530   }
531   if(typeof data.toSafeJSON === 'function') {
532     return data.toSafeJSON();
533   }
534   if(data.hasOwnProperty('_safeStringifyDepth')) {
535     depth_limit = data._safeStringifyDepth;
536     delete data._safeStringifyDepth;
537   }
538   const seen = new WeakSet();
539
540   function escapeString(str) {
541     return str.replace(/[\&<>"\'=\\]/g, function (s) {
542       return ({
543         '&': '&amp;',
544         '<': '&lt;',
545         '>': '&gt;',
546         '"': '&quot;',
547         "'": '&#39;',
548         '=': '&#x60;',
549         '/': '&#x2F;',
550       })[s];
551     });
552   }
553
554   function helper(value, depth, path) {
555     if(depth > depth_limit) {
556       return '{...}';
557     }
558   }
559
560   function helper(value, depth, path) {
561     if(depth > depth_limit) {
562       return '{...}';
563     }
564   }
565
566   function helper(value, depth, path) {
567     if(depth > depth_limit) {
568       return '{...}';
569     }
570   }
571
572   function helper(value, depth, path) {
573     if(depth > depth_limit) {
574       return '{...}';
575     }
576   }
577
578   function helper(value, depth, path) {
579     if(depth > depth_limit) {
580       return '{...}';
581     }
582   }
583
584   function helper(value, depth, path) {
585     if(depth > depth_limit) {
586       return '{...}';
587     }
588   }
589
590   function helper(value, depth, path) {
591     if(depth > depth_limit) {
592       return '{...}';
593     }
594   }
595
596   function helper(value, depth, path) {
597     if(depth > depth_limit) {
598       return '{...}';
599     }
600   }
601
602   function helper(value, depth, path) {
603     if(depth > depth_limit) {
604       return '{...}';
605     }
606   }
607
608   function helper(value, depth, path) {
609     if(depth > depth_limit) {
610       return '{...}';
611     }
612   }
613
614   function helper(value, depth, path) {
615     if(depth > depth_limit) {
616       return '{...}';
617     }
618   }
619
620   function helper(value, depth, path) {
621     if(depth > depth_limit) {
622       return '{...}';
623     }
624   }
625
626   function helper(value, depth, path) {
627     if(depth > depth_limit) {
628       return '{...}';
629     }
630   }
631
632   function helper(value, depth, path) {
633     if(depth > depth_limit) {
634       return '{...}';
635     }
636   }
637
638   function helper(value, depth, path) {
639     if(depth > depth_limit) {
640       return '{...}';
641     }
642   }
643
644   function helper(value, depth, path) {
645     if(depth > depth_limit) {
646       return '{...}';
647     }
648   }
649
650   function helper(value, depth, path) {
651     if(depth > depth_limit) {
652       return '{...}';
653     }
654   }
655
656   function helper(value, depth, path) {
657     if(depth > depth_limit) {
658       return '{...}';
659     }
660   }
661
662   function helper(value, depth, path) {
663     if(depth > depth_limit) {
664       return '{...}';
665     }
666   }
667
668   function helper(value, depth, path) {
669     if(depth > depth_limit) {
670       return '{...}';
671     }
672   }
673
674   function helper(value, depth, path) {
675     if(depth > depth_limit) {
676       return '{...}';
677     }
678   }
679
680   function helper(value, depth, path) {
681     if(depth > depth_limit) {
682       return '{...}';
683     }
684   }
685
686   function helper(value, depth, path) {
687     if(depth > depth_limit) {
688       return '{...}';
689     }
690   }
691
692   function helper(value, depth, path) {
693     if(depth > depth_limit) {
694       return '{...}';
695     }
696   }
697
698   function helper(value, depth, path) {
699     if(depth > depth_limit) {
700       return '{...}';
701     }
702   }
703
704   function helper(value, depth, path) {
705     if(depth > depth_limit) {
706       return '{...}';
707     }
708   }
709
710   function helper(value, depth, path) {
711     if(depth > depth_limit) {
712       return '{...}';
713     }
714   }
715
716   function helper(value, depth, path) {
717     if(depth > depth_limit) {
718       return '{...}';
719     }
720   }
721
722   function helper(value, depth, path) {
723     if(depth > depth_limit) {
724       return '{...}';
725     }
726   }
727
728   function helper(value, depth, path) {
729     if(depth > depth_limit) {
730       return '{...}';
731     }
732   }
733
734   function helper(value, depth, path) {
735     if(depth > depth_limit) {
736       return '{...}';
737     }
738   }
739
740   function helper(value, depth, path) {
741     if(depth > depth_limit) {
742       return '{...}';
743     }
744   }
745
746   function helper(value, depth, path) {
747     if(depth > depth_limit) {
748       return '{...}';
749     }
750   }
751
752   function helper(value, depth, path) {
753     if(depth > depth_limit) {
754       return '{...}';
755     }
756   }
757
758   function helper(value, depth, path) {
759     if(depth > depth_limit) {
760       return '{...}';
761     }
762   }
763
764   function helper(value, depth, path) {
765     if(depth > depth_limit) {
766       return '{...}';
767     }
768   }
769
770   function helper(value, depth, path) {
771     if(depth > depth_limit) {
772       return '{...}';
773     }
774   }
775
776   function helper(value, depth, path) {
777     if(depth > depth_limit) {
778       return '{...}';
779     }
780   }
781
782   function helper(value, depth, path) {
783     if(depth > depth_limit) {
784       return '{...}';
785     }
786   }
787
788   function helper(value, depth, path) {
789     if(depth > depth_limit) {
790       return '{...}';
791     }
792   }
793
794   function helper(value, depth, path) {
795     if(depth > depth_limit) {
796       return '{...}';
797     }
798   }
799
800   function helper(value, depth, path) {
801     if(depth > depth_limit) {
802       return '{...}';
803     }
804   }
805
806   function helper(value, depth, path) {
807     if(depth > depth_limit) {
808       return '{...}';
809     }
810   }
811
812   function helper(value, depth, path) {
813     if(depth > depth_limit) {
814       return '{...}';
815     }
816   }
817
818   function helper(value, depth, path) {
819     if(depth > depth_limit) {
820       return '{...}';
821     }
822   }
823
824   function helper(value, depth, path) {
825     if(depth > depth_limit) {
826       return '{...}';
827     }
828   }
829
830   function helper(value, depth, path) {
831     if(depth > depth_limit) {
832       return '{...}';
833     }
834   }
835
836   function helper(value, depth, path) {
837     if(depth > depth_limit) {
838       return '{...}';
839     }
840   }
841
842   function helper(value, depth, path) {
843     if(depth > depth_limit) {
844       return '{...}';
845     }
846   }
847
848   function helper(value, depth, path) {
849     if(depth > depth_limit) {
850       return '{...}';
851     }
852   }
853
854   function helper(value, depth, path) {
855     if(depth > depth_limit) {
856       return '{...}';
857     }
858   }
859
860   function helper(value, depth, path) {
861     if(depth > depth_limit) {
862       return '{...}';
863     }
864   }
865
866   function helper(value, depth, path) {
867     if(depth > depth_limit) {
868       return '{...}';
869     }
870   }
871
872   function helper(value, depth, path) {
873     if(depth > depth_limit) {
874       return '{...}';
875     }
876   }
877
878   function helper(value, depth, path) {
879     if(depth > depth_limit) {
880       return '{...}';
881     }
882   }
883
884   function helper(value, depth, path) {
885     if(depth > depth_limit) {
886       return '{...}';
887     }
888   }
889
890   function helper(value, depth, path) {
891     if(depth > depth_limit) {
892       return '{...}';
893     }
894   }
895
896   function helper(value, depth, path) {
897     if(depth > depth_limit) {
898       return '{...}';
899     }
900   }
901
902   function helper(value, depth, path) {
903     if(depth > depth_limit) {
904       return '{...}';
905     }
906   }
907
908   function helper(value, depth, path) {
909     if(depth > depth_limit) {
910       return '{...}';
911     }
912   }
913
914   function helper(value, depth, path) {
915     if(depth > depth_limit) {
916       return '{...}';
917     }
918   }
919
920   function helper(value, depth, path) {
921     if(depth > depth_limit) {
922       return '{...}';
923     }
924   }
925
926   function helper(value, depth, path) {
927     if(depth > depth_limit) {
928       return '{...}';
929     }
930   }
931
932   function helper(value, depth, path) {
933     if(depth > depth_limit) {
934       return '{...}';
935     }
936   }
937
938   function helper(value, depth, path) {
939     if(depth > depth_limit) {
940       return '{...}';
941     }
942   }
943
944   function helper(value, depth, path) {
945     if(depth > depth_limit) {
946       return '{...}';
947     }
948   }
949
950   function helper(value, depth, path) {
951     if(depth > depth_limit) {
952       return '{...}';
953     }
954   }
955
956   function helper(value, depth, path) {
957     if(depth > depth_limit) {
958       return '{...}';
959     }
960   }
961
962   function helper(value, depth, path) {
963     if(depth > depth_limit) {
964       return '{...}';
965     }
966   }
967
968   function helper(value, depth, path) {
969     if(depth > depth_limit) {
970       return '{...}';
971     }
972   }
973
974   function helper(value, depth, path) {
975     if(depth > depth_limit) {
976       return '{...}';
977     }
978   }
979
980   function helper(value, depth, path) {
981     if(depth > depth_limit) {
982       return '{...}';
983     }
984   }
985
986   function helper(value, depth, path) {
987     if(depth > depth_limit) {
988       return '{...}';
989     }
990   }
991
992   function helper(value, depth, path) {
993     if(depth > depth_limit) {
994       return '{...}';
995     }
996   }
997
998   function helper(value, depth, path) {
999     if(depth > depth_limit) {
1000       return '{...}';
1001     }
1002   }
1003
1004   function helper(value, depth, path) {
1005     if(depth > depth_limit) {
1006       return '{...}';
1007     }
1008   }
1009
1010   function helper(value, depth, path) {
1011     if(depth > depth_limit) {
1012       return '{...}';
1013     }
1014   }
1015
1016   function helper(value, depth, path) {
1017     if(depth > depth_limit) {
1018       return '{...}';
1019     }
1020   }
1021
1022   function helper(value, depth, path) {
1023     if(depth > depth_limit) {
1024       return '{...}';
1025     }
1026   }
1027
1028   function helper(value, depth, path) {
1029     if(depth > depth_limit) {
1030       return '{...}';
1031     }
1032   }
1033
1034   function helper(value, depth, path) {
1035     if(depth > depth_limit) {
1036       return '{...}';
1037     }
1038   }
1039
1040   function helper(value, depth, path) {
1041     if(depth > depth_limit) {
1042       return '{...}';
1043     }
1044   }
1045
1046   function helper(value, depth, path) {
1047     if(depth > depth_limit) {
1048       return '{...}';
1049     }
1050   }
1051
1052   function helper(value, depth, path) {
1053     if(depth > depth_limit) {
1054       return '{...}';
1055     }
1056   }
1057
1058   function helper(value, depth, path) {
1059     if(depth > depth_limit) {
1060       return '{...}';
1061     }
1062   }
1063
1064   function helper(value, depth, path) {
1065     if(depth > depth_limit) {
1066       return '{...}';
1067     }
1068   }
1069
1070   function helper(value, depth, path) {
1071     if(depth > depth_limit) {
1072       return '{...}';
1073     }
1074   }
1075
1076   function helper(value, depth, path) {
1077     if(depth > depth_limit) {
1078       return '{...}';
1079     }
1080   }
1081
1082   function helper(value, depth, path) {
1083     if(depth > depth_limit) {
1084       return '{...}';
1085     }
1086   }
1087
1088   function helper(value, depth, path) {
1089     if(depth > depth_limit) {
1090       return '{...}';
1091     }
1092   }
1093
1094   function helper(value, depth, path) {
1095     if(depth > depth_limit) {
1096       return '{...}';
1097     }
1098   }
1099
1100   function helper(value, depth, path) {
1101     if(depth > depth_limit) {
1102       return '{...}';
1103     }
1104   }
1105
1106   function helper(value, depth, path) {
1107     if(depth > depth_limit) {
1108       return '{...}';
1109     }
1110   }
1111
1112   function helper(value, depth, path) {
1113     if(depth > depth_limit) {
1114       return '{...}';
1115     }
1116   }
1117
1118   function helper(value, depth, path) {
1119     if(depth > depth_limit) {
1120       return '{...}';
1121     }
1122   }
1123
1124   function helper(value, depth, path) {
1125     if(depth > depth_limit) {
1126       return '{...}';
1127     }
1128   }
1129
1130   function helper(value, depth, path) {
1131     if(depth > depth_limit) {
1132       return '{...}';
1133     }
1134   }
1135
1136   function helper(value, depth, path) {
1137     if(depth > depth_limit) {
1138       return '{...}';
1139     }
1140   }
1141
1142   function helper(value, depth, path) {
1143     if(depth > depth_limit) {
1144       return '{...}';
1145     }
1146   }
1147
1148   function helper(value, depth, path) {
1149     if(depth > depth_limit) {
1150       return '{...}';
1151     }
1152   }
1153
1154   function helper(value, depth, path) {
1155     if(depth > depth_limit) {
1156       return '{...}';
1157     }
1158   }
1159
1160   function helper(value, depth, path) {
1161     if(depth > depth_limit) {
1162       return '{...}';
1163     }
1164   }
1165
1166   function helper(value, depth, path) {
1167     if(depth > depth_limit) {
1168       return '{...}';
1169     }
1170   }
1171
1172   function helper(value, depth, path) {
1173     if(depth > depth_limit) {
1174       return '{...}';
1175     }
1176   }
1177
1178   function helper(value, depth, path) {
1179     if(depth > depth_limit) {
1180       return '{...}';
1181     }
1182   }
1183
1184   function helper(value, depth, path) {
1185     if(depth > depth_limit) {
1186       return '{...}';
1187     }
1188   }
1189
1190   function helper(value, depth, path) {
1191     if(depth > depth_limit) {
1192       return '{...}';
1193     }
1194   }
1195
1196   function helper(value, depth, path) {
1197     if(depth > depth_limit) {
1198       return '{...}';
1199     }
1200   }
1201
1202   function helper(value, depth, path) {
1203     if(depth > depth_limit) {
1204       return '{...}';
1205     }
1206   }
1207
1208   function helper(value, depth, path) {
1209     if(depth > depth_limit) {
1210       return '{...}';
1211     }
1212   }
1213
1214   function helper(value, depth, path) {
1215     if(depth > depth_limit) {
1216       return '{...}';
1217     }
1218   }
1219
1220   function helper(value, depth, path) {
1221     if(depth > depth_limit) {
1222       return '{...}';
1223     }
1224   }
1225
1226   function helper(value, depth, path) {
1227     if(depth > depth_limit) {
1228       return '{...}';
1229     }
1230   }
1231
1232   function helper(value, depth, path) {
1233     if(depth > depth_limit) {
1234       return '{...}';
1235     }
1236   }
1237
1238   function helper(value, depth, path) {
1239     if(depth > depth_limit) {
1240       return '{...}';
1241     }
1242   }
1243
1244   function helper(value, depth, path) {
1245     if(depth > depth_limit) {
1246       return '{...}';
1247     }
1248   }
1249
1250   function helper(value, depth, path) {
1251     if(depth > depth_limit) {
1252       return '{...}';
1253     }
1254   }
1255
1256   function helper(value, depth, path) {
1257     if(depth > depth_limit) {
1258       return '{...}';
1259     }
1260   }
1261
1262   function helper(value, depth, path) {
1263     if(depth > depth_limit) {
1264       return '{...}';
1265     }
1266   }
1267
1268   function helper(value, depth, path) {
1269     if(depth > depth_limit) {
1270       return '{...}';
1271     }
1272   }
1273
1274   function helper(value, depth, path) {
1275     if(depth > depth_limit) {
1276       return '{...}';
1277     }
1278   }
1279
1280   function helper(value, depth, path) {
1281     if(depth > depth_limit) {
1282       return '{...}';
1283     }
1284   }
1285
1286   function helper(value, depth, path) {
1287     if(depth > depth_limit) {
1288       return '{...}';
1289     }
1290   }
1291
1292   function helper(value, depth, path) {
1293     if(depth > depth_limit) {
1294       return '{...}';
1295     }
1296   }
1297
1298   function helper(value, depth, path) {
1299     if(depth > depth_limit) {
1300       return '{...}';
1301     }
1302   }
1303
1304   function helper(value, depth, path) {
1305     if(depth > depth_limit) {
1306       return '{...}';
1307     }
1308   }
1309
1310   function helper(value, depth, path) {
1311     if(depth > depth_limit) {
1312       return '{...}';
1313     }
1314   }
1315
1316   function helper(value, depth, path) {
1317     if(depth > depth_limit) {
1318       return '{...}';
1319     }
1320   }
1321
1322   function helper(value, depth, path) {
1323     if(depth > depth_limit) {
1324       return '{...}';
1325     }
1326   }
1327
1328   function helper(value, depth, path) {
1329     if(depth > depth_limit) {
1330       return '{...}';
1331     }
1332   }
1333
1334   function helper(value, depth, path) {
1335     if(depth > depth_limit) {
1336       return '{...}';
1337     }
1338   }
1339
1340   function helper(value, depth, path) {
1341     if(depth > depth_limit) {
1342       return '{...}';
1343     }
1344   }
1345
1346   function helper(value, depth, path) {
1347     if(depth > depth_limit) {
1348       return '{...}';
1349     }
1350   }
1351
1352   function helper(value, depth, path) {
1353     if(depth > depth_limit) {
1354       return '{...}';
1355     }
1356   }
1357
1358   function helper(value, depth, path) {
1359     if(depth > depth_limit) {
1360       return '{...}';
1361     }
1362   }
1363
1364   function helper(value, depth, path) {
1365     if(depth > depth_limit) {
1366       return '{...}';
1367     }
1368   }
1369
1370   function helper(value, depth, path) {
1371     if(depth > depth_limit) {
1372       return '{...}';
1373     }
1374   }
1375
1376   function helper(value, depth, path) {
1377     if(depth > depth_limit) {
1378       return '{...}';
1379     }
1380   }
1381
1382   function helper(value, depth, path) {
1383     if(depth > depth_limit) {
1384       return '{...}';
1385     }
1386   }
1387
1388   function helper(value, depth, path) {
1389     if(depth > depth_limit) {
1390       return '{...}';
1391     }
1392   }
1393
1394   function helper(value, depth, path) {
1395     if(depth > depth_limit) {
1396       return '{...}';
1397     }
1398   }
1399
1400   function helper(value, depth, path) {
1401     if(depth > depth_limit) {
1402       return '{...}';
1403     }
1404   }
1405
1406   function helper(value, depth, path) {
1407     if(depth > depth_limit) {
1408       return '{...}';
1409     }
1410   }
1411
1412   function helper(value, depth, path) {
1413     if(depth > depth_limit) {
1414       return '{...}';
1415     }
1416   }
1417
1418   function helper(value, depth, path) {
1419     if(depth > depth_limit) {
1420       return '{...}';
1421     }
1422   }
1423
1424   function helper(value, depth, path) {
1425     if(depth > depth_limit) {
1426       return '{...}';
1427     }
1428   }
1429
1430   function helper(value, depth, path) {
1431     if(depth > depth_limit) {
1432       return '{...}';
1433     }
1434   }
1435
1436   function helper(value, depth, path) {
1437     if(depth > depth_limit) {
1438       return '{...}';
1439     }
1440   }
1441
1442   function helper(value, depth, path) {
1443     if(depth > depth_limit) {
1444       return '{...}';
1445     }
1446   }
1447
1448   function helper(value, depth, path) {
1449     if(depth > depth_limit) {
1450       return '{...}';
1451     }
1452   }
1453
1454   function helper(value, depth, path) {
1455     if(depth > depth_limit) {
1456       return '{...}';
1457     }
1458   }
1459
1460   function helper(value, depth, path) {
1461     if(depth > depth_limit) {
1462       return '{...}';
1463     }
1464   }
1465
1466   function helper(value, depth, path) {
1467     if(depth > depth_limit) {
1468       return '{...}';
1469     }
1470   }
1471
1472   function helper(value, depth, path) {
1473     if(depth > depth_limit) {
1474       return '{...}';
1475     }
1476   }
1477
1478   function helper(value, depth, path) {
1479     if(depth > depth_limit) {
1480       return '{...}';
1481     }
1482   }
1483
1484   function helper(value, depth, path) {
1485     if(depth > depth_limit) {
1486       return '{...}';
1487     }
1488   }
1489
1490   function helper(value, depth, path) {
1491     if(depth > depth_limit) {
1492       return '{...}';
1493     }
1494   }
1495
1496   function helper(value, depth, path) {
1497     if(depth > depth_limit) {
1498       return '{...}';
1499     }
1500   }
1501
1502   function helper(value, depth, path) {
1503     if(depth > depth_limit) {
1504       return '{...}';
1505     }
1506   }
1507
1508   function helper(value, depth, path) {
1509     if(depth > depth_limit) {
1510       return '{...}';
1511     }
1512   }
1513
1514   function helper(value, depth, path) {
1515     if(depth > depth_limit) {
1516       return '{...}';
1517     }
1518   }
1519
1520   function helper(value, depth, path) {
1521     if(depth > depth_limit) {
1522       return '{...}';
1523     }
1524   }
1525
1526   function helper(value, depth, path) {
1527     if(depth > depth_limit) {
1528       return '{...}';
1529     }
1530   }
1531
1532   function helper(value, depth, path) {
1533     if(depth > depth_limit) {
1534       return '{...}';
1535     }
1536   }
1537
1538   function helper(value, depth, path) {
1539     if(depth > depth_limit) {
1540       return '{...}';
1541     }
1542   }
1543
1544   function helper(value, depth, path) {
1545     if(depth > depth_limit) {
1546       return '{...}';
1547     }
1548   }
1549
1550   function helper(value, depth, path) {
1551     if(depth > depth_limit) {
1552       return '{...}';
1553     }
1554   }
1555
1556   function helper(value, depth, path) {
1557     if(depth > depth_limit) {
1558       return '{...}';
1559     }
1560   }
1561
1562   function helper(value, depth, path) {
1563     if(depth > depth_limit) {
1564       return '{...}';
1565     }
1566   }
1567
1568   function helper(value, depth, path) {
1569     if(depth > depth_limit) {
1570       return '{...}';
1571     }
1572   }
1573
1574   function helper(value, depth, path) {
1575     if(depth > depth_limit) {
1576       return '{...}';
1577     }
1578   }
1579
1580   function helper(value, depth, path) {
1581     if(depth > depth_limit) {
1582       return '{...}';
1583     }
1584   }
1585
1586   function helper(value, depth, path) {
1587     if(depth > depth_limit) {
1588       return '{...}';
1589     }
1590   }
1591
1592   function helper(value, depth, path) {
1593     if(depth > depth_limit) {
1594       return '{...}';
1595     }
1596   }
1597
1598   function helper(value, depth, path) {
1599     if(depth > depth_limit) {
1600       return '{...}';
1601     }
1602   }
1603
1604   function helper(value, depth, path) {
1605     if(depth > depth_limit) {
1606       return '{...}';
1607     }
1608   }
1609
1610   function helper(value, depth, path) {
1611     if(depth > depth_limit) {
1612       return '{...}';
1613     }
1614   }
1615
1616   function helper(value, depth, path) {
1617     if(depth > depth_limit) {
1618       return '{...}';
1619     }
1620   }
1621
1622   function helper(value, depth, path) {
1623     if(depth > depth_limit) {
1624       return '{...}';
1625     }
1626   }
1627
1628   function helper(value, depth, path) {
1629     if(depth > depth_limit) {
1630       return '{...}';
1631     }
1632   }
1633
1634   function helper(value, depth, path) {
1635     if(depth > depth_limit) {
1636       return '{...}';
1637     }
1638   }
1639
1640   function helper(value, depth, path) {
1641     if(depth > depth_limit) {
1642       return '{...}';
1643     }
1644   }
1645
1646   function helper(value, depth, path) {
1647     if(depth > depth_limit) {
1648       return '{...}';
1649     }
1650   }
1651
1652   function helper(value, depth, path) {
1653     if(depth > depth_limit) {
1654       return '{...}';
1655     }
1656   }
1657
1658   function helper(value, depth, path) {
1659     if(depth > depth_limit) {
1660       return '{...}';
1661     }
1662   }
1663
1664   function helper(value, depth, path) {
1665     if(depth > depth_limit) {
1666       return '{...}';
1667     }
1668   }
1669
1670   function helper(value, depth, path) {
1671     if(depth > depth_limit) {
1672       return '{...}';
1673     }
1674   }
1675
1676   function helper(value, depth, path) {
1677     if(depth > depth_limit) {
1678       return '{...}';
1679     }
1680   }
1681
1682   function helper(value, depth, path) {
1683     if(depth > depth_limit) {
1684       return '{...}';
1685     }
1686   }
1687
1688   function helper(value, depth, path) {
1689     if(depth > depth_limit) {
1690       return '{...}';
1691     }
1692   }
1693
1694   function helper(value, depth, path) {
1695     if(depth > depth_limit) {
1696       return '{...}';
1697     }
1698   }
1699
1700   function helper(value, depth, path) {
1701     if(depth > depth_limit) {
1702       return '{...}';
1703     }
1704   }
1705
1706   function helper(value, depth, path) {
1707     if(depth > depth_limit) {
1708       return '{...}';
1709     }
1710   }
1711
1712   function helper(value, depth, path) {
1713     if(depth > depth_limit) {
1714       return '{...}';
1715     }
1716   }
1717
1718   function helper(value, depth, path) {
1719     if(depth > depth_limit) {
1720       return '{...}';
1721     }
1722   }
1723
1724   function helper(value, depth, path) {
1725     if(depth > depth_limit) {
1726       return '{...}';
1727     }
1728   }
1729
1730   function helper(value, depth, path) {
1731     if(depth > depth_limit) {
1732       return '{...}';
1733     }
1734   }
1735
1736   function helper(value, depth, path) {
1737     if(depth > depth_limit) {
1738       return '{...}';
1739     }
1740   }
1741
1742   function helper(value, depth, path) {
1743     if(depth > depth_limit) {
1744       return '{...}';
1745     }
1746   }
1747
1748   function helper(value, depth, path) {
1749     if(depth > depth_limit) {
1750       return '{...}';
1751     }
1752   }
1753
1754   function helper(value, depth, path) {
1755     if(depth > depth_limit) {
1756       return '{...}';
1757     }
1758   }
1759
1760   function helper(value, depth, path) {
1761     if(depth > depth_limit
```

```

557     if (value === null && typeof value === 'object') {
558       if (seen.has(value)) {
559         return '[ Circular Reference ]';
560       }
561       seen.add(value);
562
563       // Handle arrays separately
564       if (Array.isArray(value)) {
565         return value.map((item, index) => helper(item, depth + 1, path.
566           concat(index)));
567       }
568
569       const result = {};
570
571       // Retrieve property descriptors without invoking getters
572       const descriptors = Object.getOwnPropertyDescriptors(value);
573       const keys = Reflect.ownKeys(descriptors); // Includes symbol
574         properties
575
576       for (const key of keys) {
577         const descriptor = descriptors[key];
578         const newPath = path.concat(key); // Build the full path
579
580         try {
581           if ('value' in descriptor) {
582             // Data property; process the value
583             result[key] = helper(descriptor.value, depth + 1, newPath);
584           } else if (typeof descriptor.get === 'function') {
585             // Accessor property with a getter
586             result[key] = '[Getter]';
587           } else {
588             result[key] = '[Unknown Property]';
589           }
590         } catch (err) {
591           result[key] = '[Error: ${escapeString(err.message)}]';
592         }
593       }
594       return result;
595     } else if (typeof value === 'string') {
596       // Escape strings to prevent HTML injection
597       return escapeString(value);
598     }
599
600     return JSON.stringify(helper(data, 0, []), null, 2);
601   }
602
603   /**
604    * Escapes special HTML characters in a string to prevent HTML injection.
605    * @param {string} string - The string to escape.
606    * @returns {string} - The escaped string with HTML characters replaced.
607    */
608   escapeHTML(string) {
609     var escapeHtmlMap = {

```

```

610      '&': '&amp;',
611      '<': '&lt;',
612      '>': '&gt;',
613      '"': '&quot;',
614      "'": '&#39;',
615      '/': '&#x2F;',
616      '`': '&#x60;',
617      '=': '&#x3D;';
618  };
619
620  return String(string).replace(/&lt;>"/'='\\/]/{g, function(s) {
621    return escapeHtmlMap[s];
622  });
623 }
624
625 /**
626 * Escapes special LaTeX characters in a string to prevent LaTeX injection.
627 * @param {string} string - The string to escape.
628 * @returns {string} - The escaped string with LaTeX characters replaced.
629 */
630 escapeLatex(string) {
631   if(typeof string !== 'string') {
632     return string;
633   }
634   return string
635     .replace(/\textbackslash/g, '\\textbackslash{}')
636     .replace(/\&/g, '\\&')
637     .replace(/\%/g, '\\%')
638     .replace(/\$/g, '\\
639 % -----
640 % End of document
641 % -----
642 \end{document})
643   .replace(/\#/g, '\\#')
644   .replace(/\_g, '\\_')
645   .replace(/\{/g, '\\{')
646   .replace(/\}/g, '\\}')
647   .replace(/\~/g, '\\textasciitilde{}')
648   .replace(/\^/g, '\\textasciicircum{}')
649   .replace(/\`/g, '\\textasciigrave{}');
650 }
651
652 /**
653 * Generates a unique object key by appending a number to the original key
654 * if it already exists.
655 * @param {object} object - Object to add unique key.
656 * @param {string} key - The original object key.
657 * @returns {string} A unique object key.
658 */
659 getKey(object, key) {
660   var i = 0;
661   var unique_key = key;
662   while(hasKey(object, unique_key)) {
663     i = i+1;

```

```
664         unique_key = key+i;
665     }
666     return unique_key;
667 }
668
669 /**
670 * Generates a random string of the specified length.
671 * @param {number} num - The desired length of the random string.
672 * @returns {string} A random string.
673 */
674 randomString(num) {
675     return Math.random().toString(36).substr(2, num);
676 }
677
678 /**
679 * Calculates the number of decimal places in a number.
680 * @param {number} num - The number to evaluate.
681 * @returns {number} The count of decimal places.
682 */
683 countDecimalPlaces(num) {
684     return num > 1 ? 0 : (num.toString().split('.')[1] || '').match(/^\d*/)[0].length+1;
685 }
686 }
687
688 exports.PRDC_JSLAB_LIB_FORMAT = PRDC_JSLAB_LIB_FORMAT;
```

Listing 80 - format.js

```
1 /**
2 * @file JSLAB FreeCADLink submodule
3 * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4 * PR-DC, Republic of Serbia
5 * info@pr-dc.com
6 */
7
8 /**
9 * Class for JSLAB FreeCADLink.
10 */
11 class PRDC_JSLAB_FREECAD_LINK {
12
13 /**
14 * Initializes a new instance of the FreeCADLink.
15 * @param {Object} js1 Reference to the main JSLAB object.
16 */
17 constructor(js1) {
18     this.js1 = js1;
19 }
20
21 /**
22 * Starts the FreeCAD application and establishes a TCP connection for
23 * remote procedure calls.
24 * Attempts to start FreeCAD if it's not running and connects to its TCP
25 * server.
26 * @param {string} exe - The executable path of FreeCAD.
27 * @param {Object} options - Configuration options such as port and host.
```

```

26   */
27   async start(exe, options) {
28     this.exe = exe;
29
30     this.port = 11077;
31     this.host = 'localhost';
32     this.timeout = 3000; // [ms]
33     this.script_timeout = 30000; // [ms]
34     this.startup_timeout = 10000; // [ms]
35
36     this.loaded = false;
37
38     if(options.hasOwnProperty('port')) {
39       this.port = options.port;
40     }
41     if(options.hasOwnProperty('host')) {
42       this.host = options.host;
43     }
44     if(options.hasOwnProperty('timeout')) {
45       this.timeout = options.timeout;
46     }
47     if(options.hasOwnProperty('script_timeout')) {
48       this.script_timeout = options.script_timeout;
49     }
50     if(options.hasOwnProperty('startup_timeout')) {
51       this.startup_timeout = options.startup_timeout;
52     }
53
54     var [flag, pids] = this.jsl.system.isProgramRunning('FreeCAD.exe');
55     if(flag) {
56       this.loaded = true;
57       this.jsl.env disp('@FreeCADLink: '+language.string(179));
58       await this.findServer();
59     } else {
60       this.jsl.system.exec('"' + this.exe + '" --single-instance');
61       var [flag, pids] = this.jsl.system.isProgramRunning('FreeCAD.exe');
62       if(flag) {
63         this.loaded = true;
64         await this.findServer();
65       } else {
66         this.jsl.env disp('@FreeCADLink: '+language.string(180));
67       }
68     }
69   }
70
71 /**
72 * Attempts to locate the FreeCAD TCP server within the network, respecting
73 * the startup timeout.
74 * Checks if the TCP server is reachable by sending a 'PING' command.
75 */
76 async findServer() {
77   var t = tic;
78   while(true) {
79     if(await this.send('PING|', 1000)) {
      break;
    }
  }
}

```

```

80     } else if(toc(t) > this.startup_timeout/1000) {
81         this.loaded = false;
82         this.jsl.env disp('@FreeCADLink: '+language.string(181));
83         break;
84     }
85     await waitSeconds(0.5);
86 }
87
88 /**
89 * Sends a message to the FreeCAD TCP server and waits for a response.
90 * Manages the TCP communication by ensuring message integrity and handling
91 * timeouts.
92 * @param {string} message - The message to send.
93 * @param {number} timeout - Timeout in milliseconds to wait for a response.
94 * @returns {Buffer|boolean} - The response from the server or false if the
95 * request times out.
96 */
97 async send(message, timeout) {
98     var obj = this;
99     var buf_in = [];
100    var N_in;
101    var got_header = false;
102
103    if(this.jsl.format.isUndefined(timeout)) {
104        timeout = this.timeout;
105    }
106    if(this.loaded) {
107        var t = tic;
108
109        this.tcp_com = this.jsl.networking.tcp(host, port, function() {
110            var length = message.length;
111            var buf_out = Buffer.alloc(2 + length);
112            buf_out.writeUInt16BE(length, 0);
113            buf_out.write(message, 2, 'utf8');
114            obj.tcp_com.write(buf_out);
115        });
116        while(true) {
117            await waitMSeconds(1);
118            if(toc(t) < timeout/1000) {
119                if(!got_header && this.tcp_com.availableBytes() > 2) {
120                    got_header = true;
121                    var data = this.tcp_com.read();
122                    var header = data.splice(0, 2);
123                    N_in = (header[0] << 8) | header[1];
124                    buf_in.push(...data);
125                    if(buf_in.length === N_in) {
126                        break;
127                    }
128                } else if(got_header && this.tcp_com.availableBytes() > 0) {
129                    var data = this.tcp_com.read();
130                    if(data.length) {
131                        buf_in.push(...data);
132                        if(buf_in.length >= N_in) {
133                            buf_in = buf_in.slice(0, N_in);
134                        }
135                    }
136                }
137            }
138        }
139    }
140
141    return buf_in;
142}

```

```

133                     break;
134                 }
135             }
136         } else {
137             break;
138         }
139     }
140 }
141
142     this.tcp_com.close();
143     if(buf_in.length) {
144         return Buffer.from(buf_in).toString();
145     }
146 }
147     return false;
148 }
149
150 /**
151 * Parses the input from FreeCAD responses to identify errors and data.
152 * Splits the message by '|' and checks for error or data messages.
153 * @param {string} message - The message received from FreeCAD.
154 * @returns {Array} - An array of parsed message components.
155 */
156 inputPraser(message) {
157     var params = [];
158     var str = message.toString();
159
160     if(str.length) {
161         params = str.split('|');
162         if(params.length && params[0] === 'ERR') {
163             disp('@FreeCADLink: Error = ' + params[1]);
164         }
165     }
166     return params;
167 }
168
169 /**
170 * Displays a message from FreeCAD in the JSLAB interface.
171 * Parses and displays messages specifically tagged as 'MSG' from FreeCAD.
172 * @param {string} message - The message received from FreeCAD.
173 */
174 showMessage(message) {
175     var params = this.inputPraser(message);
176     if(params.length && params[0] === 'MSG') {
177         this.jsl.env.disp('@FreeCADLink: Message = ' + params[1]);
178     }
179 }
180
181 /**
182 * Closes the FreeCAD application gracefully.
183 * Sends a quit command and handles the termination of the TCP connection.
184 */
185 async quit() {
186     if(this.loaded) {
187         var response = await this.send('CMD|QUIT', 1000);

```

```

188     return this.inputPraser(response);
189 } else {
190     this.jsl.env disp('@FreeCADLink: '+language.string(182));
191 }
192 return false;
193 }

194 /**
195 * Opens a specified file in FreeCAD.
196 * Sends a command to open a file and handles responses to confirm file
197 * access.
198 * @param {string} filePath - The path to the file to be opened.
199 * @param {number} timeout - Timeout in milliseconds to wait for a response.
200 */
201 async open(filePath, timeout) {
202 if(this.loaded) {
203     if(exist(filePath)) {
204         var response = await this.send('CMD|OPEN|' + filePath, timeout);
205         return this.inputPraser(response);
206     } else {
207         this.jsl.env disp('@FreeCADLink: '+language.string(183));
208     }
209 } else {
210     this.jsl.env disp('@FreeCADLink: '+language.string(182));
211 }
212 return false;
213 }

214 /**
215 * Imports a file into the current FreeCAD document.
216 * Sends an import command and handles responses to confirm the import
217 * operation.
218 * @param {string} filePath - The path of the file to be imported.
219 * @param {number} timeout - Timeout in milliseconds to wait for a response.
220 */
221 async importFile(filePath, timeout) {
222 if(this.loaded) {
223     if(exist(filePath)) {
224         var response = await this.send('CMD|IMPORT|' + filePath, timeout);
225         return this.inputPraser(response);
226     } else {
227         this.jsl.env disp('@FreeCADLink: '+language.string(183));
228     }
229 } else {
230     this.jsl.env disp('@FreeCADLink: '+language.string(182));
231 }
232 return false;
233 }

234 /**
235 * Creates a new document in FreeCAD, optionally specifying a filename.
236 * Sends a command to create a new document and handles the document
237 * creation response.
238 * @param {string} filename - Optional filename for the new document.
239 * @param {number} timeout - Timeout in milliseconds to wait for a response.

```

```

240     */
241     async newDocument(filename, timeout) {
242         if(this.loaded) {
243             var cmd = 'CMD|NEW';
244             if(!this.jsl.format.isUndefined(filename)) {
245                 cmd = cmd+"|"+filename;
246             }
247             var response = await this.send(cmd, timeout);
248             var params = this.inputPraser(response);
249             var name = "";
250             if(params.length && params[0] == 'DAT') {
251                 name = params[1];
252             }
253             return name;
254         } else {
255             this.jsl.env.disp('@FreeCADLink: '+language.string(182));
256         }
257         return false;
258     }
259
260 /**
261 * Saves the current document in FreeCAD.
262 * Sends a save command and handles responses to confirm the save operation.
263 * @param {number} timeout - Timeout in milliseconds to wait for a response.
264 */
265 async save(timeout) {
266     if(this.loaded) {
267         var response = await this.send('CMD|SAVE', timeout);
268         return this.inputPraser(response);
269     } else {
270         this.jsl.env.disp('@FreeCADLink: '+language.string(182));
271     }
272     return false;
273 }
274
275 /**
276 * Saves the current document in FreeCAD under a new filename.
277 * Sends a save as command and handles responses to confirm the operation.
278 * @param {string} filePath - The new file path for the document.
279 * @param {number} timeout - Timeout in milliseconds to wait for a response.
280 */
281 async saveAs(filePath, timeout) {
282     if(this.loaded) {
283         var response = await this.send('CMD|SAVEAS| ' + filePath, timeout);
284         return this.inputPraser(response);
285     } else {
286         this.jsl.env.disp('@FreeCADLink: '+language.string(182));
287     }
288     return false;
289 }
290
291 /**
292 * Closes the current document in FreeCAD.
293 * Sends a close command and handles responses to confirm the document
294 * closure.

```

```

294     * @param {number} timeout - Timeout in milliseconds to wait for a response.
295     */
296   async close(timeout) {
297     if(this.loaded) {
298       var response = await this.send('CMD|CLOSE', timeout);
299       return this.inputPraser(response);
300     } else {
301       this.jsl.env.disp('@FreeCADLink: '+language.string(182));
302     }
303     return false;
304   }
305
306 /**
307  * Executes a command in FreeCAD and returns the evaluation result.
308  * Sends an evaluate command with the specified command string.
309  * @param {string} command - The command to be evaluated in FreeCAD.
310  * @param {number} timeout - Timeout in milliseconds to wait for a response.
311  */
312   async evaluate(command, timeout) {
313     if(this.loaded) {
314       var response = await this.send('EVAL|' + command, timeout);
315       return this.inputPraser(response);
316     } else {
317       this.jsl.env.disp('@FreeCADLink: '+language.string(182));
318     }
319     return false;
320   }
321
322 /**
323  * Runs a script in FreeCAD with optional parameters.
324  * Sends a script command along with parameters and handles the script
325  * execution response.
326  * @param {string} script - The script to run.
327  * @param {string|array} param - Parameters to pass to the script.
328  * @param {number} timeout - Timeout in milliseconds to wait for a response.
329  */
330   async callScript(script, param, timeout) {
331     if(this.loaded) {
332       if(this.jsl.format.isUndefined(timeout)) {
333         timeout = this.script_timeout;
334       }
335       if(this.jsl.format.isUndefined(param)){
336         param = '';
337       } else if(isArray(param)){
338         param = param.join(' | ');
339       }
340       var response = await this.send('SCRIPT|' + script + '|'+ param, timeout
341         );
342       return this.inputPraser(response);
343     } else {
344       this.jsl.env.disp('@FreeCADLink: '+language.string(182));
345     }
346     return false;
347   }

```

```

347  /**
348   * Retrieves the area of the selected object in FreeCAD.
349   * Sends a measure area command and parses the response to extract the area
350   * value.
351   * @param {number} timeout - Timeout in milliseconds to wait for a response.
352   */
353   async getArea(timeout) {
354     if(this.loaded) {
355       var response = await this.send('MSR|A', timeout);
356       var params = this.inputPraser(response);
357       var area = "";
358       if(params.length && params[0] === 'DAT') {
359         area = params[1];
360       }
361       return area;
362     } else {
363       this.jsl.env disp('@FreeCADLink: '+language.string(182));
364     }
365     return false;
366   }

367 /**
368   * Retrieves the volume of the selected object in FreeCAD.
369   * Sends a measure volume command and parses the response to extract the
370   * volume value.
371   * @param {number} timeout - Timeout in milliseconds to wait for a response.
372   */
373   async getVolume(timeout) {
374     if(this.loaded) {
375       var response = await this.send('MSR|V', timeout);
376       var params = this.inputPraser(response);
377       var vol = "";
378       if(params.length && params[0] === 'DAT') {
379         vol = params[1];
380       }
381       return vol;
382     } else {
383       this.jsl.env disp('@FreeCADLink: '+language.string(182));
384     }
385     return false;
386   }
387 }

388 exports.PRDC_JSLAB_FREECAD_LINK = PRDC_JSLAB_FREECAD_LINK;

```

Listing 81 - freecad-link.js

```

1 /**
2  * @file JSLAB library geography map 3D submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 /**
9  * Class for map 3D.

```

```

10  /*
11  class PRDC_JSLAB_GEOGRAPHY_MAP_3D {
12
13  /**
14  * Creates a new 3D map instance.
15  * @param {Object} js1 - The JSLAB instance or environment reference.
16  */
17 constructor(js1, token) {
18   this.js1 = js1;
19   this.token = token;
20
21   // Default camera parameters
22   this.latitude = 44.8768331;
23   this.longitude = 20.112352;
24   this.height = 1000; // meters
25   this.heading = 0; // degrees
26   this.pitch = -30; // degrees
27
28   this.wid;
29   this.ready = false;
30 }
31
32 /**
33 * Opens a window with Cesium and initializes 3D map.
34 * @returns {Promise<void>}
35 */
36 async createWindow() {
37   var wid = this.js1.windows.openWindow('cesium.html');
38   this.wid = wid;
39   this.win = this.js1.windows.open_windows[wid];
40   await this.js1.windows.open_windows[wid].ready;
41   var context = this.js1.windows.open_windows[wid].context;
42   context.imports_ready = false;
43   while(!context.imports_ready) {
44     if(typeof context.Cesium != 'undefined') {
45       context.imports_ready = true;
46     }
47     await this.js1.non_blocking.waitMSeconds(1);
48   }
49   context.map_3d_cont = context.document.getElementById('map-3d-cont');
50   context.map_3d_cont.style = 'position: absolute; top:0; left:0; right:0;
      bottom:0;';
51
52 // Initialize Cesium Viewer
53 context.Cesium.Ion.defaultAccessToken = this.token;
54 this.viewer = new context.Cesium.Viewer(context.map_3d_cont, {
55   terrainProvider: await context.Cesium.CesiumTerrainProvider.
      fromIonAssetId(1),
56   timeline: false,
57   sceneModePicker: false,
58   baseLayerPicker: false,
59   geocoder: false,
60   infoBox: false,
61   selectionIndicator: false
62 });

```

```

63
64     this.context = context;
65     this.Cesium = context.Cesium;
66     this.ready = true;
67 }
68
69 /**
70 * Sets the camera view to the specified latitude, longitude, and height.
71 * @param {number} lat - Latitude.
72 * @param {number} lon - Longitude.
73 * @param {number} height - Height in meters.
74 * @param {number} [heading=0] - Heading in degrees.
75 * @param {number} [pitch=-30] - Pitch in degrees.
76 */
77 setView(lat, lon, height, heading = 0, pitch = -30) {
78     this.latitude = lat;
79     this.longitude = lon;
80     this.height = height;
81     this.heading = heading;
82     this.pitch = pitch;
83
84     if(this.ready && this.viewer) {
85         this.viewer.camera.flyTo({
86             destination: this.context.Cesium.Cartesian3.fromDegrees(lon, lat,
87                 height),
88             orientation: {
89                 heading: this.context.Cesium.Math.toRadians(heading),
90                 pitch: this.context.Cesium.Math.toRadians(pitch),
91                 roll: 0.0
92             }
93         });
94     }
95
96 /**
97 * Adds a new entity to the 3D map.
98 * @param {Object} data - The data representing the entity to add.
99 * @returns {PRDC_JSLAB_GEOGRAPHY_MAP_3D_ENTITY} The newly created map
100    entity.
101 */
102 addEntity(data) {
103     return new PRDC_JSLAB_GEOGRAPHY_MAP_3D_ENTITY(this.jsl, this, data);
104 }
105 /**
106 * Removes all entities from the 3D map viewer.
107 */
108 removeAllEntities() {
109     this.viewer.entities.removeAll();
110 }
111
112 /**
113 * Animates the camera to fly to the specified entity.
114 * @param {Object} entity - The entity to fly to.
115 */

```

```

116     flyTo(entity) {
117         if(hasKey(entity, 'entity')) {
118             this.viewer.flyTo(entity.entity);
119         }
120     }
121 }
122
123 exports.PRDC_JSLAB_GEOGRAPHY_MAP_3D = PRDC_JSLAB_GEOGRAPHY_MAP_3D;
124
125 class PRDC_JSLAB_GEOGRAPHY_MAP_3D_ENTITY {
126
127     /**
128      * Creates a new 3D map entity and adds it to the viewer.
129      * @param {Object} js1 - The JSL environment object.
130      * @param {Object} map_3d - The 3D map instance where the entity will be
131      * added.
132      * @param {Object} data - The data representing the entity.
133      */
134     constructor(js1, map_3d, data) {
135         this.js1 = js1;
136         this.map_3d = map_3d;
137         this.data = data;
138
139         this.entity = this.map_3d.viewer.entities.add(data);
140     }
141
142     /**
143      * Animates the camera to fly to this entity.
144      */
145     flyTo() {
146         this.map_3d.viewer.flyTo(this.entity);
147     }

```

Listing 82 - geography-map-3d.js

```

1  /**
2   * @file JSLAB library geography map submodule
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   */
7
8  /**
9   * Class for map.
10  */
11 class PRDC_JSLAB_GEOGRAPHY_MAP {
12
13     /**
14      * Creates a new map instance.
15      * @param {Object} js1 - The JSLAB instance or environment reference.
16      * @param {string} [tileset='OpenStreetMap'] - The name of the tileset to
17      * use.
18      */
19      constructor(js1, tileset = 'OpenStreetMap') {
        this.js1 = js1;

```

```

20
21 // Default map parameters
22 this.lat = 44.8768331;
23 this.lon = 20.112352;
24 this.zoom = 12;
25 this.tileset = tileset;
26 this.tileset_url;

27
28 this.tilesets = {
29   // OpenStreetMap Standard
30   'OpenStreetMap': 'https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',
31
32   // Stamen Design
33   'Stamen Toner': 'https://stamen-tiles.a.ssl.fastly.net/toner/{z}/{x}/{y}
34   }.png',
35   'Stamen Watercolor': 'https://stamen-tiles.a.ssl.fastly.net/watercolor/
36   {z}/{x}/{y}.jpg',
37   'Stamen Terrain': 'https://stamen-tiles.a.ssl.fastly.net/terrain/{z}/{x}
38  }/{y}.jpg',
39
40   // CartoDB (CARTO)
41   'Carto Positron': 'https://{s}.basemaps.cartocdn.com/light_all/{z}/{x}/{
42   y}{r}.png',
43   'Carto DarkMatter': 'https://{s}.basemaps.cartocdn.com/dark_all/{z}/{x}
44  }/{y}{r}.png',
45
46   // OpenTopoMap
47   'OpenTopoMap': 'https://{s}.tile.opentopomap.org/{z}/{x}/{y}.png',
48
49   // Esri
50   'Esri WorldStreetMap': 'https://server.arcgisonline.com/ArcGIS/rest/
51   services/World_Street_Map/MapServer/tile/{z}/{y}/{x}',
52   'Esri WorldImagery': 'https://server.arcgisonline.com/ArcGIS/rest/
53   services/World_Imagery/MapServer/tile/{z}/{y}/{x}',
54   'Esri DarkGrayCanvas': 'https://server.arcgisonline.com/ArcGIS/rest/
55   services/Dark_Gray_Base/MapServer/tile/{z}/{y}/{x}',

56   // Wikimedia Maps
57   'Wikimedia Standard': 'https://maps.wikimedia.org/osm-intl/{z}/{x}/{y}.
58   png',
59   'Wikimedia Cycle Map': 'https://maps.wikimedia.org/osm-intl-cycle/{z}/{x}
60  }/{y}.png',
61   'Wikimedia Transport Map': 'https://maps.wikimedia.org/osm-intl-
62   transport/{z}/{x}/{y}.png',
63 };

64
65 if(hasKey(this.tilesets, this.tileset)) {
66   this.tileset_url = this.tilesets[this.tileset];
67 } else {
68   this.tileset_url = this.tileset;
69 }
70
71 this.wid;
72 this.ready = false;
73 }
```

```

64
65  /**
66   * Opens a window with Leaflet and initializes the map.
67   * @returns {Promise<void>}
68   */
69 async createWindow() {
70   var wid = this.jsl.windows.openWindow('leaflet.html');
71   this.wid = wid;
72   this.win = this.jsl.windows.open_windows[wid];
73   await this.jsl.windows.open_windows[wid].ready;
74   var context = this.jsl.windows.open_windows[wid].context;
75   context.imports_ready = false;
76   while(!context.imports_ready) {
77     if(typeof context.L != 'undefined') {
78       context.imports_ready = true;
79     }
80     await this.jsl.non_blocking.waitMSeconds(1);
81   }
82   this.context = context;
83   context.map_cont = context.document.getElementById('map-cont');
84   context.map_cont.style = 'position: absolute; top:0; left:0; right:0; bottom
85   :0;';
86   this.leaflet_obj = context.L.map('map-cont',
87     { attributionControl: false });
88   ).setView([this.lat, this.lon], this.zoom);
89   context.L.tileLayer(this.tileset_url).addTo(this.leaflet_obj);
90   this.ready = true;
91 }

92 /**
93  * Sets the center of the map to the specified latitude and longitude.
94  * This will update the internal state and move the Leaflet map if it is
95  * ready.
96  * @param {number} lat - Latitude.
97  * @param {number} lon - Longitude.
98  */
99 setCenter(lat, lon) {
100   this.lat = lat;
101   this.lon = lon;
102   if(this.ready && this.leaflet_obj) {
103     this.leaflet_obj.setView([lat, lon], this.zoom);
104   }
105 }

106 /**
107  * Sets the zoom level of the map.
108  * This will update the internal state and adjust the Leaflet map if it is
109  * ready.
110  * @param {number} zoom - The zoom level.
111  */
112 setZoom(zoom) {
113   this.zoom = zoom;
114   if(this.ready && this.leaflet_obj) {
115     this.leaflet_obj.setZoom(zoom);
116   }
117 }
```

```

116   }
117
118  /**
119   * Adds a marker to the map at the specified latitude and longitude.
120   * @param {number} lat - Latitude.
121   * @param {number} lon - Longitude.
122   * @returns {PRDC_JSLAB_GEOGRAPHY_MAP_MARKER|null} - The marker instance or
123   *         null if map is not ready.
124   */
125  addMarker(lat, lon) {
126    return new PRDC_JSLAB_GEOGRAPHY_MAP_MARKER(this.jsl, this, lat, lon);
127  }
128
129 exports.PRDC_JSLAB_GEOGRAPHY_MAP = PRDC_JSLAB_GEOGRAPHY_MAP;
130
131 class PRDC_JSLAB_GEOGRAPHY_MAP_MARKER {
132
133  /**
134   * Creates a new marker and adds it to the map.
135   * @param {Object} jsl - The JSLAB instance or environment reference.
136   * @param {PRDC_JSLAB_GEOGRAPHY_MAP} map - The map instance.
137   * @param {number} lat - Latitude.
138   * @param {number} lon - Longitude.
139   */
140  constructor(jsl, map, lat, lon) {
141    this.jsl = jsl;
142    this.map = map;
143
144    this.leaflet_obj = this.map.context.L.marker([lat, lon]).addTo(this.map.
145      leaflet_obj);
146    this.lat = lat;
147    this.lon = lon;
148  }
149
150  /**
151   * Sets a custom icon for the marker.
152   * @param {Object} iconOptions - Leaflet icon options: { iconUrl: '...', ,
153   *           iconSize: [...], iconAnchor: [...] , etc. }
154   */
155  setIcon(iconOptions) {
156    if(this.leaflet_obj && this.map.ready) {
157      let customIcon = this.map.context.L.icon(iconOptions);
158      this.leaflet_obj.setIcon(customIcon);
159      this.leaflet_obj.setRotationOrigin('center center');
160    }
161
162  /**
163   * Sets a custom icon for the marker.
164   * @param {Object} iconOptions - Leaflet divIcon options: { iconUrl: '...', ,
165   *           iconSize: [...], iconAnchor: [...] , etc. }
166   */
167  setDivIcon(iconOptions) {
168    if(this.leaflet_obj && this.map.ready) {

```

```

167     let customIcon = this.map.context.L.divIcon(iconOptions);
168     this.leaflet_obj.setIcon(customIcon);
169     this.leaflet_obj.setRotationOrigin('center center');
170   }
171 }
172
173 /**
174 * Sets the position of the marker.
175 * @param {number} lat - New latitude.
176 * @param {number} lon - New longitude.
177 */
178 setPosition(lat, lon) {
179   this.lat = lat;
180   this.lon = lon;
181   if(this.leaflet_obj && this.map.ready) {
182     this.leaflet_obj.setLatLng([lat, lon]);
183   }
184 }
185
186 /**
187 * Sets the rotation (in degrees) of the marker.
188 * Leaflet markers can be rotated via CSS transform if using a DivIcon or a
189 * custom class.
190 * This example uses a simple inline style transform if the marker's icon
191 * supports it.
192 * @param {number} rotation - Rotation angle in degrees.
193 */
194 setRotation(rotation) {
195   if(this.leaflet_obj && this.map.ready) {
196     this.leaflet_obj.setRotationAngle(rotation);
197   }
}

```

Listing 83 - geography-map.js

```

1 /**
2 * @file JSLAB library geography submodule
3 * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4 * PR-DC, Republic of Serbia
5 * info@pr-dc.com
6 */
7
8 var { PRDC_JSLAB_GEOGRAPHY_MAP } = require('./geography-map');
9 var { PRDC_JSLAB_GEOGRAPHY_MAP_3D } = require('./geography-map-3d');
10
11 /**
12 * Class for JSLAB geography submodule.
13 */
14 class PRDC_JSLAB_LIB_GEOGRAPHY {
15
16 /**
17 * Initializes the geography submodule.
18 * @param {Object} js1 Reference to the main JSLAB object.
19 */
20 constructor(js1) {

```

```

1 var obj = this;
2 this.jsl = jsl;
3 }
4
5 /**
6 * Initializes and returns a new 2D web map instance.
7 * @param {...*} args - Arguments for configuring the web map.
8 * @returns {Promise<PRDC_JSLAB_GEOGRAPHY_MAP>} The initialized 2D web map
9 * instance.
10 */
11 async webmap(...args) {
12     var map = new PRDC_JSLAB_GEOGRAPHY_MAP(this.jsl, ...args);
13     await map.createWindow();
14     return map;
15 }
16
17 /**
18 * Initializes and returns a new 3D geoglobe instance.
19 * @param {...*} args - Arguments for configuring the 3D geoglobe.
20 * @returns {Promise<PRDC_JSLAB_GEOGRAPHY_MAP_3D>} The initialized 3D
21 * geoglobe instance.
22 */
23 async geoglobe(...args) {
24     var map_3d = new PRDC_JSLAB_GEOGRAPHY_MAP_3D(this.jsl, ...args);
25     await map_3d.createWindow();
26     return map_3d;
27 }
28
29 /**
30 * Calculates the bounding box and center from an array of tile coordinates.
31 * @param {Array<Object>} tile_coordinates - An array of objects with tile
32 * coordinates, each having properties 'x', 'y', and 'z' for tile X and Y
33 * coordinates and zoom level, respectively.
34 * @returns {Object} An object containing the bounds as an array of '[  

35 * min_lat, min_lng]' and '[max_lat, max_lng]', and the center as '[  

36 * latitude, longitude]'.
37 */
38 calculateTilesBoundingBox(tile_coordinates) {
39     var min_lat = Number.MAX_VALUE,
40         max_lat = -Number.MAX_VALUE,
41         min_lng = Number.MAX_VALUE,
42         max_lng = -Number.MAX_VALUE;
43
44     tile_coordinates.forEach(function(coord) {
45         var n = Math.pow(2, coord.z);
46         var tileBounds = {
47             min_lat: (2 * Math.atan(Math.exp(Math.PI - (2 * Math.PI * coord.y) / n)) - Math.PI / 2) * (180 / Math.PI),
48             max_lat: (2 * Math.atan(Math.exp(Math.PI - (2 * Math.PI * (coord.y + 1)) / n)) - Math.PI / 2) * (180 / Math.PI),
49             min_lng: ((coord.x) / n) * 360 - 180,
50             max_lng: ((coord.x + 1) / n) * 360 - 180
51         };
52
53         min_lat = Math.min(min_lat, tileBounds.min_lat);
54         max_lat = Math.max(max_lat, tileBounds.max_lat);
55         min_lng = Math.min(min_lng, tileBounds.min_lng);
56         max_lng = Math.max(max_lng, tileBounds.max_lng);
57     });
58
59     return {
60         bounds: [min_lat, max_lat, min_lng, max_lng],
61         center: [(min_lng + max_lng) / 2, (min_lat + max_lat) / 2]
62     };
63 }
64
65 /**
66 * Converts a coordinate from geographic to Web Mercator projection.
67 * @param {Object} coord - A coordinate object with properties 'x' and 'y'.
68 * @param {number} z - The zoom level.
69 * @returns {Object} The converted coordinate object.
70 */
71 convertToMercator(coord, z) {
72     var n = Math.pow(2, z);
73     var x = coord.x * (Math.PI * 2) / n;
74     var y = coord.y * (Math.PI * 2) / n;
75     var lat = (2 * Math.atan(Math.exp(Math.PI * y / n))) - Math.PI / 2;
76     var lon = (2 * Math.atan(Math.exp(Math.PI * x / n))) - Math.PI / 2;
77
78     return {
79         x: lon * 180 / Math.PI,
80         y: lat * 180 / Math.PI
81     };
82 }
83
84 /**
85 * Converts a coordinate from Web Mercator projection to geographic.
86 * @param {Object} coord - A coordinate object with properties 'x' and 'y'.
87 * @param {number} z - The zoom level.
88 * @returns {Object} The converted coordinate object.
89 */
90 convertFromMercator(coord, z) {
91     var n = Math.pow(2, z);
92     var x = coord.x * (Math.PI * 2) / n;
93     var y = coord.y * (Math.PI * 2) / n;
94     var lat = (2 * Math.atan(Math.exp(Math.PI * y / n))) - Math.PI / 2;
95     var lon = (2 * Math.atan(Math.exp(Math.PI * x / n))) - Math.PI / 2;
96
97     return {
98         x: lon * 180 / Math.PI,
99         y: lat * 180 / Math.PI
100    };
101 }

```

```

68     max_lat = Math.max(max_lat, tileBounds.max_lat);
69     min_lng = Math.min(min_lng, tileBounds.min_lng);
70     max_lng = Math.max(max_lng, tileBounds.max_lng);
71   });
72
73   var center = [(min_lat + max_lat) / 2, (min_lng + max_lng) / 2];
74   var bounds = [[min_lat, min_lng], [max_lat, max_lng]];
75
76   return { bounds, center };
77 }
78
79 /**
80 * Calculates a new latitude and longitude based on a starting point,
81 * distance, and bearing using the Haversine formula.
82 * @param {number} lat - The latitude of the starting point.
83 * @param {number} lon - The longitude of the starting point.
84 * @param {number} distance - The distance from the starting point in meters
85 * @param {number} bearing - The bearing in degrees from north.
86 * @returns {Array<number>} An array containing the latitude and longitude
87 * of the calculated point.
88 */
89 offsetLatLon(lat, lon, distance, bearing) {
90   var dist_rad = distance / 6371000;
91   var bearing_rad = (bearing * Math.PI) / 180;
92   var lat_rad = (lat * Math.PI) / 180;
93   var lng_rad = (lon * Math.PI) / 180;
94   var new_lat_rad = Math.asin(Math.sin(lat_rad) * Math.cos(dist_rad) +
95     Math.cos(lat_rad) * Math.sin(dist_rad) * Math.cos(bearing_rad));
96   var new_lng_rad = lng_rad + Math.atan2(Math.sin(bearing_rad) * Math.sin(
97     dist_rad) * Math.cos(lat_rad),
98     Math.cos(dist_rad) - Math.sin(lat_rad) * Math.sin(new_lat_rad)));
99
100  return [(new_lat_rad * 180) / Math.PI, (new_lng_rad * 180) / Math.PI];
101}
102
103 /**
104 * Calculates the distance between two points on Earth using the Haversine
105 * formula.
106 * @param {number} lat1 - The latitude of the first point.
107 * @param {number} lon1 - The longitude of the first point.
108 * @param {number} lat2 - The latitude of the second point.
109 * @param {number} lon2 - The longitude of the second point.
110 * @returns {number} The distance between the two points in meters.
111 */
112 latLonDistance(lat1, lon1, lat2, lon2) {
113   var dLat = (lat2 - lat1) * Math.PI / 180;
114   var dLon = (lon2 - lon1) * Math.PI / 180;
115   return 6371000 * 2 * Math.asin(Math.sqrt(Math.sin(dLat/2) * Math.sin(dLat
116     /2) +
117     Math.cos(lat1 * Math.PI / 180) * Math.cos(lat2 * Math.PI / 180) *
118     Math.sin(dLon/2) * Math.sin(dLon/2))); // [m]
119 }
120
121 /**
122 * Calculates the distance between two points on Earth including altitude

```

```

difference using the Haversine formula.

117  * @param {number} lat1 - The latitude of the first point.
118  * @param {number} lon1 - The longitude of the first point.
119  * @param {number} alt1 - The altitude of the first point in meters.
120  * @param {number} lat2 - The latitude of the second point.
121  * @param {number} lon2 - The longitude of the second point.
122  * @param {number} alt2 - The altitude of the second point in meters.
123  * @returns {number} The 3D distance between the two points in meters.
124  */
125 latLonAltDistance(lat1, lon1, alt1, lat2, lon2, alt2) {
126   var L = latLonDistance(lat1, lon1, lat2, lon2);
127   return Math.sqrt(Math.pow(L, 2)+Math.pow(alt1-alt2, 2));
128 }

129 /**
130  * Checks if the latitude and longitude values have been updated.
131  * @param {Object} latlon - An object containing the current and last values
132  * of latitude and longitude.
133  * @returns {boolean} True if the latitude and/or longitude values have been
134  * updated; false otherwise.
135  */
136 checkNewLatLon(latlon) {
137   if(this.jsl.format.isUndefined(latlon.value)) {
138     return false;
139   } else if(this.jsl.format.isUndefined(latlon.last_value)) {
140     return true;
141   } else {
142     return (latlon.value[0] != latlon.last_value[0]) || (latlon.value[1] !=
143       latlon.last_value[1]);
144   }
145 /**
146  * Converts latitude, longitude, and altitude to Cartesian coordinates.
147  * @param {number|number[]} lat - Latitude in degrees or array of latitudes.
148  * @param {number|number[]} lon - Longitude in degrees or array of
149  * longitudes.
150  * @param {number|number[]} alt - Altitude in meters or array of altitudes.
151  * @returns {Cesium.Cartesian3|Cesium.Cartesian3[]} Cartesian coordinate(s).
152  */
153 latLonAlt2cartesian(lat, lon, alt) {
154   var isArray = Array.isArray(lat) && Array.isArray(lon) && Array.isArray(
155     alt);
156
157   if(isArray) {
158     return lat.map((latitude, index) =>
159       this.jsl.env.Cesium.Cartesian3.fromDegrees(lon[index], latitude, alt[
160         index]));
161   } else {
162     return this.jsl.env.Cesium.Cartesian3.fromDegrees(lon, lat, alt);
163   }
164 }

```

165 exports.PRDC_JSLAB_LIB_GEOGRAPHY = PRDC_JSLAB_LIB_GEOGRAPHY;

Listing 84 - geography.js

```

1  /**
2   * @file JSLAB library geometry Space Search submodule
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   * @version 0.0.1
7   */
8
9  /**
10  * Class for N-dimensional Space Search.
11  */
12 class PRDC_JSLAB_GEOMETRY_SPACE_SERACH {
13
14  /**
15   * Displays the size of elements based on bounds and subdivisions.
16   * @param {Array} bounds - Array of [min, max] for each dimension.
17   * @param {Array} subdivisionPerDepth - Subdivision factors per depth and
18   * dimension.
19  */
20  dispElementSize(bounds, subdivisionPerDepth) {
21    var numDimensions = bounds.length;
22    // Calculate the initial differences between bounds for each dimension
23    var dq = bounds.map((bound, index) => (bound[1] - bound[0]) /
24      subdivisionPerDepth[0][index]);
25
26    // Copy dq to dqmin for further subdivision calculations
27    var dqmin = [...dq];
28
29    // Iterate through subdivisionPerDepth starting from the second depth
30    // level
31    for(var i = 1; i < subdivisionPerDepth.length; i++) {
32      for(var j = 0; j < numDimensions; j++) {
33        dqmin[j] /= subdivisionPerDepth[i][j];
34      }
35
36      // Display the results for initial and minimal element dimensions
37      disp(` Dimenzije inicialnih elemenata: ${dq.map(val => val.toFixed(2)).
38          join('x')}`);
39      disp(` Dimenzije najmanjih elemenata: ${dqmin.map(val => val.toFixed(2)).
40          join('x')}`);
41    }
42
43    /**
44     * Splits the search space into smaller intervals for parallel processing.
45     * @param {Array.<Array.<number>>} x_lim - The limits of the search space,
46     * where each sub-array represents [start, end] for a dimension.
47     * @param {Array.<Array.<number>>} k - The parameters to optimize.
48     * @param {number} N_proc - The number of processors to divide the work
49     * among.
50     * @returns {Array.<Array.<number>>} An array containing the split search
51     * spaces and the adjusted parameters.
52  }

```

```

45  */
46  splitSearchSpace(x_lim, k, N_proc) {
47    const [start, end] = x_lim[0];
48    const interval = (end - start) / N_proc;
49
50    var k_out = [...k];
51    k_out[0][0] = k_out[0][0] / N_proc;
52
53    return [Array.from({ length: N_proc }, (_, i) => [
54      [start + i * interval, start + (i + 1) * interval],
55      ...x_lim.slice(1)
56    ]), k_out];
57  }
58
59 /**
60 * Executes a provided function in parallel across the split search spaces.
61 * @param {Array.<Array.<number>>} x_lim - The limits of the search space,
62 *   where each sub-array represents [start, end] for a dimension.
63 * @param {Array.<Array.<number>>} k - The parameters to optimize.
64 * @param {Object} context - The execution context containing necessary
65 *   configurations and states.
66 * @param {Function} setup - The setup function to initialize the parallel
67 *   environment.
68 * @param {Function} fun - The function to execute in parallel on each split
69 *   of the search space.
70 * @returns {Promise<Array.<Array.<number>>>} A promise that resolves to
71 *   arrays of input and output results from the parallel execution.
72 */
73
74 async runParallel(x_lim, k, context, setup, fun) {
75   var N_proc = parallel.getProcessorsNum();
76
77   var funStr = fun.toString();
78   var funBody = funStr.slice(funStr.indexOf("{") + 1, funStr.lastIndexOf("}");
79   var funArgs = funStr.slice(funStr.indexOf("(") + 1, funStr.indexOf(")"));
80
81   var [x_lim_parallel, k_parallel] = this.splitSearchSpace(x_lim, k, N_proc);
82
83   var res = await parallel.parfor(0, N_proc-1, 1, N_proc,
84     Object.assign(context, {x_lim_parallel, k_parallel,
85       funArgs, funBody}), setup, function(i) {
86       var new_fun = new Function(funArgs, funBody);
87       return crawler.run(x_lim_parallel[i], k_parallel, new_fun);
88     });
89   var Nin = res.map(pair => pair[0]).flat();
90   var Nout = res.map(pair => pair[1]).flat();
91   return [Nin, Nout];
92 }
93
94 /**
95 * Executes the space search algorithm.
96 * @param {number[][][]} bounds - Array of [min, max] for each dimension.
97 * @param {number[][][]} subdivisionPerDepth - Subdivision factors for each
98 *   depth and dimension.
99 */

```

```

92   * @param {function} conditionFunction - Function that determines if a point
93     satisfies the condition.
94   * @returns {Array<Array<number>>} - Arrays of points inside and outside the
95     condition.
96   */
97   run(bounds, subdivisionPerDepth, conditionFunction) {
98     var numDimensions = bounds.length;
99     var maxDepth = subdivisionPerDepth.length;
100
101    // Initial starting point
102    var startCoordinates = bounds.map(([min]) => min);
103
104    // Calculate initial step sizes (dq)
105    var initialStepSizes = bounds.map(
106      ([min, max], idx) => (max - min) / subdivisionPerDepth[0][idx]
107    );
108
109    // Generate step sizes for each depth
110    var stepSizesPerDepth = [initialStepSizes];
111    for(var depth = 1; depth < maxDepth; depth++) {
112      var previousStepSizes = stepSizesPerDepth[depth - 1];
113      var newStepSizes = previousStepSizes.map(
114        (size, idx) => size / subdivisionPerDepth[depth][idx]
115      );
116      stepSizesPerDepth.push(newStepSizes);
117    }
118
119    // Compute the values at the boundary points
120    var cornerShifts = this.generateCornerShifts(numDimensions);
121    var boundaryPoints = cornerShifts.map(shift => {
122      return shift.map((s, idx) => startCoordinates[idx] + s * (bounds[idx][1]
123        - bounds[idx][0]));
124    });
125    var boundaryValues = boundaryPoints.map(point => (conditionFunction(point)
126      ? 1 : 0));
127
128    // Call makeNodesND
129    var [Nin, Nout] = this.makeNodesND(
130      startCoordinates,
131      boundaryValues,
132      0,
133      stepSizesPerDepth,
134      subdivisionPerDepth,
135      conditionFunction
136    );
137
138    return [Nin, Nout];
139  }
140
141  /**
142   * Recursively creates nodes in N dimensions based on subdivisions.
143   * @param {number[]} startCoordinates - Starting coordinates for the current
144     grid.
145   * @param {number[]} boundaryValues - Values at the boundary points of the
146     current hypercube.

```

```

141  * @param {number} currentDepth - Current depth of the recursion.
142  * @param {number[][][]} stepSizesPerDepth - Step sizes for each depth.
143  * @param {number[][][]} subdivisionPerDepth - Subdivision factors for each
144  *      depth.
145  * @param {function} conditionFunction - User-defined condition function.
146  * @returns {Array<Array<number>>} - Arrays of points inside and outside the
147  *      condition.
148  */
149 makeNodesND(
150   startCoordinates,
151   boundaryValues,
152   currentDepth,
153   stepSizesPerDepth,
154   subdivisionPerDepth,
155   conditionFunction
156 ) {
157   var numDimensions = startCoordinates.length;
158   var maxDepth = subdivisionPerDepth.length; // maximum depth
159   var Nin = [];
160   var Nout = [];
161
162   // Generate coordinate arrays for each dimension
163   var stepSizes = stepSizesPerDepth[currentDepth];
164   var numSteps = subdivisionPerDepth[currentDepth];
165   var coordsArray = [];
166   for(var idx = 0; idx < numDimensions; idx++) {
167     var steps = numSteps[idx];
168     var stepSize = stepSizes[idx];
169     var start = startCoordinates[idx];
170     var arr = Array.from(
171       { length: steps + 1 },
172       (_, i) => start + i * stepSize
173     );
174     coordsArray.push(arr);
175   }
176
177   // Initialize N-dimensional grid
178   var gridShape = coordsArray.map(A => A.length);
179   var N = createFilledArray(...gridShape, null);
180
181   // Mark whether a node needs to be calculated
182   var nf = createFilledArray(...gridShape, 1);
183
184   // Set nf to 0 for corner points (boundary points)
185   var cornerIndices = this.getCornerIndices(gridShape);
186   cornerIndices.forEach(indices => {
187     setValueAt(nf, indices, 0);
188   });
189
190   // Set the values at the corner points and add them to Nin or Nout
191   cornerIndices.forEach((indices, idx) => {
192     var coords = indices.map((index, dim) => coordsArray[dim][index]);
193     var value = boundaryValues[idx];
194     setValueAt(N, indices, [...coords, value]);
195   });

```

```

194   // On first level , add corner points to Nin or Nout
195   if(currentDepth == 0) {
196     if(value) {
197       Nin.push(coords);
198     } else {
199       Nout.push(coords);
200     }
201   }
202 });
203
204 // Now compute the values at all nodes where nf is 1
205 var indicesList = this.generateIndicesList(gridShape, nf);
206 for(var indices of indicesList) {
207   var coords = indices.map((index, dim) => coordsArray[dim][index]);
208   var e = conditionFunction(coords) ? 1 : 0;
209   setValueAt(N, indices, [...coords, e]);
210
211   if(e) {
212     Nin.push(coords);
213   } else {
214     Nout.push(coords);
215   }
216 }
217
218 // Check for further subdivision
219 if(currentDepth < maxDepth - 1) {
220   var innerCubeIndicesList = this.generateInnerCubeIndicesList(gridShape);
221   for(var indices of innerCubeIndicesList) {
222     var cubeCorners = this.getCubeCorners(N, indices);
223     var boundaryValues_sub = cubeCorners.map(corner => corner[corner.length - 1]);
224
225     var allSame = boundaryValues_sub.every(val => val ===
226       boundaryValues_sub[0]);
227     if(!allSame) {
228       var startCoords_sub = cubeCorners[0].slice(0, numDimensions);
229       var [Nin_sub, Nout_sub] = this.makeNodesND(
230         startCoords_sub,
231         boundaryValues_sub,
232         currentDepth + 1,
233         stepSizesPerDepth,
234         subdivisionPerDepth,
235         conditionFunction
236       );
237       Nin = Nin.concat(Nin_sub);
238       Nout = Nout.concat(Nout_sub);
239     }
240   }
241   return [Nin, Nout];
242 }
243
244 /**
245 * Generates all corner shifts for a hypercube in N dimensions.

```

```

247  * @param {number} numDimensions - Number of dimensions.
248  * @returns {number[][]} - Array of shifts for each corner.
249  */
250 generateCornerShifts(numDimensions) {
251   var shifts = [];
252   var totalCorners = 1 << numDimensions;
253   for(var i = 0; i < totalCorners; i++) {
254     var shift = [];
255     for(var j = 0; j < numDimensions; j++) {
256       shift.push((i >> j) & 1);
257     }
258     shifts.push(shift);
259   }
260   return shifts;
261 }
262
263 /**
264  * Retrieves the indices of corner points in the grid.
265  * @param {number[]} gridShape - Shape of the grid.
266  * @returns {number[][]} - List of corner indices.
267  */
268 getCornerIndices(gridShape) {
269   var numDimensions = gridShape.length;
270   var cornerShifts = this.generateCornerShifts(numDimensions);
271   return cornerShifts.map(shift => shift.map((s, idx) => s * (gridShape[idx] - 1)));
272 }
273
274 /**
275  * Generates a list of indices for nodes that need to be calculated.
276  * @param {number[]} gridShape - Shape of the grid.
277  * @param {Array} nf - N-dimensional array indicating nodes to compute.
278  * @returns {number[][][]} - List of node indices where nf is 1.
279  */
280 generateIndicesList(gridShape, nf) {
281   var indicesList = [];
282
283   var generateIndices = (indices, dim) => {
284     if(dim === gridShape.length) {
285       if(getValueAt(nf, indices) === 1) {
286         indicesList.push([...indices]);
287       }
288       return;
289     }
290     for(var i = 0; i < gridShape[dim]; i++) {
291       indices[dim] = i;
292       generateIndices(indices, dim + 1);
293     }
294   };
295   generateIndices(new Array(gridShape.length).fill(0), 0);
296   return indicesList;
297 }
298
299 /**
300  * Generates a list of starting indices for inner hypercubes.

```

```

301   * @param {number[]} gridShape - Shape of the grid.
302   * @returns {number[][]} - List of inner cube starting indices.
303   */
304 generateInnerCubeIndicesList(gridShape) {
305   var indicesList = [];
306
307   var generateIndices = (indices, dim) => {
308     if(dim === gridShape.length) {
309       indicesList.push([... indices]);
310       return;
311     }
312     if(gridShape[dim] <= 1) {
313       // No cubes along this dimension
314       return;
315     }
316     for(var i = 0; i < gridShape[dim] - 1; i++) {
317       indices[dim] = i;
318       generateIndices(indices, dim + 1);
319     }
320   };
321   generateIndices(new Array(gridShape.length).fill(0), 0);
322   return indicesList;
323 }
324
325 /**
326  * Retrieves the corner coordinates and values of a hypercube.
327  * @param {Array} N - N-dimensional grid.
328  * @param {number[]} indices - Starting indices of the hypercube.
329  * @returns {Array} - Array of corner coordinates and values.
330  */
331 getCubeCorners(N, indices) {
332   var numDimensions = indices.length;
333   var cornerShifts = this.generateCornerShifts(numDimensions);
334   var cubeCorners = cornerShifts.map(shift => {
335     var cornerIndices = indices.map((idx, dim) => idx + shift[dim]);
336     return getValueAt(N, cornerIndices);
337   });
338   return cubeCorners;
339 }
340 }
341
342 exports.PRDC_JSLAB_GEOMETRY_SPACE_SERACH = PRDC_JSLAB_GEOMETRY_SPACE_SERACH;

```

Listing 85 - geometry-spacesearch.js

```

1 /**
2  * @file JSLAB library geometry submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 var { PRDC_JSLAB_GEOMETRY_SPACE_SERACH } = require('./geometry-spacesearch');
9
10 /**
11  * Class for JSLAB geometry submodule.

```

```

12  */
13 class PRDC_JSLAB_LIB_GEOMETRY {
14
15  /**
16   * Initializes a new instance of the geometry submodule, providing access to
17   * geometry manipulation utilities.
18   * @param {Object} js1 Reference to the main JSLAB object.
19   */
20  constructor(js1) {
21    var obj = this;
22    this.js1 = js1;
23  }
24
25  /**
26   * Creates an instance of PRDC_JSLAB_LIB_OPTIM_SPACE_SERACH.
27   */
28  spaceSearch(... args) {
29    return new PRDC_JSLAB_GEOMETRY_SPACE_SERACH(... args);
30  }
31
32  /**
33   * Returns the shortest distance from point P to the line defined by (A, i)
34   * and the closest point (P1) on that line.
35   * @param {number[]} P - point [Px, Py, Pz]
36   * @param {number[]} A - a point on the line
37   * @param {number[]} i - a unit direction vector of the line
38   * @returns {{ d: number, P1: number[] }}
39   *   d - shortest distance
40   *   P1 - point on the line with the smallest distance to P
41   */
42  pointLineDistance(P, A, i) {
43    // P1 = A + dot(P - A, i) * i
44    const PA = minus(P, A);
45    const distAlongI = dot(PA, i);
46    const P1 = plus(A, scale(i, distAlongI));
47
48    // Distance = || P - P1 ||
49    const d = norm(minus(P, P1));
50
51    return { d, P1 };
52  }
53  /**
54   * Returns the intersection points of a circle (center O, radius r)
55   * in a plane with a line passing through point P with direction i.
56   * @param {number[]} P - point on the line
57   * @param {number[]} i - direction vector of the line (unit)
58   * @param {number[]} O - center of the circle
59   * @param {number} r - circle radius
60   * @returns {{ P1: number[] | null, P2: number[] | null, flag: number }}
61   *   flag = 0 -> intersection (two points)
62   *   flag = 1 -> tangent (one point)
63   *   flag = 2 -> no intersection
64   */
65  lineCircleIntersection(P, i, O, r) {
    let P1 = null;

```

```

66   let P2 = null;
67   let flag = 0;
68
69   // Distance from O to line & the closest point A
70   const { d, P1: A } = this.pointLineDistance(O, P, i);
71
72   if(d < r) {
73     // Two intersection points
74     const h = Math.sqrt(r * r - d * d); // half of the chord length
75     P1 = plus(A, scale(i, h));
76     P2 = plus(A, scale(i, -h));
77     flag = 0; // intersection
78   } else if(Math.abs(d - r) <= EPS) {
79     // Tangent
80     P1 = A;
81     flag = 1;
82   } else {
83     // No intersection
84     flag = 2;
85   }
86
87   return { P1, P2, flag };
88 }
89
90 /**
91 * Returns the line (point P, direction i) that is the intersection
92 * of two planes, or indicates if they are the same or parallel.
93 * @param {number[]} P1 - a point in plane 1
94 * @param {number[]} n1 - normal to plane 1
95 * @param {number[]} P2 - a point in plane 2
96 * @param {number[]} n2 - normal to plane 2
97 * @returns {{ P: number[] | null, i: number[] | null, flag: number }}
98 *   flag = 0 -> planes intersect
99 *   flag = 1 -> planes are the same
100 *  flag = 2 -> planes are parallel (no intersection)
101 */
102 planesIntersection(P1, n1, P2, n2) {
103   let P = null;
104   let i = null;
105   let flag = 0;
106
107   const V = cross3D(n1, n2, 1);
108
109   const V_norm = norm(V);
110   if(V_norm > EPS) {
111     // planes intersect
112     i = scale(V, 1.0 / V_norm); // unit direction
113
114     // Solve for a point on the intersection line:
115     // We want to solve the system:
116     // dot(n1, X) = dot(n1, P1)
117     // dot(n2, X) = dot(n2, P2)
118     //
119     // We'll attempt x=0, y=0, z=0 approach or check sub-determinants.
120     const A = [

```

```

121      [ n1[0] ,  n1[1] ,  n1[2] ] ,
122      [ n2[0] ,  n2[1] ,  n2[2] ] ,
123    ];
124    const B = [ dot(n1, P1) ,  dot(n2, P2) ];
125
126    // We try ignoring one coordinate at a time (k=1 to 3):
127    let solved = false ;
128    for(let k = 0; k < 3; k++) {
129      // Indices [0,1,2], skip k => j
130      const j = [0, 1, 2].filter(idx => idx !== k);
131
132      // Build a 2x2 submatrix from A, using columns j[0], j[1]
133      const subA = [
134        [A[0][j[0]], A[0][j[1]]] ,
135        [A[1][j[0]], A[1][j[1]]] ,
136      ];
137      const detSubA = subA[0][0] * subA[1][1] - subA[0][1] * subA[1][0];
138
139      if(Math.abs(detSubA) > EPS) {
140        // We can solve
141        // subA * C = B
142        // C is 2x1 => we solve by 2x2 inverse
143        const invDet = 1.0 / detSubA;
144        const C0 = invDet * ( B[0]*subA[1][1] - B[1]*subA[0][1] );
145        const C1 = invDet * ( -B[0]*subA[1][0] + B[1]*subA[0][0] );
146
147        // Build full solution X
148        const X = [0, 0, 0];
149        X[j[0]] = C0;
150        X[j[1]] = C1;
151        P = X;
152        solved = true ;
153        break;
154      }
155    }
156    if(!solved) {
157      // fallback: planes might be very close , etc .
158      P = [0, 0, 0];
159    }
160  } else {
161    // Check if they are the same plane
162    // We test if P2 satisfies plane 1 => dot(n1, P1-P2)=0 (within EPS).
163    const diff = minus(P1, P2);
164    if(Math.abs(dot(n1, diff)) <= EPS) {
165      // same plane
166      flag = 1;
167    } else {
168      // parallel planes
169      flag = 2;
170    }
171  }
172
173  return { P, i, flag };
174}
175

```

```

176  /**
177   * Checks if point P lies on the line segment A–B.
178   * Returns 1 if on segment, 0 otherwise.
179   * @param {number[]} P
180   * @param {number[]} A
181   * @param {number[]} B
182   * @returns {number} 1 (on segment), 0 (not on segment)
183   */
184  isPointOnLine(P, A, B) {
185    // i = (B - A) / ||B - A||
186    const AB = minus(B, A);
187    const i = scale(AB, 1.0 / norm(AB));
188
189    // dot(A - P, i) and dot(B - P, i)
190    const d1 = dot(minus(A, P), i);
191    const d2 = dot(minus(B, P), i);
192
193    // If both dot products have the same sign, P is outside the segment
194    if((d1 > 0 && d2 > 0) || (d1 < 0 && d2 < 0)) {
195      return 0;
196    }
197    return 1;
198  }
199
200 /**
201  * Returns the overlapping segment (if any) between two segments P1–P2 and
202  * P3–P4,
203  * or indicates no overlap.
204  * @param {number[]} P1
205  * @param {number[]} P2
206  * @param {number[]} P3
207  * @param {number[]} P4
208  * @returns {{ A: number[] | null, B: number[] | null, flag: number, id1: number
209  *           | null, id2: number | null }}
210  */
211  linesOverlap(P1, P2, P3, P4) {
212    let A = null;
213    let B = null;
214    let id1 = null;
215    let id2 = null;
216    let flag = 0;
217
218    // i = (P2 - P1) / ||P2 - P1||
219    const v = minus(P2, P1);
220    const len = norm(v);
221    const i = scale(v, 1.0 / len);
222
223    const P = [P1, P2, P3, P4];
224
225    const tArr = [];
226    tArr.push([0, 1, 1]);
227    tArr.push([dot(minus(P2, P1), i), 2, 1]);
228    tArr.push([dot(minus(P3, P1), i), 3, 2]);
229    tArr.push([dot(minus(P4, P1), i), 4, 2]);

```

```

229   // Sort rows by the first column
230   tArr.sort((a, b) => a[0] - b[0]);
231
232   if(Math.abs(tArr[0][2] - tArr[1][2]) <= EPS) {
233     // no overlap
234     flag = 1;
235   } else {
236     id1 = tArr[1][1];
237     id2 = tArr[2][1];
238     A = P[id1 - 1];
239     B = P[id2 - 1];
240     flag = 0;
241   }
242
243   return { A, B, flag, id1, id2 };
244 }
245
246 /**
247 * Finds the minimal 3D distance between all pairs of points in two arrays.
248 *
249 * @param {number[][]} P1i - Array of 3D points (e.g. [[x1, y1, z1], [x2, y2
250 , z2], ...]).
251 * @param {number[][]} P2i - Another array of 3D points.
252 * @returns {{ L: number, P1: number[], P2: number[] }}
253 *   L - The minimal distance found.
254 *   P1 - The point in P1i corresponding to the minimal distance.
255 *   P2 - The point in P2i corresponding to the minimal distance.
256 */
257 minPointsDistance3D(P1i, P2i) {
258   var P1ia = P1i.flat();
259   var P2ia = P2i.flat();
260
261   let L = Infinity;
262   let I = -1;
263   let J = -1;
264
265   for(let i = 0; i < P1ia.length; i += 3) {
266     for(let j = 0; j < P2ia.length; j += 3) {
267       const dx = P1ia[i] - P2ia[j];
268       const dy = P1ia[i+1] - P2ia[j+1];
269       const dz = P1ia[i+2] - P2ia[j+2];
270
271       const dist = Math.sqrt(dx*dx + dy*dy + dz*dz);
272
273       if (dist < L) {
274         L = dist;
275         I = i;
276         J = j;
277       }
278     }
279   }
280
281   const closestP1 = [P1ia[I], P1ia[I + 1], P1ia[I + 2]];
282   const closestP2 = [P2ia[J], P2ia[J + 1], P2ia[J + 2]];

```

```

283     return {
284         L,
285         P1: closestP1 ,
286         P2: closestP2
287     };
288 }
289
290 /**
291 * Generates a rotation matrix to rotate from vector a to vector b.
292 * @param {number[]} a - The initial unit vector.
293 * @param {number[]} b - The target unit vector.
294 * @returns {number[][]} - The resulting rotation matrix.
295 */
296 getRotationMatrix(a, b) {
297     // a and b are unit vectors
298     var v = this.jsl.array.cross3D(a, b, 1);
299     var s = this.jsl.env.math.norm(v);
300     var c = this.jsl.array.dotVector(a, b);

301     if(c === -1) { // Vectors are opposite
302         // Find a vector orthogonal to 'a'
303         var orthogonal = [0, 0, 0];
304         if(this.jsl.env.math.abs(a[0]) < this.jsl.env.math.abs(a[1]) &&
305             this.jsl.env.math.abs(a[0]) < this.jsl.env.math.abs(a[2])) {
306             orthogonal = [0, -a[2], a[1]];
307         } else if(this.jsl.env.math.abs(a[1]) < this.jsl.env.math.abs(a[2])) {
308             orthogonal = [-a[2], 0, a[0]];
309         } else {
310             orthogonal = [-a[1], a[0], 0];
311         }
312     }
313     orthogonal = this.jsl.array.normalizeVector(orthogonal);

314     // Compute rotation matrix using Householder reflection
315     var o = orthogonal;
316     return [
317         [-1 + 2 * o[0] * o[0], 2 * o[0] * o[1], 2 * o[0] * o[2],
318          2 * o[1] * o[0], -1 + 2 * o[1] * o[1], 2 * o[1] * o[2],
319          2 * o[2] * o[0], 2 * o[2] * o[1], -1 + 2 * o[2] * o[2]],
320     ];
321 } else if(s === 0) { // Vectors are the same
322     // Return identity matrix
323     return [1, 0, 0, 0, 1, 0, 0, 0, 1];
324 } else {
325     // Compute skew-symmetric cross-product matrix of v
326     var vx = this.jsl.array.skewVector(v);

327     // Compute R = I + vx + vx * vx * ((1 - c) / (s * s))
328     var I = [1, 0, 0, 0, 1, 0, 0, 0, 1];

329     var vx2 = this.jsl.array.multiply(vx, vx, 3, 3, 3);
330
331     var factor = (1 - c) / (s * s);

332     var R = this.jsl.array.plus(this.jsl.array.plus(I, vx), this.jsl.array.
333     scale(vx2, factor));

```

```

337         return R;
338     }
339 }
340 }
341 /**
342 * Transforms coordinates by scaling , rotating , and translating them.
343 * @param {number[][]} coordinates - Array of coordinate points.
344 * @param {number} scale_factor - Factor by which to scale the coordinates.
345 * @param {number[][]} rotation_matrix - Matrix used to rotate the
346   coordinates .
347 * @param {number[]} translation - Vector used to translate the coordinates.
348 * @returns {number[][]} - The transformed coordinates.
349 */
350 transform(coordinates , scale_factor , rotation_matrix , translation) {
351   var obj = this;
352   return coordinates.map(function(coordinate) {
353     var transformed = obj.jsl.array.plus(translation , obj.jsl.array.multiply
354       (rotation_matrix , obj.jsl.array.scale(coordinate , scale_factor) , 3 ,
355        3 , 1));
356     return transformed;
357   });
358 /**
359 * Creates 3D vectors for plotting based on provided parameters.
360 * @param {number[]} xi - X coordinates of vector origins.
361 * @param {number[]} yi - Y coordinates of vector origins.
362 * @param {number[]} zi - Z coordinates of vector origins.
363 * @param {number[]} ui - X components of vectors.
364 * @param {number[]} vi - Y components of vectors.
365 * @param {number[]} wi - Z components of vectors.
366 * @param {number} scale - Scale factor for the vectors.
367 * @param {number} angle_factor - Angle factor for arrowheads.
368 * @param {Object} opts - Additional plotting options.
369 * @returns {Object} - An object containing line and head trace data for
370   plotting.
371 */
372 createVectors3D(xi , yi , zi , ui , vi , wi , scale = 0.3 , angle_factor = 0.4 ,
373   opts) {
374   if(!Array.isArray(xi)) xi = [xi];
375   if(!Array.isArray(yi)) yi = [yi];
376   if(!Array.isArray(zi)) zi = [zi];
377   if(!Array.isArray(ui)) ui = [ui];
378   if(!Array.isArray(vi)) vi = [vi];
379   if(!Array.isArray(wi)) wi = [wi];
380   // Define the unit arrow once
381   var arrowhead_length = scale * 1;
382   var arrowhead_width = arrowhead_length * this.jsl.env.math.tan(
383     angle_factor);
384   // Shaft points (unit arrow along x-axis)
385   var shaft_start = [0, 0, 0];
386   var shaft_end = [1 - arrowhead_length , 0, 0];

```

```

386
387 // Arrowhead base points
388 var arrow_tip = [1, 0, 0]; // Tip at the end of the shaft
389 var arrow_left = [1 - arrowhead_length, arrowhead_width, 0];
390 var arrow_right = [1 - arrowhead_length, -arrowhead_width, 0];
391
392 // All points of the unit arrow
393 var unit_arrow_points = [shaft_start, shaft_end, arrow_tip, arrow_left,
394   arrow_right];
395
396 var vectors = {
397   line: {
398     x: [],
399     y: [],
400     z: [],
401     type: 'scatter3d', color: '#00f', mode: 'lines', showLegend: false
402   },
403   head: {
404     x: [],
405     y: [],
406     z: [],
407     i: [],
408     j: [],
409     k: [],
410     type: 'mesh3d', color: '#00f', opacity: 1, flatShading: true,
411     showScale: false, showLegend: false, lighting: {ambient: 1}
412   }
413 };
414
415 if(typeof opts == 'object') {
416   Object.assign(vectors.line, opts);
417   Object.assign(vectors.head, opts);
418 }
419
420 var vertex_index = 0;
421 var x_axis = [1, 0, 0];
422
423 for(var i = 0; i < xi.length; i++) {
424   var x0 = xi[i];
425   var y0 = yi[i];
426   var z0 = zi[i];
427   var u = ui[i];
428   var v = vi[i];
429   var w = wi[i];
430
431   var vector = [u, v, w];
432
433   // Calculate the length of the vector
434   var length = this.jsl.env.math.norm(vector);
435   if(length == 0) continue; // Skip zero-length vectors
436
437   // Normalize the direction vector
438   var dir = this.jsl.array.normalizeVector(vector);
439
440   // Compute rotation matrix to rotate from x-axis to dir

```

```

440 var R = this.getRotationMatrix(x_axis, dir);
441
442 // Scale factor is the length of the vector
443 var scale_factor = length;
444
445 // Translation vector is the starting point (x0, y0, z0)
446 var translation = [x0, y0, z0];
447
448 // Transform the unit arrow points
449 var transformed_points = this.transform(unit_arrow_points, scale_factor,
450   R, translation);
451
452 // Extract points
453 var shaft_start_rot = transformed_points[0];
454 var shaft_end_rot = transformed_points[1];
455 var arrow_tip_rot = transformed_points[2];
456 var arrow_left_rot = transformed_points[3];
457 var arrow_right_rot = transformed_points[4];
458
459 // Add shaft line to lines arrays
460 vectors.line.x.push(shaft_start_rot[0], shaft_end_rot[0], null);
461 vectors.line.y.push(shaft_start_rot[1], shaft_end_rot[1], null);
462 vectors.line.z.push(shaft_start_rot[2], shaft_end_rot[2], null);
463
464 // Add arrowhead edges to lines arrays
465 vectors.line.x.push(
466   arrow_tip_rot[0], arrow_left_rot[0], null,
467   arrow_left_rot[0], arrow_right_rot[0], null,
468   arrow_right_rot[0], arrow_tip_rot[0], null
469 );
470 vectors.line.y.push(
471   arrow_tip_rot[1], arrow_left_rot[1], null,
472   arrow_left_rot[1], arrow_right_rot[1], null,
473   arrow_right_rot[1], arrow_tip_rot[1], null
474 );
475 vectors.line.z.push(
476   arrow_tip_rot[2], arrow_left_rot[2], null,
477   arrow_left_rot[2], arrow_right_rot[2], null,
478   arrow_right_rot[2], arrow_tip_rot[2], null
479 );
480
481 // Add arrowhead vertices to mesh arrays
482 vectors.head.x.push(arrow_tip_rot[0], arrow_left_rot[0], arrow_right_rot
483   [0]);
484 vectors.head.y.push(arrow_tip_rot[1], arrow_left_rot[1], arrow_right_rot
485   [1]);
486 vectors.head.z.push(arrow_tip_rot[2], arrow_left_rot[2], arrow_right_rot
487   [2]);
488
489 // Define the face for the current arrowhead
490 vectors.head.i.push(vertex_index);
491 vectors.head.j.push(vertex_index + 1);
492 vectors.head.k.push(vertex_index + 2);
493
494 // Update vertex index for the next arrow

```

```

491         vertex_index += 3;
492     }
493     return vectors;
494 }
495
496 /**
497 * Creates 3D disks for plotting based on provided parameters.
498 * @param {number[]} xi - X coordinates of disk centers.
499 * @param {number[]} yi - Y coordinates of disk centers.
500 * @param {number[]} zi - Z coordinates of disk centers.
501 * @param {number[]} ri - Radii of the disks.
502 * @param {number[]} ui - X components of normal vectors (for disk
503 orientation).
504 * @param {number[]} vi - Y components of normal vectors (for disk
505 orientation).
506 * @param {number[]} wi - Z components of normal vectors (for disk
507 orientation).
508 * @param {Object} opts - Additional plotting options.
509 * @returns {Object} - An object containing line and area trace data for
510 plotting.
511 */
512 createDisks3D(xi, yi, zi, ri, ui, vi, wi, opts) {
513   if(!Array.isArray(xi)) xi = [xi];
514   if(!Array.isArray(yi)) yi = [yi];
515   if(!Array.isArray(zi)) zi = [zi];
516   if(!Array.isArray(ri)) ri = [ri];
517   if(!Array.isArray(ui)) ui = [ui];
518   if(!Array.isArray(vi)) vi = [vi];
519   if(!Array.isArray(wi)) wi = [wi];
520   if(ri.length === 1 && xi.length > 1) ri = new Array(xi.length).fill(ri[0])
521   ;
522
523   var segments = opts.segments || 32;
524   delete opts.segments;
525
526   var disks = {
527     line: {
528       x: [],
529       y: [],
530       z: [],
531       type: 'scatter3d', color: '#00f', mode: 'lines', showLegend: false
532     },
533     area: {
534       x: [],
535       y: [],
536       z: [],
537       i: [],
538       j: [],
539       k: [],
540       type: 'mesh3d', color: '#00f', opacity: 1, flatShading: true,
541       showScale: false, showLegend: false, lighting: {ambient: 1}
542     }
543   };
544
545   if(typeof opts === 'object') {

```

```

541   Object.assign(disks.line, opts);
542   Object.assign(disks.area, opts);
543 }
544
545   var vertex_index = 0;
546   var z_axis = [0, 0, 1]; // Assuming disk normal initially aligns with x-
547   axis
548
549   // Create a unit circle in the XY plane for the disk
550   var points = [...this.circle(1, segments)];
551
552   for(var i = 0; i < xi.length; i++) {
553     var x0 = xi[i];
554     var y0 = yi[i];
555     var z0 = zi[i];
556     var radius = ri[i];
557     var normal = [ui[i], vi[i], wi[i]];
558
559     if(radius === 0) continue; // Skip zero-radius disks
560
561     // Normalize the normal vector to ensure it's a unit vector
562     var dir = this.jsl.array.normalizeVector(normal);
563
564     // Compute rotation matrix to rotate from x-axis to the normal vector
565     var R = this.getRotationMatrix(z_axis, dir);
566
567     // Translation vector is the disk center point
568     var translation = [x0, y0, z0];
569
570     var circle = [...points];
571
572     // Transform the circle points to the correct position and orientation
573     var transformed_points = this.transform(circle, radius, R, translation);
574
575     // Add circle outline to line data for edges
576     for(var j = 0; j < transformed_points.length; j++) {
577       var point = transformed_points[j];
578       disks.line.x.push(point[0]);
579       disks.line.y.push(point[1]);
580       disks.line.z.push(point[2]);
581       if(j === transformed_points.length - 1) {
582         disks.line.x.push(transformed_points[0][0], null); // Connect last
583         to first point
584         disks.line.y.push(transformed_points[0][1], null);
585         disks.line.z.push(transformed_points[0][2], null);
586       }
587     }
588   }
589
590   // Add all points for mesh (area) data
591   for(var j = 0; j < transformed_points.length; j++) {
592     var point = transformed_points[j];
593     disks.area.x.push(point[0]);

```

```

594     disks.area.y.push(point[1]);
595     disks.area.z.push(point[2]);
596   }
597
598   // Triangulation for mesh
599   for(var j = 1; j < transformed_points.length - 1; j++) {
600     disks.area.i.push(vertex_index);
601     disks.area.j.push(vertex_index + j);
602     disks.area.k.push(vertex_index + j + 1);
603   }
604   vertex_index += transformed_points.length;
605 }
606 return disks;
607 }
608 */
609 * Creates a rectangular planes in 3D space, oriented by a normal vector [u,
610   v, w].
611 * @param {number[]} xi - X coordinates of planes centers.
612 * @param {number[]} yi - Y coordinates of planes centers.
613 * @param {number[]} zi - Z coordinates of planes centers.
614 * @param {number[]} width_i - Width of the rectangle.
615 * @param {number[]} height_i - Height of the rectangle.
616 * @param {number[]} ui - X component of the plane's normal vector.
617 * @param {number[]} vi - Y component of the plane's normal vector.
618 * @param {number[]} wi - Z component of the plane's normal vector.
619 * @param {Object} opts - Additional plotting options (color, opacity,
620   etc.).
621 * @returns {Object} - An object containing line and area trace data
622   for plotting.
623 */
624 createPlanes3D(xi, yi, zi, width_i, height_i, ui, vi, wi, opts) {
625   if(!Array.isArray(xi)) xi = [xi];
626   if(!Array.isArray(yi)) yi = [yi];
627   if(!Array.isArray(zi)) zi = [zi];
628   if(!Array.isArray(width_i)) width_i = [width_i];
629   if(!Array.isArray(height_i)) height_i = [height_i];
630   if(!Array.isArray(ui)) ui = [ui];
631   if(!Array.isArray(vi)) vi = [vi];
632   if(!Array.isArray(wi)) wi = [wi];
633   if(width_i.length === 1 && xi.length > 1) width_i = new Array(xi.length)
634     .fill(width_i[0]);
635   if(height_i.length === 1 && xi.length > 1) height_i = new Array(xi.length)
636     .fill(height_i[0]);
637
638   const planes = {
639     line: {
640       x: [],
641       y: [],
642       z: [],
643       type: 'scatter3d',
644       color: '#00f',
645       mode: 'lines',
646       showLegend: false
647     }
648   }
649   return planes;
650 }

```

```

644 },
645 area: {
646   x: [],
647   y: [],
648   z: [],
649   i: [],
650   j: [],
651   k: [],
652   type: 'mesh3d',
653   color: '#00f',
654   opacity: 1,
655   flatShading: true,
656   showScale: false,
657   showLegend: false,
658   lighting: { ambient: 1 }
659 }
660 };
661
662 // Merge any provided opts into our line & area objects
663 if(typeof opts === 'object') {
664   Object.assign(planes.line, opts);
665   Object.assign(planes.area, opts);
666 }
667
668 // Reference axis (z-axis) that our rectangle initially lies in (XY-plane)
669 .
670 // We will rotate from z-axis to our normal.
671 const z_axis = [0, 0, 1];
672
673 for(var i = 0; i < ui.length; i++) {
674   // Create a symmetrical rectangle in the XY-plane, centered at (0,0,0)
675   const rect_coords = this.symRectangle(width_i[i], height_i[i], 0);
676
677   // Convert that flat array into point-triplets:
678   var rectangle_points = [];
679   for(let i = 0; i < rect_coords.length; i += 3) {
680     rectangle_points.push([rect_coords[i], rect_coords[i + 1], rect_coords[i + 2]]);
681   }
682
683   // Normal vector
684   const normal = [ui[i], vi[i], wi[i]];
685
686   // Normalize the plane normal
687   const dir = this.jsl.array.normalizeVector(normal);
688
689   // Compute rotation matrix to rotate a plane (lying in XY-plane) so its
690   // normal aligns with 'dir'
691   const R = this.getRotationMatrix(z_axis, dir);
692
693   // Rotate these points according to R
694   // No scaling or translation is applied here. If you want to shift it to
695   // [x0,y0,z0], just add that translation.
696   rectangle_points = this.transform(rectangle_points, 1.0, R, [xi[i], yi[i],
697   zi[i]]);
```

```

694
695    // Build the line trace: push each consecutive segment plus a null to
696    // break the stroke
697    for(let j = 0; j < rectangle_points.length; j++) {
698      var [x, y, z] = rectangle_points[j];
699      planes.line.x.push(x);
700      planes.line.y.push(y);
701      planes.line.z.push(z);
702    }
703    // Insert null to break the line
704    planes.line.x.push(null);
705    planes.line.y.push(null);
706    planes.line.z.push(null);

707    // Build the mesh: we only need the first 4 unique corners for a
708    // rectangle
709    // (the 5th is a repeat of the 1st).
710    // Triangulate the rectangle as two triangles: (0,1,2) and (0,2,3)
711    const n = 4; // We only take indices 0..3
712    const base_index = 0;

713    for(let j = 0; j < n; j++) {
714      planes.area.x.push(rectangle_points[j][0]);
715      planes.area.y.push(rectangle_points[j][1]);
716      planes.area.z.push(rectangle_points[j][2]);
717    }
718    // Two triangles to form the quad
719    planes.area.i.push(base_index, base_index);
720    planes.area.j.push(base_index + 1, base_index + 2);
721    planes.area.k.push(base_index + 2, base_index + 3);
722  }
723  return planes;
724}
725
726 /**
727 * Creates a lines in 3D space.
728 * @param {number[]} x1i - X1 coordinates of lines.
729 * @param {number[]} y1i - Y1 coordinates of lines.
730 * @param {number[]} z1i - Z1 coordinates of lines.
731 * @param {number[]} x2i - X2 coordinates of lines.
732 * @param {number[]} y2i - Y2 coordinates of lines.
733 * @param {number[]} z2i - Z2 coordinates of lines.
734 * @returns {Object} - lines object.
735 */
736 createLines3D(x1i, y1i, z1i, x2i, y2i, z2i, opts) {
737   if(!Array.isArray(x1i)) x1i = [x1i];
738   if(!Array.isArray(y1i)) y1i = [y1i];
739   if(!Array.isArray(z1i)) z1i = [z1i];
740   if(!Array.isArray(x2i)) x2i = [x2i];
741   if(!Array.isArray(y2i)) y2i = [y2i];
742   if(!Array.isArray(z2i)) z2i = [z2i];

743   const lines = {
744     x: [],
745     y: [],
746

```

```
747     z: [] ,
748     type: 'scatter3d',
749     color: '#00f',
750     mode: 'lines',
751     showLegend: false
752   };
753
754   // Merge any provided opts into our line & area objects
755   if(typeof opts === 'object') {
756     Object.assign(lines, opts);
757   }
758
759   for(var i = 0; i < x1i.length; i++) {
760     lines.x.push(x1i[i], x2i[i], null);
761     lines.y.push(y1i[i], y2i[i], null);
762     lines.z.push(z1i[i], z2i[i], null);
763   }
764   return lines;
765 }
766
767 /**
768 * Creates a points in 3D space.
769 * @param {number[]} xi - X coordinates of points.
770 * @param {number[]} yi - Y coordinates of points.
771 * @param {number[]} zi - Z coordinates of points.
772 * @returns {Object} - points object.
773 */
774 createPoints3D(xi, yi, zi, opts) {
775   if(!Array.isArray(xi)) xi = [xi];
776   if(!Array.isArray(yi)) yi = [yi];
777   if(!Array.isArray(zi)) zi = [zi];
778
779   const points = {
780     x: [],
781     y: [],
782     z: [],
783     type: 'scatter3d',
784     color: '#00f',
785     mode: 'markers',
786     showLegend: false
787   };
788
789   // Merge any provided opts into our line & area objects
790   if(typeof opts === 'object') {
791     Object.assign(points, opts);
792   }
793
794   for(var i = 0; i < xi.length; i++) {
795     points.x.push(xi[i], null);
796     points.y.push(yi[i], null);
797     points.z.push(zi[i], null);
798   }
799   return points;
800 }
801
```

```

802  /**
803   * Creates a points in 3D space.
804   * @param {number[]} xi - X coordinates of points.
805   * @param {number[]} yi - Y coordinates of points.
806   * @param {number[]} zi - Z coordinates of points.
807   * @returns {Object} - points object.
808  */
809  createText3D(xi, yi, zi, texti, dxi, dyi, dzi, opts) {
810    if(!Array.isArray(xi)) xi = [xi];
811    if(!Array.isArray(yi)) yi = [yi];
812    if(!Array.isArray(zi)) zi = [zi];
813    if(!Array.isArray(texti)) texti = [texti];
814    if(!Array.isArray(dxi)) dxi = [dxi];
815    if(!Array.isArray(dyi)) dyi = [dyi];
816    if(!Array.isArray(dzi)) dzi = [dzi];
817    if(dxi.length === 1 && xi.length > 1) dxi = new Array(xi.length).fill(dxi[0]);
818    if(dyi.length === 1 && yi.length > 1) dyi = new Array(yi.length).fill(dyi[0]);
819    if(dzi.length === 1 && zi.length > 1) dzi = new Array(zi.length).fill(dzi[0]);
820
821    const texts = {
822      x: [],
823      y: [],
824      z: [],
825      text: [],
826      type: 'scatter3d',
827      textposition: 'center middle',
828      textfont: {
829        size: 18,
830        color: '#f00'
831      },
832      mode: 'text',
833      showLegend: false
834    };
835
836    // Merge any provided opts into our line & area objects
837    if(typeof opts === 'object') {
838      Object.assign(texts, opts);
839    }
840
841    for(var i = 0; i < xi.length; i++) {
842      texts.x.push(plus(xi[i], dxi[i]));
843      texts.y.push(plus(yi[i], dyi[i]));
844      texts.z.push(plus(zi[i], dzi[i]));
845      texts.text.push(texti[i]);
846    }
847    return texts;
848  }
849
850 /**
851  * Creates a symmetrical rectangle in 3D space.
852  * @param {number} W - Width of the rectangle.
853  * @param {number} H - Height of the rectangle.

```

```

854  * @param {number} [Z=0] - Z-coordinate for the rectangle plane.
855  * @returns {number[]} - Array of vertex coordinates for the rectangle.
856  */
857  symRectangle(W, H, Z = 0) {
858    return [W/2, H/2, Z,
859            -W/2, H/2, Z,
860            -W/2, -H/2, Z,
861            W/2, -H/2, Z,
862            W/2, H/2, Z];
863  }
864
865  /**
866   * Helper method to generate points for a circle in the XY plane.
867   * @param {number} radius - Radius of the circle.
868   * @param {number} segments - Number of segments for the circle.
869   * @returns {number[][]} - Array of points forming the circle.
870   */
871  circle(radius, segments) {
872    var points = [];
873    for(var i = 0; i < segments; i++) {
874      var angle = 2 * Math.PI * i / segments;
875      points.push([radius * Math.cos(angle), radius * Math.sin(angle), 0]);
876    }
877    return points;
878  }
879
880  /**
881   * Helper method to generate points for a disk in the XY plane.
882   * @param {number} radius - Radius of the disk.
883   * @param {number} segments_a - Number of angular segments for the disk.
884   * @param {number} segments_r - Number of radial segments for the disk.
885   * @returns {number[][][]} - Array of points forming the disk.
886   */
887  disk(radius, segments_a, segments_r) {
888    var points = [];
889    points.push([0, 0, 0]);
890    for(var j = 1; j <= segments_r; j++) {
891      var r = radius * j / segments_r;
892      for(var i = 1; i <= segments_a; i++) {
893        var angle = 2 * Math.PI * i / segments_a;
894        points.push([r * Math.cos(angle), r * Math.sin(angle), 0]);
895      }
896    }
897    return points;
898  }
899
900  /**
901   * Generates the boundary of a 3D shape based on points and a shrink factor.
902   * @param {number[][][]} points - Array of points defining the shape.
903   * @param {number} [shrink=0.5] - Factor by which to shrink the boundary.
904   * @returns {Array} - An array containing boundary facets and the volume.
905   */
906  boundary3D(points, shrink = 0.5) {
907    var shp = new this.jsl.env.AlphaShape3D();
908    shp.newShape(points);

```

```

909
910  var Acrit = shp.getCriticalAlpha('one-region');
911  var spec = shp.getAlphaSpectrum();
912
913  var idx = spec.indexOf(Acrit);
914  var subspec = spec.slice(idx);
915
916  var idx = Math.max(Math.ceil((1 - shrink) * subspec.length) - 1, 0);
917  var alphaval = subspec[idx];
918
919  shp.setAlpha(alphaval);
920  var V = shp.getVolume();
921  var bf = shp.getBoundaryFacets();
922  shp = null;
923  return [bf, V];
924 }
925
926 /**
927 * Writes geometry data to an OFF file.
928 * @param {string} filename - The path to the OFF file.
929 * @param {number[][]} vertices - Array of vertex coordinates.
930 * @param {number[][]} faces - Array of face indices.
931 */
932 writeOff(filename, vertices, faces) {
933   var shp = new this.jsl.env.AlphaShape3D();
934   shp.writeOff(filename, vertices, faces);
935   shp = null;
936 }
937
938 /**
939 * Reads an OFF file and returns the vertices and faces.
940 * @param {string} filename - The path to the OFF file.
941 * @returns {{ vertices: number[][], faces: number[][] }} - An object
942 * containing vertices and faces arrays.
943 * @throws Will throw an error if the file cannot be read or is not a valid
944 * OFF file.
945 */
946 readOff(filename) {
947   var data = this.jsl.env.readFileSync(filename, 'utf8');
948   var tokens = data.split(/\s+/).filter(token => token.length > 0);
949   if(tokens[0] !== 'OFF') {
950     this.jsl.env.error('@readOff: '+language.string(193));
951   }
952   tokens.shift();
953   if(tokens.length < 3) {
954     this.jsl.env.error('@readOff: '+language.string(194));
955   }
956   var j = 0;
957   tokens = tokens.map(x => parseFloat(x));
958   var nvert = tokens[j++];
959   var nface = tokens[j++];
960   if(!isNaN(tokens[j])) {
961     j++;

```

```

962     }
963
964     // Read vertex coordinates
965     var vertices = createArray(nvert);
966     var k = 0;
967     for(var i = 0; i < nvert; i++) {
968         if(j + 2 >= tokens.length) {
969             this.jsl.env.error('@readOff: '+language.string(195));
970             break;
971         }
972         vertices[k++] = [tokens[j++], tokens[j++], tokens[j++]];
973     }
974
975     // Read face data
976     var faces = createArray(nface);
977     var k = 0;
978     for(let i = 0; i < nface; i++) {
979         if(j >= tokens.length) {
980             this.jsl.env.error('@readOff: '+language.string(196));
981             break;
982         }
983         var vertices_per_face = tokens[j++];
984         var face = [];
985         for(let v = 0; v < vertices_per_face; v++) {
986             face.push(tokens[j++]);
987         }
988         faces[k++] = face;
989     }
990
991     return [vertices, faces];
992 }
993 }
994
995 exports.PRDC_JSLAB_LIB_GEOMETRY = PRDC_JSLAB_LIB_GEOMETRY;

```

Listing 86 - geometry.js

```

1  /**
2   * @file JSLAB sandbox window init file
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   */
7
8 // Modules
9 // -----
10 const helper = require('../js/helper.js');
11 const { PRDC_APP_CONFIG } = require('../config/config');
12 const { PRDC_JSLAB_LANGUAGE } = require('../js/language');
13
14 global.app_path = process.argv.find(e => e.startsWith('--app-path=')).split('=')
15   [1].replace(/\js\\?$/,'');
16
17 const { PRDC_JSLAB_LIB } = require('../js/sandbox/jslab');
18
19 // Global variables

```

```

19 var config = new PRDC_APP_CONFIG();
20 global.language = new PRDC_JSLAB_LANGUAGE();
21 var js1 = new PRDC_JSLAB_LIB(config);
22
23 if(config.TEST) {
24     const { PRDC_JSLAB_TESTER } = require("../js/tester.js");
25     tester = new PRDC_JSLAB_TESTER('sandbox');
26     tester.runTests();
27 }

```

Listing 87 - init-sandbox.js

```

1 /**
2  * @file Init worker
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  * @version 0.0.1
7  */
8 "use strict";
9
10 global.fs = require('fs');
11 global.path = require('path');
12 global.app_path = process.argv.find(e => e.startsWith('--app-path=')).split('=')
13   )[1].replace(/\js\?$/,'');
14
15 // Global variables
16 global.win = self;
17 global.worker_module;
18
19 /**
20  * Handle messages
21 */
22 self.addEventListener("message", function(e) {
23     if(e.data.type === 'configureWorker') {
24         var { PRDC_WORKER } = require(e.data.module_path);
25         global.worker_module = new PRDC_WORKER();
26     } else if(global.worker_module && e.data.hasOwnProperty('method')) {
27         global.worker_module[e.data.method](e.data);
28     }
29 });

```

Listing 88 - init-worker.js

```

1 /**
2  * @file JSLAB electron environment
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 if(!global.is_worker) {
9     var { ipcRenderer } = require('electron');
10 }
11
12 const { PRDC_JSLAB_FREECAD_LINK } = require('../freecad-link');

```

```

13 const { PRDC_JSLAB_OPENMODELICA_LINK } = require('./om-link');
14
15 const fs = require("fs");
16 const os = require('os');
17 const net = require('net');
18 const udp = require('dgram');
19 const cp = require("child_process");
20 const path = require("path");
21 const tcpPortUsed = require('tcp-port-used');
22 const { pathEqual } = require('path-equal');
23 const { Readable, Writable } = require('stream');
24 const { NativeModule } = require(app_path + '/build/Release/native_module');
25 const { AlphaShape3D } = require(app_path + '/build/Release/alpha_shape_3d');
26 const { extractFull } = require('node-7z');
27 const seedrandom = require('seedrandom');
28 const bin7zip = require('7zip-bin').path7za;
29 const PDFDocument = require('pdfkit');
30 const SVGtoPDF = require('svg-to-pdfkit');
31 const { PolynomialRegression } = require('ml-regression-polynomial');
32 const recast = require('recast');
33 const babel_parser = require('@babel/parser');
34 var SourceMapConsumer = require("source-map").SourceMapConsumer;
35 var { SerialPort } = require('serialport');
36
37 /**
38 * Class for JSLAB electron environment.
39 */
40 class PRDC_JSLAB_ENV {
41
42 /**
43 * Constructs a electron environment submodule object with access to JSLAB's
44 * electron environment functions.
45 * @constructor
46 * @param {Object} jsl - Reference to the main JSLAB object.
47 */
48 constructor(jsl) {
49   var obj = this;
50   this.jsl = jsl;
51
52   if(!global.is_worker) {
53     this.context = window;
54     this.debug = ipcRenderer.sendSync("sync-message", "get-debug-flag");
55     this.version = ipcRenderer.sendSync("sync-message", "get-app-version");
56     this.exe_path = ipcRenderer.sendSync("sync-message", "get-path", "exe");
57     this.platform = ipcRenderer.sendSync("sync-message", "get-platform");
58     this.speech = new SpeechSynthesisUtterance();
59     this.speech.voice = speechSynthesis.getVoices()[0];
60     this.navigator = navigator;
61     this.processors_number = navigator.hardwareConcurrency;
62   } else {
63     this.context = global;
64     this.debug = global.debug;
65     this.version = global.version;
66     this.exe_path = undefined;
67     this.platform = global.platform;

```

```

67   this . processors _ number = undefined ;
68 }
69 this . native _ module = new NativeModule () ;
70 this . AlphaShape3D = AlphaShape3D ;
71 this . bin7zip = bin7zip ;
72 this . seedRandom = seedrandom ;
73 this . extractFull = extractFull ;

74
75 this . Cesium = Cesium ;
76 this . process _ pid = process . pid ;
77 this . math = this . context . math ;
78 this . fmin = this . context . fmin ;
79 this . PDFDocument = PDFDocument ;
80 this . SVGtoPDF = SVGtoPDF ;
81 this . PolynomialRegression = PolynomialRegression ;
82 this . os = os ;
83 this . net = net ;
84 this . udp = udp ;
85 this . tcpPortUsed = tcpPortUsed ;
86 this . recast = recast ;
87 this . babel _ parser = babel _ parser ;
88 this . SourceMapConsumer = SourceMapConsumer ;
89 this . SourceMapConsumer . initialize ({
90   "lib / mappings . wasm": app _ path + '/ node _ modules / source - map / lib / mappings .
91   wasm ' ,
92 });
93 this . SerialPort = SerialPort ;

94 this . online = this . navigator . onLine ;
95 function onOnlineChange () {
96   obj . online = obj . navigator . onLine ;
97 }
98 this . context . addEventListener ( 'online' , onOnlineChange ) ;
99 this . context . addEventListener ( 'offline' , onOnlineChange ) ;

100
101 this . context . freecad _ link = new PRDC_JSLAB_FREECAD_LINK (this . jsl ) ;
102 this . context . om _ link = new PRDC_JSLAB_OPENMODELICA_LINK (this . jsl ) ;
103
104 // On IPC message
105 if (! global . is _ worker ) {
106   ipcRenderer . on ( "SandboxWindow" , function ( event , action , data ) {
107     switch ( action ) {
108       case "eval - code" :
109         obj . jsl . eval . evalCodeFromMain ( ... data ) ;
110         break ;
111       case "get - completions" :
112         ipcRenderer . send ( "completions -" + data [ 0 ] , obj . jsl . basic .
113           getCompletions ( data ) ) ;
114         break ;
115       case "stop - loop" :
116         obj . jsl . setStopLoop ( data ) ;
117         break ;
118       case "run - last - script" :
119         obj . jsl . eval . runLast () ;
120         break ;

```

```

120     case "set-current-path":
121         obj.jsl.setPath(data);
122         break;
123     case "set-saved-paths":
124         obj.jsl.setSavedPaths(data);
125         break;
126     case "set-language":
127         language.set(data);
128         obj.figures._updateLanguage();
129         obj.windows._updateLanguage();
130         break;
131     }
132   });
133 }
134
135 // Functions
136 this.setImmediate = this.context.setImmediate;
137 this.clearImmediate = this.context.clearImmediate;
138 this.pathEqual = pathEqual;
139 this.Readable = Readable;
140 this.Writable = Writable;
141
142 // Which properties and methods to export to context
143 this.exports = [ 'debug', 'version', 'platform' ];
144 }
145
146 /**
147 * Opens a window based on the provided ID.
148 * @param {number} id - The ID of the window to open.
149 * @returns {((boolean|BrowserWindow))} - The opened window or false if the
150 * window does not exist.
151 */
152 openWindow(wid, file = "blank.html") {
153   if(!global.isWorker) {
154     var obj = this;
155     var sub_context = this.context.open(file, wid);
156     sub_context.addEventListener("error", function(err) {
157       if(err && err.hasOwnProperty('error')) {
158         obj.error(err.error.stack);
159       } else {
160         obj.error(err.message);
161       }
162     });
163
164     var loadCheckInterval = setInterval(function() {
165       try {
166         if(isFinite(sub_context.wid)) {
167           clearInterval(loadCheckInterval);
168         }
169       } catch {
170         sub_context.close();
171         obj.jsl.windows._closedWindow(wid);
172         clearInterval(loadCheckInterval);
173         obj.error('@openWindow: '+language.string(174));
174       }
175     }, 100);
176   }
177 }

```

```
174 }, 10) ;  
175  
176     return [sub_context, new Promise(function(resolve) {  
177         sub_context.addEventListener("DOMContentLoaded", function() {  
178             sub_context.wid = wid;  
179  
180             sub_context.show = function() {  
181                 return obj.jsl.windows.open_windows[wid].show();  
182             };  
183             sub_context.hide = function() {  
184                 return obj.jsl.windows.open_windows[wid].hide();  
185             };  
186             sub_context.focus = function() {  
187                 return obj.jsl.windows.open_windows[wid].focus();  
188             };  
189             sub_context.minimize = function() {  
190                 return obj.jsl.windows.open_windows[wid].minimize();  
191             };  
192             sub_context.center = function() {  
193                 return obj.jsl.windows.open_windows[wid].center();  
194             };  
195             sub_context.moveTop = function() {  
196                 return obj.jsl.windows.open_windows[wid].moveTop();  
197             };  
198  
199             sub_context.setSize = function(width, height) {  
200                 return obj.jsl.windows.open_windows[wid].setSize(width, height);  
201             };  
202             sub_context.setPos = function(left, top) {  
203                 return obj.jsl.windows.open_windows[wid].setPos(left, top);  
204             };  
205             sub_context.setResizable = function(state) {  
206                 return obj.jsl.windows.open_windows[wid].setResizable(state);  
207             };  
208             sub_context.setMovable = function(state) {  
209                 return obj.jsl.windows.open_windows[wid].setMovable(state);  
210             };  
211             sub_context.setAspectRatio = function(aspect_ratio) {  
212                 return obj.jsl.windows.open_windows[wid].setMovable(aspect_ratio);  
213             };  
214             sub_context.setOpacity = function(opacity) {  
215                 return obj.jsl.windows.open_windows[wid].setOpacity(opacity);  
216             };  
217             sub_context.setTitle = function(title) {  
218                 return obj.jsl.windows.open_windows[wid].setTitle(title);  
219             };  
220  
221             sub_context.getSize = function() {  
222                 return obj.jsl.windows.open_windows[wid].getSize();  
223             };  
224             sub_context.getPos = function() {  
225                 return obj.jsl.windows.open_windows[wid].getPos();  
226             };  
227  
228 }
```

```

229     sub_context.openDevTools = function() {
230       return obj.jsl.windows.open_windows[wid].openDevTools();
231     };
232
233     sub_context.document.addEventListener("keydown", function(e) {
234       if(e.ctrlKey && e.key.toLowerCase() == 'c') {
235         if(obj.getWinSelectionText(sub_context) == "") {
236           // No selected text
237           obj.jsl.setStopLoop(true);
238           e.stopPropagation();
239           e.preventDefault();
240         }
241       }
242     });
243     resolve(sub_context);
244   }, false);
245 });
246 }
247 }
248
249 /**
250 * Retrieves the selected text from a given window.
251 * @param {Window} context - The window context to get the selection text
252 * from.
253 * @returns {string} - The selected text.
254 */
255 getWinSelectionText(context) {
256   var text = "";
257   if(context.getSelection) {
258     text = context.getSelection().toString();
259   } else if(context.document.selection && context.document.selection.type != "Control") {
260     text = context.document.selection.createRange().text;
261   }
262   return text;
263 }
264
265 /**
266 * Closes a window or all windows based on the provided ID.
267 * @param {number} id - The ID of the window to close, or "all" to close all
268 * windows.
269 */
270 closeWindow(wid) {
271   if(!global.is_worker) {
272     var obj = this;
273     if(wid == "all") {
274       Object.keys(this.jsl.windows.open_windows).forEach(function(key) {
275         obj.jsl.windows.open_windows[key].context.close();
276         obj.jsl.windows._closedWindow(key);
277       });
278       return true;
279     } else if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
280       obj.jsl.windows.open_windows[key].context.close();
281       obj.jsl.windows._closedWindow(key);
282       return true;
283     }
284   }
285 }

```

```

281         }
282     }
283     return false;
284 }
285
286 /**
287 * Shows the specified window.
288 * @param {number} wid - The ID of the window to show.
289 * @returns {boolean|undefined} - Returns false if the window ID is invalid.
290 */
291 showWindow(wid) {
292     if(!global.is_worker) {
293         if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
294             ipcRenderer.sendSync("sync-message", "call-sub-win-method",
295                 [wid, 'show']);
296             return true;
297         }
298     }
299     return false;
300 }
301
302 /**
303 * Hides the specified window.
304 * @param {number} wid - The ID of the window to hide.
305 * @returns {boolean|undefined} - Returns false if the window ID is invalid.
306 */
307 hideWindow(wid) {
308     if(!global.is_worker) {
309         if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
310             ipcRenderer.sendSync("sync-message", "call-sub-win-method",
311                 [wid, 'hide']);
312             return true;
313         }
314     }
315     return false;
316 }
317
318 /**
319 * Brings the specified window to the foreground.
320 * @param {number} wid - The ID of the window to focus.
321 * @returns {boolean|undefined} - Returns false if the window ID is invalid.
322 */
323 focusWindow(wid) {
324     if(!global.is_worker) {
325         if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
326             ipcRenderer.sendSync("sync-message", "call-sub-win-method",
327                 [wid, 'focus']);
328             return true;
329         }
330     }
331     return false;
332 }
333
334 /**
335 * Minimizes the specified window.

```

```

336  * @param {number} wid - The ID of the window to minimize.
337  * @returns {boolean|undefined} - Returns false if the window ID is invalid.
338  */
339  minimizeWindow(wid) {
340    if(!global.is_worker) {
341      if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
342        ipcRenderer.sendSync("sync-message", "call-sub-win-method",
343          [wid, 'minimize']);
344        return true;
345      }
346    }
347    return false;
348  }

349 /**
350 * Centers the specified window on the screen.
351 * @param {number} wid - The ID of the window to center.
352 * @returns {boolean|undefined} - Returns false if the window ID is invalid.
353 */
354 centerWindow(wid) {
355   if(!global.is_worker) {
356     if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
357       ipcRenderer.sendSync("sync-message", "call-sub-win-method",
358         [wid, 'center']);
359       return true;
360     }
361   }
362   return false;
363 }

364 /**
365 * Moves the specified window to the top.
366 * @param {number} wid - The ID of the window to move to top.
367 * @returns {boolean|undefined} - Returns false if the window ID is invalid.
368 */
369 moveTopWindow(wid) {
370   if(!global.is_worker) {
371     if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
372       ipcRenderer.sendSync("sync-message", "call-sub-win-method",
373         [wid, 'moveTop']);
374       return true;
375     }
376   }
377   return false;
378 }

379 /**
380 * Sets the size of a specified window.
381 * @param {number} wid - The ID of the window.
382 * @param {number} width - The new width of the window.
383 * @param {number} height - The new height of the window.
384 * @returns {boolean|undefined} - Returns false if the window ID is invalid.
385 */
386 setWindowSize(wid, width, height) {
387   if(!global.is_worker) {
388
389
390

```

```

391     if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
392         return ipcRenderer.sendSync("sync-message", "call-sub-win-method",
393             [wid, "setSize", width, height]);
394     }
395     return false;
396 }
397
398 /**
399 * Sets the position of a specified window.
400 * @param {number} wid - The ID of the window.
401 * @param {number} left - The new left position of the window.
402 * @param {number} top - The new top position of the window.
403 * @returns {boolean|undefined} - Returns false if the window ID is invalid.
404 */
405 setWindowPos(wid, left, top) {
406     if(!global.is_worker) {
407         if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
408             return ipcRenderer.sendSync("sync-message", "call-sub-win-method",
409                 [wid, "setPosition", left, top]);
410         }
411     }
412     return false;
413 }
414
415 /**
416 * Sets whether the specified window is resizable.
417 * @param {number} wid - The ID of the window.
418 * @param {boolean} state - The resizable state to set.
419 * @returns {boolean|undefined} - Returns false if the window ID is invalid.
420 */
421 setWindowResizable(wid, state) {
422     if(!global.is_worker) {
423         if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
424             return ipcRenderer.sendSync("sync-message", "call-sub-win-method",
425                 [wid, "setResizable", state]);
426         }
427     }
428     return false;
429 }
430
431 /**
432 * Sets whether the specified window is movable.
433 * @param {number} wid - The ID of the window.
434 * @param {boolean} state - The movable state to set.
435 * @returns {boolean|undefined} - Returns false if the window ID is invalid.
436 */
437 setWindowMovable(wid, state) {
438     if(!global.is_worker) {
439         if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
440             return ipcRenderer.sendSync("sync-message", "call-sub-win-method",
441                 [wid, "setMovable", state]);
442         } else {
443             return false;
444         }
445     }

```

```

446     }
447 }
448
449 /**
450 * Sets the aspect ratio of the specified window.
451 * @param {number} wid - The ID of the window.
452 * @param {number} aspect_ratio - The aspect ratio to set (width/height).
453 * @returns {boolean|undefined} - Returns false if the window ID is invalid.
454 */
455 setWindowAspectRatio(wid, aspect_ratio) {
456   if(!global.is_worker) {
457     if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
458       return ipcRenderer.sendSync("sync-message", "call-sub-win-method",
459         [wid, "setAspectRatio", aspect_ratio]);
460     }
461   }
462   return false;
463 }
464
465 /**
466 * Sets the opacity of the specified window.
467 * @param {number} wid - The ID of the window.
468 * @param {number} opacity - The opacity level to set (0.0 to 1.0).
469 * @returns {boolean|undefined} - Returns false if the window ID is invalid.
470 */
471 setWindowOpacity(wid, opacity) {
472   if(!global.is_worker) {
473     if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
474       return ipcRenderer.sendSync("sync-message", "call-sub-win-method",
475         [wid, "setOpacity", opacity]);
476     }
477   }
478   return false;
479 }
480
481 /**
482 * Sets the title of the specified window if not running in a worker thread.
483 * @param {string} wid - The window ID.
484 * @param {string} title - The new title for the window.
485 * @returns {boolean|undefined} False if the window does not exist,
486         undefined if in a worker thread.
487 */
488 setWindowTitle(wid, title) {
489   if(!global.is_worker) {
490     if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
491       this.jsl.windows.open_windows[wid].context.document.title = title;
492     }
493   }
494   return false;
495 }
496 /**
497 * Retrieves the size of a specified window.
498 * @param {number} wid - The ID of the window.
499 * @returns {Array|boolean} - Returns an array [width, height] or false if

```



```
      the window ID is invalid.  
500   */  
501   getWindowSize(wid) {  
502     if(!global.is_worker) {  
503       if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {  
504         return ipcRenderer.sendSync("sync-message", "call-sub-win-method",  
505           [wid, "getSize"]);  
506       }  
507     }  
508     return false;  
509   }  
510  
511   /**  
512    * Retrieves the position of a specified window.  
513    * @param {number} wid - The ID of the window.  
514    * @returns {Array|boolean} - Returns an array [left, top] or false if the  
515    window ID is invalid.  
516    */  
517   getWindowPos(wid) {  
518     if(!global.is_worker) {  
519       if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {  
520         return ipcRenderer.sendSync("sync-message", "call-sub-win-method",  
521           [wid, "getPosition"]);  
522       }  
523     }  
524     return false;  
525   }  
526  
527   /**  
528    * Opens the developer tools for a specified window in the renderer process.  
529    * Only available if not in a worker context.  
530    * @param {string} wid - The window ID.  
531    * @returns {boolean} True if the developer tools were opened; otherwise,  
532    * false.  
533    */  
534   openWindowDevTools(wid) {  
535     if(!global.is_worker) {  
536       if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {  
537         return ipcRenderer.sendSync("sync-message", "open-sub-win-devtools",  
538           wid);  
539       }  
540     }  
541     return false;  
542   }  
543  
544   /**  
545    * Opens the developer tools for the sandbox environment.  
546    */  
547   openSandboxDevTools() {  
548     ipcRenderer.send("MainProcess", "show-sandbox-dev-tools");  
549   }  
550  
551   /**  
552    * Clears local storage.  
553    */
```

```
551 clearStorage() {
552     if(! global.is_worker) {
553         ipcRenderer.send("MainWindow", "clear-storage");
554     }
555 }
556
557 /**
558 * Creates a directory at the specified path. If the directory already
559 exists, no action is taken.
560 * @param {string} directory - The path where the directory will be created.
561 * @returns {boolean} True if the directory was created or already exists,
562 *               false if an error occurred.
563 */
564 makeDirectory(directory) {
565     try {
566         fs.mkdirSync(directory, { recursive: true });
567         return true;
568     } catch(err) {
569         this.jsl._console.log(err);
570         if(err.code === 'EEXIST') {
571             return true;
572         } else {
573             return false;
574         }
575     }
576 }
577 /**
578 * Checks if a directory exists at the specified path.
579 * @param {string} directory - The path to the directory.
580 * @returns {boolean} - True if the directory exists, false otherwise.
581 */
582 checkDirectory(directory) {
583     var lstat = fs.lstatSync(directory, { throwIfNoEntry: false });
584     if(lstat !== undefined && lstat.isDirectory()) {
585         return true;
586     } else {
587         return false;
588     }
589 }
590 /**
591 * Checks if a file exists at the specified path.
592 * @param {string} file_path - The path to the file.
593 * @returns {boolean} - True if the file exists, false otherwise.
594 */
595 checkFile(file_path) {
596     var lstat = fs.lstatSync(file_path, { throwIfNoEntry: false });
597     if(lstat !== undefined && lstat.isFile()) {
598         return true;
599     } else {
600         return false;
601     }
602 }
603
```

```
604  /**
605   * Checks if the provided path is an absolute path.
606   * @param {string} file_path The file path to check.
607   * @returns {boolean} True if the path is absolute, false otherwise.
608   */
609  pathIsAbsolute(file_path) {
610    if(typeof file_path === 'string') {
611      return path.isAbsolute(file_path);
612    } else {
613      return false;
614    }
615  }
616
617  /**
618   * Joins all given path segments together using the platform-specific
619   * separator as a delimiter.
620   * @param {...string} paths The path segments to join.
621   * @returns {string} The combined path.
622   */
623  pathJoin(...arg) {
624    return path.join(...arg);
625  }
626
627  /**
628   * Retrieves the platform-specific path separator.
629   * @returns {string} The path separator.
630   */
631  pathGetSep() {
632    return path.sep;
633  }
634
635  /**
636   * Extracts the basename of a path, possibly removing the specified
637   * extension.
638   * @param {...string} paths The path to extract the basename from, followed
639   * by an optional extension to remove.
640   * @returns {string} The basename of the path.
641   */
642  pathBaseName(...arg) {
643    return path.basename(...arg);
644  }
645
646  /**
647   * Retrieves the file name without its extension from the given file path.
648   * @param {string} file_path - The path to the file.
649   * @returns {string} - The file name without the extension.
650   */
651  pathFileName(file_path) {
652    return path.parse(file_path).name;
653  }
654
655  /**
656   * Extracts the directory of file.
657   * @param {String} path The filesystem path from which to extract the
658   * directory.
```

```
655     * @returns {String} The directory from the given path.  
656     */  
657     pathDirName(file_path) {  
658         return path.dirname(file_path);  
659     }  
660  
661     /**  
662      * Retrieves the file extension from the given file path.  
663      * @param {string} file_path - The path to the file.  
664      * @returns {string} - The file extension.  
665      */  
666     pathExtName(file_path) {  
667         return path.extname(file_path);  
668     }  
669  
670     /**  
671      * Resolves a sequence of path segments into an absolute path.  
672      * @param {string} path_in - The path or sequence of paths to resolve.  
673      * @returns {string} - The resolved absolute path.  
674      */  
675     pathResolve(path_in) {  
676         return path.resolve(path_in);  
677     }  
678  
679     /**  
680      * Normalizes a given path, resolving '..' and '.' segments.  
681      * @param {string} path - The path to normalize.  
682      * @returns {string} - The normalized path.  
683      */  
684     pathNormalize(path_in) {  
685         return path.normalize(path_in);  
686     }  
687  
688     /**  
689      * Parses a file path into its component parts.  
690      * @param {string} path - The file path to parse.  
691      * @returns {Object} An object containing properties like 'root', 'dir', 'base', 'ext', and 'name'.  
692      */  
693     pathParse(path_in) {  
694         return path.parse(path_in);  
695     }  
696  
697     /**  
698      * Reads the content of a file synchronously.  
699      * @param {string} file_path The path to the file.  
700      * @returns {((Buffer|string|false))} The file content or false in case of an error.  
701      */  
702     readFileSync(...arg) {  
703         try {  
704             return fs.readFileSync(...arg);  
705         } catch(err) {  
706             this.jsl._console.log(err);  
707             return false;
```



```
708     }
709 }
710
711 /**
712 * Copies a file synchronously with the given arguments.
713 * @param {...any} arg - Arguments to pass to fs.copyFileSync.
714 * @returns {boolean} - Returns true if the copy was successful, false
715 * otherwise.
716 */
717 copyFileSync (...arg) {
718     try {
719         return fs.copyFileSync (...arg);
720     } catch (err) {
721         this.jsl._console.log(err);
722         return false;
723     }
724 }
725 /**
726 * Writes data to a file synchronously.
727 * @param {string} file_path - The path to the file.
728 * @param {any} data - The data to write.
729 * @param {boolean} throw_flag - Whether to throw an error on failure.
730 * @returns {boolean} - Returns true if the write was successful, false
731 * otherwise.
732 */
733 writeFileSync (file_path , data , throw_flag) {
734     try {
735         return fs.writeFileSync (file_path , data);
736     } catch (err) {
737         this.jsl._console.log(err);
738         if (throw_flag) {
739             this.error ('@writeFileSync: '+err);
740         }
741         return false;
742     }
743 }
744 /**
745 * Removes a file or directory synchronously.
746 * @param {string} path - The path to remove.
747 * @param {boolean} [throw_flag=true] - Whether to throw an error on failure
748 *
749 * @returns {boolean} - Returns true if the removal was successful, false
750 * otherwise.
751 */
752 rmSync (path_in , throw_flag = true) {
753     try {
754         return fs.rmSync (path_in , { recursive: true , force: true });
755     } catch (err) {
756         this.jsl._console.log(err);
757         if (throw_flag) {
758             this.error ('@rmSync: '+err);
759         }
760         return false;
761     }
762 }
```

```
759     }
760   }
761
762 /**
763 * Reads the contents of a directory synchronously.
764 * @param {string} folder The path to the directory.
765 * @returns {string[]|false} An array of filenames or false in case of an
766 * error.
767 */
768   readdirSync (... args) {
769     try {
770       return fs.readdirSync (... args);
771     } catch (err) {
772       this.jsl._console.log(err);
773       this.error('@readdirSync: '+err);
774       return false;
775     }
776   }
777 /**
778 * Displays a dialog to open files , asynchronously returning the selected
779 * files' paths .
780 * @param {Object} options The options for the dialog.
781 * @returns {Promise<string[]>} A promise that resolves to the paths of
782 * selected files .
783 */
784 showOpenDialog(options) {
785   if (!global.is_worker) {
786     return ipcRenderer.invoke("dialog", "showOpenDialog", options);
787   }
788   return false;
789 }
790 /**
791 * Displays a dialog to open files , synchronously returning the selected
792 * files' paths .
793 * @param {Object} options The options for the dialog.
794 * @returns {string[]} The paths of selected files .
795 */
796 showOpenDialogSync(options) {
797   if (!global.is_worker) {
798     return ipcRenderer.sendSync("dialog", "showOpenDialogSync", options);
799   }
800   return false;
801 }
802 /**
803 * Displays a dialog to save file , asynchronously returning the selected
804 * files' paths .
805 * @param {Object} options The options for the dialog.
806 * @returns {Promise<string[]>} A promise that resolves to the paths of
807 * selected files .
808 */
809 showSaveDialog(options) {
810   if (!global.is_worker) {
```

```
808     return ipcRenderer.invoke("dialog", "showSaveDialog", options);
809 }
810 return false;
811 }

813 /**
814 * Displays a dialog to save file, synchronously returning the selected
815 * files' paths.
816 * @param {Object} options The options for the dialog.
817 * @returns {string[]} The paths of selected files.
818 */
819 showSaveDialogSync(options) {
820     if(!global.is_worker) {
821         return ipcRenderer.sendSync("dialog", "showSaveDialogSync", options);
822     }
823     return false;
824 }

825 /**
826 * Displays a message box, synchronously returning the index of the clicked
827 * button.
828 * @param {Object} options The options for the message box.
829 * @returns {number} The index of the clicked button.
830 */
831 showMessageBox(options) {
832     if(!global.is_worker) {
833         return ipcRenderer.sendSync("dialog", "showMessageBoxSync", options);
834     }
835     return false;
836 }

837 /**
838 * Checks if a loop stop flag has been set, indicating whether to halt
839 * execution.
840 * @returns {boolean} True if the loop should stop, false otherwise.
841 */
842 checkStopLoop() {
843     if(!global.is_worker) {
844         return ipcRenderer.sendSync("sync-message", "check-stop-loop");
845     }
846     return false;
847 }

848 /**
849 * Resets the loop stop flag to allow continued execution.
850 * @returns {undefined} No return value.
851 */
852 resetStopLoop() {
853     if(!global.is_worker) {
854         return ipcRenderer.sendSync("sync-message", "reset-stop-loop");
855     }
856     return false;
857 }

858 /**
859 */
```

```

860  * Opens the editor window and optionally loads a script.
861  * @param {string} filename - The path to the script file to open.
862  * @param {number} lineno - Line number to highlight.
863  */
864  editor(filename, lineno) {
865    if(!global.is_worker) {
866      ipcRenderer.send("MainProcess", "show-editor");
867      if(typeof filename == "string") {
868        ipcRenderer.send("EditorWindow", "open-script", [filename, lineno]);
869      }
870    }
871  }
872
873 /**
874  * Sends a message to display in the main window of the application.
875  * @param {...any} args - The messages to send for display in the main
876  * window.
877  */
878 disp(...args) {
879   if(!global.is_worker) {
880     var obj = this;
881     args.forEach(function(msg) {
882       ipcRenderer.send("MainWindow", "disp", obj.jsl.format.safeStringify(
883         msg));
884     });
885   }
886 /**
887  * Sends a message to display in the main window of the application with
888  * monospaced font.
889  * @param {...any} args - The messages to send for display in the main
890  * window with monospaced font.
891  */
892 dispMonospaced(...args) {
893   if(!global.is_worker) {
894     var obj = this;
895     args.forEach(function(msg) {
896       ipcRenderer.send("MainWindow", "disp-monospaced", obj.jsl.format.
897         safeStringify(msg));
898     });
899   }
900 /**
901  * Sends a message to display latex in the main window of the application.
902  * @param {string} expr The expression to be displayed.
903  */
904 dispLatex(expr) {
905   if(!global.is_worker) {
906     ipcRenderer.send("MainWindow", "disp-latex", expr);
907   }
908 /**
909  */

```

```
910     * Triggers the display of CMD help content in the main window.
911     */
912 cmd_help() {
913     if(!global.is_worker) {
914         ipcRenderer.send("MainWindow", "help");
915     }
916 }
917
918 /**
919  * Triggers the display of informational content in the main window.
920 */
921 info() {
922     if(!global.is_worker) {
923         ipcRenderer.send("MainWindow", "info");
924     }
925 }
926
927 /**
928  * Opens the settings interface in the main window.
929 */
930 settings() {
931     if(!global.is_worker) {
932         ipcRenderer.send("MainWindow", "settings");
933     }
934 }
935
936 /**
937  * Sends an error message to be displayed in the main window.
938  * @param {string} msg The error message to be displayed.
939 */
940 error(msg, throw_flag = true) {
941     if(!global.is_worker) {
942         this.jsl.stop_loop = true;
943         this.jsl.onStopLoop(false);
944         this.jsl.no_ans = true;
945         this.jsl.ignore_output = true;
946         this.jsl.onEvaluated();
947         if(throw_flag) {
948             throw new Error(msg);
949         } else {
950             ipcRenderer.send("MainWindow", "error", this.jsl.format.safeStringify(
951                         msg));
952         }
953     }
954
955 /**
956  * Sends an internal error message to be displayed, indicating an error
957  * within the application's internals.
958  * @param {string} msg The internal error message.
959 */
960 errorInternal(msg) {
961     if(!global.is_worker) {
962         ipcRenderer.send("MainWindow", "internal-error", msg);
963     }
964 }
```

```
963     }
964
965     /**
966      * Sends a warning message to be displayed in the main window.
967      * @param {string} msg The warning message.
968      */
969     warn(msg) {
970       if(!global.is_worker) {
971         ipcRenderer.send("MainWindow", "warn", msg);
972       }
973     }
974
975     /**
976      * Clears the command window, removing all current content.
977      */
978     clc() {
979       if(!global.is_worker) {
980         ipcRenderer.send("MainWindow", "clear");
981       }
982     }
983
984     /**
985      * Lists the contents of the current directory and returns them.
986      * @returns {string[]} The contents of the current directory.
987      */
988     listFolderContents() {
989       return fs.readdirSync(this.jsl.current_path);
990     }
991
992     /**
993      * Requests an update of the file browser to reflect current directory or
994      * file changes.
995      */
996     updateFileBrowser() {
997       if(!global.is_worker) {
998         ipcRenderer.send("MainWindow", "update-file-browser");
999       }
1000
1001    /**
1002      * Displays the result of an operation in the workspace area of the main
1003      * window.
1004      * @param {string} data The data or result to display.
1005      */
1006     showAns(data) {
1007       if(!global.is_worker) {
1008         ipcRenderer.send("MainWindow", "show-ans", data);
1009       }
1010
1011    /**
1012      * Requests an update of the workspace to reflect changes in variables or
1013      * state.
1014      */
1015     updateWorkspace() {
```

```

1015     if (! global.is_worker) {
1016         ipcRenderer.send("MainWindow", "update-workspace");
1017     }
1018 }
1019
1020 /**
1021 * Sets the workspace with the provided data, replacing the current state.
1022 */
1023 setWorkspace() {
1024     if (! global.is_worker) {
1025         ipcRenderer.send("MainWindow", "set-workspace", this.jsl.getWorkspace());
1026         ;
1027     }
1028
1029 /**
1030 * Saves a new path to the application's memory for quick access.
1031 * @param {string} new_path The path to save.
1032 */
1033 savePath(new_path) {
1034     if (! global.is_worker) {
1035         ipcRenderer.send("MainWindow", "save-path", new_path);
1036     }
1037 }
1038
1039 /**
1040 * Removes a previously saved path from the application's memory.
1041 * @param {string} saved_path The path to remove.
1042 */
1043 removePath(saved_path) {
1044     if (! global.is_worker) {
1045         ipcRenderer.send("MainWindow", "remove-path", saved_path);
1046     }
1047 }
1048
1049 /**
1050 * Sets the application's status message.
1051 * @param {string} state The current state of the application.
1052 * @param {string} txt The text message to display as status.
1053 */
1054 setStatus(state, txt) {
1055     if (! global.is_worker) {
1056         ipcRenderer.send("MainWindow", "set-status", [state, txt]);
1057     }
1058 }
1059
1060 /**
1061 * Updates the application's statistics, typically displayed in the status
1062 * bar or a similar area.
1063 * @param {object} stats The statistical data to set.
1064 */
1065 setStats(stats) {
1066     if (! global.is_worker) {
1067         ipcRenderer.send("MainWindow", "set-stats", stats);
1068     }

```

```

1068 }
1069
1070 /**
1071 * Notifies the main window that code evaluation has started.
1072 */
1073 codeEvaluating() {
1074   if(!global.is_worker) {
1075     ipcRenderer.send("MainWindow", "code-evaluating");
1076   }
1077 }
1078
1079 /**
1080 * Notifies the main window that code evaluation has finished.
1081 */
1082 codeEvaluated() {
1083   if(!global.is_worker) {
1084     ipcRenderer.send("MainWindow", "code-evaluated");
1085     ipcRenderer.send("MainProcess", "code-evaluated");
1086   }
1087 }
1088
1089 /**
1090 * Checks if a script resides within the current active directory or a saved
1091 * directory, and if not, prompts to find the script.
1092 * @param {string} script_path The path of the script to check.
1093 * @returns {boolean} Returns true if the script directory is unknown,
1094 *   prompting for location; otherwise false.
1095 */
1096 checkScriptDir(script_path) {
1097   var script_dir = this.addPathSep(path.dirname(script_path));
1098   if(pathEqual(script_dir, this.jsl.current_path) ||
1099     this.jsl.saved_paths.includes(script_dir)) {
1100     return false;
1101   } else {
1102     ipcRenderer.send("MainWindow", "unknown-script-dir");
1103     return true;
1104   }
1105 }
1106
1107 /**
1108 * Ensures a path string ends with a path separator, appending one if
1109 * necessary.
1110 * @param {string} path_str The path string to modify.
1111 * @returns {string} The modified path string with a trailing separator.
1112 */
1113 addPathSep(path_str) {
1114   if(path_str && path_str[path_str.length - 1] != path.sep) {
1115     path_str += path.sep;
1116   }
1117   return path_str;
1118 }
1119
1120 /**
1121 * Changes the current working directory to the specified path.
1122 * @param {string} new_path The path to set as the current working directory

```

```

1120 .
1121 */
1122 cd(new_path) {
1123   if(!global.is_worker) {
1124     ipcRenderer.send("MainWindow", "set-current-path", new_path,
1125       undefined, false);
1126   }
1127 }
1128 /**
1129 * Retrieves a default path based on a specified type, e.g., documents,
1130 * downloads.
1131 * @param {string} type The type of default path to retrieve.
1132 * @returns {string} The default path for the specified type.
1133 */
1134 getDefaultPath(type) {
1135   if(!global.is_worker) {
1136     return ipcRenderer.sendSync("sync-message", "get-path", type) + path.sep
1137   }
1138   return false;
1139 }
1140 /**
1141 * Opens the specified folder in the file manager.
1142 * @param {string} file_path The path of the folder to open.
1143 */
1144 openFolder(file_path) {
1145   if(!global.is_worker) {
1146     return ipcRenderer.send("MainProcess", "open-folder", file_path);
1147   }
1148   return false;
1149 }
1150 /**
1151 * Opens the specified directory in the file manager. This method is similar
1152 * to 'openFolder'.
1153 * @param {string} file_path The path of the directory to open.
1154 */
1155 openDir(file_path) {
1156   if(!global.is_worker) {
1157     return ipcRenderer.send("MainProcess", "open-dir", file_path);
1158   }
1159   return false;
1160 }
1161 /**
1162 * Shows the specified file in the folder using the file manager.
1163 * @param {string} file_path The path of the file to show.
1164 */
1165 showFileInFolder(file_path) {
1166   if(!global.is_worker) {
1167     return ipcRenderer.send("MainProcess", "show-file-in-folder", file_path)
1168   }
1169 }

```

```
1170     return false;
1171 }
1172
1173 /**
1174 * Shows the specified file in the directory using the file manager. This is
1175 * similar to 'showFileInFolder'.
1176 * @param {string} file_path The path of the file to highlight.
1177 */
1178 showFileInDir(file_path) {
1179     if(!global.is_worker) {
1180         return ipcRenderer.send("MainProcess", "show-file-in-dir", file_path);
1181     }
1182     return false;
1183 }
1184 /**
1185 * Retrieves desktop sources synchronously by sending an IPC message.
1186 * @returns {DesktopSource[]|undefined} An array of desktop sources if not
1187 * in a worker, otherwise undefined.
1188 */
1189 getDesktopSources() {
1190     if(!global.is_worker) {
1191         return ipcRenderer.sendSync("get-desktop-sources");
1192     }
1193     return false;
1194 }
1195 /**
1196 * Executes a system command assynchronously.
1197 * @param {...*} args Command arguments.
1198 */
1199 exec(...args){
1200     return cp.exec(...args);
1201 }
1202
1203 /**
1204 * Executes a system command synchronously and returns the result.
1205 * @param {...*} args Command arguments.
1206 * @returns {*} The result of the command execution.
1207 */
1208 execSync(...args){
1209     return cp.execSync(...args);
1210 }
1211
1212 /**
1213 * Spawns child process synchronously.
1214 * @param {...*} args Command arguments.
1215 */
1216 spawnSync(...args){
1217     return cp.spawnSync(...args);
1218 }
1219
1220 /**
1221 * Spawns child process assynchronously.
1222 * @param {...*} args Command arguments.
```

```

1223  */
1224 spawn(... args){
1225   return cp.spawn(... args);
1226 }
1227
1228 /**
1229 * Resets the sandbox environment by sending a synchronous IPC message if
1230 * not in a worker.
1231 * @returns {void}
1232 */
1233 resetSandbox() {
1234   if(!global.is_worker) {
1235     ipcRenderer.sendSync("sync-message", "reset-sandbox");
1236   }
1237
1238 /**
1239 * Resets the app.
1240 * @returns {void}
1241 */
1242 resetApp() {
1243   if(!global.is_worker) {
1244     ipcRenderer.sendSync("sync-message", "reset-app");
1245   }
1246 }
1247
1248
1249 exports.PRDC_JSLAB_ENV = PRDC_JSLAB_ENV;

```

Listing 89 - jslab-env-electron.js

```

1 /**
2  * @file JSLAB library eval
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 /**
9  * Class for JSLAB library eval.
10 */
11 class PRDC_JSLAB_EVAL {
12
13 /**
14  * Constructs a eval submodule object with access to JSLAB's electron
15  * environment functions.
16  * @constructor
17  * @param {Object} jsl - Reference to the main JSLAB object.
18  */
19 constructor(jsl) {
20   var obj = this;
21   this.jsl = jsl;
22
23   // Errors handling
24   this.jsl.context.addEventListener('unhandledrejection', function(e) {
25     if(e && e.reason) {

```



```

77     obj.jsl.onEvaluated();
78     if(err.name === 'JslabError') {
79       obj.jsl.env.error(err.message, false);
80     } else {
81       obj.rewriteError(err.stack.toString());
82     }
83   }
84
85   try {
86     var data = await this.evalString(code);
87     if(obj.jsl.no_ans === false) {
88       obj.jsl.context.ans = data;
89     }
90     if(show_output && obj.jsl.ignore_output === false) {
91       obj.jsl.env.showAns(prettyPrint(data));
92     }
93     obj.jsl.onEvaluated();
94   } catch(err) {
95     onError(err);
96   }
97 }
98
99 /**
100 * Evaluates a string of code and handles errors and output.
101 * @param {String} code The code string to evaluate.
102 * @returns {*} The result of the evaluated code.
103 */
104 async evalString(code) {
105   var rewrite_result = this.rewriteCode(code);
106   if(rewrite_result) {
107     this.source_maps.push(rewrite_result.map);
108     this.transformed_codes.push(rewrite_result.code);
109     this.source_codes.push(code);
110
111     var current_source_code = this.current_source_code;
112     var current_source_map = this.current_source_map;
113     this.current_source_code = code;
114     this.current_source_map = rewrite_result.map;
115
116     var result = await this.jsl._eval(rewrite_result.code);
117
118     this.current_source_code = current_source_code;
119     this.current_source_map = current_source_map;
120     return result;
121   }
122   return false;
123 }
124
125 /**
126 * Executes a script file within the JSLAB environment.
127 * @param {String} script_path The path to the script file to be executed.
128 * @param {Array} [lines] Specifies a range of lines to execute, if provided
129 * @param {Boolean} [silent=false] If true, suppresses output from the
130   script.

```

```

130  /*
131   * @param {string} script_path - The path to the script file to run.
132   * @param {number} lines - The number of lines to run from the script.
133   * @param {boolean} silent - Whether to suppress output.
134   */
135  async runScript(script_path, lines, silent = false) {
136    script_path = this.jsl.pathResolve(script_path);
137    if(script_path) {
138      this.jsl.current_script = script_path;
139      this.jsl.jsl_file_name = this.jsl.env.pathBaseName(script_path);
140
141      var script_code = this.jsl.env.readFileSync(script_path);
142      if(script_code === false) {
143        this.jsl.env.error('@runScript: '+language.string(103)+': '+
144          script_path, false);
145      } else {
146        script_code = script_code.toString();
147        if(lines !== undefined) {
148          var code_lines = script_code.split('\n');
149          if(typeof lines === 'number') {
150            if(code_lines.length >= lines) {
151              code_lines[lines-1];
152            } else {
153              this.jsl.env.error('@runScript: '+language.string(104)+': '\n '+
154                language.string(105)+': '+script_path, false);
155            }
156          } else {
157            if(code_lines.length >= lines[0] && code_lines.length >= lines[1])
158              {
159                code_lines = code_lines.slice(lines[0]-1, lines[1]-1);
160              } else {
161                this.jsl.env.error('@runScript: '+language.string(104)+': '\n '+
162                  language.string(105)+': '+script_path);
163              }
164            }
165          }
166        return await this.evalString(script_code, !silent);
167      }
168    }
169    return false;
170  }
171
172 /**
173 * Re-evaluates the last script file that was executed in the environment.
174 */
175 async runLast() {
176   var cmd;
177   if(this.jsl.last_script_lines !== undefined) {
178     cmd = 'run(' + JSON.stringify(this.jsl.last_script_path) + ', ' + this.
179       jsl.last_script_lines.toString() + '", undefined, true)';
180   } else {
181     cmd = 'run(' + JSON.stringify(this.jsl.last_script_path) + ', undefined,
182       false, true)';
183   }
184   await this.evalCodeFromMain(cmd);
185 }
186
187 /**
188 * Extracts and logs error information from the stack trace.
189 */

```

```

179  * @param {String} stack The error stack trace.
180  */
181  async rewriteError(stack, on_rewrite = false) {
182   this.jsl._console.log(stack, on_rewrite);
183
184   const regex_normal = /at\s(.*)\s\(((.*\.\.*?)(\d+)(\d+)\))/;
185   const regex_eval = /eval at evalString\s*\((.*:(\d+)(\d+))$/;
186   const regex_rewrite = /\((\d+)(\d+)\)/;
187   var lines = stack.split('\n');
188   var msg = lines[0];
189
190   if(on_rewrite) {
191     msg = msg.replace(/\(\d+:\d+\)/g, ',');
192     let matchs = lines[0].match(regex_rewrite);
193     if(matchs) {
194       let line = parseInt(matchs[1]);
195       let column = parseInt(matchs[2]);
196
197       msg += '\n' + language.string(114) + " ";
198       msg += "(" + this.jsl.current_script + ")" + language.string(112) + ": "
199       + line + ", " + language.string(113) + ":" + column;
200     }
201     throw {
202       name: 'JslabError',
203       message: msg
204     };
205   } else {
206     for(let i = 1; i < lines.length; i++) {
207       if(lines[i].includes("eval at evalString ()")) {
208         msg += '\n' + language.string(114) + " ";
209         let matchs = lines[i].match(regex_eval);
210         let line = parseInt(matchs[1]);
211         let column = parseInt(matchs[2]);
212         var result = await this.getOriginalPosition(end(this.source_maps),
213           line, column);
214
215         msg += "(" + this.jsl.current_script + ")" + language.string(112) + ": "
216         + result.line + ", " + language.string(113) + ":" + result.column;
217         break;
218       } else {
219         let matchs = lines[i].match(regex_normal);
220         if(matchs) {
221           msg += '\n' + language.string(114) + " ";
222           let expression = matchs[1];
223           let path = matchs[2];
224           let line = parseInt(matchs[3]);
225           let column = parseInt(matchs[4]);
226           msg += expression + " (" + path + ")" + language.string(112) + ": "
227           + line + ", " + language.string(113) + ":" + column;
228         }
229       }
230     }
231     this.jsl.env.error(msg, false);
232   }
233 }
```

```

229 }
230
231 /**
232 * Extracts the position (line and column) of an expression from an error
233 * stack trace , providing context for debugging .
234 * @returns {Array} An array containing the line number, column number, and
235 * script path of the expression causing the error .
236 */
237
238
239 const regex_normal = /at \s(.*)\s\(((.*\\.*?)(:(\\d+):(\\d+))|)/;
240 const regex_eval = /eval at evalString\s*\((.*:(\\d+):(\\d+))$/;
241 var lines = stack .split ('\\n');
242 var line;
243 var script;
244
245 for (let i = 2; i < lines.length; i++) {
246   if (lines[i].includes ("eval at evalString ()")) {
247     let matchs = lines[i].match (regex_eval);
248     line = parseInt (matchs[1]);
249     column = parseInt (matchs[2]);
250     script = this .jsl.current_script;
251     break;
252   } else {
253     let matchs = lines[i].match (regex_normal);
254     if (matchs) {
255       line = parseInt (matchs[3]);
256       column = parseInt (matchs[4]);
257       script = matchs[2];
258       break;
259     }
260   }
261 }
262
263 var result = await this .getOriginalPosition (end (this .source_maps), line ,
264   column);
265 return [result.line, result.column, script];
266
267 /**
268 * Retrieves the original source position from a source map .
269 * @param {Object} map - The source map object .
270 * @param {number} line - The line number in the generated code .
271 * @param {number} column - The column number in the generated code .
272 * @returns {Promise<Object>} An object containing the original source
273 * position , including source file , line , and column .
274 */
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
889

```

```

280     return result;
281 }
282
283 /**
284 * Extracts the body of a given function as a string.
285 * @param {Function} fun - The function from which to extract the body.
286 * @returns {string|undefined} The body of the function as a string, or 'undefined' if it cannot be extracted.
287 */
288 getFunctionBody(fun) {
289   if(typeof fun === 'function') {
290     const funStr = fun.toString();
291     const bodyMatch = funStr.match(/(\s*\S*)/);
292     if(bodyMatch && bodyMatch[1]) {
293       return bodyMatch[1];
294     }
295   }
296   return undefined;
297 }
298
299 /**
300 * Rewrites a string of code using Recast and Babel Parser.
301 * @param {String} code The code string to evaluate.
302 * @returns {String} The rewritten code.
303 */
304 rewriteCode(code) {
305   const obj = this;
306
307   // Trick from devtools: Wrap code in parentheses if it's an object literal
308   if(/^\s*\{/.test(code) && /\}\s*$/ .test(code)) {
309     code = '(' + code + ')';
310   }
311
312   if(config.DEBUG_PRE_TRANSFORMED_CODE) {
313     obj.jsl._console.log(code);
314   }
315
316   // Parse the code into an AST using Recast with Babel parser
317   let ast;
318   try {
319     ast = this.jsl.env.recast.parse(code, {
320       parser: {
321         parse(source) {
322           return obj.jsl.env.babel_parser.parse(source, {
323             sourceType: 'module',
324             plugins: [
325               'jsx',
326               'typescript',
327               'classProperties',
328               'dynamicImport',
329               'optionalChaining',
330               'nullishCoalescingOperator',
331               '@babel/plugin-syntax-top-level-await'
332             ],
333             tolerant: true

```

```

334           });
335       },
336     },
337     sourceFileName: "source.js"
338   });
339 } catch(err) {
340   this.rewriteError(err.stack, true);
341   return false;
342 }
343
344 // Function to check for forbidden names in patterns
345 function checkPattern(pattern) {
346   if(pattern.type === 'Identifier') {
347     if(config.FORBIDDEN_NAMES.includes(pattern.name)) {
348       throw {
349         name: 'JslabError',
350         message: `${language.string(185)}: ${pattern.name} ${language.
351           string(184)} `,
352       };
353     }
354   } else if(pattern.type === 'ObjectPattern') {
355     pattern.properties.forEach((prop) => {
356       if(prop.type === 'RestElement') {
357         checkPattern(prop.argument);
358       } else {
359         checkPattern(prop.value);
360       }
361     });
362   } else if(pattern.type === 'ArrayPattern') {
363     pattern.elements.forEach((element) => {
364       if(element) checkPattern(element);
365     });
366   } else if(pattern.type === 'RestElement') {
367     checkPattern(pattern.argument);
368   } else if(pattern.type === 'AssignmentPattern') {
369     checkPattern(pattern.left);
370   }
371 }
372
373 // Traverse the AST to check for forbidden names
374 this.jsl.env.recast.types.visit(ast, {
375   visitVariableDeclarator(path) {
376     checkPattern(path.node.id);
377     this.traverse(path);
378   },
379   visitFunctionDeclaration(path) {
380     const node = path.node;
381     if(node.id && config.FORBIDDEN_NAMES.includes(node.id.name)) {
382       throw {
383         name: 'JslabError',
384         message: `${language.string(186)}: ${node.id.name} ${language.
385           string(184)} `,
386       };
387     }
388   }
389 }
390 this.traverse(path);

```

```

387 },
388 visitClassDeclaration(path) {
389   const node = path.node;
390   if(node.id && config.FORBIDDEN_NAMES.includes(node.id.name)) {
391     throw {
392       name: 'JslabError',
393       message: `${language.string(187)}: ${node.id.name} ${language.
394         string(184)}`,
395     };
396   }
397   this.traverse(path);
398 },
399 visitImportDeclaration(path) {
400   path.node.specifiers.forEach((specifier) => {
401     if(config.FORBIDDEN_NAMES.includes(specifier.local.name)) {
402       throw {
403         name: 'JslabError',
404         message: `${language.string(188)}: ${specifier.local.name} ${{
405           language.string(184)}`,
406       };
407     }
408   });
409   this.traverse(path);
410 },
411 visitAssignmentExpression(path) {
412   checkPattern(path.node.left);
413   this.traverse(path);
414 },
415 const b = this.jsl.env.recast.types.builders;
416
417 // Helper function to transform patterns to assign to jsl.context
418 function transformPatternToContext(pattern) {
419   if(pattern.type === 'Identifier') {
420     // Replace identifier with jsl.context.identifier
421     return b.memberExpression(
422       b.memberExpression(b.identifier('jsl'), b.identifier('context')),
423       b.identifier(pattern.name),
424       false
425     );
426   } else if(pattern.type === 'ObjectPattern') {
427     return b.objectPattern(
428       pattern.properties.map((prop) => {
429         if(prop.type === 'RestElement') {
430           return b.restElement(transformPatternToContext(prop.argument));
431         }
432         return b.objectProperty(
433           prop.key,
434           transformPatternToContext(prop.value),
435           prop.computed,
436           false
437         );
438       })
439     );
440   }
441 }

```

```

440     } else if(pattern.type === 'ArrayPattern') {
441       return b.arrayPattern(
442         pattern.elements.map((element) => {
443           if(element) {
444             return transformPatternToContext(element);
445           }
446           return null;
447         })
448       );
449     } else if(pattern.type === 'RestElement') {
450       return b.restElement(transformPatternToContext(pattern.argument));
451     } else if(pattern.type === 'AssignmentPattern') {
452       return b.assignmentPattern(
453         transformPatternToContext(pattern.left),
454         pattern.right
455       );
456     }
457     return pattern;
458   }
459
460   let tempVarCounter = 0;
461   let functionDepth = 0;
462
463   // Helpers to track function-like scopes
464   function enterFunctionScope() {
465     functionDepth++;
466   }
467   function leaveFunctionScope() {
468     functionDepth--;
469   }
470
471   const visitObj = {
472     // If top-level, after the function declaration, assign it to js1.context
473     visitFunctionDeclaration(path) {
474       const node = path.node;
475       if(functionDepth === 0 && node.id) {
476         enterFunctionScope();
477         this.traverse(path);
478         leaveFunctionScope();
479
480         // Insert assignment after the function declaration
481         const funcName = node.id.name;
482         const assignment = b.expressionStatement(
483           b.assignmentExpression(
484             '=',
485             b.memberExpression(
486               b.memberExpression(b.identifier('js1'), b.identifier('context')),
487               b.identifier(funcName)
488             ),
489             b.identifier(funcName)
490           )
491         );
492         path.insertAfter(assignment);

```

```
493     return false;
494 } else {
495     enterFunctionScope();
496     this.traverse(path);
497     leaveFunctionScope();
498     return false;
499 }
500 },
501 },
502
503 // Same logic for arrow functions and others - no change required
504 visitFunctionExpression(path) {
505     enterFunctionScope();
506     this.traverse(path);
507     leaveFunctionScope();
508     return false;
509 },
510 visitArrowFunctionExpression(path) {
511     enterFunctionScope();
512     this.traverse(path);
513     leaveFunctionScope();
514     return false;
515 },
516 visitClassMethod(path) {
517     enterFunctionScope();
518     this.traverse(path);
519     leaveFunctionScope();
520     return false;
521 },
522 visitObjectMethod(path) {
523     enterFunctionScope();
524     this.traverse(path);
525     leaveFunctionScope();
526     return false;
527 },
528 visitClassPrivateMethod(path) {
529     enterFunctionScope();
530     this.traverse(path);
531     leaveFunctionScope();
532     return false;
533 },
534
535 // If top-level class declaration, after it assign it to jsl.context
536 visitClassDeclaration(path) {
537     const node = path.node;
538     if(functionDepth === 0 && node.id) {
539         enterFunctionScope();
540         this.traverse(path);
541         leaveFunctionScope();
542
543         const className = node.id.name;
544         const assignment = b.expressionStatement(
545             b.assignmentExpression(
546                 '_',
547                 b.memberExpression(
```

```

548         b.memberExpression(b.identifier('jsl'), b.identifier('context')
549             )),
550         b.identifier(className)
551     ),
552     b.identifier(className)
553   );
554   path.insertAfter(assignment);
555
556   return false;
557 } else {
558   enterFunctionScope();
559   this.traverse(path);
560   leaveFunctionScope();
561   return false;
562 }
563 },
564
565 visitVariableDeclaration(path) {
566   const node = path.node;
567   const parentType = path.parent.node.type;
568
569   // Determine if we should rewrite based on kind and scope
570   // var: rewrite only if top-level (functionDepth === 0)
571   // let/const: rewrite only if top-level (parent is Program)
572   const isTopLevel = (functionDepth === 0);
573   const shouldRewrite =
574     ((node.kind === 'var') && isTopLevel) ||
575     ((node.kind === 'let' || node.kind === 'const') && parentType ===
576      'Program');
577
578   if (!shouldRewrite) {
579     // Just keep as is
580     this.traverse(path);
581     return false;
582   }
583
584   const newAssignments = [];
585
586   node.declarations.forEach((decl) => {
587     if (decl.id.type === 'ArrayPattern') {
588       // Handle ArrayPattern by creating a temporary variable and
589       // individual assignments
590       const tempVarName = `temp${tempVarCounter++}`;
591       // Create a temporary init variable if needed
592       const tempDecl = b.variableDeclaration('var', [
593         b.variableDeclarator(b.identifier(tempVarName), decl.init)
594       ]);
595       newAssignments.push(tempDecl);
596
597       // For each element in the ArrayPattern, assign to jsl.context
598       decl.id.elements.forEach((element, index) => {
599         if (element && element.type === 'Identifier') {
600           const transformedLeft = b.memberExpression(
601             b.memberExpression(b.identifier('jsl'), b.identifier('

```

```

          context'))),
600   b.identifier(element.name),
601   false
602 );
603 const rightAccess = b.memberExpression(
604   b.identifier(tempVarName),
605   b.numericLiteral(index),
606   true
607 );
608 newAssignments.push(
609   b.expressionStatement(
610     b.assignmentExpression('=', transformedLeft, rightAccess)
611     )
612   );
613 } else if(element && element.type === 'RestElement') {
614   // Handle RestElement (e.g., ...rest)
615   const transformedLeft = transformPatternToContext(element.
616     argument);
617   const rightSlice = b.callExpression(
618     b.memberExpression(
619       b.identifier(tempVarName),
620       b.identifier('slice'),
621       false
622       ),
623       [b.numericLiteral(index)]
624     );
625   newAssignments.push(
626     b.expressionStatement(
627       b.assignmentExpression('=', transformedLeft, rightSlice)
628       )
629     );
630   });
631 }
632 } else {
633   // For normal patterns or single identifiers
634   const transformedId = transformPatternToContext(decl.id);
635   const assignment = b.expressionStatement(
636     b.assignmentExpression('=', transformedId, decl.init || b.
637       identifier('undefined'))
638     );
639   newAssignments.push(assignment);
640 }
641 });
642 // Check if this declaration is part of a ForStatement init
643 if(path.parent.node.type === 'ForStatement' && path.parent.node.init
644   === node) {
645   // For a ForStatement init, we need a single expression, not
646   // multiple statements
647   const exprs = [];
648   newAssignments.forEach((stmt) => {
649     if(stmt.type === 'VariableDeclaration' && stmt.declarations.length
650       === 1) {
651       // Convert var temp = init into (temp = init)

```

```
649         const declInit = stmt.declarations[0].init || b.identifier(`  
650             undefined`);  
651         exprs.push(b.assignmentExpression('=', b.identifier(stmt.  
652             declarations[0].id.name), declInit));  
653     } else if(stmt.type === 'ExpressionStatement') {  
654         exprs.push(stmt.expression);  
655     }  
656 );  
657  
658     let initExpr;  
659     if(exprs.length === 1) {  
660         initExpr = exprs[0];  
661     } else {  
662         initExpr = b.sequenceExpression(exprs);  
663     }  
664  
665     path.replace(initExpr);  
666     return false;  
667 } else {  
668     // Normal top-level or block-level declaration: replace with  
669     // multiple statements  
670     path.replace(...newAssignments);  
671     return false;  
672 },  
673  
674 visitImportDeclaration(path) {  
675     const node = path.node;  
676     const newStatements = [];  
677  
678     node.specifiers.forEach((specifier) => {  
679         let requireCall;  
680         if(specifier.type === 'ImportDefaultSpecifier') {  
681             // const defaultExport = require('module').default;  
682             requireCall = b.memberExpression(  
683                 b.callExpression(b.identifier('require'), [node.source]),  
684                 b.identifier('default'),  
685                 false  
686             );  
687         } else {  
688             // const { namedExport } = require('module');  
689             requireCall = b.callExpression(b.identifier('require'), [node.  
690                 source]);  
691         }  
692  
693         const left = b.memberExpression(  
694             b.memberExpression(b.identifier('jsl'), b.identifier('context')),  
695             b.identifier(specifier.local.name),  
696             false  
697         );  
698  
699         const right =  
700             specifier.type === 'ImportDefaultSpecifier' ? requireCall  
701             : b.memberExpression(requireCall, b.identifier(specifier.imported.  
702                 name), false);
```

```
699
700     newStatements.push(
701         b.expressionStatement(
702             b.assignmentExpression('=', left, right)
703         )
704     );
705 });
706
707     path.replace(... newStatements);
708     return false;
709 }
710 };
711
712 this.jsl.env.recast.types.visit(ast, visitObj);
713
714 // Ensure the value of the last expression is returned
715 const finalBody = ast.program.body;
716 if(finalBody.length > 0) {
717     const lastNode = finalBody[finalBody.length - 1];
718     if(lastNode.type === 'ExpressionStatement') {
719         // Replace it with a return statement
720         finalBody[finalBody.length - 1] = b.returnStatement(lastNode.
721             expression);
722     } else if(lastNode.type !== 'ReturnStatement') {
723         // Not an expression or return statement, so append 'return undefined
724         ;
725         finalBody.push(
726             b.returnStatement(b.identifier('undefined'))
727         );
728     } else {
729         // If the body is empty, add 'return undefined;'
730         finalBody.push(
731             b.returnStatement(b.identifier('undefined'))
732         );
733     }
734
735 // Reconstruct the AST with the transformed body
736 const transformedAST = b.program(finalBody);
737
738 // Generate code from the transformed AST with Recast
739 const result = this.jsl.env.recast.print(transformedAST, {
740     sourceMapName: 'transformed.js.map',
741 });
742
743 // Wrap the code in an async IIFE to allow top-level await
744 const transformedCode = `(async () => { ${result.code} })();`;
745
746 if(config.DEBUG_TRANSFORMED_CODE) {
747     obj.jsl._console.log(transformedCode);
748     obj.jsl._console.log(result.map);
749 }
750
751 return { code: transformedCode, map: result.map };
}
```

```

752 }
753
754 exports.PRDC_JSLAB_EVAL = PRDC_JSLAB_EVAL;

```

Listing 90 - jslab-eval.js

```

1  /**
2   * @file JSLAB library override submodule
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   */
7
8 /**
9  * Class for JSLAB override submodule.
10 */
11 class PRDC_JSLAB_OVERRIDE {
12
13 /**
14  * Initializes the override submodule, setting up a secure execution
15  * environment by deleting or overriding global properties and methods.
16  * @param {Object} jsl Reference to the main JSLAB object.
17 */
18 constructor(jsl) {
19   var obj = this;
20   this.jsl = jsl;
21
22   this.jsl.builtin_workspace = Object.getOwnPropertyNames(this.jsl.context);
23   this._Module = require('module');
24
25   // Overrides
26   this.jsl._console = this.jsl.context.console;
27   this.jsl._eval = this.jsl.context.eval;
28   this.jsl._require = this._Module.prototype.require;
29   this.jsl._requestAnimationFrame = this.jsl.context.requestAnimationFrame.
30     bind(this.context);
31   this.jsl._cancelAnimationFrame = this.jsl.context.cancelAnimationFrame.
32     bind(this.context);
33   this.jsl._setInterval = setInterval.bind(this.jsl.context);
34   this.jsl._clearInterval = clearInterval.bind(this.jsl.context);
35   this.jsl._setTimeout = setTimeout.bind(this.jsl.context);
36   this.jsl._clearTimeout = clearTimeout.bind(this.jsl.context);
37   this.jsl._setImmediate = this.jsl.env.setImmediate;
38   this.jsl._clearImmediate = this.jsl.env.clearImmediate;
39   this.jsl._Promise = this.jsl.context.Promise;
40   if(!global.is_worker) {
41     this.jsl._requestIdleCallback = this.jsl.context.requestIdleCallback.
42       bind(this.context);
43     this.jsl._cancelIdleCallback = this.jsl.context.cancelIdleCallback.bind(
44       this.context);
45   }
46   this.jsl._isNaN = this.jsl.context.isNaN;
47
48   // Add toJSON methods to some classes
49   if(!Gamepad.prototype.toJSON) {
50     Gamepad.prototype.toJSON = function() {

```

```

46      return {
47        id: this.id,
48        index: this.index,
49        connected: this.connected,
50        timestamp: this.timestamp,
51        mapping: this.mapping,
52        axes: Array.from(this.axes),
53        buttons: this.buttons.map(button => ({
54          pressed: button.pressed,
55          value: button.value
56        }))
57      };
58    };
59  }
60  if (!MediaDeviceInfo.prototype.toJSON) {
61    MediaDeviceInfo.prototype.toJSON = function() {
62      return {
63        deviceId: this.deviceId,
64        kind: this.kind,
65        label: this.label,
66        groupId: this.groupId
67      };
68    };
69  }
70
71 // Assign environment properties to context
72 this.jsl.env.exports.forEach(function(prop) {
73   if(config.DEBUG_FUN_SHADOW && obj.jsl.context.hasOwnProperty(prop)) {
74     obj.jsl._console.log('Shadowing function/property: ' + prop + ' with
75       env');
76   } else if(config.DEBUG_NEW_FUN) {
77     obj.jsl._console.log('Adding function/property to context: ' + prop +
78       ' from env');
79   }
80   obj.jsl.context[prop] = obj.jsl.env.prop;
81 });
82
83 // Assign libraries properties and methods to context
84 if(this.jsl.env.math) {
85   Object.getOwnPropertyNames(this.jsl.env.math).forEach(function(prop) {
86     if(config.DEBUG_FUN_SHADOW && obj.jsl.context.hasOwnProperty(prop)) {
87       obj.jsl._console.log('Shadowing function/property: ' + prop + ' with
88         math lib');
89     } else if(config.DEBUG_NEW_FUN) {
90       obj.jsl._console.log('Adding function/property to context: ' + prop +
91         ' from math lib');
92     }
93
94     var prop_out = prop;
95     if(config.MATHJS_PREVENT_OVERRIDE.includes(prop)) {
96       prop_out = 'mathjs_' + prop;
97     }
98     obj.jsl.context[prop_out] = obj.jsl.env.math[prop];
99   });
100 }

```

```

97
98 // Construct objects of submodules
99 var submodules = {};
100 config .SUBMODULES[ 'builtin' ].forEach(function(module) {
101   var exp = require ('./'+module .file );
102   submodules [ module .name ] = new exp [ module .class _name ]( obj .jsl );
103 });
104
105 config .SUBMODULES[ 'lib' ].forEach(function(lib) {
106   var exp = require ('./'+lib .file );
107   obj .jsl .context [ lib .name ] = new exp [ lib .class _name ]( obj .jsl );
108   obj .jsl [ lib .name ] = obj .jsl .context [ lib .name ];
109 });
110
111 // Assign submodule properties and methods to context
112 Object .getOwnPropertyNames (submodules) .forEach(function(submodule) {
113   obj .jsl [ submodule ] = submodules [ submodule ]; // Assign submodules
114   Object .getOwnPropertyNames (submodules [ submodule ]) .forEach(function(prop)
115   {
116     if (! [ 'jsl' ].includes (prop) || ! prop .startsWith ('_')) {
117       if (config .DEBUG_FUN_SHADOW && obj .jsl .context .hasOwnProperty (prop))
118       {
119         obj .jsl ._console .log ('Shadowing property: ' + prop + ' with
120           submodule ' + submodule);
121       } else if (config .DEBUG_NEW_FUN) {
122         obj .jsl ._console .log ('Adding property to context: ' + prop + '
123           from submodule ' + submodule);
124       }
125       obj .jsl .context [ prop ] = submodules [ submodule ] [ prop ];
126     }
127   });
128
129 Object .getOwnPropertyNames (Object .getPrototypeOf (submodules [ submodule ])) .
130   forEach(function(prop) {
131   if (! [ 'constructor' ].includes (prop)) {
132     if (config .DEBUG_FUN_SHADOW && obj .jsl .context .hasOwnProperty (prop))
133     {
134       obj .jsl ._console .log ('Shadowing function: ' + prop + ' with
135           submodule ' + submodule);
136     } else if (config .DEBUG_NEW_FUN) {
137       obj .jsl ._console .log ('Adding function to context: ' + prop + '
138           from submodule ' + submodule);
139     }
140     obj .jsl .context [ prop ] = submodules [ submodule ] [ prop ].bind (submodules [
141       submodule ]);
142   }
143 });
144 });
145
146 this .jsl .initial _ workspace = Object .getOwnPropertyNames (this .jsl .context );
147
148 // Execute override submodule
149 this .execute ();
150 Object .getOwnPropertyNames (Object .getPrototypeOf (this )) .forEach(function(
151   prop) {

```

```

142   if (!['constructor', 'execute'].includes(prop)) {
143     if (config.DEBUG_FUN_SHADOW && obj.jsl.context.hasOwnProperty(prop)) {
144       obj.jsl._console.log('Shadowing function: ' + prop + ' with
145                             submodule override');
146     } else if (config.DEBUG_NEW_FUN) {
147       obj.jsl._console.log('Adding function to context: ' + prop + ' from
148                             submodule override');
149     }
150     obj.jsl.context[prop] = obj[prop].bind(obj);
151   }
152
153   setTimeout(function() {
154     obj.jsl.context.Promise = obj.Promise;
155     obj.jsl.context.console = obj.console;
156   }, 500);
157
158 /**
159 * Executes overrides
160 */
161 execute() {
162   var obj = this;
163
164   this.deleted = {};
165   this.delete_globals = [
166     'eval', 'name', 'closed', 'length', 'frameElement', 'navigator', '',
167     'styleMedia', 'onsearch', 'isSeureContext', 'trustedTypes', '',
168     'onappinstalled', 'onbeforeinstallprompt', 'customElements', 'history',
169     'navigation', 'locationbar', 'menubar', 'personalbar', 'onunload',
170     'scheduler', 'chrome', 'scrollbars', 'clientInformation', '',
171     'onstorage', 'launchQueue', 'originAgentCluster', 'isSecureContext',
172     'statusbar', 'toolbar', 'status', 'origin', 'credentialless', '',
173     'external', 'screen', 'innerWidth', 'innerHeight', 'scrollX', '',
174     'pageXOffset', 'scrollY', 'pageYOffset', 'visualViewport', 'screenX',
175     'screenY', 'outerWidth', 'outerHeight', 'screenLeft', 'screenTop',
176     'onbeforexrselect', 'onabort', 'onbeforeinput', 'onblur', 'oncancel',
177     'oncanplay', 'oncanplaythrough', 'onchange', 'onclick', 'onclose',
178     'oncontextlost', 'oncontextmenu', 'oncontextrestored', 'oncuechange',
179     'ondblclick', 'ondrag', 'ondragend', 'ondragenter', 'ondragleave',
180     'ondragover', 'ondragstart', 'ondrop', 'ondurationchange',
181     'onemptied', 'onended', 'onfocus', 'onformdata', 'oninput',
182     'oninvalid', 'onload', 'onloadededata', 'onloadedmetadata',
183     'onloadstart', 'onmousedown', 'onmouseenter', 'onmouseleave',
184     'onmousemove', 'onmouseout', 'onmouseover', 'onmouseup',
185     'onmousewheel', 'onpause', 'onplay', 'onplaying', 'onprogress',
186     'onratechange', 'onreset', 'onresize', 'onscroll',
187     'onsecuritypolicyviolation', 'onseeked', 'onseeking', 'onselect',
188     'onslotchange', 'onstalled', 'onsubmit', 'onsuspend', 'ontimeupdate',
189     'ontoggle', 'onvolumechange', 'onwaiting', 'onwebkitanimationend',
190     'onwebkitanimationiteration', 'onwebkitanimationstart',
191     onwebkittransitionend', 'onwheel', 'onauxclick',
192     ongotpointercapture', 'onlostpointercapture', 'onpointerdown',
193     onpointermove', 'onpointerrawupdate', 'onpointerup',
194     onpointercancel', 'onpointerover', 'onpointerout', 'onpointerenter',
195   ];

```

```

    'onpointerleave', 'onselectstart', 'onselectionchange', ,
    onanimationend', 'onanimationiteration', 'onanimationstart', ,
    ontransitionrun', 'ontransitionstart', 'ontransitionend', ,
    ontransitioncancel', 'onafterprint', 'onbeforeprint', ,
    onbeforeunload', 'onhashchange', 'onlanguagechange', 'onmessage', ,
    onmessageerror', 'onoffline', 'ononline', 'onpagehide', 'onpageshow',
    , 'onpopstate', 'ondevicemotion', 'ondeviceorientation',
    ondeviceorientationabsolute', 'onbeforematch', 'onbeforetoggle', ,
    onscrollend', 'oncontentvisibilityautostatechange', 'cookieStore', ,
    caches',
167   ];
168
169 // Delete globals
170 this.delete_globals.forEach(function(prop) {
171   obj.deleted[prop] = obj.jsl.env.context[prop];
172   delete obj.jsl.env.context[prop];
173 });
174
175 /**
176 * Custom implementation of console methods to integrate with JSLAB's
177 * display and logging mechanisms.
178 */
179 this.console = {
180   log: function(...arg) {
181     obj.jsl.env.disp(...arg);
182     obj.jsl.no_ans = true;
183     obj.jsl.ignore_output = true;
184   },
185   warn: function(...arg) {
186     obj.jsl.env.warn(...arg);
187     obj.jsl.no_ans = true;
188     obj.jsl.ignore_output = true;
189   },
190   info: function(...arg) {
191     obj.jsl.env.disp(...arg);
192     obj.jsl.no_ans = true;
193     obj.jsl.ignore_output = true;
194   },
195   error: function(...arg) {
196     obj.jsl.env.error(...arg);
197     obj.jsl.no_ans = true;
198     obj.jsl.ignore_output = true;
199   },
200   trace: function() {
201     obj.jsl.notImplemented();
202   },
203   assert: function(expression, msg) {
204     if(!expression) {
205       obj.jsl.env.error(msg);
206       obj.jsl.no_ans = true;
207       obj.jsl.ignore_output = true;
208     }
209   },
210   table: function(...arg) {
211     obj.jsl.notImplemented();

```



```
211 },
212 clear: function() {
213   obj.jsl.basic._clear();
214 },
215 debug: function() {
216   obj.jsl.notImplemented();
217 },
218 dir: function() {
219   obj.jsl.notImplemented();
220 },
221 dirxml: function() {
222   obj.jsl.notImplemented();
223 },
224 time: function() {
225   obj.jsl.notImplemented();
226 },
227 timeLog: function() {
228   obj.jsl.notImplemented();
229 },
230 timeEnd: function() {
231   obj.jsl.notImplemented();
232 },
233 timeStamp: function() {
234   obj.jsl.notImplemented();
235 },
236 count: function() {
237   obj.jsl.notImplemented();
238 },
239 countreset: function() {
240   obj.jsl.notImplemented();
241 },
242 group: function() {
243   obj.jsl.notImplemented();
244 },
245 groupCollapsed: function() {
246   obj.jsl.notImplemented();
247 },
248 groupEnd: function() {
249   obj.jsl.notImplemented();
250 },
251 profile: function() {
252   obj.jsl.notImplemented();
253 },
254 profileEnd: function() {
255   obj.jsl.notImplemented();
256 }
257 };
258 /**
259 * Provides a custom Promise implementation with enhanced functionality to
260 * integrate with JSLAB's execution flow.
261 */
262 this.Promise = class extends this.jsl._Promise {
263   constructor(executor) {
264     if(obj.jsl.basic.checkStopLoop() || !obj.jsl.basic.checkStopLoop()) {
```

```

265     obj.jsl.promises_number += 1;
266     obj.jsl.last_promise_id += 1;
267     let promises_id = obj.jsl.last_promise_id;
268     obj.jsl.onStatsChange();
269     let data = super(function(_resolve, _reject) {
270       return executor(function(...args) {
271         obj.jsl.clearPromise(promises_id);
272         _resolve(...args);
273       }, function(...args) {
274         obj.jsl.clearPromise(promises_id);
275         _reject(...args);
276       });
277     });
278     data.loop_stoped = false;
279     obj.jsl.started_promises[promises_id] = data;
280     return data;
281   } else {
282     let data = super(function(resolve, reject) {
283       reject();
284     });
285     return data;
286   }
287 }
288
289 // Override the 'then' method
290 then(...args) {
291   if(!this.loop_stoped) {
292     let newPromise = super.then(...args);
293     return newPromise;
294   }
295   return false;
296 }
297
298 // Override the 'catch' method
299 catch(...args) {
300   if(!this.loop_stoped) {
301     let newPromise = super.catch(...args);
302     return newPromise;
303   }
304   return false;
305 }
306
307 // Override the 'finally' method
308 finally(...args) {
309   if(!this.loop_stoped) {
310     let newPromise = super.finally(...args);
311     return newPromise;
312   }
313   return false;
314 }
315 };
316
317 this._Module.prototype.require = function(...args) {
318   var name = obj.jsl.pathResolve(args[0], this);
319   if(name) {

```

```

320     if (!obj.jsl.required_modules.includes(name)) {
321         obj.jsl.required_modules.push(name);
322     }
323     args[0] = name;
324     return obj.jsl._require.apply(this, args);
325 }
326 return false;
327 };
328 }
329
330 /**
331 * Schedules a function to be called on the next animation frame in a non-
332 * blocking manner.
333 * @param {Function} fun The function to call on the next animation frame.
334 * @returns {number} The request ID of the animation frame request.
335 */
336 requestAnimationFrame(fun) {
337     var obj = this;
338     var request_id = this.jsl._requestAnimationFrame(function() {
339         fun();
340         obj.jsl.array.removeElementByValue(obj.jsl.started_animation_frames,
341             request_id);
342         obj.jsl.onStatsChange();
343     });
344     this.jsl.started_animation_frames.push(request_id);
345     this.jsl.onStatsChange();
346     return request_id;
347 }
348 /**
349 * Cancels an animation frame request.
350 * @param {number} request_id The ID of the request to cancel.
351 */
352 cancelAnimationFrame(request_id) {
353     this.jsl._cancelAnimationFrame(request_id);
354     this.jsl.array.removeElementByValue(this.jsl.started_animation_frames,
355         request_id);
356     this.jsl.onStatsChange();
357 }
358 /**
359 * Schedules a function to run during the browser's idle periods in a non-
360 * blocking manner.
361 * @param {Function} fun The function to execute during idle time.
362 * @param {Object} options Optional settings for the idle callback.
363 * @returns {number} The request ID of the idle callback request.
364 */
365 requestIdleCallback(fun, options) {
366     var obj = this;
367     var request_id = this.jsl.requestIdleCallback(function() {
368         fun(options);
369         obj.jsl.array.removeElementByValue(obj.jsl.started_idle_callbacks,
370             request_id);
371         obj.jsl.onStatsChange();
372     });
373 }
```

```
370     this.jsl.started_idle_callbacks.push(request_id);
371     this.jsl.onStatsChange();
372     return request_id;
373 }
374
375 /**
376 * Cancels an idle callback request.
377 * @param {number} request_id The ID of the request to cancel.
378 */
379 cancelIdleCallback(request_id) {
380     this.jsl._cancelIdleCallback(request_id);
381     this.jsl.array.removeElementByValue(this.jsl.started_idle_callbacks,
382         request_id);
383     this.jsl.onStatsChange();
384 }
385
386 /**
387 * Sets a repeated interval in a non-blocking manner.
388 * @param {Function} fun The function to execute at each interval.
389 * @param {number} delay The number of milliseconds between each execution
390 *          of the function.
391 * @returns {number} The interval ID.
392 */
393 setInterval(...arg) {
394     var request_id = this.jsl._setInterval(...arg);
395     this.jsl.started_intervals.push(request_id);
396     this.jsl.onStatsChange();
397     return request_id;
398 }
399
400 /**
401 * Clears a repeated interval.
402 * @param {number} request_id The ID of the interval to clear.
403 */
404 clearInterval(request_id) {
405     this.jsl._clearInterval(request_id);
406     this.jsl.array.removeElementByValue(this.jsl.started_intervals, request_id
407         );
408     this.jsl.onStatsChange();
409 }
410
411 /**
412 * Sets a timeout to execute a function once after a delay in a non-blocking
413 * manner.
414 * @param {Function} fun The function to execute after the delay.
415 * @param {number} delay The delay in milliseconds before the function is
416 *          executed.
417 * @returns {number} The timeout ID.
418 */
419 setTimeout(fun, delay, ...arg) {
420     var obj = this;
421     var request_id = this.jsl._setTimeout(function() {
422         fun(...arg);
423         obj.jsl.array.removeElementByValue(obj.jsl.started_timeouts, request_id)
424             ;
425     });
426 }
```

```

419     obj.jsl.onStatsChange() ;
420 }, delay);
421 this.jsl.started_timeouts.push(request_id);
422 this.jsl.onStatsChange();
423 return request_id;
424 }

425 /**
426 * Clears a timeout.
427 * @param {number} request_id The ID of the timeout to clear.
428 */
429 clearTimeout(request_id) {
430   this.jsl._clearTimeout(request_id);
431   this.jsl.array.removeElementByValue(this.jsl.started_timeouts, request_id)
432   ;
433   this.jsl.onStatsChange();
434 }

435 /**
436 * Schedules a function to be executed immediately in a non-blocking manner.
437 * @param {Function} fun The function to execute.
438 * @returns {number} The immediate ID.
439 */
440 setImmediate(fun) {
441   var obj = this;
442   var request_id = this.jsl._setImmediate(function() {
443     fun();
444     obj.jsl.array.removeElementByValue(obj.jsl.started_immediates,
445       request_id);
446     obj.jsl.onStatsChange();
447   });
448   this.started_immediates.push(request_id);
449   this.jsl.onStatsChange();
450   return request_id;
451 }

452 /**
453 * Clears an immediate.
454 * @param {number} request_id The ID of the immediate to clear.
455 */
456 clearImmediate(request_id) {
457   this.jsl._clearImmediate(request_id);
458   this.jsl.array.removeElementByValue(this.jsl.started_immediates,
459     request_id);
460   this.jsl.onStatsChange();
461 }
462 }
463
464 exports.PRDC_JSLAB_OVERRIDE = PRDC_JSLAB_OVERRIDE;

```

Listing 91 - jslab-override.js

```

1 /**
2  * @file JSLAB ploter echarts based
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia

```



```
5  * info@pr-dc.com
6  */
7
8 /**
9  * Class for JSLAB ploter echarts based.
10 */
11 class PRDC_JSLAB_PLOTER {
12
13 /**
14  * Create ploter object.
15 */
16 constructor() {
17     this.library = 'echarts';
18 }
19 }
20
21 exports.PRDC_JSLAB_PLOTER = PRDC_JSLAB_PLOTER;
```

Listing 92 - jslab-ploter-echarts.js

```
1 /**
2  * @file JSLAB ploter plotly based
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 /**
9  * Class for JSLAB ploter plotly based.
10 */
11 class PRDC_JSLAB_PLOTER {
12
13 /**
14  * Initializes the plotter object , setting Plotly as the underlying library
15  * for plotting operations.
16 * @param {Object} jsl - Reference to the JSLAB environment.
17 */
18 constructor(jsl) {
19     this.jsl = jsl;
20     this.library = 'plotly';
21 }
22
23 /**
24  * Sets the plot for the given figure identifier , displaying data and
25  * configuration.
26 * @param {number} fid - The identifier for the figure to update.
27 */
28 async plot(fid) {
29     if(!this.jsl.figures.open_figures.hasOwnProperty(fid)) {
30         this.jsl.env.error('@ploter/plot: '+language.string(172));
31         return;
32     }
33     var obj = this;
34     var figure = this.jsl.figures.open_figures[fid];
```

```

35   var plot = figure.plot;
36   var context_plot = figure.context.plot;
37
38   var plot_cont = figure.dom.querySelector('#figure-content .plot-cont');
39   if(!plot_cont) {
40     context_plot.setCont();
41     plot_cont = context_plot.plot_cont;
42   }
43
44   // Plot traces
45   var traces = [];
46   var plot_traces = structuredClone(plot.traces);
47   if(!Array.isArray(plot_traces)) {
48     plot_traces = [plot_traces];
49   }
50   var ci = 0;
51   figure.layout_3d = false;
52
53   plot_traces.forEach(function(trace_options) {
54     if(!figure.layout_3d && trace_options.hasOwnProperty('z')) {
55       figure.layout_3d = true; // It is 3D plot
56     }
57
58     var type;
59     if(trace_options.hasOwnProperty('type')) {
60       type = trace_options.type;
61     }
62     if(type != 'pie') {
63       var trace = {};
64       trace.type = type;
65
66       Object.keys(trace_options).forEach(function(key) {
67         if(['x', 'y', 'z'].includes(key) && !Array.isArray(trace_options[key])) {
68           trace[key] = [trace_options[key]];
69         } else if(!['mesh3d'].includes(type) && key == 'color' || key == 'width') {
70           if(!trace.hasOwnProperty('line')) {
71             trace.line = {};
72           }
73           trace.line[key] = trace_options[key];
74         } else {
75           var key_lc = key.toLowerCase();
76           if(trace.hasOwnProperty(key_lc) && typeof trace[key_lc] == 'object') {
77             Object.assign(trace[key_lc], trace_options[key]);
78           } else {
79             trace[key_lc] = trace_options[key];
80           }
81         }
82       });
83
84       if(!['mesh3d'].includes(type) && (!trace.hasOwnProperty('color') || !trace.line.hasOwnProperty('color'))) {
85         if(!trace.hasOwnProperty('line')) {

```



```
139     plot_bgcolor: "#fff",
140     paper_bgcolor: "#fff",
141     showlegend: plot.traces.length > 1,
142     legend: {
143       x: 0.98,
144       xanchor: 'right',
145       y: 0.98,
146       bgcolor: '#fff',
147       bordercolor: '#000',
148       borderwidth: 1,
149       borderpad: 10
150     },
151     font: {
152       family: 'Roboto',
153       size: 18
154     }
155   };
156
157   if(figure.layout_3d) {
158     Object.assign(layout,
159     {
160       margin: {
161         l: 0,
162         r: 0,
163         b: 0,
164         t: 0,
165         pad: 0
166       },
167       scene: {
168         xaxis: {
169           showgrid: true,
170           zeroline: false,
171           showline: true,
172           ticks: 'inside',
173           ticklen: 8,
174           linewidth: 1,
175           tickwidth: 1,
176           gridwidth: 0.5,
177           tickcolor: '#000',
178           linecolor: '#000',
179           gridcolor: '#999',
180           mirror: 'ticks',
181           autorange: true,
182           title: {
183             text: 'x [m]'
184           },
185           tickangle: 0
186         },
187         yaxis: {
188           showgrid: true,
189           zeroline: false,
190           showline: true,
191           ticks: 'inside',
192           ticklen: 8,
193           linewidth: 1,
```

```
194         tickwidth: 1,
195         gridwidth: 0.5,
196         tickcolor: '#000',
197         linecolor: '#000',
198         gridcolor: '#999',
199         mirror: 'ticks',
200         autorange: true,
201         title: {
202             text: 'y [m]',
203         },
204         tickangle: 0
205     },
206     zaxis: {
207         showgrid: true,
208         zeroline: false,
209         showline: true,
210         ticks: 'inside',
211         ticklen: 8,
212         linewidth: 1,
213         tickwidth: 1,
214         gridwidth: 0.5,
215         tickcolor: '#000',
216         linecolor: '#000',
217         gridcolor: '#999',
218         mirror: 'ticks',
219         autorange: true,
220         title: {
221             text: 'z [m]',
222         },
223         tickangle: 0
224     },
225     aspectratio: {
226         x: 1, y: 1, z: 1
227     },
228     camera: {
229         projection: { type: 'orthographic' }
230     },
231     dragmode: 'orbit' // "zoom" | "pan" | "orbit"
232 }
233 }
234 );
235 } else {
236     Object.assign(layout,
237     {
238         margin: {
239             l: 60,
240             r: 15,
241             b: 60,
242             t: 15,
243             pad: 5
244         },
245         xaxis: {
246             showgrid: true,
247             zeroline: false,
248             showline: true,
```

```
249     automargin: true,
250     mirror: 'ticks',
251     ticks: 'inside',
252     ticklen: 8,
253     tickwidth: 0.5,
254     tickcolor: '#000',
255     linecolor: '#000',
256     linewidth: 0.5
257   },
258   yaxis: {
259     showgrid: true,
260     zeroline: false,
261     showline: true,
262     automargin: true,
263     mirror: 'ticks',
264     ticks: 'inside',
265     ticklen: 8,
266     tickwidth: 0.5,
267     tickcolor: '#000',
268     linecolor: '#000',
269     linewidth: 0.5
270   }
271 }
272 );
273 }
274
275 // Options
276 if(plot.options.hasOwnProperty('legend')) {
277   Object.assign(layout.legend, plot.options.legend);
278 }
279 if(plot.options.hasOwnProperty('showLegend')) {
280   layout.showlegend = plot.options.showLegend;
281 }
282 if(plot.options.hasOwnProperty('legendLocation')) {
283   switch(plot.options.legendLocation.toLowerCase()) {
284     case 'north':
285       layout.legend.xanchor = 'center',
286       layout.legend.yanchor = 'top',
287       layout.legend.x = 0.5,
288       layout.legend.y = 0.98;
289       break;
290     case 'south':
291       layout.legend.xanchor = 'center',
292       layout.legend.yanchor = 'bottom',
293       layout.legend.x = 0.5,
294       layout.legend.y = 0.02;
295       break;
296     case 'east':
297       layout.legend.xanchor = 'right',
298       layout.legend.yanchor = 'center',
299       layout.legend.x = 0.98,
300       layout.legend.y = 0.5;
301       break;
302     case 'west':
303       layout.legend.xanchor = 'left',
```

```
304     layout.legend.yanchor = 'center',
305     layout.legend.x = 0.02,
306     layout.legend.y = 0.5;
307     break;
308   case 'northeast':
309     layout.legend.xanchor = 'right',
310     layout.legend.yanchor = 'top',
311     layout.legend.x = 0.98,
312     layout.legend.y = 0.98;
313     break;
314   case 'northwest':
315     layout.legend.xanchor = 'left',
316     layout.legend.yanchor = 'top',
317     layout.legend.x = 0.02,
318     layout.legend.y = 0.98;
319     break;
320   case 'southeast':
321     layout.legend.xanchor = 'right',
322     layout.legend.yanchor = 'bottom',
323     layout.legend.x = 0.98,
324     layout.legend.y = 0.02;
325     break;
326   case 'southwest':
327     layout.legend.xanchor = 'left',
328     layout.legend.yanchor = 'bottom',
329     layout.legend.x = 0.02,
330     layout.legend.y = 0.02;
331     break;
332   case 'northoutside':
333     layout.legend.xanchor = 'center',
334     layout.legend.yanchor = 'bottom',
335     layout.legend.x = 0.5,
336     layout.legend.y = 1.02;
337     break;
338   case 'southoutside':
339     layout.legend.xanchor = 'center',
340     layout.legend.yanchor = 'top',
341     layout.legend.x = 0.5,
342     layout.legend.y = -0.02;
343     break;
344   case 'eastoutside':
345     layout.legend.xanchor = 'left',
346     layout.legend.yanchor = 'center',
347     layout.legend.x = 1.02,
348     layout.legend.y = 0.5;
349     break;
350   case 'westoutside':
351     layout.legend.xanchor = 'right',
352     layout.legend.yanchor = 'center',
353     layout.legend.x = -0.02,
354     layout.legend.y = 0.5;
355     break;
356   case 'northeastoutside':
357     layout.legend.xanchor = 'left',
358     layout.legend.yanchor = 'bottom',
```

```

359     layout.legend.x = 1.02,
360     layout.legend.y = 1.02;
361     break;
362   case 'northwestoutside':
363     layout.legend.xanchor = 'right',
364     layout.legend.yanchor = 'bottom',
365     layout.legend.x = -0.02,
366     layout.legend.y = 1.02;
367     break;
368   case 'southeastoutside':
369     layout.legend.xanchor = 'left',
370     layout.legend.yanchor = 'top',
371     layout.legend.x = 1.02,
372     layout.legend.y = -0.02;
373     break;
374   case 'southwestoutside':
375     layout.legend.xanchor = 'right',
376     layout.legend.yanchor = 'top',
377     layout.legend.x = -0.02,
378     layout.legend.y = -0.02;
379     break;
380   }
381 }
382 if(plot.options.hasOwnProperty('legendOrientation')) {
383   if(plot.options.legendOrientation == 'horizontal' ||
384     plot.options.legendOrientation == 'h') {
385     layout.legend.orientation = 'h';
386   } else {
387     layout.legend.orientation = 'v';
388   }
389 }
390
391 if(figure.layout_3d) {
392   if(plot.options.hasOwnProperty('xlim')) {
393     layout.scene.xaxis.range = plot.options.xlim;
394   }
395   if(plot.options.hasOwnProperty('ylim')) {
396     layout.scene.yaxis.range = plot.options.ylim;
397   }
398   if(plot.options.hasOwnProperty('zlim')) {
399     layout.scene.zaxis.range = plot.options.zlim;
400   }
401 } else {
402   if(plot.options.hasOwnProperty('xlim')) {
403     layout.xaxis.range = plot.options.xlim;
404   }
405   if(plot.options.hasOwnProperty('ylim')) {
406     layout.yaxis.range = plot.options.ylim;
407   }
408 }
409
410 if(plot.options.hasOwnProperty('font')) {
411   layout.font = plot.options.font;
412 }
413 if(plot.options.hasOwnProperty('margin')) {

```

```

414     var margin = plot.options.margin;
415     if(margin.hasOwnProperty('top')) {
416         layout.margin.t = margin.top;
417     }
418     if(margin.hasOwnProperty('bottom')) {
419         layout.margin.b = margin.bottom;
420     }
421     if(margin.hasOwnProperty('left')) {
422         layout.margin.l = margin.left;
423     }
424     if(margin.hasOwnProperty('right')) {
425         layout.margin.r = margin.right;
426     }
427 }
428
429 if(plot.hasOwnProperty('title_val')) {
430     layout.title.text = plot.title_val;
431     layout.title.font = {
432         size: 18
433     };
434     layout.margin.t = 40;
435 }
436
437 if(plot.hasOwnProperty('legend_state') &&
438     ['on', 'off', 1, 0].includes(plot.legend_state)) {
439     layout.showlegend = plot.legend_state === 'on' || plot.legend_state;
440 }
441
442 if(figure.layout_3d) {
443     if(plot.hasOwnProperty('xlabel_val')) {
444         layout.scene.xaxis.title = plot.xlabel_val;
445         layout.scene.xaxis.automargin = true;
446         layout.scene.xaxis.standoff = 20;
447     }
448     if(plot.hasOwnProperty('ylabel_val')) {
449         layout.scene.yaxis.title = plot.ylabel_val;
450         layout.scene.yaxis.automargin = true;
451         layout.scene.yaxis.standoff = 20;
452     }
453     if(plot.hasOwnProperty('zlabel_val')) {
454         layout.scene.zaxis.title = plot.zlabel_val;
455         layout.scene.zaxis.automargin = true;
456         layout.scene.zaxis.standoff = 20;
457     }
458 } else {
459     if(plot.hasOwnProperty('xlabel_val')) {
460         layout.xaxis.title = plot.xlabel_val;
461         layout.xaxis.automargin = true;
462         layout.xaxis.standoff = 20;
463     }
464     if(plot.hasOwnProperty('ylabel_val')) {
465         layout.yaxis.title = plot.ylabel_val;
466         layout.yaxis.automargin = true;
467         layout.yaxis.standoff = 20;
468 }

```

```

469     if(plot.axis_style_val === 'equal') {
470       layout.yaxis.scaleanchor = "x";
471     } else {
472       layout.yaxis.scaleanchor = false;
473     }
474   }
475
476   context_plot.relayout(layout);
477   context_plot.resize();
478
479   if(plot.lim_update) {
480     plot.lim_update = false;
481     var layout2 = {};
482     if(figure.layout_3d) {
483       if(plot.hasOwnProperty('xlim_val')) {
484         if(plot.xlim_val[0] === 'auto') {
485           plot.xlim_val[0] = plot_cont._fullLayout.scene.xaxis.range[0];
486         }
487         if(plot.xlim_val[1] === 'auto') {
488           plot.xlim_val[1] = plot_cont._fullLayout.scene.xaxis.range[1];
489         }
490         layout2['scene.xaxis.range'] = plot.xlim_val;
491       }
492       if(plot.hasOwnProperty('ylim_val')) {
493         if(plot.ylim_val[0] === 'auto') {
494           plot.ylim_val[0] = plot_cont._fullLayout.scene.yaxis.range[0];
495         }
496         if(plot.ylim_val[1] === 'auto') {
497           plot.ylim_val[1] = plot_cont._fullLayout.scene.yaxis.range[1];
498         }
499         layout2['scene.yaxis.range'] = plot.ylim_val;
500       }
501       if(plot.hasOwnProperty('zlim_val')) {
502         if(plot.zlim_val[0] === 'auto') {
503           plot.zlim_val[0] = plot_cont._fullLayout.scene.zaxis.range[0];
504         }
505         if(plot.zlim_val[1] === 'auto') {
506           plot.zlim_val[1] = plot_cont._fullLayout.scene.zaxis.range[1];
507         }
508         layout2['scene.zaxis.range'] = plot.zlim_val;
509       }
510     } else {
511       if(plot.hasOwnProperty('xlim_val')) {
512         if(plot.xlim_val[0] === 'auto') {
513           plot.xlim_val[0] = plot_cont._fullLayout.xaxis.range[0];
514         }
515         if(plot.xlim_val[1] === 'auto') {
516           plot.xlim_val[1] = plot_cont._fullLayout.xaxis.range[1];
517         }
518         layout2['xaxis.range'] = plot.xlim_val;
519       }
520       if(plot.hasOwnProperty('ylim_val')) {
521         if(plot.ylim_val[0] === 'auto') {
522           plot.ylim_val[0] = plot_cont._fullLayout.yaxis.range[0];
523         }
524       }
525     }
526   }
527 }
```

```

524     if (plot.ylim_val[1] === 'auto') {
525         plot.ylim_val[1] = plot_cont._fullLayout.yaxis.range[1];
526     }
527     layout2['yaxis.range'] = plot.ylim_val;
528 }
529 }
530 context_plot.relayout(layout2);
531 context_plot.resize();
532 }
533
534 if (figure.layout_3d) {
535     var layout3 = {};
536
537     // Equal axes
538     var xRange = plot_cont._fullLayout.scene.xaxis.range;
539     if (plot.hasOwnProperty('xlim_val')) {
540         xRange = plot.xlim_val;
541         layout3['scene.xaxis.range'] = plot.xlim_val;
542     }
543     var yRange = plot_cont._fullLayout.scene.yaxis.range;
544     if (plot.hasOwnProperty('ylim_val')) {
545         yRange = plot.ylim_val;
546         layout3['scene.yaxis.range'] = plot.ylim_val;
547     }
548     var zRange = plot_cont._fullLayout.scene.zaxis.range;
549     if (plot.hasOwnProperty('zlim_val')) {
550         zRange = plot.zlim_val;
551         layout3['scene.zaxis.range'] = plot.zlim_val;
552     }
553
554     var zoom = 1;
555     if (typeof plot.zoom_val !== 'undefined') {
556         zoom = plot.zoom_val;
557     }
558
559     if (plot.axis_style_val === 'equal') {
560         var xRangeSize = Math.abs(xRange[1] - xRange[0]);
561         var yRangeSize = Math.abs(yRange[1] - yRange[0]);
562         var zRangeSize = Math.abs(zRange[1] - zRange[0]);
563         var maxRange = Math.max(xRangeSize, yRangeSize, zRangeSize);
564
565         layout3["scene.aspectratio"] = {
566             x: xRangeSize / maxRange * zoom,
567             y: yRangeSize / maxRange * zoom,
568             z: zRangeSize / maxRange * zoom
569         };
570     } else {
571         layout3["scene.aspectratio"] = {
572             x: zoom,
573             y: zoom,
574             z: zoom
575         };
576     }
577
578     var radius = maxRange / 2;

```

```

579     var azimuthRad = plot.view_val[0] * Math.PI / 180;
580     var elevationRad = plot.view_val[1] * Math.PI / 180;
581
582     var eye = {
583         x: radius * Math.cos(elevationRad) * Math.cos(azimuthRad),
584         y: radius * Math.cos(elevationRad) * Math.sin(azimuthRad),
585         z: radius * Math.sin(elevationRad)
586     };
587
588     var up = { x: 0, y: 0, z: 1 };
589
590     layout3['scene.camera.eye'] = eye;
591     layout3['scene.camera.up'] = up;
592
593     context_plot.relayout(layout3);
594     context_plot.resize();
595 }
596 }
597
598 /**
599 * Updates plot data for a specified figure.
600 * @param {string} fid - The identifier of the figure to update.
601 * @param {Object} traces - The trace data to be updated in the plot.
602 * @param {number} N - The new data length or index for updating the plot.
603 */
604 updateData(fid, traces, N) {
605     if(!this.jsl.figures.open_figures.hasOwnProperty(fid)) {
606         this.jsl.env.error('@ploter/updateData: '+language.string(172));
607         return;
608     }
609     var figure = this.jsl.figures.open_figures[fid];
610     figure.context.plot.restyle(traces, [N]);
611 }
612
613 /**
614 * Handles plot resizing for a specified figure identifier.
615 * @param {number} fid - The identifier for the figure to resize.
616 */
617 onResize(fid) {
618     if(!this.jsl.figures.open_figures.hasOwnProperty(fid)) {
619         this.jsl.env.error('@ploter/onResize: '+language.string(172));
620         return;
621     }
622     var figure = this.jsl.figures.open_figures[fid];
623     figure.context.plot.resize();
624 }
625
626 /**
627 * Removes the plot container for a specified figure identifier.
628 * @param {number} fid - The identifier for the figure from which to remove
629 *          the plot container.
630 */
631 remove(fid) {
632     if(!this.jsl.figures.open_figures.hasOwnProperty(fid)) {
633         this.jsl.env.error('@ploter/remove: '+language.string(172));

```

```

633         return;
634     }
635     var figure = this.jsl.figures.open_figures[ fid ];
636     var plot_cont = figure.context.plot.plot_cont;
637     if( plot_cont ) {
638         plot_cont.remove();
639     }
640 }
641 }
642
643 exports.PRDC_JSLAB_PLOTER = PRDC_JSLAB_PLOTER;

```

Listing 93 - jslab-ploter-plotly.js

```

1  /**
2  * @file JSLAB library
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 const { PRDC_JSLAB_EVAL } = require('./jslab-eval');
9 const { PRDC_JSLAB_OVERRIDE } = require('./jslab-override');
10 const { PRDC_JSLAB_ENV } = require('./jslab-env-electron');
11
12 /**
13 * Class for JSLAB library.
14 */
15 class PRDC_JSLAB_LIB {
16
17 /**
18 * Constructs the JSLAB library environment, initializing submodules and
19 * setting up the execution context.
20 * @param {Object} config Configuration options for the JSLAB library.
21 */
22 constructor(config) {
23     var obj = this;
24     this.ready = false;
25     this.config = config;
26
27     // Library built in properties
28     this.previous_workspace = {};
29     this.previous_properties = [];
30
31     this.stop_loop = false;
32     this.last_script_path;
33     this.last_script_lines;
34     this.last_script_silent;
35
36     this.current_script;
37
38     this.required_modules = [];
39     this.started_timeouts = [];
40     this.started_intervals = [];
41     this.started_animation_frames = [];
42     this.started_idle_callbacks = [];

```

```

42   this.started_immediates = [];
43   this.cleanup_registry = [];
44   this.started_promises = {};
45   this.promises_number = 0;
46   this.last.promise_id = 0;
47
48   this.env = new PRDC_JSLAB_ENV(this);
49   this.context = this.env.context;
50   this.setDepthSafeStringify(this.context, 1);
51   this.context._ = Symbol('_');
52   this.eval = new PRDC_JSLAB_EVAL(this);
53
54   this.debug = this.env.debug;
55   this.includes_path = this.env.getDefaultPath('includes');
56
57   if(this.config.PLOTER === 'plotly') {
58     var { PRDC_JSLAB_PLOTER } = require('../jslab-ploter-plotly');
59   } else if(this.config.PLOTER === 'echarts') {
60     var { PRDC_JSLAB_PLOTER } = require('../jslab-ploter-echarts');
61   }
62   this.ploter = new PRDC_JSLAB_PLOTER(this);
63
64   this.override = new PRDC_JSLAB_OVERRIDE(this);
65
66   this.ready = true;
67 }
68
69 /**
70 * Invoked at the beginning of a code evaluation to set up necessary flags
71 * and state.
72 */
73 onEvaluating() {
74   this.env.codeEvaluating();
75   this.ignore_output = false;
76   this.no_ans = false;
77   this.stop_loop = false;
78   this.env.resetStopLoop();
79   this.env.setStatus('busy', language.string(88));
80 }
81
82 /**
83 * Invoked after code evaluation to finalize the state and update the
84 * environment accordingly.
85 */
86 onEvaluated() {
87   this.env.setWorkspace();
88   this.env.codeEvaluated();
89   this.env.setStatus('ready', language.string(87));
90 }
91
92 /**
93 * Triggers when there are changes in the stats, such as the number of
94 * promises or timeouts, and updates the environment state.
95 */
96 onStatsChange() {

```

```

94   this.env.setWorkspace();
95   this.env.setStats({
96     'required_modules': this.required_modules.length,
97     'promises': this.promises_number,
98     'timeouts': this.started_timeouts.length,
99     'immediates': this.started_immediates.length,
100    'intervals': this.started_intervals.length,
101    'animation_frames': this.started_animation_frames.length,
102    'idle_callbacks': this.started_idle_callbacks.length
103  });
104}
105
106 /**
107 * Clears the current workspace, stopping any ongoing operations and
108 * removing any dynamic properties.
109 */
110 clear() {
111   this.doCleanup();
112   this.context.sym.clear();
113   this.context.parallel.terminate();
114
115   this.clearEventListeners();
116   this.clearImmediates();
117   this.clearAnimationFrames();
118   this.clearIntervals();
119   this.clearTimeouts();
120   this.clearIdleCallbacks();
121   this.stopPromises();
122   this.stopSubprocesses();
123   this.unrequireAll();
124   this.onStatsChange();
125
126   var obj = this;
127   this.previous_properties.forEach(function(property) {
128     if(typeof obj.context[property] != 'function' || (typeof obj.context[
129       property] == 'function' && obj.context[property]._jsl_saved)) {
130       delete obj.context[property];
131     }
132   });
133   this.previous_properties = [];
134   this.previous_workspace = {};
135   this.no_ans = true;
136   this.ignore_output = true;
137 }
138 /**
139 * Sets the current path in the environment, adjusting how file paths are
140 * resolved.
141 * @param {String} new_path The new path to set as the current working
142 * directory.
143 */
144 setPath(new_path) {
145   this.current_path = this.env.addPathSep(new_path);
146 }

```

```

145 /**
146 * Saves the properties of the current workspace to restore them later if
147 * needed.
148 */
149 savePreviousWorkspace() {
150   var obj = this;
151   this.previous_properties = this.getWorkspaceProperties();
152   this.previous_properties.forEach(function(property) {
153     if(typeof obj.context[property] === 'function') {
154       obj.context[property]._jsl_saved = true;
155     }
156     obj.previous_workspace[property] = obj.context[property];
157     delete obj.context[property];
158   });
159 }
160 /**
161 * Restores the properties of the workspace that were saved by 'savePreviousWorkspace'.
162 */
163 loadPreviousWorkspace() {
164   var obj = this;
165   var workspace = this.getWorkspaceProperties();
166   workspace.forEach(function(property) {
167     if(obj.previous_properties.includes(property)) {
168       // This property is defined again in new code
169       obj.previous_properties.splice(
170         obj.previous_properties.indexOf(property), 1);
171     }
172   });
173   // Add saved properties back to global scope
174   this.previous_properties.forEach(function(property) {
175     obj.context[property] = obj.previous_workspace[property];
176   });
177 }
178 /**
179 * Retrieves a list of custom properties added to the global context since
180 * the initial load of the library.
181 * @returns {Array} A list of property names that have been added to the
182 * global context.
183 */
184 getWorkspaceProperties() {
185   var initial_props = this.initial_workspace;
186   var current_prop_list = Object.getOwnPropertyNames(this.context);
187   var workspace = [];
188
189   for(var i = 0, l = current_prop_list.length; i < l; ++i) {
190     var prop_name = current_prop_list[i];
191     if(initial_props.indexOf(prop_name) === -1) {
192       workspace.push(prop_name);
193     }
194   }
195   return workspace;

```

```

196     }
197
198  /**
199   * Constructs a detailed view of the current workspace, including the types
200   * and names of properties.
201   * @returns {Array} An array of arrays, each containing the name, type, and
202   * constructor name (if applicable) of each property.
203   */
204   getWorkspace() {
205     var initial_props = this.initial_workspace;
206     var current_prop_list = Object.getOwnPropertyNames(this.context);
207     var workspace = [];
208
209     for(var i = 0, l = current_prop_list.length; i < l; ++i) {
210       var prop_name = current_prop_list[i];
211       if(initial_props && initial_props.indexOf(prop_name) === -1) {
212         var prop = this.context[prop_name];
213         var type = typeof prop;
214         var name = 'none';
215
216         if(type !== 'undefined') {
217           if(prop && prop.constructor) {
218             name = prop.constructor.name;
219           } else {
220             name = '/';
221           }
222         }
223       }
224
225       return workspace;
226     }
227
228  /**
229   * Generates the initialization script for the worker environment.
230   * @returns {string} – The worker initialization script as a string.
231   */
232   getWorkerInit() {
233     return `
234       global.app_path = ${JSON.stringify(app_path)};
235       global.is_worker = true;
236       global.debug = ${this.env.debug};
237       global.version = '${this.env.version}';
238       global.platform = '${this.env.platform}';
239
240
241       const helper = require(app_path + '/js/helper.js');
242       const { PRDC_APP_CONFIG } = require(app_path + '/config/config');
243
244       importScripts(app_path + '/lib/luxon-3.4.4/luxon-3.4.4.min.js');
245       importScripts(app_path + '/lib/math-11.8.2/math-11.8.2.min.js');
246       importScripts(app_path + '/lib/sprintf-1.1.3/sprintf-1.1.3.min.js');
247       importScripts(app_path + '/lib/sympy-0.26.2/pyodide.js');
248

```

```

249 const { PRDC_JSLAB_LIB } = require(app_path + '/js/sandbox/jslab');
250
251 // Global variables
252 var config = new PRDC_APP_CONFIG();
253 var jsl = new PRDC_JSLAB_LIB(config);
254 jsl.current_path = ${JSON.stringify(this.current_path)};
255 jsl.includes_path = ${JSON.stringify(this.includes_path)};
256 jsl.saved_paths = JSON.parse('${JSON.stringify(this.saved_paths)})';
257 ';
258 }
259
260 /**
261 * Resolves the absolute path for a file, searching through predefined
262 * directories and optionally using module context.
263 * @param {string} file_path - The file path to resolve.
264 * @param {object} [this_module] - Optional module context for resolution.
265 * @returns {string|boolean} - The resolved path, or 'false' if not found.
266 */
267 pathResolve(file_path, this_module) {
268   var obj = this;
269   if(typeof file_path === 'string') {
270     if(!this.env.pathIsAbsolute(file_path)) {
271       if(this_module) {
272         var file_path_temp = this.env.pathJoin(this_module.path, file_path);
273         if(this.env.checkFile(file_path_temp)) {
274           return file_path_temp;
275         }
276       }
277       var file_paths = [];
278       var file_path_temp = this.env.pathJoin(this.current_path, file_path);
279       if(this.env.checkFile(file_path_temp)) {
280         file_paths.push(file_path_temp);
281       }
282       file_path_temp = this.env.pathJoin(this.includes_path, file_path);
283       if(this.env.checkFile(file_path_temp)) {
284         file_paths.push(file_path_temp);
285       }
286       this.saved_paths.forEach(function(saved_path) {
287         file_path_temp = obj.env.pathJoin(saved_path, file_path);
288         if(obj.env.checkFile(file_path_temp)) {
289           file_paths.push(file_path_temp);
290         }
291       });
292       if(file_paths.length > 2) {
293         var N = file_paths.length;
294         var str = '@pathResolve: '+language.string(106)+' ' + file_paths[0]
295             + ', '+language.string(107)+': [\n';
296         for(var i = 1; i < N-1; i++) {
297           str += ' ' + file_paths[i] + ',\n';
298         }
299         str += ' ' + file_paths[N-1] + '\n';
300         this.env.disp(str);
301       } else if(file_paths.length > 1) {
302         this.env.disp('@pathResolve: '+language.string(106)+' ' + file_paths

```

```

      [0] + ' '+language.string(108)+': [\n' + file_paths[1] + '\n]' ;
    } else if(file_paths.length == 0) {
      try {
        if(this_module) {
          return this.override._Module._resolveFilename(file_path,
            this_module, false);
        } else {
          return require.resolve(file_path);
        }
      } catch(err) {
        this._console.log(err);
        this.env.error('@pathResolve: '+language.string(109)+' '+
          file_path + ' '+language.string(110)+'.');
        return false;
      }
    }
    file_path = file_paths[0];
  }
} else {
  this.env.error('@pathResolve: '+language.string(111)+'.');
  return false;
}
return file_path;
}

/***
 * Sets the flag to stop loop execution within the JSLAB environment.
 * @param {Boolean} data The flag indicating whether to stop loop execution.
 */
setStopLoop(data) {
  this.stop_loop = data;
  this.onStopLoop(false);
  this.onEvaluated();
  this.env.disp(language.string(90));
}

/***
 * Handles the process of stopping loop execution and cleaning up
 * asynchronous operations.
 * @param {Boolean} [throw_error=true] Indicates whether to throw an error
 * when stopping the loop.
 * @throws {Object} An error object with a custom message if 'throw_error'
 * is true and the loop needs to be stopped.
 */
onStopLoop(throw_error = true) {
  if(this.stop_loop) {
    this.clearEventListeners();
    this.clearImmediates();
    this.clearAnimationFrames();
    this.clearIntervals();
    this.clearTimeouts();
    this.clearIdleCallbacks();
    this.stopPromises();
    this.stopSubprocesses();
  }
}

```

```
350     this.doCleanup();
351     this.onStatsChange();
352     if(throw_error) {
353         throw {name: 'JslabError', message: language.string(90)};
354     }
355 }
356 }
357
358 /**
359 * Sets the paths saved in the environment for easier access to files and
360 * modules.
361 * @param {Array} data An array of paths to be saved for future use.
362 */
363 setSavedPaths(data) {
364     this.saved_paths = data;
365 }
366
367 /**
368 * Placeholder function for features that have not been implemented.
369 */
370 notImplemented() {
371     obj.env.error(language.string(115));
372 }
373
374 /**
375 * Clears all modules that have been required during the session.
376 */
377 unrequireAll() {
378     var obj = this;
379     this.required_modules.forEach(function(module) {
380         try {
381             var name = require.resolve(module);
382             if(name) {
383                 delete require.cache[name];
384             }
385         } catch(err) {
386             obj._console.log(err);
387         }
388     });
389     this.required_modules = [];
390     Object.keys(require.cache).forEach(function(key) { delete require.cache[key]; });
391 }
392
393 /**
394 * Stops all promises that have been started within the environment.
395 */
396 stopPromises() {
397     var obj = this;
398     Object.keys(this.started_promises).forEach(function(key) {
399         obj.started_promises[key].loop_stoped = true;
400         delete obj.started_promises[key];
401     });
402     this.promises_number = 0;
403 }
```

```
403
404  /**
405   * Registers an object for cleanup with a specified cleanup function.
406   * @param {Object} obj - The object to be registered for cleanup.
407   * @param {Function} fun - The function to execute during cleanup.
408   */
409 addForCleanup(obj, fun) {
410   this.cleanup_registry.push({obj: obj, fun: fun});
411 }
412
413 /**
414  * Do cleanup on all registered objects;
415  */
416 doCleanup() {
417   for(var entry of this.cleanup_registry) {
418     if(isFunction(entry.fun)) {
419       try {
420         entry.fun();
421     } catch {};
422   } else if(isFunction(entry.obj._jslabCleanup)) {
423     try {
424       entry.obj._jslabCleanup();
425     } catch {};
426   }
427   }
428   this.cleanup_registry = [];
429 }
430
431 /**
432  * Removes a specific promise from the tracking list based on its ID.
433  * @param {Number} id The ID of the promise to remove.
434  */
435 clearPromise(id) {
436   this.promises_number -= 1;
437   if(this.promises_number < 0) {
438     this.promises_number = 0;
439   }
440   this.onStatsChange();
441   delete this.started_promises[id];
442 }
443
444 /**
445  * Clears all timeouts that have been set within the environment.
446  */
447 clearTimeouts() {
448   var obj = this;
449   this.started_timeouts.forEach(function(timeout) {
450     obj._clearTimeout(timeout);
451   });
452   this.started_timeouts = [];
453 }
454
455 /**
456  * Clears all intervals that have been set within the environment.
457  */
458
```

```
458 clearIntervals() {
459     var obj = this;
460     this.started_intervals.forEach(function(interval) {
461         obj._clearInterval(interval);
462     });
463     this.started_intervals = [];
464 }
465
466 /**
467 * Cancels all animation frames that have been requested within the
468 * environment.
469 */
470 clearAnimationFrames() {
471     var obj = this;
472     this.started_animation_frames.forEach(function(animation_frames) {
473         obj._cancelAnimationFrame(animation_frames);
474     });
475     this.started_animation_frames = [];
476 }
477
478 /**
479 * Cancels all idle callbacks that have been requested within the
480 * environment.
481 */
482 clearIdleCallbacks() {
483     var obj = this;
484     this.started_idle_callbacks.forEach(function(idle_callback) {
485         obj._cancelIdleCallback(idle_callback);
486     });
487     this.started_idle_callbacks = [];
488 }
489
490 /**
491 * Clears all immediate executions that have been set within the environment
492 *
493 */
494 clearImmediates() {
495     var obj = this;
496     this.started_immediates.forEach(function(immediate) {
497         obj._clearImmediate(immediate);
498     });
499     this.started_immediates = [];
500 }
501
502 /**
503 * Removes all event listeners that have been added to the document body,
504 * effectively clearing any remaining event bindings.
505 */
506 clearEventListeners() {
507     document.body.parentNode.replaceChild(document.body.cloneNode(true),
508                                         document.body);
509 }
510
511 /**
512 * Lists all subprocesses.
```

```

508  */
509  stopSubprocesses() {
510    var pids = this.listSubprocesses();
511    pids.forEach(function(pid) {
512      killProcess(pid);
513    });
514  }
515
516 /**
517 * Sets the safe stringify depth for the given data.
518 * @param {Object} data - The data object to modify.
519 * @param {number} depth - The depth level for safe stringification.
520 * @returns {Object} The modified data object with the set depth.
521 */
522 setDepthSafeStringify(data, depth) {
523   data._safeStringifyDepth = depth;
524   return data;
525 }
526
527 /**
528 * Lists all subprocesses.
529 */
530 listSubprocesses() {
531   var output = this.env.execSync(`wmic process where (ParentProcessId=${this
532     .env.process_id}) get ProcessId,CommandLine`).toString();
533   var pids = output.match(/(?<=\$)\d+(?=\$*)/gm);
534   pids.pop();
535   return pids.map((pid) => Number(pid));
536 }
537 /**
538 * Gets properties missing documentation.
539 * @param {Array|String} [workspace=this.initial_workspace] - Workspace to
540   check.
541 * @param {boolean} [without_builtin=true] - Exclude built-in properties.
542 * @returns {Array<string>} Missing property names.
543 */
544 _getMissingDocs(workspace = this.initial_workspace, without_builtin = true)
545 {
546   var missing = [];
547   for(var prop of workspace) {
548     if(!prop.startsWith('_') &&
549       (!without_builtin ||
550        !this.builtin_workspace.includes(prop))) {
551       try {
552         help(prop);
553       } catch {
554         missing.push(prop);
555       }
556     }
557   }
558   return missing;
559 }
560 /**

```

```

560   * Writes templates for missing docs to a file.
561   * @param {string} path - File path for missing docs.
562   * @param {Array|string} [workspace=this.initial_workspace] - Workspace to
563   * check.
564   */
565   _writeMissingDocsToFile(path, workspace = this.initial_workspace) {
566     var workspace_array = workspace;
567     if (!Array.isArray(workspace)) {
568       workspace_array = Object.keys(workspace);
569     }
570
571     var missing = this._getMissingDocs(workspace_array);
572     var str = ',';
573     for(var prop of missing) {
574       var kind = 'function member';
575       if (!Array.isArray(workspace)) {
576         kind = typeof workspace[prop] === 'function' ? 'function' : 'member';
577       }
578       str += '
579     /**
580      * Description
581      * @name ${prop}
582      * @kind ${kind}
583      * @param {}
584      * @returns {}
585      * @memberof PRDC_JSLAB_LIB
586      */
587    ';
588  }
589  this.env.writeFileSync(path, str);
590 }
591 /**
592  * Awaits the provided promise if the loop has not been stopped.
593  * @param {Promise} p - The promise to await.
594  * @returns {Promise|undefined} - Resolves if 'p' is awaited; does nothing
595  * if the loop is stopped.
596  */
597 async promiseOrStoped(p) {
598   if (!p.loop_stoped) {
599     await p;
600   }
601 }
602 }
603
604 exports.PRDC_JSLAB_LIB = PRDC_JSLAB_LIB;

```

Listing 94 - jslab.js

```

1 /**
2  * @file JSLAB test
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */

```



```
7
8 const { PRDC_JSLAB_TESTS } = require('../tester');
9 var tests = new PRDC_JSLAB_TESTS();
10
11 tests.add('Expect tic to be equal to last tic', function() {
12     return tic === jsl.context.last_tic;
13 }
14 );
15
16 tests.add('Joining two paths with path separator', function() {
17     return 'a'+pathSep()+'b' === jsl.env.pathJoin('a', 'b');
18 }
19 );
20
21 exports.MODULE_TESTS = tests;
```

Listing 95 - jslab.test.js

```
1 /**
2  * @file JSLAB library math submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 /**
9  * Class for JSLAB math submodule.
10 */
11 class PRDC_JSLAB_LIB_MATH {
12
13 /**
14  * Constructs the math submodule, initializing mathematical constants and
15  * functions.
16  * @param {Object} jsl Reference to the main JSLAB object.
17 */
18 constructor(jsl) {
19     var obj = this;
20     this.jsl = jsl;
21
22     // Additional math constants
23     if(this.jsl.env.math) {
24         /**
25          * Pi number.
26          * @type {number}
27         */
28         this.Pi = this.jsl.env.math.pi;
29
30         /**
31          * Coefficient for converting degrees to radians.
32          * @type {number}
33         */
34         this.d2r = this.jsl.env.math.pi/180;
35
36         /**
37          * Coefficient for converting radians to degrees.
```

```

38      * @type {number}
39      */
40      this.r2d = 1/this.d2r;
41  }
42
43 /**
44  * Floating-point relative accuracy
45  * @type {number}
46  */
47 this.eps = 1E-7;
48
49 /**
50  * Floating-point relative accuracy
51  * @type {number}
52  */
53 this.EPS = this.eps;
54 }
55
56 /**
57  * Seeds the random number generator with the provided arguments.
58  * @param {...any} args - Arguments used to seed the random generator.
59  * @returns {any} The result from the seeded random generator.
60  */
61 seedRandom(...args) {
62   return this.jsl.env.seedRandom(...args);
63 }
64
65 /**
66  * Performs linear interpolation on a set of data points.
67  * @param {Array} x The x-values of the data points.
68  * @param {Array} y The y-values of the data points, corresponding to each x
69  * -value.
70  * @param {Number|Array} xq The x-value(s) for which to interpolate a y-
71  * value.
72  * @param {String} mode The mode of interpolation. Use 'extrap' for
73  * extrapolation.
74  * @returns {Number|Array} The interpolated y-value(s) at xq.
75  */
76 interp(x, y, xq, mode = 'none') {
77   // Helper function for scalar interpolation
78   const interpolate = (xqi) => {
79     var i = x.indexOf(xqi);
80     if(i >= 0) {
81       return y[i];
82     }
83
84     var x1 = [...x].reverse().find(v => v <= xqi);
85     var x2 = x.find(v => v >= xqi);
86
87     // Handle extrapolation if mode is 'extrap'
88     if(mode === 'extrap') {
89       if(xqi < x[0]) {
90         // Left-side extrapolation
91         const gradient = (y[1] - y[0]) / (x[1] - x[0]);
92         return y[0] + (xqi - x[0]) * gradient;
93       }
94     }
95   }
96 }

```

```

90     } else if(xqi > x[x.length - 1]) {
91       // Right-side extrapolation
92       const lastIdx = x.length - 1;
93       const gradient = (y[lastIdx] - y[lastIdx - 1]) / (x[lastIdx] - x[
94         lastIdx - 1]);
95       return y[lastIdx] + (xqi - x[lastIdx]) * gradient;
96     }
97   }
98
99   // Handle interpolation inside the data range
100  if(x1 !== undefined && x2 !== undefined && x1 !== x2) {
101    var y1 = y[x.indexOf(x1)];
102    var y2 = y[x.indexOf(x2)];
103    return y1 + (xqi - x1) * (y2 - y1) / (x2 - x1);
104  }
105
106  // Return NaN if no valid interpolation point is found
107  return NaN;
108};

109 // Check if xq is an array
110 if(Array.isArray(xq)) {
111   return xq.map(interpolate);
112 } else {
113   return interpolate(xq);
114 }
115 }

116 /**
117 * Calculates the output of a bilinear function based on input value,
118 * midpoint, and mid-value.
119 * @param {number} x - The input value for the function.
120 * @param {number} midPoint - The midpoint of the function where the slope
121 * changes.
122 * @param {number} midValue - The value of the function at the midpoint.
123 * @returns {number} The output value of the bilinear function.
124 */
125 bilinearFunction(x, midPoint, midValue) {
126   var sign = 1;
127   if(x < 0) {
128     sign = -1;
129   }
130   x = Math.abs(x);
131   if(x <= midPoint) {
132     return sign * (midValue / midPoint) * x;
133   } else {
134     return sign * (((1 - midValue) / (1 - midPoint)) * (x - midPoint) +
135     midValue);
136   }
137 }
138 /**
139 * Generates a random number between a specified range.
140 * @param {Number} [min=0] The lower bound of the range.
141 * @param {Number} [max] The upper bound of the range.

```

```

141   * @returns {Number} A random number within the specified range.
142   */
143 random(min, max) {
144   if(isNaN(Number(max))) return Math.random();
145   if(isNaN(Number(min))) min = 0;
146   return Math.random() * (max - min) + min;
147 }
148
149 /**
150  * Generates a random integer within a specified range.
151  * @param {Number} [min=0] The lower bound of the range.
152  * @param {Number} [max] The upper bound of the range.
153  * @returns {Number} A random integer within the specified range.
154  */
155 randInt(min, max) {
156   if(isNaN(Number(max))) return NaN;
157   if(isNaN(Number(min))) min = 0;
158   return Math.floor(Math.random() * (max - min + 1) + min);
159 }
160
161 /**
162  * Computes the arc cosine of x, with the result in degrees.
163  * @param {Number} x The value to compute the arc cosine for.
164  * @returns {Number} The arc cosine of x in degrees.
165  */
166acosd(x) {
167   return this.jsl.env.math.dotMultiply(this.jsl.env.math.acos(x), this.r2d);
168 }
169
170 /**
171  * Computes the arc cotangent of x, with the result in degrees.
172  * @param {Number} x The value to compute the arc cotangent for.
173  * @returns {Number} The arc cotangent of x in degrees.
174  */
175acotd(x) {
176   return this.jsl.env.math.dotMultiply(this.jsl.env.math.acot(x), this.r2d);
177 }
178
179 /**
180  * Computes the arc cosecant of x, with the result in degrees.
181  * @param {Number} x The value to compute the arc cosecant for.
182  * @returns {Number} The arc cosecant of x in degrees.
183  */
184acsed(x) {
185   return this.jsl.env.math.dotMultiply(this.jsl.env.math.acsc(x), this.r2d);
186 }
187
188 /**
189  * Computes the arc secant of x, with the result in degrees.
190  * @param {Number} x The value to compute the arc secant for.
191  * @returns {Number} The arc secant of x in degrees.
192  */
193asecd(x) {
194   return this.jsl.env.math.dotMultiply(this.jsl.env.math.asec(x), this.r2d);
195 }
```

```

196
197 /**
198 * Computes the arc sine of x, with the result in degrees.
199 * @param {Number} x The value to compute the arc sine for.
200 * @returns {Number} The arc sine of x in degrees.
201 */
202 asind(x) {
203     return this.jsl.env.math.dotMultiply(this.jsl.env.math.asin(x), this.r2d);
204 }
205
206 /**
207 * Computes the arc tangent of x, with the result in degrees.
208 * @param {Number} x The value to compute the arc tangent for.
209 * @returns {Number} The arc tangent of x in degrees.
210 */
211 atan(x) {
212     return this.jsl.env.math.dotMultiply(this.jsl.env.math.atan(x), this.r2d);
213 }
214
215 /**
216 * Computes the arc tangent of the quotient of its arguments, with the
217 * result in degrees.
218 * @param {Number} y The y coordinate.
219 * @param {Number} x The x coordinate.
220 * @returns {Number} The arc tangent of y/x in degrees.
221 */
222 atan2d(y, x) {
223     return this.jsl.env.math.dotMultiply(this.jsl.env.math.atan2(y, x), this.r2d);
224 }
225
226 /**
227 * Computes the cosine of x, where x is in degrees.
228 * @param {Number} x The angle in degrees.
229 * @returns {Number} The cosine of x in degrees.
230 */
231 cosd(x) {
232     return this.jsl.env.math.cos(this.jsl.env.math.dotMultiply(x, this.d2r));
233 }
234
235 /**
236 * Computes the cotangent of x, where x is in degrees.
237 * @param {Number} x The angle in degrees.
238 * @returns {Number} The cotangent of x.
239 */
240 cotd(x) {
241     return this.jsl.env.math.cot(this.jsl.env.math.dotMultiply(x, this.d2r));
242 }
243
244 /**
245 * Computes the cosecant of x, where x is in degrees.
246 * @param {Number} x The angle in degrees.
247 * @returns {Number} The cosecant of x.
248 */
249 cscd(x) {

```

```

249     return this.jsl.env.math.csc(this.jsl.env.math.dotMultiply(x, this.d2r));
250 }
251
252 /**
253 * Computes the secant of x, where x is in degrees.
254 * @param {Number} x The angle in degrees.
255 * @returns {Number} The secant of x.
256 */
257 secd(x) {
258   return this.jsl.env.math.sec(this.jsl.env.math.dotMultiply(x, this.d2r));
259 }
260
261 /**
262 * Computes the sine of x, where x is in degrees.
263 * @param {Number} x The angle in degrees.
264 * @returns {Number} The sine of x.
265 */
266 sind(x) {
267   return this.jsl.env.math.sin(this.jsl.env.math.dotMultiply(x, this.d2r));
268 }
269
270 /**
271 * Computes the tangent of x, where x is in degrees.
272 * @param {Number} x The angle in degrees.
273 * @returns {Number} The tangent of x.
274 */
275 tand(x) {
276   return this.jsl.env.math.tan(this.jsl.env.math.dotMultiply(x, this.d2r));
277 }
278
279 /**
280 * Computes the characteristic polynomial of a matrix or the polynomial from
281 * roots.
282 * @param {Array} A - If 'A' is a matrix (2D array), computes its
283 * characteristic polynomial.
284 *           If 'A' is an array of roots, computes the polynomial
285 * with those roots.
286 * @returns {Array} - Coefficients of the resulting polynomial.
287 */
288 poly(A) {
289   if(A[0].length) {
290     return charpoly(A);
291   }
292
293   let p = [1];
294
295   for(const root of A) {
296     p = p.map((coef) => coef * -root)
297       .concat(0)
298       .map((coef, index, arr) => coef + (arr[index + 1] || 0));
299   }
300
301   return p;
302 }

```

```

301  /**
302   * Fits a polynomial of degree n to the given data points and returns the
303   * coefficients (highest degree first).
304   * @param {number[]} x - The array of x-values.
305   * @param {number[]} y - The array of y-values corresponding to each x-value
306   *
307   * @param {number} n - The degree of the polynomial to fit.
308   * @returns {number[]} The coefficients of the fitted polynomial in
309   * descending order.
310   */
311 polyfit(x, y, n) {
312   var p = new this.jsl.env.PolynomialRegression(x, y, n);
313   return p.coefficients.reverse(); // Reverse to match MATLAB's coefficient
314   * order
315 }
316 /**
317  * Evaluates a polynomial with given coefficients at specified x-values.
318  * @param {number[]} p - The coefficients of the polynomial in descending
319  * order.
320  * @param {number[]} x_in - The array of x-values at which to evaluate the
321  * polynomial.
322  * @returns {number[]} The resulting y-values after evaluating the
323  * polynomial at x_in.
324  */
325 polyval(p, x_in) {
326   var y_out = [];
327   y_out.length = x_in.length;
328   for(var k = 0; k < x_in.length; k++) {
329     var y = p[0];
330     for(var i = 1; i < p.length; i++) {
331       y = y * x_in[k] + p[i];
332     }
333     y_out[k] = y;
334   }
335   return y_out;
336 }
337 /**
338  * Computes the roots of a polynomial with the given coefficients.
339  * @param {number[]} p - Array of polynomial coefficients, ordered from
340  * highest degree to constant term.
341  * @returns {number[]} Array of roots (real or complex) of the polynomial.
342  */
343 roots(p) {
344   return this.jsl.env.native_module.roots(p);
345 }
346 /**
347  * Generates a string representation of a polynomial based on the provided
348  * coefficients and options.
349  * @param {number[]} p - An array of polynomial coefficients, ordered from
350  * highest degree to constant term.
351  * @param {Object} [opts] - Optional settings for the polynomial string.
352  * @param {string} [opts.x_symbol='x'] - The symbol to use for the variable

```

```

      x.
346  * @param {string} [opts.y_symbol='y'] - The symbol to use for the variable
347  * @param {number} [opts.precision=7] - The number of decimal places for
348  * coefficients.
349  * @param {string} [opts.lang] - The language format for the output ('tex',
350  * 'c', etc.).
351  * @returns {string} The formatted polynomial string.
352  */
353  polystr(p, opts) {
354    let degree = p.length - 1;
355    let x_symbol = 'x';
356    let y_symbol = 'y';
357    let precision = 7;
358    let lang;
359    if(opts) {
360      if(opts.hasOwnProperty('x_symbol')) {
361        x_symbol = opts.x_symbol;
362      }
363      if(opts.hasOwnProperty('y_symbol')) {
364        y_symbol = opts.y_symbol;
365      }
366      if(opts.hasOwnProperty('precision')) {
367        precision = opts.precision;
368      }
369      if(opts.hasOwnProperty('lang')) {
370        lang = opts.lang;
371      }
372    }
373    let str = y_symbol+ '= ';
374    for(let i = 0; i < p.length; i++) {
375      let current_power = degree - i;
376      let coef = p[i];
377      // Skip terms with zero coefficient
378      if(coef === 0) {
379        continue;
380      }
381      // Handle the sign
382      let sign_str = '';
383      if(coef > 0 && i !== 0) {
384        sign_str = '+' ;
385      } else if(coef < 0) {
386        sign_str = '-' ;
387        coef = -coef; // Convert to positive for display
388      }
389      // Form the term based on the power of x
390      let term_str;
391      if(current_power > 1) {
392        if(lang === 'tex') {
393          term_str = `${sign_str}\backslash\backslash num\{\${coef.toExponential(precision)}\}\backslash\backslash,
394          ${x_symbol}^{\${current_power}}`;

```

```

396     } else if(lang == 'c') {
397         term_str = '${sign_str}${coef.toExponential(precision)}*pow(${x_symbol}, ${current_power})';
398     } else {
399         term_str = '${sign_str}${coef.toExponential(precision)}*${x_symbol}^${current_power}';
400     }
401     } else if(current_power == 1) {
402         if(lang == 'tex') {
403             term_str = '${sign_str}\\num\\${coef.toExponential(precision)}\\';
404         } else {
405             term_str = '${sign_str}${coef.toExponential(precision)}*${x_symbol}';
406         }
407     } else {
408         if(lang == 'tex') {
409             term_str = '${sign_str}\\num\\${coef.toExponential(precision)}';
410         } else {
411             term_str = '${sign_str}${coef.toExponential(precision)}';
412         }
413     }
414 }
415
416 // Append the term to the polynomial string
417 str += term_str;
418 }
419
420 return str;
421 }
422
423 /**
424 * Generates a C language formatted string representation of a polynomial.
425 * @param {number[]} p - An array of polynomial coefficients, ordered from
426 * highest degree to constant term.
427 * @param {Object} [opts] - Optional settings for the polynomial string.
428 * @returns {string} The polynomial string formatted for C language.
429 */
430 polystrc(p, opts) {
431     if(opts) {
432         opts.lang = 'c';
433     } else {
434         opts = {lang: 'c'};
435     }
436     return polystr(p, opts);
437 }
438 /**
439 * Generates a LaTeX formatted string representation of a polynomial.
440 * @param {number[]} p - An array of polynomial coefficients, ordered from
441 * highest degree to constant term.
442 * @param {Object} [opts] - Optional settings for the polynomial string.
443 * @returns {string} The polynomial string formatted for LaTeX.
444 */
445 polystrtex(p, opts) {

```

```

445     if(opts) {
446         opts.lang = 'tex';
447     } else {
448         opts = {lang: 'tex'};
449     }
450     return polystr(p, opts);
451 }
452
453 /**
454 * Filters out spikes in a sequence by replacing sudden large changes with
455 * the previous value.
456 * @param {number[]} x - The input sequence of numbers.
457 * @param {number} dx_max - The maximum allowed difference between
458 * consecutive elements before considering it a spike.
459 * @param {number} n - The maximum number of consecutive spikes to correct.
460 * @returns {number[]} The filtered sequence with spikes removed.
461 */
462 spikeFilter(x, dx_max, n) {
463     var c = 0;
464     if(x.length > 1) {
465         for(var i = 1; i < x.length; i++) {
466             if(Math.abs(x[i]-x[i-1]) > dx_max && c < n){
467                 c = c+1;
468                 x[i] = x[i-1];
469             } else {
470                 c = 0;
471             }
472         }
473     }
474     return x;
475 }
476 /**
477 * Calculates the magnitude (absolute value) of a complex number or a real
478 * number.
479 * @param {number|Object} num - A real number or an object with 'real' and
480 * 'imag' properties.
481 * @returns {number} The magnitude of the number.
482 */
483 magnitude(num) {
484     if(typeof num === 'object' && num.real !== undefined && num.imag !==
485         undefined) {
486         return Math.hypot(num.real, num.imag);
487     } else {
488         return Math.abs(num);
489     }
490 }
491 /**
492 * Compares two numbers (real or complex) according to Octave's rules.
493 * @param {number|Object} a - First number to compare.
494 * @param {number|Object} b - Second number to compare.
495 * @returns {number} -1 if a < b, 1 if a > b, 0 if equal.

```

```

495     */
496     compareComplex(a, b) {
497         const mag_A = magnitude(a);
498         const mag_B = magnitude(b);
499
500         if(mag_A < mag_B) return -1;
501         if(mag_A > mag_B) return 1;
502
503         // Magnitudes are equal; compare real parts
504         const real_A = (typeof a === 'object') ? a.real : a;
505         const real_B = (typeof b === 'object') ? b.real : b;
506
507         if(real_A < real_B) return -1;
508         if(real_A > real_B) return 1;
509
510         // Real parts are equal; compare imaginary parts
511         const imag_A = (typeof a === 'object') ? a.imag : 0;
512         const imag_B = (typeof b === 'object') ? b.imag : 0;
513
514         if(imag_A < imag_B) return -1;
515         if(imag_A > imag_B) return 1;
516
517         // Numbers are equal
518         return 0;
519     }
520
521 /**
522 * Finds the minimum value in an array of numbers, which can include both
523 * real numbers and complex numbers.
524 * Complex numbers are represented as objects with 'real' and 'imag'
525 * properties.
526 * The comparison follows Octave's rules:
527 * 1. Compare magnitudes (absolute values) of the numbers.
528 * 2. If magnitudes are equal, compare real parts.
529 * 3. If real parts are equal, compare imaginary parts.
530 *
531 * @param {Array<number|Object>} arr - An array of numbers or complex number
532 * objects.
533 * @returns {number|Object} The minimum value found in the array.
534 * @throws {TypeError} If the input is not an array.
535 */
536 min(arr) {
537     if(!Array.isArray(arr)) {
538         this.jsl.env.error('@min: '+language.string(190));
539         return;
540     }
541     arr = arr.filter(num => !isNaN(num));
542
543     // Separate real numbers and complex numbers
544     const real_numbers = [];
545     const complex_numbers = [];
546     for(const num of arr) {
547         if(typeof num === 'object' && num.real !== undefined && num.imag !==
548             undefined) {
549             complex_numbers.push(num);
550         } else {
551             real_numbers.push(num);
552         }
553     }
554
555     if(complex_numbers.length) {
556         const minComplex = compareComplex(...complex_numbers);
557         if(minComplex < 0) {
558             return complex_numbers[0];
559         } else {
560             return compareComplex(minComplex, ...real_numbers);
561         }
562     } else {
563         return compareComplex(...real_numbers);
564     }
565 }

```

```

546     } else {
547         real_numbers.push(num);
548     }
549 }
550
551 // Find min among real numbers using existing function
552 let min_real = real_numbers.length > 0 ? this.jsl.env.math.min(
553     real_numbers) : null;
554
555 // Find min among complex numbers
556 let min_complex = complex_numbers.length > 0 ? complex_numbers[0] : null;
557 for(let i = 1; i < complex_numbers.length; i++) {
558     if(compareComplex(complex_numbers[i], min_complex) < 0) {
559         min_complex = complex_numbers[i];
560     }
561 }
562
563 // Compare min_real and min_complex
564 if(min_real !== null && min_complex !== null) {
565     return compareComplex(min_real, min_complex) < 0 ? min_real :
566         min_complex;
567 } else if(min_real !== null) {
568     return min_real;
569 } else {
570     return min_complex;
571 }
572 /**
573 * Finds the maximum value in an array of numbers, which can include both
574 * real numbers and complex numbers.
575 * Complex numbers are represented as objects with 'real' and 'imag'
576 * properties.
577 * The comparison follows Octave's rules:
578 * 1. Compare magnitudes (absolute values) of the numbers.
579 * 2. If magnitudes are equal, compare real parts.
580 * 3. If real parts are equal, compare imaginary parts.
581 *
582 * @param {Array<number|Object>} arr - An array of numbers or complex number
583 * objects.
584 * @returns {number|Object} The maximum value found in the array.
585 * @throws {TypeError} If the input is not an array.
586 */
587 max(arr) {
588     if(!Array.isArray(arr)) {
589         this.jsl.env.error('@max: '+language.string(190));
590         return;
591     }
592     arr = arr.filter(num => !isNaN(num));
593
594     // Separate real numbers and complex numbers
595     const real_numbers = [];
596     const complex_numbers = [];
597     for(const num of arr) {
598         if(typeof num === 'object' && num.real !== undefined && num.imag !==

```

```

      undefined) {
  596   complex_numbers.push(num);
  597 } else {
  598   real_numbers.push(num);
  599 }
 600 }

// Find max among real numbers using existing function
 602 let max_real = real_numbers.length > 0 ? this.jsl.env.math.max(
 603   real_numbers) : null;

 604 // Find max among complex numbers
 605 let max_complex = complex_numbers.length > 0 ? complex_numbers[0] : null;
 606 for(let i = 1; i < complex_numbers.length; i++) {
 607   if(compareComplex(complex_numbers[i], max_complex) > 0) {
 608     max_complex = complex_numbers[i];
 609   }
 610 }

 612 // Compare max_real and max_complex
 613 if(max_real !== null && max_complex !== null) {
 614   return compareComplex(max_real, max_complex) > 0 ? max_real :
 615     max_complex;
 616 } else if(max_real !== null) {
 617   return max_real;
 618 } else {
 619   return max_complex;
 620 }

 621 }

 622 /**
 623 * Extracts the real part of a number or an array of numbers.
 624 * Handles mixed inputs containing both real numbers and complex numbers.
 625 *
 626 * @param {number|Object|Array<number|Object>} input - A number, complex
 627 *   number object, or array thereof.
 628 * @returns {number|Array<number>} The real part(s) of the input.
 629 * @throws {TypeError} If the input is not a number, complex number object,
 630 *   or array.
 631 */
 632 real(input) {
 633   if(Array.isArray(input)) {
 634     return input.map(real);
 635   } else if(typeof input === 'object' && input !== null && 'real' in input
 636     && 'imag' in input) {
 637     return real(input.real);
 638   } else if(typeof input === 'number') {
 639     return input;
 640   } else {
 641     this.jsl.env.error('@real: '+language.string(192));
 642   }
 643 }

 644 /**
 645 * Extracts the imaginary part of a number or an array of numbers.

```

```

645  * Handles mixed inputs containing both real numbers and complex numbers.
646  *
647  * @param {number|Object|Array<number|Object>} input - A number, complex
648  * number object, or array thereof.
649  * @returns {number|Array<number>} The imaginary part(s) of the input.
650  * @throws {TypeError} If the input is not a number, complex number object,
651  * or array.
652  */
653  imag(input) {
654    if(Array.isArray(input)) {
655      return input.map(imag);
656    } else if(typeof input === 'object' && input !== null && 'real' in input
657      && 'imag' in input) {
658      return imag(input.imag);
659    } else if(typeof input === 'number') {
660      return 0;
661    } else {
662      this.jsl.env.error('@imag: '+language.string(192));
663    }
664    return false;
665  }
666  /**
667   * Performs cumulative trapezoidal integration on the provided data.
668   * @param {...any} args - Arguments required for cumulative trapezoidal
669   * integration.
670   * @returns {any} The result of the cumulative trapezoidal integration.
671   */
672  cumtrapz(...args) {
673    return this.jsl.env.native_module.cumtrapz(...args);
674  }
675  /**
676   * Performs trapezoidal integration on the provided data.
677   * @param {...any} args - Arguments required for trapezoidal integration.
678   * @returns {any} The result of the trapezoidal integration.
679   */
680  trapz(...args) {
681    return this.jsl.env.native_module.trapz(...args);
682  }
683  /**
684   * Compute the mean squared error (MSE) between two arrays.
685   * @param {Array<number>} A - The first array.
686   * @param {Array<number>} B - The second array.
687   * @returns {number} - The mean squared error between A and B.
688   * @throws {Error} - If A and B have different lengths or are not arrays.
689   */
690  mse(A, B) {
691    if(!Array.isArray(A) || !Array.isArray(B)) {
692      throw new Error("Both inputs must be arrays.");
693    }
694    if(A.length !== B.length) {
695      throw new Error("Input arrays must have the same length.");
696    }
697  }
698  /**
699   * Compute the dot product of two arrays.
700   * @param {Array<number>} A - The first array.
701   * @param {Array<number>} B - The second array.
702   * @returns {number} - The dot product of A and B.
703   */
704  dotProduct(A, B) {
705    if(!Array.isArray(A) || !Array.isArray(B)) {
706      throw new Error("Both inputs must be arrays.");
707    }
708    if(A.length !== B.length) {
709      throw new Error("Input arrays must have the same length.");
710    }
711    let result = 0;
712    for(let i = 0; i < A.length; i++) {
713      result += A[i] * B[i];
714    }
715    return result;
716  }
717  /**
718   * Compute the element-wise product of two arrays.
719   * @param {Array<number>} A - The first array.
720   * @param {Array<number>} B - The second array.
721   * @returns {Array<number>} - The element-wise product of A and B.
722   */
723  elementWiseProduct(A, B) {
724    if(!Array.isArray(A) || !Array.isArray(B)) {
725      throw new Error("Both inputs must be arrays.");
726    }
727    if(A.length !== B.length) {
728      throw new Error("Input arrays must have the same length.");
729    }
730    let result = [];
731    for(let i = 0; i < A.length; i++) {
732      result.push(A[i] * B[i]);
733    }
734    return result;
735  }
736  /**
737   * Compute the element-wise difference of two arrays.
738   * @param {Array<number>} A - The first array.
739   * @param {Array<number>} B - The second array.
740   * @returns {Array<number>} - The element-wise difference of A and B.
741   */
742  elementWiseDifference(A, B) {
743    if(!Array.isArray(A) || !Array.isArray(B)) {
744      throw new Error("Both inputs must be arrays.");
745    }
746    if(A.length !== B.length) {
747      throw new Error("Input arrays must have the same length.");
748    }
749    let result = [];
750    for(let i = 0; i < A.length; i++) {
751      result.push(A[i] - B[i]);
752    }
753    return result;
754  }
755  /**
756   * Compute the element-wise quotient of two arrays.
757   * @param {Array<number>} A - The first array.
758   * @param {Array<number>} B - The second array.
759   * @returns {Array<number>} - The element-wise quotient of A and B.
760   */
761  elementWiseQuotient(A, B) {
762    if(!Array.isArray(A) || !Array.isArray(B)) {
763      throw new Error("Both inputs must be arrays.");
764    }
765    if(A.length !== B.length) {
766      throw new Error("Input arrays must have the same length.");
767    }
768    let result = [];
769    for(let i = 0; i < A.length; i++) {
770      result.push(A[i] / B[i]);
771    }
772    return result;
773  }
774  /**
775   * Compute the element-wise remainder of two arrays.
776   * @param {Array<number>} A - The first array.
777   * @param {Array<number>} B - The second array.
778   * @returns {Array<number>} - The element-wise remainder of A and B.
779   */
780  elementWiseRemainder(A, B) {
781    if(!Array.isArray(A) || !Array.isArray(B)) {
782      throw new Error("Both inputs must be arrays.");
783    }
784    if(A.length !== B.length) {
785      throw new Error("Input arrays must have the same length.");
786    }
787    let result = [];
788    for(let i = 0; i < A.length; i++) {
789      result.push(A[i] % B[i]);
790    }
791    return result;
792  }
793  /**
794   * Compute the element-wise power of two arrays.
795   * @param {Array<number>} A - The first array.
796   * @param {Array<number>} B - The second array.
797   * @returns {Array<number>} - The element-wise power of A and B.
798   */
799  elementWisePower(A, B) {
800    if(!Array.isArray(A) || !Array.isArray(B)) {
801      throw new Error("Both inputs must be arrays.");
802    }
803    if(A.length !== B.length) {
804      throw new Error("Input arrays must have the same length.");
805    }
806    let result = [];
807    for(let i = 0; i < A.length; i++) {
808      result.push(Math.pow(A[i], B[i]));
809    }
810    return result;
811  }
812  /**
813   * Compute the element-wise logarithm of two arrays.
814   * @param {Array<number>} A - The first array.
815   * @param {Array<number>} B - The second array.
816   * @returns {Array<number>} - The element-wise logarithm of A and B.
817   */
818  elementWiseLogarithm(A, B) {
819    if(!Array.isArray(A) || !Array.isArray(B)) {
820      throw new Error("Both inputs must be arrays.");
821    }
822    if(A.length !== B.length) {
823      throw new Error("Input arrays must have the same length.");
824    }
825    let result = [];
826    for(let i = 0; i < A.length; i++) {
827      result.push(Math.log(A[i] / B[i]));
828    }
829    return result;
830  }
831  /**
832   * Compute the element-wise sine of two arrays.
833   * @param {Array<number>} A - The first array.
834   * @param {Array<number>} B - The second array.
835   * @returns {Array<number>} - The element-wise sine of A and B.
836   */
837  elementWiseSine(A, B) {
838    if(!Array.isArray(A) || !Array.isArray(B)) {
839      throw new Error("Both inputs must be arrays.");
840    }
841    if(A.length !== B.length) {
842      throw new Error("Input arrays must have the same length.");
843    }
844    let result = [];
845    for(let i = 0; i < A.length; i++) {
846      result.push(Math.sin(A[i] / B[i]));
847    }
848    return result;
849  }
850  /**
851   * Compute the element-wise cosine of two arrays.
852   * @param {Array<number>} A - The first array.
853   * @param {Array<number>} B - The second array.
854   * @returns {Array<number>} - The element-wise cosine of A and B.
855   */
856  elementWiseCosine(A, B) {
857    if(!Array.isArray(A) || !Array.isArray(B)) {
858      throw new Error("Both inputs must be arrays.");
859    }
860    if(A.length !== B.length) {
861      throw new Error("Input arrays must have the same length.");
862    }
863    let result = [];
864    for(let i = 0; i < A.length; i++) {
865      result.push(Math.cos(A[i] / B[i]));
866    }
867    return result;
868  }
869  /**
870   * Compute the element-wise tangent of two arrays.
871   * @param {Array<number>} A - The first array.
872   * @param {Array<number>} B - The second array.
873   * @returns {Array<number>} - The element-wise tangent of A and B.
874   */
875  elementWiseTangent(A, B) {
876    if(!Array.isArray(A) || !Array.isArray(B)) {
877      throw new Error("Both inputs must be arrays.");
878    }
879    if(A.length !== B.length) {
880      throw new Error("Input arrays must have the same length.");
881    }
882    let result = [];
883    for(let i = 0; i < A.length; i++) {
884      result.push(Math.tan(A[i] / B[i]));
885    }
886    return result;
887  }
888  /**
889   * Compute the element-wise cotangent of two arrays.
890   * @param {Array<number>} A - The first array.
891   * @param {Array<number>} B - The second array.
892   * @returns {Array<number>} - The element-wise cotangent of A and B.
893   */
894  elementWiseCotangent(A, B) {
895    if(!Array.isArray(A) || !Array.isArray(B)) {
896      throw new Error("Both inputs must be arrays.");
897    }
898    if(A.length !== B.length) {
899      throw new Error("Input arrays must have the same length.");
900    }
901    let result = [];
902    for(let i = 0; i < A.length; i++) {
903      result.push(Math.cot(A[i] / B[i]));
904    }
905    return result;
906  }
907  /**
908   * Compute the element-wise secant of two arrays.
909   * @param {Array<number>} A - The first array.
910   * @param {Array<number>} B - The second array.
911   * @returns {Array<number>} - The element-wise secant of A and B.
912   */
913  elementWiseSecant(A, B) {
914    if(!Array.isArray(A) || !Array.isArray(B)) {
915      throw new Error("Both inputs must be arrays.");
916    }
917    if(A.length !== B.length) {
918      throw new Error("Input arrays must have the same length.");
919    }
920    let result = [];
921    for(let i = 0; i < A.length; i++) {
922      result.push(Math.sec(A[i] / B[i]));
923    }
924    return result;
925  }
926  /**
927   * Compute the element-wise cosecant of two arrays.
928   * @param {Array<number>} A - The first array.
929   * @param {Array<number>} B - The second array.
930   * @returns {Array<number>} - The element-wise cosecant of A and B.
931   */
932  elementWiseCosecant(A, B) {
933    if(!Array.isArray(A) || !Array.isArray(B)) {
934      throw new Error("Both inputs must be arrays.");
935    }
936    if(A.length !== B.length) {
937      throw new Error("Input arrays must have the same length.");
938    }
939    let result = [];
940    for(let i = 0; i < A.length; i++) {
941      result.push(Math.cosec(A[i] / B[i]));
942    }
943    return result;
944  }
945  /**
946   * Compute the element-wise hyperbolic sine of two arrays.
947   * @param {Array<number>} A - The first array.
948   * @param {Array<number>} B - The second array.
949   * @returns {Array<number>} - The element-wise hyperbolic sine of A and B.
950   */
951  elementWiseHyperbolicSine(A, B) {
952    if(!Array.isArray(A) || !Array.isArray(B)) {
953      throw new Error("Both inputs must be arrays.");
954    }
955    if(A.length !== B.length) {
956      throw new Error("Input arrays must have the same length.");
957    }
958    let result = [];
959    for(let i = 0; i < A.length; i++) {
960      result.push(Math.sinh(A[i] / B[i]));
961    }
962    return result;
963  }
964  /**
965   * Compute the element-wise hyperbolic cosine of two arrays.
966   * @param {Array<number>} A - The first array.
967   * @param {Array<number>} B - The second array.
968   * @returns {Array<number>} - The element-wise hyperbolic cosine of A and B.
969   */
970  elementWiseHyperbolicCosine(A, B) {
971    if(!Array.isArray(A) || !Array.isArray(B)) {
972      throw new Error("Both inputs must be arrays.");
973    }
974    if(A.length !== B.length) {
975      throw new Error("Input arrays must have the same length.");
976    }
977    let result = [];
978    for(let i = 0; i < A.length; i++) {
979      result.push(Math.cosh(A[i] / B[i]));
980    }
981    return result;
982  }
983  /**
984   * Compute the element-wise hyperbolic tangent of two arrays.
985   * @param {Array<number>} A - The first array.
986   * @param {Array<number>} B - The second array.
987   * @returns {Array<number>} - The element-wise hyperbolic tangent of A and B.
988   */
989  elementWiseHyperbolicTangent(A, B) {
990    if(!Array.isArray(A) || !Array.isArray(B)) {
991      throw new Error("Both inputs must be arrays.");
992    }
993    if(A.length !== B.length) {
994      throw new Error("Input arrays must have the same length.");
995    }
996    let result = [];
997    for(let i = 0; i < A.length; i++) {
998      result.push(Math.tanh(A[i] / B[i]));
999    }
1000   return result;
1001 }
1002 
```

```

696     }
697
698     const n = A.length;
699     const mse = A.reduce((sum, a, i) => {
700       const diff = a - B[i];
701       return sum + diff * diff;
702     }, 0) / n;
703
704     return mse;
705   }
706
707 /**
708 * Calculates the coefficients of the characteristic polynomial of a square
709 * matrix.
710 * @param {number[][]} matrix - A square matrix (2D array) for which the
711 *   characteristic polynomial is computed.
712 * @returns {number[]} - An array of coefficients of the characteristic
713 *   polynomial.
714 * @throws {Error} - Throws an error if the input is not a square matrix or
715 *   has less than 2 rows/columns.
716 */
717 charpoly(matrix) {
718   const n = matrix.length;
719   const m = matrix[0].length;
720
721   if(n !== m || n < 1) {
722     throw new Error("Argument 'matrix' must be a square matrix.");
723   }
724
725   let p = ones(n+1);
726   let a1 = [...matrix];
727   for(let k = 2; k <= n; k++) {
728     p[k-1] = -1 * sum(this.jsl.env.math.diag(a1)) / (k - 1);
729     a1 = this.jsl.env.math.multiply(matrix, plus(a1,
730       this.jsl.env.math.multiply(p[k-1],
731       this.jsl.env.math.diag(ones(n)))));
732   }
733
734   p[n] = -1 * sum(this.jsl.env.math.diag(a1)) / n;
735
736   return p;
737 }
738 }
739
740 exports.PRDC_JSLAB_LIB_MATH = PRDC_JSLAB_LIB_MATH;

```

Listing 96 - math.js

```

1 /**
2 * @file JSLAB library math.js doc.
3 * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4 * PR-DC, Republic of Serbia

```

```

5   * info@pr-dc.com
6   */
7
8 /**
9  * Class for JSLAB math.js doc.
10 */
11 class PRDC_JSLAB_MATHJS_DOC {
12
13     constructor() {
14
15         /**
16          * Test whether a value is a Complex number.
17          * @name isComplex
18          * @kind function
19          * @param { *} x - The value to test.
20          * @returns { boolean } Returns true if 'x' is a Complex number, false
21          * otherwise.
22          * @memberof PRDC_JSLAB_MATHJS_DOC
23          */
24
25         /**
26          * Test whether a value is a BigNumber.
27          * @name isBigNumber
28          * @kind function
29          * @param { *} x - The value to test.
30          * @returns { boolean } Returns true if 'x' is a BigNumber, false
31          * otherwise.
32          * @memberof PRDC_JSLAB_MATHJS_DOC
33          */
34
35         /**
36          * Test whether a value is a Fraction.
37          * @name isFraction
38          * @kind function
39          * @param { *} x - The value to test.
40          * @returns { boolean } Returns true if 'x' is a Fraction, false otherwise
41
42         /**
43          * Test whether a value is a Unit.
44          * @name isUnit
45          * @kind function
46          * @param { *} x - The value to test.
47          * @returns { boolean } Returns true if 'x' is a Unit, false otherwise.
48          * @memberof PRDC_JSLAB_MATHJS_DOC
49          */
50
51         /**
52          * Test whether a value is a Matrix.
53          * @name isMatrix
54          * @kind function
55          * @param { *} x - The value to test.
56          * @returns { boolean } Returns true if 'x' is a Matrix, false otherwise.

```



```
57     * @memberof PRDC_JSLAB_MATHJS_DOC
58     */
59
60 /**
61  * Test whether a value is a collection (Array or Matrix).
62  * @name isCollection
63  * @kind function
64  * @param { *} x - The value to test.
65  * @returns { boolean } Returns true if 'x' is an Array or Matrix, false
66  * otherwise.
67  * @memberof PRDC_JSLAB_MATHJS_DOC
68  */
69
70 /**
71  * Test whether a value is a DenseMatrix.
72  * @name isDenseMatrix
73  * @kind function
74  * @param { *} x - The value to test.
75  * @returns { boolean } Returns true if 'x' is a DenseMatrix, false
76  * otherwise.
77  * @memberof PRDC_JSLAB_MATHJS_DOC
78  */
79
80 /**
81  * Test whether a value is a SparseMatrix.
82  * @name isSparseMatrix
83  * @kind function
84  * @param { *} x - The value to test.
85  * @returns { boolean } Returns true if 'x' is a SparseMatrix, false
86  * otherwise.
87  * @memberof PRDC_JSLAB_MATHJS_DOC
88  */
89
90 /**
91  * Test whether a value is a Range.
92  * @name isRange
93  * @kind function
94  * @param { *} x - The value to test.
95  * @returns { boolean } Returns true if 'x' is a Range, false otherwise.
96  * @memberof PRDC_JSLAB_MATHJS_DOC
97  */
98
99 /**
100  * Test whether a value is an Index.
101  * @name isIndex
102  * @kind function
103  * @param { *} x - The value to test.
104  * @returns { boolean } Returns true if 'x' is an Index, false otherwise.
105  * @memberof PRDC_JSLAB_MATHJS_DOC
106  */
107
108 /**
109  * Test whether a value is a boolean.
110  * @name isBoolean
111  * @kind function
```

```
109 * @param { * } x - The value to test.
110 * @returns { boolean } Returns true if 'x' is a boolean, false otherwise.
111 * @memberof PRDC_JSLAB_MATHJS_DOC
112 */
113
114 /**
115 * Test whether a value is a ResultSet.
116 * @name isResultSet
117 * @kind function
118 * @param { * } x - The value to test.
119 * @returns { boolean } Returns true if 'x' is a ResultSet, false
120 * otherwise.
121 * @memberof PRDC_JSLAB_MATHJS_DOC
122 */
123
124 /**
125 * Test whether a value is a Help object.
126 * @name isHelp
127 * @kind function
128 * @param { * } x - The value to test.
129 * @returns { boolean } Returns true if 'x' is a Help object, false
130 * otherwise.
131 * @memberof PRDC_JSLAB_MATHJS_DOC
132 */
133
134 /**
135 * Test whether a value is a Date.
136 * @name isDate
137 * @kind function
138 * @param { * } x - The value to test.
139 * @returns { boolean } Returns true if 'x' is a Date object, false
140 * otherwise.
141 * @memberof PRDC_JSLAB_MATHJS_DOC
142 */
143
144 /**
145 * Test whether a value is a RegExp.
146 * @name isRegExp
147 * @kind function
148 * @param { * } x - The value to test.
149 * @returns { boolean } Returns true if 'x' is a RegExp object, false
150 * otherwise.
151 * @memberof PRDC_JSLAB_MATHJS_DOC
152 */
153
154 /**
155 * Test whether a value is an AccessorNode.
156 * @name isAccessorNode
157 * @kind function
158 * @param { * } x - The value to test.
159 * @returns { boolean } Returns true if 'x' is an AccessorNode, false
160 * otherwise.
161 * @memberof PRDC_JSLAB_MATHJS_DOC
162 */
163
```

```
159  /**
160  * Test whether a value is an ArrayNode.
161  * @name isArrayNode
162  * @kind function
163  * @param { *} x - The value to test.
164  * @returns { boolean } Returns true if 'x' is an ArrayNode, false
165  * otherwise.
166  * @memberof PRDC_JSLAB_MATHJS_DOC
167  */
168 /**
169 * Test whether a value is an AssignmentNode.
170 * @name isAssignmentNode
171 * @kind function
172 * @param { *} x - The value to test.
173 * @returns { boolean } Returns true if 'x' is an AssignmentNode, false
174 * otherwise.
175 * @memberof PRDC_JSLAB_MATHJS_DOC
176 */
177 /**
178 * Test whether a value is a BlockNode.
179 * @name isBlockNode
180 * @kind function
181 * @param { *} x - The value to test.
182 * @returns { boolean } Returns true if 'x' is a BlockNode, false
183 * otherwise.
184 * @memberof PRDC_JSLAB_MATHJS_DOC
185 */
186 /**
187 * Test whether a value is a ConditionalNode.
188 * @name isConditionalNode
189 * @kind function
190 * @param { *} x - The value to test.
191 * @returns { boolean } Returns true if 'x' is a ConditionalNode, false
192 * otherwise.
193 * @memberof PRDC_JSLAB_MATHJS_DOC
194 */
195 /**
196 * Test whether a value is a ConstantNode.
197 * @name isConstantNode
198 * @kind function
199 * @param { *} x - The value to test.
200 * @returns { boolean } Returns true if 'x' is a ConstantNode, false
201 * otherwise.
202 * @memberof PRDC_JSLAB_MATHJS_DOC
203 */
204 /**
205 * Test whether a value is a FunctionAssignmentNode.
206 * @name isFunctionAssignmentNode
207 * @kind function
208 * @param { *} x - The value to test.
```

```
209 * @returns { boolean } Returns true if 'x' is a FunctionAssignmentNode ,  
210   false otherwise.  
211 * @memberof PRDC_JSLAB_MATHJS_DOC  
212 */  
213 /**  
214 * Test whether a value is a FunctionNode.  
215 * @name isFunctionNode  
216 * @kind function  
217 * @param { * } x - The value to test.  
218 * @returns { boolean } Returns true if 'x' is a FunctionNode , false  
219   otherwise.  
220 * @memberof PRDC_JSLAB_MATHJS_DOC  
221 */  
222 /**  
223 * Test whether a value is an IndexNode.  
224 * @name isIndexNode  
225 * @kind function  
226 * @param { * } x - The value to test.  
227 * @returns { boolean } Returns true if 'x' is an IndexNode , false  
228   otherwise.  
229 * @memberof PRDC_JSLAB_MATHJS_DOC  
230 */  
231 /**  
232 * Test whether a value is a Node.  
233 * @name isNode  
234 * @kind function  
235 * @param { * } x - The value to test.  
236 * @returns { boolean } Returns true if 'x' is a Node , false otherwise.  
237 * @memberof PRDC_JSLAB_MATHJS_DOC  
238 */  
239 /**  
240 * Test whether a value is an ObjectNode.  
241 * @name isObjectNode  
242 * @kind function  
243 * @param { * } x - The value to test.  
244 * @returns { boolean } Returns true if 'x' is an ObjectNode , false  
245   otherwise.  
246 * @memberof PRDC_JSLAB_MATHJS_DOC  
247 */  
248 /**  
249 * Test whether a value is an OperatorNode.  
250 * @name isOperatorNode  
251 * @kind function  
252 * @param { * } x - The value to test.  
253 * @returns { boolean } Returns true if 'x' is an OperatorNode , false  
254   otherwise.  
255 * @memberof PRDC_JSLAB_MATHJS_DOC  
256 */  
257 /**
```

```
259 * Test whether a value is a ParenthesisNode.
260 * @name isParenthesisNode
261 * @kind function
262 * @param { *} x - The value to test.
263 * @returns { boolean } Returns true if 'x' is a ParenthesisNode, false
264 * otherwise.
265 * @memberof PRDC_JSLAB_MATHJS_DOC
266 */
267 /**
268 * Test whether a value is a RangeNode.
269 * @name isRangeNode
270 * @kind function
271 * @param { *} x - The value to test.
272 * @returns { boolean } Returns true if 'x' is a RangeNode, false
273 * otherwise.
274 * @memberof PRDC_JSLAB_MATHJS_DOC
275 */
276 /**
277 * Test whether a value is a RelationalNode.
278 * @name isRelationalNode
279 * @kind function
280 * @param { *} x - The value to test.
281 * @returns { boolean } Returns true if 'x' is a RelationalNode, false
282 * otherwise.
283 * @memberof PRDC_JSLAB_MATHJS_DOC
284 */
285 /**
286 * Test whether a value is a SymbolNode.
287 * @name isSymbolNode
288 * @kind function
289 * @param { *} x - The value to test.
290 * @returns { boolean } Returns true if 'x' is a SymbolNode, false
291 * otherwise.
292 * @memberof PRDC_JSLAB_MATHJS_DOC
293 */
294 /**
295 * Test whether a value is a Chain.
296 * @name isChain
297 * @kind function
298 * @param { *} x - The value to test.
299 * @returns { boolean } Returns true if 'x' is a Chain, false otherwise.
300 * @memberof PRDC_JSLAB_MATHJS_DOC
301 */
302 /**
303 * Subscribe to an event.
304 * @name on
305 * @kind function
306 * @param { string } event - The event name to subscribe to.
307 * @param { function } callback - The callback function to execute when
308 * the event occurs.
```

```
309     * @returns { void }
310     * @memberof PRDC_JSLAB_MATHJS_DOC
311     */
312
313 /**
314  * Unsubscribe from an event.
315  * @name off
316  * @kind function
317  * @param { string } event - The event name to unsubscribe from.
318  * @param { function } [callback] - The callback function to remove.
319  * @returns { void }
320  * @memberof PRDC_JSLAB_MATHJS_DOC
321  */
322
323 /**
324  * Subscribe to an event once.
325  * @name once
326  * @kind function
327  * @param { string } event - The event name to subscribe to.
328  * @param { function } callback - The callback function to execute when
329  *   the event occurs.
330  * @returns { void }
331  * @memberof PRDC_JSLAB_MATHJS_DOC
332  */
333 /**
334  * Emit an event, triggering all bound callbacks.
335  * @name emit
336  * @kind function
337  * @param { string } event - The event name to emit.
338  * @param { * } [data] - The data to pass to the event handlers.
339  * @returns { void }
340  * @memberof PRDC_JSLAB_MATHJS_DOC
341  */
342
343 /**
344  * An object containing functions for parsing and evaluating expressions.
345  * @name expression
346  * @kind member
347  * @memberof PRDC_JSLAB_MATHJS_DOC
348  */
349
350 /**
351  * Import functions or constants into math.js.
352  * @name import
353  * @kind function
354  * @param { Object | Array } object - An object or array of objects with
355  *   functions or constants to import.
356  * @param { Object } [options] - Optional import options.
357  * @returns { void }
358  * @memberof PRDC_JSLAB_MATHJS_DOC
359  */
360 /**
361  * Create a new, isolated math.js instance.
```

```
362     * @name create
363     * @kind function
364     * @param { Object } [config] - Optional configuration options.
365     * @param { Function[] } [factories] - Optional list of factories to
366     * include.
367     * @returns { PRDC_JSLAB_MATHJS_DOC } Returns a new instance of math.js.
368     * @memberof PRDC_JSLAB_MATHJS_DOC
369     */
370
370 /**
371 * Factory function to create new functions.
372 * @name factory
373 * @kind function
374 * @param { string } name - The name of the function.
375 * @param { string[] } dependencies - Array of dependency names.
376 * @param { function } create - Function to create the new function.
377 * @returns { function } Returns the created function.
378 * @memberof PRDC_JSLAB_MATHJS_DOC
379 */
380
381 /**
382 * Calculate the absolute value of a number.
383 * @name abs
384 * @kind function
385 * @param { number | BigNumber | Complex | Array | Matrix } x - A number
386 * or array with numbers.
387 * @returns { number | BigNumber | Complex | Array | Matrix } Returns the
388 * absolute value of 'x'.
389 * @memberof PRDC_JSLAB_MATHJS_DOC
390 */
391
391 /**
392 * A node representing access to a property or index.
393 * @name AccessorNode
394 * @kind function
395 * @param { Node } object - The object being accessed.
396 * @param { IndexNode } index - The index used to access the object.
397 * @returns { AccessorNode } Returns a new AccessorNode.
398 * @memberof PRDC_JSLAB_MATHJS_DOC
399 */
400
400 /**
401 * Calculate the inverse cosine of a value.
402 * @name acos
403 * @kind function
404 * @param { number | Complex | Array | Matrix } x - Function input.
405 * @returns { number | Complex | Array | Matrix } The arc cosine of 'x'.
406 * @memberof PRDC_JSLAB_MATHJS_DOC
407 */
408
409 /**
410 * Calculate the inverse hyperbolic cosine of a value.
411 * @name acosh
412 * @kind function
413 * @param { number | Complex | Array | Matrix } x - Function input.
```

```

414   * @returns { number | Complex | Array | Matrix } The inverse hyperbolic
415   * cosine of 'x'.
416   * @memberof PRDC_JSLAB_MATHJS_DOC
417   */
418 
419 /**
420 * Calculate the inverse cotangent of a value.
421 * @name acot
422 * @kind function
423 * @param { number | Complex | Array | Matrix } x - Function input.
424 * @returns { number | Complex | Array | Matrix } The inverse cotangent of
425 * 'x'.
426 * @memberof PRDC_JSLAB_MATHJS_DOC
427 */
428 
429 /**
430 * Calculate the inverse hyperbolic cotangent of a value.
431 * @name acoth
432 * @kind function
433 * @param { number | Complex | Array | Matrix } x - Function input.
434 * @returns { number | Complex | Array | Matrix } The inverse hyperbolic
435 * cotangent of 'x'.
436 * @memberof PRDC_JSLAB_MATHJS_DOC
437 */
438 
439 /**
440 * Calculate the inverse cosecant of a value.
441 * @name acsc
442 * @kind function
443 * @param { number | Complex | Array | Matrix } x - Function input.
444 * @returns { number | Complex | Array | Matrix } The inverse cosecant of
445 * 'x'.
446 * @memberof PRDC_JSLAB_MATHJS_DOC
447 */
448 
449 /**
450 * Calculate the inverse hyperbolic cosecant of a value.
451 * @nameacsch
452 * @kind function
453 * @param { number | Complex | Array | Matrix } x - Function input.
454 * @returns { number | Complex | Array | Matrix } The inverse hyperbolic
455 * cosecant of 'x'.
456 * @memberof PRDC_JSLAB_MATHJS_DOC
457 */
458 
459 /**
460 * Add two scalar values, 'x + y'.
461 * @name addScalar
462 * @kind function
463 * @param { number | BigNumber | Fraction | Complex } x - First value to
464 * add.
465 * @param { number | BigNumber | Fraction | Complex } y - Second value to
466 * add.
467 * @returns { number | BigNumber | Fraction | Complex } Sum of 'x' and 'y'.

```

```

461   * @memberof PRDC_JSLAB_MATHJS_DOC
462   */
463
464  /**
465   * Logical AND. Returns true if both inputs are true.
466   * @name and
467   * @kind function
468   * @param { boolean | Array | Matrix } x - First input.
469   * @param { boolean | Array | Matrix } y - Second input.
470   * @returns { boolean | Array | Matrix } Returns true when both 'x' and 'y'
471   *       are true.
472   * @memberof PRDC_JSLAB_MATHJS_DOC
473   */
474
475 /**
476  * Apply a function to each entry in a matrix or array.
477  * @name apply
478  * @kind function
479  * @param { Array | Matrix } x - The input array or matrix.
480  * @param { number } dim - The dimension along which to apply the function
481  *
482  * @param { function } callback - The function to apply.
483  * @returns { Array | Matrix } The result of applying the function along
484  *       the specified dimension.
485  * @memberof PRDC_JSLAB_MATHJS_DOC
486  */
487
488 /**
489  * Calculate the argument of a complex number.
490  * @name arg
491  * @kind function
492  * @param { number | Complex | Array | Matrix } x - Function input.
493  * @returns { number | Array | Matrix } The argument of 'x'.
494  * @memberof PRDC_JSLAB_MATHJS_DOC
495  */
496
497 /**
498  * An array node representing an array in the expression tree.
499  * @name ArrayNode
500  * @kind function
501  * @param { Node[] } items - An array of nodes.
502  * @returns { ArrayNode } Returns a new ArrayNode.
503  * @memberof PRDC_JSLAB_MATHJS_DOC
504  */
505
506 /**
507  * Calculate the inverse secant of a value.
508  * @name asec
509  * @kind function
510  * @param { number | Complex | Array | Matrix } x - Function input.
511  * @returns { number | Complex | Array | Matrix } The inverse secant of 'x'
512  *       .
513  * @memberof PRDC_JSLAB_MATHJS_DOC
514  */

```

```

512 /**
513 * Calculate the inverse hyperbolic secant of a value.
514 * @name asech
515 * @kind function
516 * @param { number | Complex | Array | Matrix } x - Function input.
517 * @returns { number | Complex | Array | Matrix } The inverse hyperbolic
518 * secant of 'x'.
519 * @memberof PRDC_JSLAB_MATHJS_DOC
520 */
521 /**
522 * Calculate the inverse sine of a value.
523 * @name asin
524 * @kind function
525 * @param { number | Complex | Array | Matrix } x - Function input.
526 * @returns { number | Complex | Array | Matrix } The inverse sine of 'x'.
527 * @memberof PRDC_JSLAB_MATHJS_DOC
528 */
529 /**
530 * Calculate the inverse hyperbolic sine of a value.
531 * @name asinh
532 * @kind function
533 * @param { number | Complex | Array | Matrix } x - Function input.
534 * @returns { number | Complex | Array | Matrix } The inverse hyperbolic
535 * sine of 'x'.
536 * @memberof PRDC_JSLAB_MATHJS_DOC
537 */
538 /**
539 * An assignment node representing variable assignment.
540 * @name AssignmentNode
541 * @kind function
542 * @param { Node } object - The symbol or AccessorNode to assign to.
543 * @param { Node } value - The value to assign.
544 * @returns { AssignmentNode } Returns a new AssignmentNode.
545 * @memberof PRDC_JSLAB_MATHJS_DOC
546 */
547 */

548 /**
549 * Calculate the inverse tangent of a value.
550 * @name atan
551 * @kind function
552 * @param { number | Complex | Array | Matrix } x - Function input.
553 * @returns { number | Complex | Array | Matrix } The inverse tangent of 'x'.
554 * @memberof PRDC_JSLAB_MATHJS_DOC
555 */
556 */

557 /**
558 * Calculate the inverse tangent of 'y/x'.
559 * @name atan2
560 * @kind function
561 * @param { number | BigNumber | Array | Matrix } y - Dividend.
562 * @param { number | BigNumber | Array | Matrix } x - Divisor.
563 */

```

```

564   * @returns { number | BigNumber | Array | Matrix } The inverse tangent of
565   *          'y/x'.
566   * @memberof PRDC_JSLAB_MATHJS_DOC
567   */
568
569 /**
570 * Calculate the inverse hyperbolic tangent of a value.
571 * @name atanh
572 * @kind function
573 * @param { number | Complex | Array | Matrix } x - Function input.
574 * @returns { number | Complex | Array | Matrix } The inverse hyperbolic
575 *          tangent of 'x'.
576 * @memberof PRDC_JSLAB_MATHJS_DOC
577 */
578
579 /**
580 * Atomic mass constant , expressed in kg.
581 * @name atomicMass
582 * @kind member
583 * @memberof PRDC_JSLAB_MATHJS_DOC
584 * @type { number }
585 */
586
587 /**
588 * Avogadro's number , approximately '6.022e23' mol-1.
589 * @name avogadro
590 * @kind member
591 * @memberof PRDC_JSLAB_MATHJS_DOC
592 * @type { number }
593 */
594
595 /**
596 * Compute the Bell Numbers , 'B(n)'.
597 * @name bellNumbers
598 * @kind function
599 * @param { number } n - The input value.
600 * @returns { number } Returns the nth Bell number.
601 * @memberof PRDC_JSLAB_MATHJS_DOC
602 */
603
604 /**
605 * BigNumber constructor .
606 * @name BigNumber
607 * @kind function
608 * @param { number | string | BigNumber } value - The numeric value.
609 * @returns { BigNumber } Returns a new BigNumber instance.
610 * @memberof PRDC_JSLAB_MATHJS_DOC
611 */
612
613 /**
614 * Create a BigNumber with arbitrary precision .
615 * @name bignumber
616 * @kind function
617 * @param { number | string | BigNumber } value - The numeric value.
618 * @returns { BigNumber } Returns a BigNumber instance.

```

```
617     * @memberof PRDC_JSLAB_MATHJS_DOC
618     */
619
620     /**
621      * Format a number as binary.
622      * @name bin
623      * @kind function
624      * @param { number | BigNumber } n - The number to format.
625      * @returns { string } The binary representation of 'n'.
626      * @memberof PRDC_JSLAB_MATHJS_DOC
627      */
628
629     /**
630      * Bitwise AND operation.
631      * @name bitAnd
632      * @kind function
633      * @param { number | BigNumber | Array | Matrix } x - First value.
634      * @param { number | BigNumber | Array | Matrix } y - Second value.
635      * @returns { number | BigNumber | Array | Matrix } Result of 'x AND y'.
636      * @memberof PRDC_JSLAB_MATHJS_DOC
637      */
638
639     /**
640      * Bitwise NOT operation.
641      * @name bitNot
642      * @kind function
643      * @param { number | BigNumber | Array | Matrix } x - Input value.
644      * @returns { number | BigNumber | Array | Matrix } Result of 'NOT x'.
645      * @memberof PRDC_JSLAB_MATHJS_DOC
646      */
647
648     /**
649      * Bitwise OR operation.
650      * @name bitOr
651      * @kind function
652      * @param { number | BigNumber | Array | Matrix } x - First value.
653      * @param { number | BigNumber | Array | Matrix } y - Second value.
654      * @returns { number | BigNumber | Array | Matrix } Result of 'x OR y'.
655      * @memberof PRDC_JSLAB_MATHJS_DOC
656      */
657
658     /**
659      * Bitwise XOR operation.
660      * @name bitXor
661      * @kind function
662      * @param { number | BigNumber | Array | Matrix } x - First value.
663      * @param { number | BigNumber | Array | Matrix } y - Second value.
664      * @returns { number | BigNumber | Array | Matrix } Result of 'x XOR y'.
665      * @memberof PRDC_JSLAB_MATHJS_DOC
666      */
667
668     /**
669      * A node representing a block of statements.
670      * @name BlockNode
671      * @kind function
```



```
672     * @param { Object[] } blocks - An array of statements.  
673     * @returns { BlockNode } Returns a new BlockNode.  
674     * @memberof PRDC_JSLAB_MATHJS_DOC  
675     */  
676  
677     /**  
678      * The Bohr magneton in units of 'J/T'.  
679      * @name bohrMagneton  
680      * @kind member  
681      * @memberof PRDC_JSLAB_MATHJS_DOC  
682      * @type { number }  
683      */  
684  
685     /**  
686      * The Bohr radius in meters.  
687      * @name bohrRadius  
688      * @kind member  
689      * @memberof PRDC_JSLAB_MATHJS_DOC  
690      * @type { number }  
691      */  
692  
693     /**  
694      * The Boltzmann constant in 'J/K'.  
695      * @name boltzmann  
696      * @kind member  
697      * @memberof PRDC_JSLAB_MATHJS_DOC  
698      * @type { number }  
699      */  
700  
701     /**  
702      * Parse a value into a boolean.  
703      * @name boolean  
704      * @kind function  
705      * @param { * } x - The value to parse.  
706      * @returns { boolean } The parsed boolean value.  
707      * @memberof PRDC_JSLAB_MATHJS_DOC  
708      */  
709  
710     /**  
711      * The Catalan's constant.  
712      * @name catalan  
713      * @kind member  
714      * @memberof PRDC_JSLAB_MATHJS_DOC  
715      * @type { number }  
716      */  
717  
718     /**  
719      * Calculate the cube root of a value.  
720      * @name cbrt  
721      * @kind function  
722      * @param { number | BigNumber | Complex | Array | Matrix } x - Function  
    input.  
723      * @returns { number | BigNumber | Complex | Array | Matrix } The cube  
    root of 'x'.  
724      * @memberof PRDC_JSLAB_MATHJS_DOC
```

```
725 */  
726  
727 /**  
728 * Round a value towards plus infinity.  
729 * @name ceil  
730 * @kind function  
731 * @param { number | BigNumber | Array | Matrix } x - Input value.  
732 * @returns { number | BigNumber | Array | Matrix } The rounded value.  
733 * @memberof PRDC_JSLAB_MATHJS_DOC  
734 */  
735  
736 /**  
737 * Create a chained operation , allowing to chain methods.  
738 * @name chain  
739 * @kind function  
740 * @param { * } value - The initial value of the chain.  
741 * @returns { Chain } A chain object.  
742 * @memberof PRDC_JSLAB_MATHJS_DOC  
743 */  
744  
745 /**  
746 * Chain constructor.  
747 * @name Chain  
748 * @kind function  
749 * @param { * } value - The initial value.  
750 * @returns { Chain } Returns a new Chain instance.  
751 * @memberof PRDC_JSLAB_MATHJS_DOC  
752 */  
753  
754 /**  
755 * Classical electron radius in meters.  
756 * @name classicalElectronRadius  
757 * @kind member  
758 * @memberof PRDC_JSLAB_MATHJS_DOC  
759 * @type { number }  
760 */  
761  
762 /**  
763 * Calculate the number of combinations of n items taken k at a time.  
764 * @name combinations  
765 * @kind function  
766 * @param { number | BigNumber } n - Total number of items.  
767 * @param { number | BigNumber } k - Number of items to choose.  
768 * @returns { number | BigNumber } Number of possible combinations.  
769 * @memberof PRDC_JSLAB_MATHJS_DOC  
770 */  
771  
772 /**  
773 * Calculate the number of combinations with replacement of n items taken  
774 * k at a time.  
775 * @name combinationsWithRep  
776 * @kind function  
777 * @param { number | BigNumber } n - Total number of items.  
778 * @param { number | BigNumber } k - Number of items to choose.  
779 * @returns { number | BigNumber } Number of possible combinations with
```

```

    replacement .
779  * @memberof PRDC_JSLAB_MATHJS_DOC
780  */
781
782 /**
783  * Compare two values numerically .
784  * @name compare
785  * @kind function
786  * @param { number | BigNumber | Fraction | Complex | Unit | string | |
787  *         Array | Matrix } x - First value to compare .
788  * @param { number | BigNumber | Fraction | Complex | Unit | string | |
789  *         Array | Matrix } y - Second value to compare .
790  * @returns { number | BigNumber | Fraction | Complex | Unit | string | |
791  *          Array | Matrix } Returns 1 when x > y , -1 when x < y , and 0 when x ==
792  *          y .
793  * @memberof PRDC_JSLAB_MATHJS_DOC
794  */
795
796 /**
797  * Compare two strings using natural order .
798  * @name compareNatural
799  * @kind function
800  * @param { string } x - First string to compare .
801  * @param { string } y - Second string to compare .
802  * @returns { number } Returns 1 when x > y , -1 when x < y , and 0 when x ==
803  *          y .
804  * @memberof PRDC_JSLAB_MATHJS_DOC
805  */
806
807 /**
808  * Compare two strings lexicographically .
809  * @name compareText
810  * @kind function
811  * @param { string } x - First string to compare .
812  * @param { string } y - Second string to compare .
813  * @returns { number } Returns 1 when x > y , -1 when x < y , and 0 when x ==
814  *          y .
815  * @memberof PRDC_JSLAB_MATHJS_DOC
816  */
817
818 /**
819  * Compile an expression into a compiled function for faster evaluation .
820  * @name compile
821  * @kind function
822  * @param { string | Object } expr - The expression to compile .
823  * @returns { Object } A compiled expression that can be evaluated with ‘
824  * eval ’ .
825  * @memberof PRDC_JSLAB_MATHJS_DOC
826  */
827
828 /**
829  * Create a complex number .
830  * @name complex
831  * @kind function
832  * @param { number | string | Complex } [ re ] - Real part or a string

```

```

      representation .
826  * @param { number } [im] - Imaginary part .
827  * @returns { Complex } Returns a Complex number .
828  * @memberof PRDC_JSLAB_MATHJS_DOC
829  */
830
831 /**
832  * Complex number constructor .
833  * @name Complex
834  * @kind function
835  * @param { number | string | Complex } [re] - Real part or a string
836  * representation .
837  * @param { number } [im] - Imaginary part .
838  * @returns { Complex } A new Complex number .
839  * @memberof PRDC_JSLAB_MATHJS_DOC
840  */
841 /**
842  * Calculate the composition count of n into k parts .
843  * @name composition
844  * @kind function
845  * @param { number | BigNumber } n - Total number of items .
846  * @param { number | BigNumber } k - Number of parts .
847  * @returns { number | BigNumber } Number of compositions .
848  * @memberof PRDC_JSLAB_MATHJS_DOC
849  */
850
851 /**
852  * Concatenate matrices or arrays along a specified dimension .
853  * @name concat
854  * @kind function
855  * @param { Array | Matrix } a - First array or matrix .
856  * @param { ...Array | ...Matrix } b - Other arrays or matrices .
857  * @param { number | BigNumber } [dim=0] - Dimension along which to
858  * concatenate .
859  * @returns { Array | Matrix } Concatenated array or matrix .
860  * @memberof PRDC_JSLAB_MATHJS_DOC
861  */
862 /**
863  * A node representing a conditional expression .
864  * @name ConditionalNode
865  * @kind function
866  * @param { Node } condition - The condition expression .
867  * @param { Node } trueExpr - Expression to evaluate when condition is
868  * true .
869  * @param { Node } falseExpr - Expression to evaluate when condition is
870  * false .
871  * @returns { ConditionalNode } A new ConditionalNode instance .
872  * @memberof PRDC_JSLAB_MATHJS_DOC
873  */
874 /**
875  * Conductance quantum in Siemens .
876  * @name conductanceQuantum

```



```

927 * Calculate the hyperbolic cotangent of a value.
928 * @name coth
929 * @kind function
930 * @param { number | Complex | Array | Matrix } x - Function input.
931 * @returns { number | Complex | Array | Matrix } The hyperbolic cotangent
932 *          of x.
933 * @memberof PRDC_JSLAB_MATHJS_DOC
934 */
935 /**
936 * Coulomb's constant in Nm^2/C^2.
937 * @name coulomb
938 * @kind member
939 * @memberof PRDC_JSLAB_MATHJS_DOC
940 * @type { number }
941 */
942 /**
943 * Count the number of elements in a matrix or array.
944 * @name count
945 * @kind function
946 * @param { Array | Matrix } x - The input array or matrix.
947 * @returns { number } The number of elements.
948 * @memberof PRDC_JSLAB_MATHJS_DOC
949 */
950 /**
951 * Create a user-defined unit and register it with the Unit system.
952 * @name createUnit
953 * @kind function
954 * @param { string } name - The name of the new unit.
955 * @param { string | Object } definition - Definition of the unit.
956 * @param { Object } [options] - Configuration options.
957 * @returns { Unit } The created unit.
958 * @memberof PRDC_JSLAB_MATHJS_DOC
959 */
960 /**
961 * Calculate the cosecant of a value.
962 * @name csc
963 * @kind function
964 * @param { number | Complex | Unit | Array | Matrix } x - Function input.
965 * @returns { number | Complex | Array | Matrix } The cosecant of x.
966 * @memberof PRDC_JSLAB_MATHJS_DOC
967 */
968 /**
969 * Calculate the hyperbolic cosecant of a value.
970 * @name csch
971 * @kind function
972 * @param { number | Complex | Array | Matrix } x - Function input.
973 * @returns { number | Complex | Array | Matrix } The hyperbolic cosecant
974 *          of x.
975 * @memberof PRDC_JSLAB_MATHJS_DOC
976 */
977 /**
978 * Calculate the hyperbolic cosecant of a value.
979 * @name csch

```

```

980
981  /**
982   * Compute the conjugate transpose of a matrix.
983   * @name ctranspose
984   * @kind function
985   * @param { Array | Matrix } x - The matrix to transpose.
986   * @returns { Array | Matrix } The conjugate transpose of x.
987   * @memberof PRDC_JSLAB_MATHJS_DOC
988   */
989
990 /**
991  * Compute the cube of a value.
992  * @name cube
993  * @kind function
994  * @param { number | BigNumber | Complex | Unit | Array | Matrix } x -
995   * Input value.
996  * @returns { number | BigNumber | Complex | Unit | Array | Matrix } The
997   * cube of x.
998  * @memberof PRDC_JSLAB_MATHJS_DOC
999  */
1000 /**
1001  * Compute the cumulative sum of a matrix or array.
1002  * @name cumsum
1003  * @kind function
1004  * @param { Array | Matrix } x - The input array or matrix.
1005  * @param { number | BigNumber } [dim] - Dimension along which to
1006   * calculate.
1007  * @returns { Array | Matrix } The cumulative sum.
1008  * @memberof PRDC_JSLAB_MATHJS_DOC
1009  */
1010 /**
1011  * Test element-wise whether two values are equal.
1012  * @name deepEqual
1013  * @kind function
1014  * @param { * } x - First value to compare.
1015  * @param { * } y - Second value to compare.
1016  * @returns { boolean } Returns true if x and y are deep equal.
1017  * @memberof PRDC_JSLAB_MATHJS_DOC
1018  */
1019 /**
1020  * Dense matrix constructor.
1021  * @name DenseMatrix
1022  * @kind function
1023  * @param { Array } data - The data for the matrix.
1024  * @returns { DenseMatrix } A new DenseMatrix instance.
1025  * @memberof PRDC_JSLAB_MATHJS_DOC
1026  */
1027 /**
1028  * Take the derivative of an expression.
1029  * @name derivative
1030  * @kind function
1031

```

```

1032   * @param { string | Node } expr - The expression to differentiate.
1033   * @param { string | Node } variable - The variable with respect to which
1034     to differentiate.
1035   * @param { Object } [options] - Optional options object.
1036   * @returns { Node } The derivative of the expression.
1037   * @memberof PRDC_JSLAB_MATHJS_DOC
1038   */
1039
1040 /**
1041  * Deuteron mass in kilograms.
1042  * @name deuteronMass
1043  * @kind member
1044  * @memberof PRDC_JSLAB_MATHJS_DOC
1045  * @type { number }
1046  */
1047 /**
1048  * Calculate the differences between adjacent values in a matrix or array.
1049  * @name diff
1050  * @kind function
1051  * @param { Array | Matrix } x - Input array or matrix.
1052  * @param { number } [dim=0] - Dimension along which to calculate the
1053    difference.
1054  * @returns { Array | Matrix } The differences.
1055  * @memberof PRDC_JSLAB_MATHJS_DOC
1056  */
1057 /**
1058  * Divide two scalar values, x / y.
1059  * @name divideScalar
1060  * @kind function
1061  * @param { number | BigNumber | Fraction | Complex } x - Numerator.
1062  * @param { number | BigNumber | Fraction | Complex } y - Denominator.
1063  * @returns { number | BigNumber | Fraction | Complex } The result of
1064    division.
1065  * @memberof PRDC_JSLAB_MATHJS_DOC
1066  */
1067 /**
1068  * Divide two matrices element-wise.
1069  * @name dotDivide
1070  * @kind function
1071  * @param { Array | Matrix } x - Numerator matrix.
1072  * @param { Array | Matrix } y - Denominator matrix.
1073  * @returns { Array | Matrix } The element-wise division.
1074  * @memberof PRDC_JSLAB_MATHJS_DOC
1075  */
1076 /**
1077  * Multiply two matrices element-wise.
1078  * @name dotMultiply
1079  * @kind function
1080  * @param { Array | Matrix } x - First matrix.
1081  * @param { Array | Matrix } y - Second matrix.
1082  * @returns { Array | Matrix } The element-wise multiplication.
1083

```



```
1084     * @memberof PRDC_JSLAB_MATHJS_DOC
1085     */
1086
1087 /**
1088 * Exponentiate two matrices element-wise.
1089 * @name dotPow
1090 * @kind function
1091 * @param { Array | Matrix } x - Base matrix.
1092 * @param { Array | Matrix } y - Exponent matrix.
1093 * @returns { Array | Matrix } The element-wise exponentiation.
1094 * @memberof PRDC_JSLAB_MATHJS_DOC
1095 */
1096
1097 /**
1098 * Euler's number, the base of natural logarithms.
1099 * @name e
1100 * @kind member
1101 * @memberof PRDC_JSLAB_MATHJS_DOC
1102 * @type { number }
1103 */
1104
1105 /**
1106 * Efimov factor.
1107 * @name efimovFactor
1108 * @kind member
1109 * @memberof PRDC_JSLAB_MATHJS_DOC
1110 * @type { number }
1111 */
1112
1113 /**
1114 * Calculate eigenvalues and eigenvectors of a matrix.
1115 * @name eigs
1116 * @kind function
1117 * @param { Array | Matrix } x - A square matrix.
1118 * @returns { Object } An object containing eigenvalues and eigenvectors.
1119 * @memberof PRDC_JSLAB_MATHJS_DOC
1120 */
1121
1122 /**
1123 * Electric constant (vacuum permittivity) in F/m.
1124 * @name electricConstant
1125 * @kind member
1126 * @memberof PRDC_JSLAB_MATHJS_DOC
1127 * @type { number }
1128 */
1129
1130 /**
1131 * Electron mass in kilograms.
1132 * @name electronMass
1133 * @kind member
1134 * @memberof PRDC_JSLAB_MATHJS_DOC
1135 * @type { number }
1136 */
1137
1138 /**
```

```
1139 * Elementary charge in coulombs.
1140 * @name elementaryCharge
1141 * @kind member
1142 * @memberof PRDC_JSLAB_MATHJS_DOC
1143 * @type { number }
1144 */
1145
1146 /**
1147 * Test whether two values are equal.
1148 * @name equal
1149 * @kind function
1150 * @param { * } x - First value to compare.
1151 * @param { * } y - Second value to compare.
1152 * @returns { boolean } Returns true if x equals y.
1153 * @memberof PRDC_JSLAB_MATHJS_DOC
1154 */
1155
1156 /**
1157 * Test whether two scalar values are equal.
1158 * @name equalScalar
1159 * @kind function
1160 * @param { number | BigNumber | Fraction | Complex } x - First value.
1161 * @param { number | BigNumber | Fraction | Complex } y - Second value.
1162 * @returns { boolean } Returns true if x equals y.
1163 * @memberof PRDC_JSLAB_MATHJS_DOC
1164 */
1165
1166 /**
1167 * Test whether two strings are equal.
1168 * @name equalText
1169 * @kind function
1170 * @param { string } x - First string.
1171 * @param { string } y - Second string.
1172 * @returns { boolean } Returns true if x equals y.
1173 * @memberof PRDC_JSLAB_MATHJS_DOC
1174 */
1175
1176 /**
1177 * Calculate the error function of a value.
1178 * @name erf
1179 * @kind function
1180 * @param { number | BigNumber | Complex | Array | Matrix } x - Input
1181 * value.
1182 * @returns { number | BigNumber | Complex | Array | Matrix } The error
1183 * function evaluated at x.
1184 * @memberof PRDC_JSLAB_MATHJS_DOC
1185 */
1186 /**
1187 * Calculate the exponential of a value.
1188 * @name exp
1189 * @kind function
1190 * @param { number | BigNumber | Complex | Array | Matrix } x - Exponent.
1191 * @returns { number | BigNumber | Complex | Array | Matrix } The
1192 * exponential of x.
```

```

1191   * @memberof PRDC_JSLAB_MATHJS_DOC
1192   */
1193
1194 /**
1195  * Calculate exp(x) - 1.
1196  * @name expm1
1197  * @kind function
1198  * @param { number | BigNumber | Complex } x - Input value.
1199  * @returns { number | BigNumber | Complex } The result of exp(x) - 1.
1200  * @memberof PRDC_JSLAB_MATHJS_DOC
1201  */
1202
1203 /**
1204  * Calculate the factorial of a value.
1205  * @name factorial
1206  * @kind function
1207  * @param { number | BigNumber | Array | Matrix } n - A non-negative
1208  * integer.
1209  * @returns { number | BigNumber | Array | Matrix } The factorial of n.
1210  * @memberof PRDC_JSLAB_MATHJS_DOC
1211  */
1212
1213 /**
1214  * Boolean value false.
1215  * @name false
1216  * @kind member
1217  * @memberof PRDC_JSLAB_MATHJS_DOC
1218  * @type { boolean }
1219  */
1220
1221 /**
1222  * Faraday constant in C/mol.
1223  * @name faraday
1224  * @kind member
1225  * @memberof PRDC_JSLAB_MATHJS_DOC
1226  * @type { number }
1227  */
1228
1229 /**
1230  * Fermi coupling constant in GeV^-2.
1231  * @name fermiCoupling
1232  * @kind member
1233  * @memberof PRDC_JSLAB_MATHJS_DOC
1234  * @type { number }
1235  */
1236
1237 /**
1238  * Compute the Fast Fourier Transform of a matrix or array.
1239  * @name fft
1240  * @kind function
1241  * @param { Array | Matrix } x - Input array or matrix.
1242  * @returns { Array | Matrix } The FFT of x.
1243  * @memberof PRDC_JSLAB_MATHJS_DOC
1244  */

```

```
1245 /**
1246 * Fibonacci heap data structure.
1247 * @name FibonacciHeap
1248 * @kind function
1249 * @param { function } [compare] - Comparison function.
1250 * @returns { FibonacciHeap } A new FibonacciHeap instance.
1251 * @memberof PRDC_JSLAB_MATHJS_DOC
1252 */
1253
1254 /**
1255 * Filter the items in an array or matrix.
1256 * @name filter
1257 * @kind function
1258 * @param { Array | Matrix } x - The input array or matrix.
1259 * @param { function } test - The test function.
1260 * @returns { Array | Matrix } The filtered array or matrix.
1261 * @memberof PRDC_JSLAB_MATHJS_DOC
1262 */
1263
1264 /**
1265 * Fine-structure constant.
1266 * @name fineStructure
1267 * @kind member
1268 * @memberof PRDC_JSLAB_MATHJS_DOC
1269 * @type { number }
1270 */
1271
1272 /**
1273 * First radiation constant in Wm^2.
1274 * @name firstRadiation
1275 * @kind member
1276 * @memberof PRDC_JSLAB_MATHJS_DOC
1277 * @type { number }
1278 */
1279
1280 /**
1281 * Round a value towards zero.
1282 * @name fix
1283 * @kind function
1284 * @param { number | BigNumber | Array | Matrix } x - Input value.
1285 * @returns { number | BigNumber | Array | Matrix } The rounded value.
1286 * @memberof PRDC_JSLAB_MATHJS_DOC
1287 */
1288
1289 /**
1290 * Flatten a multi-dimensional array or matrix into a single dimension.
1291 * @name flatten
1292 * @kind function
1293 * @param { Array | Matrix } x - The input array or matrix.
1294 * @returns { Array | Matrix } The flattened array or matrix.
1295 * @memberof PRDC_JSLAB_MATHJS_DOC
1296 */
1297
1298 /**
1299 * Round a value towards negative infinity.
```

```
1300 * @name floor
1301 * @kind function
1302 * @param { number | BigNumber | Array | Matrix } x - Input value.
1303 * @returns { number | BigNumber | Array | Matrix } The rounded value.
1304 * @memberof PRDC_JSLAB_MATHJS_DOC
1305 */
1306
1307 /**
1308 * Iterate over each element of a matrix or array.
1309 * @name forEach
1310 * @kind function
1311 * @param { Array | Matrix } x - The input array or matrix.
1312 * @param { function } callback - The function to execute on each element.
1313 * @returns { void }
1314 * @memberof PRDC_JSLAB_MATHJS_DOC
1315 */
1316
1317 /**
1318 * Format a value for display.
1319 * @name format
1320 * @kind function
1321 * @param { * } value - The value to format.
1322 * @param { Object | function } [options] - Formatting options or custom
1323 * function.
1324 * @returns { string } The formatted value.
1325 * @memberof PRDC_JSLAB_MATHJS_DOC
1326 */
1327 /**
1328 * Create a fraction.
1329 * @name fraction
1330 * @kind function
1331 * @param { number | string | Fraction } numerator - Numerator.
1332 * @param { number | string } [denominator] - Denominator.
1333 * @returns { Fraction } A new Fraction instance.
1334 * @memberof PRDC_JSLAB_MATHJS_DOC
1335 */
1336
1337 /**
1338 * Fraction constructor.
1339 * @name Fraction
1340 * @kind function
1341 * @param { number | string | Fraction } numerator - Numerator.
1342 * @param { number | string } [denominator] - Denominator.
1343 * @returns { Fraction } A new Fraction instance.
1344 * @memberof PRDC_JSLAB_MATHJS_DOC
1345 */
1346
1347 /**
1348 * A node representing a function assignment in the expression tree.
1349 * @name FunctionAssignmentNode
1350 * @kind function
1351 * @param { string } name - The name of the function being assigned.
1352 * @param { string[] } params - An array of parameter names.
1353 * @param { Node } expr - The function expression.
```

```
1354     * @returns { FunctionAssignmentNode } A new FunctionAssignmentNode
1355         instance.
1356     * @memberof PRDC_JSLAB_MATHJS_DOC
1357     */
1358 
1359     /**
1360      * A node representing a function call in the expression tree.
1361      * @name FunctionNode
1362      * @kind function
1363      * @param { string | SymbolNode } name - The name of the function or a
1364          SymbolNode.
1365      * @param { Node[] } args - An array of argument nodes.
1366      * @returns { FunctionNode } A new FunctionNode instance.
1367      * @memberof PRDC_JSLAB_MATHJS_DOC
1368      */
1369 
1370     /**
1371      * Calculate the gamma function of a value.
1372      * @name gamma
1373      * @kind function
1374      * @param { number | BigNumber | Complex | Array | Matrix } n - The input
1375          value.
1376      * @returns { number | BigNumber | Complex | Array | Matrix } The gamma of
1377          n.
1378      * @memberof PRDC_JSLAB_MATHJS_DOC
1379      */
1380 
1381     /**
1382      * The molar gas constant , in units of J/(molK).
1383      * @name gasConstant
1384      * @kind member
1385      * @memberof PRDC_JSLAB_MATHJS_DOC
1386      * @type { number }
1387      */
1388 
1389     /**
1390      * Compute the greatest common divisor of two or more values.
1391      * @name gcd
1392      * @kind function
1393      * @param { ...number | ...BigNumber | Array | Matrix } args - Two or more
1394          integer numbers.
1395      * @returns { number | BigNumber | Array | Matrix } The greatest common
1396          divisor.
1397      * @memberof PRDC_JSLAB_MATHJS_DOC
1398      */
1399 
1400     /**
1401      * Get the data type of a matrix.
1402      * @name getMatrixDataType
1403      * @kind function
1404      * @param { Matrix } matrix - The input matrix.
1405      * @returns { string } The data type of the matrix elements.
1406      * @memberof PRDC_JSLAB_MATHJS_DOC
1407      */
1408 
```

```

1403 /**
1404 * Newtonian constant of gravitation , in m^3/(kg s^2) .
1405 * @name gravitationConstant
1406 * @kind member
1407 * @memberof PRDC_JSLAB_MATHJS_DOC
1408 * @type { number }
1409 */
1410
1411 /**
1412 * Acceleration due to gravity on Earth , in m/s ^2.
1413 * @name gravity
1414 * @kind member
1415 * @memberof PRDC_JSLAB_MATHJS_DOC
1416 * @type { number }
1417 */
1418
1419 /**
1420 * Hartree energy , in joules .
1421 * @name hartreeEnergy
1422 * @kind member
1423 * @memberof PRDC_JSLAB_MATHJS_DOC
1424 * @type { number }
1425 */
1426
1427 /**
1428 * Test whether a value is a numeric value .
1429 * @name hasNumericValue
1430 * @kind function
1431 * @param { * } x - The value to test .
1432 * @returns { boolean } Returns true if x is numeric .
1433 * @memberof PRDC_JSLAB_MATHJS_DOC
1434 */
1435
1436 /**
1437 * Help object constructor .
1438 * @name Help
1439 * @kind function
1440 * @param { * } value - The function or object to get help for .
1441 * @returns { Help } A new Help instance .
1442 * @memberof PRDC_JSLAB_MATHJS_DOC
1443 */
1444
1445 /**
1446 * Format a number as hexadecimal .
1447 * @name hex
1448 * @kind function
1449 * @param { number | BigNumber } value - The value to format .
1450 * @returns { string } The hexadecimal representation .
1451 * @memberof PRDC_JSLAB_MATHJS_DOC
1452 */
1453
1454 /**
1455 * Calculate the hypotenuse of a list of values .
1456 * @name hypot
1457 * @kind function

```

```

1458  * @param { ... number | ... BigNumber | Array | Matrix } args - The input
1459  * values.
1460  * @returns { number | BigNumber | Array | Matrix } The hypotenuse.
1461  * @memberof PRDC_JSLAB_MATHJS_DOC
1462  */
1463 /**
1464  * The imaginary unit 'i'.
1465  * @name i
1466  * @kind member
1467  * @memberof PRDC_JSLAB_MATHJS_DOC
1468  * @type { Complex }
1469  */
1470 /**
1471  * Create an identity matrix.
1472  * @name identity
1473  * @kind function
1474  * @param { number | Array } size - The size of the matrix.
1475  * @param { string } [format] - The matrix format.
1476  * @returns { Matrix } An identity matrix.
1477  * @memberof PRDC_JSLAB_MATHJS_DOC
1478  */
1479 /**
1480 /**
1481  * Compute the inverse Fast Fourier Transform.
1482  * @name ifft
1483  * @kind function
1484  * @param { Array | Matrix } x - Input array or matrix.
1485  * @returns { Array | Matrix } The inverse FFT of x.
1486  * @memberof PRDC_JSLAB_MATHJS_DOC
1487  */
1488 /**
1489 /**
1490  * Get the imaginary part of a complex number.
1491  * @name im
1492  * @kind function
1493  * @param { number | BigNumber | Complex | Array | Matrix } x - Input
1494  * value.
1495  * @returns { number | BigNumber | Array | Matrix } The imaginary part of
1496  * x.
1497  * @memberof PRDC_JSLAB_MATHJS_DOC
1498 */
1499 /**
1500  * Immutable dense matrix constructor.
1501  * @name ImmutableDenseMatrix
1502  * @kind function
1503  * @param { Array } data - The data for the matrix.
1504  * @returns { ImmutableDenseMatrix } A new ImmutableDenseMatrix instance.
1505  * @memberof PRDC_JSLAB_MATHJS_DOC
1506  */
1507 /**
1508  * Index constructor for matrices.
1509 */

```

```
1510 * @name Index
1511 * @kind function
1512 * @param { ...Range | ...number } ranges - Ranges or indices.
1513 * @returns { Index } A new Index instance.
1514 * @memberof PRDC_JSLAB_MATHJS_DOC
1515 */
1516
1517 /**
1518 * A node representing an index operation in the expression tree.
1519 * @name IndexNode
1520 * @kind function
1521 * @param { Node[] } dimensions - The indices.
1522 * @returns { IndexNode } A new IndexNode instance.
1523 * @memberof PRDC_JSLAB_MATHJS_DOC
1524 */
1525
1526 /**
1527 * Compute the intersection of two sets.
1528 * @name intersect
1529 * @kind function
1530 * @param { Array | Matrix } a - First set.
1531 * @param { Array | Matrix } b - Second set.
1532 * @returns { Array | Matrix } The intersection of a and b.
1533 * @memberof PRDC_JSLAB_MATHJS_DOC
1534 */
1535
1536 /**
1537 * Inverse conductance quantum, in ohms.
1538 * @name inverseConductanceQuantum
1539 * @kind member
1540 * @memberof PRDC_JSLAB_MATHJS_DOC
1541 * @type { number }
1542 */
1543
1544 /**
1545 * Calculate the modular inverse of a value.
1546 * @name invmod
1547 * @kind function
1548 * @param { number | BigNumber } a - The value.
1549 * @param { number | BigNumber } m - The modulus.
1550 * @returns { number | BigNumber } The modular inverse.
1551 * @memberof PRDC_JSLAB_MATHJS_DOC
1552 */
1553
1554 /**
1555 * Test whether a value is an integer.
1556 * @name isInteger
1557 * @kind function
1558 * @param { number | BigNumber } x - The value to test.
1559 * @returns { boolean } Returns true if x is an integer.
1560 * @memberof PRDC_JSLAB_MATHJS_DOC
1561 */
1562
1563 /**
1564 * Test whether a value is negative.
```

```

1565   * @name isNegative
1566   * @kind function
1567   * @param { number | BigNumber } x - The value to test.
1568   * @returns { boolean } Returns true if x is negative.
1569   * @memberof PRDC_JSLAB_MATHJS_DOC
1570   */
1571
1572 /**
1573 * Test whether a value is positive.
1574 * @name isPositive
1575 * @kind function
1576 * @param { number | BigNumber } x - The value to test.
1577 * @returns { boolean } Returns true if x is positive.
1578 * @memberof PRDC_JSLAB_MATHJS_DOC
1579 */
1580
1581 /**
1582 * Test whether a number is prime.
1583 * @name isPrime
1584 * @kind function
1585 * @param { number | BigNumber } x - The value to test.
1586 * @returns { boolean } Returns true if x is prime.
1587 * @memberof PRDC_JSLAB_MATHJS_DOC
1588 */
1589
1590 /**
1591 * Test whether a value is zero.
1592 * @name isZero
1593 * @kind function
1594 * @param { number | BigNumber } x - The value to test.
1595 * @returns { boolean } Returns true if x is zero.
1596 * @memberof PRDC_JSLAB_MATHJS_DOC
1597 */
1598
1599 /**
1600 * Calculate the Kullback–Leibler divergence between two distributions.
1601 * @name kldivergence
1602 * @kind function
1603 * @param { Array | Matrix } p - First probability distribution.
1604 * @param { Array | Matrix } q - Second probability distribution.
1605 * @returns { number } The KL divergence  $D_{KL}(p || q)$ .
1606 * @memberof PRDC_JSLAB_MATHJS_DOC
1607 */
1608
1609 /**
1610 * Von Klitzing constant, in ohms.
1611 * @name klitzing
1612 * @kind member
1613 * @memberof PRDC_JSLAB_MATHJS_DOC
1614 * @type { number }
1615 */
1616
1617 /**
1618 * Compute the Kronecker product of two matrices.
1619 * @name kron

```

```
1620 * @kind function
1621 * @param { Array | Matrix } A - First matrix.
1622 * @param { Array | Matrix } B - Second matrix.
1623 * @returns { Matrix } The Kronecker product of A and B.
1624 * @memberof PRDC_JSLAB_MATHJS_DOC
1625 */
1626
1627 /**
1628 * Natural logarithm of 10.
1629 * @name LN10
1630 * @kind member
1631 * @memberof PRDC_JSLAB_MATHJS_DOC
1632 * @type { number }
1633 */
1634
1635 /**
1636 * Natural logarithm of 2.
1637 * @name LN2
1638 * @kind member
1639 * @memberof PRDC_JSLAB_MATHJS_DOC
1640 * @type { number }
1641 */
1642
1643 /**
1644 * Base 10 logarithm of e.
1645 * @name LOG10E
1646 * @kind member
1647 * @memberof PRDC_JSLAB_MATHJS_DOC
1648 * @type { number }
1649 */
1650
1651 /**
1652 * Base 2 logarithm of e.
1653 * @name LOG2E
1654 * @kind member
1655 * @memberof PRDC_JSLAB_MATHJS_DOC
1656 * @type { number }
1657 */
1658
1659 /**
1660 * Test whether value x is larger than y.
1661 * @name larger
1662 * @kind function
1663 * @param { number | BigNumber | Fraction | Complex | Unit | string | |
1664 *         Array | Matrix } x - First value.
1665 * @param { number | BigNumber | Fraction | Complex | Unit | string | |
1666 *         Array | Matrix } y - Second value.
1667 * @returns { boolean | Array | Matrix } Returns true if x > y.
1668 * @memberof PRDC_JSLAB_MATHJS_DOC
1669 */
1670
1671 /**
1672 * Test whether value x is larger than or equal to y.
1673 * @name largerEq
1674 * @kind function
```

```
1673     * @param { number | BigNumber | Fraction | Complex | Unit | string |  
1674       Array | Matrix } x - First value.  
1675     * @param { number | BigNumber | Fraction | Complex | Unit | string |  
1676       Array | Matrix } y - Second value.  
1677     * @returns { boolean | Array | Matrix } Returns true if x >= y.  
1678     * @memberof PRDC_JSLAB_MATHJS_DOC  
1679   */  
1680  
1681   /**  
1682    * Compute the least common multiple of two or more values.  
1683    * @name lcm  
1684    * @kind function  
1685    * @param { ...number | ...BigNumber | Array | Matrix } args - Two or more  
1686      integer numbers.  
1687    * @returns { number | BigNumber | Array | Matrix } The least common  
1688      multiple.  
1689    * @memberof PRDC_JSLAB_MATHJS_DOC  
1690  */  
1691  
1692  /**  
1693   * Count the number of leaf nodes in an expression tree.  
1694   * @name leafCount  
1695   * @kind function  
1696   * @param { Node } node - The root node of the expression tree.  
1697   * @returns { number } The number of leaf nodes.  
1698   * @memberof PRDC_JSLAB_MATHJS_DOC  
1699  */  
1700  
1701  /**  
1702   * Bitwise left shift operation.  
1703   * @name leftShift  
1704   * @kind function  
1705   * @param { number | BigNumber | Array | Matrix } x - Value to be shifted.  
1706   * @param { number | BigNumber | Array | Matrix } y - Amount of bits to  
1707     shift.  
1708   * @returns { number | BigNumber | Array | Matrix } The shifted value.  
1709   * @memberof PRDC_JSLAB_MATHJS_DOC  
1710  */  
1711  
1712  /**  
1713   * Compute the natural logarithm of the gamma function.  
1714   * @name lgamma  
1715   * @kind function  
1716   * @param { number } n - The input value.  
1717   * @returns { number } The natural logarithm of the gamma function at n.  
1718   * @memberof PRDC_JSLAB_MATHJS_DOC  
1719  */  
1720  
1721  /**  
1722   * Calculate the natural logarithm of a value.  
1723   * @name log  
1724   * @kind function  
1725   * @param { number | BigNumber | Complex | Array | Matrix } x - Value for  
1726     which to calculate the logarithm.  
1727   * @param { number | BigNumber | Complex } [base=e] - Base of the
```

```

1722     logarithm .
1723     * @returns { number | BigNumber | Complex | Array | Matrix } The
1724       logarithm of x.
1725     * @memberof PRDC_JSLAB_MATHJS_DOC
1726     */
1727
1728   /**
1729    * Calculate the base-10 logarithm of a value.
1730    * @name log10
1731    * @kind function
1732    * @param { number | BigNumber | Complex | Array | Matrix } x - Value for
1733      which to calculate the logarithm.
1734    * @returns { number | BigNumber | Complex | Array | Matrix } The base-10
1735      logarithm of x.
1736    * @memberof PRDC_JSLAB_MATHJS_DOC
1737    */
1738
1739 /**
1740  * Calculate the natural logarithm of 1 plus a value.
1741  * @name log1p
1742  * @kind function
1743  * @param { number | BigNumber } x - Input value.
1744  * @returns { number | BigNumber } The result of  $\log(1 + x)$ .
1745  * @memberof PRDC_JSLAB_MATHJS_DOC
1746  */
1747
1748 /**
1749  * Calculate the base-2 logarithm of a value.
1750  * @name log2
1751  * @kind function
1752  * @param { number | BigNumber | Complex | Array | Matrix } x - Value for
1753    which to calculate the logarithm.
1754  * @returns { number | BigNumber | Complex | Array | Matrix } The base-2
1755    logarithm of x.
1756  * @memberof PRDC_JSLAB_MATHJS_DOC
1757  */
1758
1759 /**
1760  * Loschmidt constant at 0°C and 1 atm, in  $m^{-3}$ .
1761  * @name loschmidt
1762  * @kind member
1763  * @memberof PRDC_JSLAB_MATHJS_DOC
1764  * @type { number }
1765  */
1766
1767 /**
1768  * Solve a linear system  $A * x = b$  where A is a lower triangular matrix.
1769  * @name lsolve
1770  * @kind function
1771  * @param { Matrix | Array } L - A lower triangular matrix.
1772  * @param { Matrix | Array } b - A column vector.
1773  * @returns { Matrix | Array } The solution vector x.
1774  * @memberof PRDC_JSLAB_MATHJS_DOC
1775  */
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3820
3821
3822
3823
3824
3825
3826
3827
38
```

```

1771  /**
1772   * Find all solutions of a linear system  $A \star x = b$  where  $A$  is a lower
1773   * triangular matrix.
1774   * @name lsolveAll
1775   * @kind function
1776   * @param { Matrix | Array } L - A lower triangular matrix.
1777   * @param { Matrix | Array } b - A column vector.
1778   * @returns { Array } An array of solutions.
1779   * @memberof PRDC_JSLAB_MATHJS_DOC
1780   */
1781 /**
1782  * Compute the LU decomposition with partial pivoting.
1783  * @name lup
1784  * @kind function
1785  * @param { Matrix | Array } A - A square matrix.
1786  * @param { number } [threshold=1e-10] - Tolerance threshold.
1787  * @returns { Object } An object containing L, U, and P matrices.
1788  * @memberof PRDC_JSLAB_MATHJS_DOC
1789  */
1790 /**
1791  * Solve a linear system using LU decomposition.
1792  * @name lusolve
1793  * @kind function
1794  * @param { Matrix | Array } A - Coefficient matrix.
1795  * @param { Matrix | Array } b - Right-hand side vector or matrix.
1796  * @param { string } [order] - Matrix storage order.
1797  * @returns { Matrix | Array } The solution vector or matrix.
1798  * @memberof PRDC_JSLAB_MATHJS_DOC
1799  */
1800 /**
1801  * Solve the Lyapunov equation  $A \star X + X \star A' = Q$ .
1802  * @name lyap
1803  * @kind function
1804  * @param { Matrix | Array } A - A square matrix.
1805  * @param { Matrix | Array } Q - A square matrix.
1806  * @returns { Matrix } Solution X of the Lyapunov equation.
1807  * @memberof PRDC_JSLAB_MATHJS_DOC
1808  */
1809 /**
1810  * Compute the median absolute deviation of a set of values.
1811  * @name mad
1812  * @kind function
1813  * @param { Array | Matrix } array - Input array.
1814  * @param { number | BigNumber } [dim] - Dimension along which to compute.
1815  * @returns { number | BigNumber | Array | Matrix } The median absolute
1816  * deviation.
1817  * @memberof PRDC_JSLAB_MATHJS_DOC
1818  */
1819 /**
1820  * Magnetic constant (vacuum permeability), in N/A^2.
1821  */
1822 /**
1823  * Magnetic constant (vacuum permeability), in N/A^2.

```

```
1824 * @name magneticConstant
1825 * @kind member
1826 * @memberof PRDC_JSLAB_MATHJS_DOC
1827 * @type { number }
1828 */
1829
1830 /**
1831 * Magnetic flux quantum, in Wb.
1832 * @name magneticFluxQuantum
1833 * @kind member
1834 * @memberof PRDC_JSLAB_MATHJS_DOC
1835 * @type { number }
1836 */
1837
1838 /**
1839 * Map a function over the elements of a matrix or array.
1840 * @name map
1841 * @kind function
1842 * @param { Array | Matrix } x - The input array or matrix.
1843 * @param { function } callback - The function to apply.
1844 * @returns { Array | Matrix } The result after applying the callback.
1845 * @memberof PRDC_JSLAB_MATHJS_DOC
1846 */
1847
1848 /**
1849 * Create a matrix.
1850 * @name matrix
1851 * @kind function
1852 * @param { Array | Matrix } [data] - The data for the matrix.
1853 * @param { string } [format] - The matrix format.
1854 * @returns { Matrix } A new Matrix instance.
1855 * @memberof PRDC_JSLAB_MATHJS_DOC
1856 */
1857
1858 /**
1859 * Matrix constructor.
1860 * @name Matrix
1861 * @kind function
1862 * @param { Array | Matrix } [data] - The data for the matrix.
1863 * @param { string } [format] - The matrix format.
1864 * @returns { Matrix } A new Matrix instance.
1865 * @memberof PRDC_JSLAB_MATHJS_DOC
1866 */
1867
1868 /**
1869 * Create a matrix from given columns.
1870 * @name matrixFromColumns
1871 * @kind function
1872 * @param { ...Array | ...Matrix } columns - Columns to construct the
1873 * matrix.
1874 * @returns { Matrix } The constructed matrix.
1875 * @memberof PRDC_JSLAB_MATHJS_DOC
1876 */
1877 /**

```

```
1878 * Create a matrix using a provided function.  
1879 * @name matrixFromFunction  
1880 * @kind function  
1881 * @param { Array } size - The size of the matrix.  
1882 * @param { function } callback - Function to generate matrix entries.  
1883 * @returns { Matrix } The generated matrix.  
1884 * @memberof PRDC_JSLAB_MATHJS_DOC  
1885 */  
1886  
1887 /**
1888 * Create a matrix from given rows.  
1889 * @name matrixFromRows  
1890 * @kind function  
1891 * @param { ...Array | ...Matrix } rows - Rows to construct the matrix.  
1892 * @returns { Matrix } The constructed matrix.  
1893 * @memberof PRDC_JSLAB_MATHJS_DOC  
1894 */  
1895  
1896 /**
1897 * Compute the arithmetic mean of a set of values.  
1898 * @name mean  
1899 * @kind function  
1900 * @param { Array | Matrix } array - Input array.  
1901 * @param { number | BigNumber } [dim] - Dimension along which to compute.  
1902 * @returns { number | BigNumber | Array | Matrix } The mean value.  
1903 * @memberof PRDC_JSLAB_MATHJS_DOC  
1904 */  
1905  
1906 /**
1907 * Compute the median of a set of values.  
1908 * @name median  
1909 * @kind function  
1910 * @param { Array | Matrix } array - Input array.  
1911 * @param { number | BigNumber } [dim] - Dimension along which to compute.  
1912 * @returns { number | BigNumber | Array | Matrix } The median value.  
1913 * @memberof PRDC_JSLAB_MATHJS_DOC  
1914 */  
1915  
1916 /**
1917 * Calculate the modulus of two numbers.  
1918 * @name mod  
1919 * @kind function  
1920 * @param { number | BigNumber | Array | Matrix } x - Dividend.  
1921 * @param { number | BigNumber | Array | Matrix } y - Divisor.  
1922 * @returns { number | BigNumber | Array | Matrix } The remainder after  
division.  
1923 * @memberof PRDC_JSLAB_MATHJS_DOC  
1924 */  
1925  
1926 /**
1927 * Compute the mode of a set of values.  
1928 * @name mode  
1929 * @kind function  
1930 * @param { Array | Matrix } array - Input array.  
1931 * @returns { Array | Matrix } The mode(s) of the array.
```

```

1932   * @memberof PRDC_JSLAB_MATHJS_DOC
1933   */
1934
1935 /**
1936   * Molar mass constant , in kg/mol.
1937   * @name molarMass
1938   * @kind member
1939   * @memberof PRDC_JSLAB_MATHJS_DOC
1940   * @type { number }
1941   */
1942
1943 /**
1944   * Molar mass of carbon-12, in kg/mol.
1945   * @name molarMassC12
1946   * @kind member
1947   * @memberof PRDC_JSLAB_MATHJS_DOC
1948   * @type { number }
1949   */
1950
1951 /**
1952   * Molar Planck constant , in J s/mol.
1953   * @name molarPlanckConstant
1954   * @kind member
1955   * @memberof PRDC_JSLAB_MATHJS_DOC
1956   * @type { number }
1957   */
1958
1959 /**
1960   * Molar volume of an ideal gas at 1 atm and 0°C, in m /mol.
1961   * @name molarVolume
1962   * @kind member
1963   * @memberof PRDC_JSLAB_MATHJS_DOC
1964   * @type { number }
1965   */
1966
1967 /**
1968   * Compute the multinomial coefficient of a list of integers.
1969   * @name multinomial
1970   * @kind function
1971   * @param { ...number | Array } args - Integer numbers.
1972   * @returns { number } The multinomial coefficient .
1973   * @memberof PRDC_JSLAB_MATHJS_DOC
1974   */
1975
1976 /**
1977   * Multiply two scalar values , x * y.
1978   * @name multiplyScalar
1979   * @kind function
1980   * @param { number | BigNumber | Fraction | Complex } x - First value.
1981   * @param { number | BigNumber | Fraction | Complex } y - Second value.
1982   * @returns { number | BigNumber | Fraction | Complex } The product of x
1983   * and y.
1984   * @memberof PRDC_JSLAB_MATHJS_DOC
1985   */

```

```
1986  /**
1987   * Neutron mass, in kilograms.
1988   * @name neutronMass
1989   * @kind member
1990   * @memberof PRDC_JSLAB_MATHJS_DOC
1991   * @type { number }
1992   */
1993
1994 /**
1995  * Logical NOT operation.
1996  * @name not
1997  * @kind function
1998  * @param { boolean | Array | Matrix } x - Input value.
1999  * @returns { boolean | Array | Matrix } The logical negation of x.
2000  * @memberof PRDC_JSLAB_MATHJS_DOC
2001  */
2002
2003 /**
2004  * Calculate the nth root of a value.
2005  * @name nthRoot
2006  * @kind function
2007  * @param { number | BigNumber | Complex | Array | Matrix } a - The value.
2008  * @param { number | BigNumber } [root=2] - The root.
2009  * @returns { number | BigNumber | Complex | Array | Matrix } The nth root
2010  *          of a.
2011  * @memberof PRDC_JSLAB_MATHJS_DOC
2012  */
2013 /**
2014  * Calculate the nth roots of a complex number.
2015  * @name nthRoots
2016  * @kind function
2017  * @param { number | Complex } a - The value.
2018  * @param { number | BigNumber } n - The degree of the root.
2019  * @returns { Array } An array of the n roots.
2020  * @memberof PRDC_JSLAB_MATHJS_DOC
2021  */
2022
2023 /**
2024  * Nuclear magneton, in J/T.
2025  * @name nuclearMagneton
2026  * @kind member
2027  * @memberof PRDC_JSLAB_MATHJS_DOC
2028  * @type { number }
2029  */
2030
2031 /**
2032  * JavaScript null value.
2033  * @name null
2034  * @kind member
2035  * @memberof PRDC_JSLAB_MATHJS_DOC
2036  * @type { null }
2037  */
2038
2039 /**
```

```
2040 * Parse a value into a number.
2041 * @name number
2042 * @kind function
2043 * @param { * } value - The value to parse.
2044 * @returns { number } The numeric value.
2045 * @memberof PRDC_JSLAB_MATHJS_DOC
2046 */
2047
2048 /**
2049 * Convert a math.js data type to a numeric type.
2050 * @name numeric
2051 * @kind function
2052 * @param { * } value - The value to convert.
2053 * @returns { number | BigNumber | Complex } The numeric representation.
2054 * @memberof PRDC_JSLAB_MATHJS_DOC
2055 */
2056
2057 /**
2058 * A node representing an object in the expression tree.
2059 * @name ObjectNode
2060 * @kind function
2061 * @param { Object } properties - Object properties as nodes.
2062 * @returns { ObjectNode } A new ObjectNode instance.
2063 * @memberof PRDC_JSLAB_MATHJS_DOC
2064 */
2065
2066 /**
2067 * Format a number as octal.
2068 * @name oct
2069 * @kind function
2070 * @param { number | BigNumber } value - The value to format.
2071 * @returns { string } The octal representation.
2072 * @memberof PRDC_JSLAB_MATHJS_DOC
2073 */
2074
2075 /**
2076 * A node representing an operator in the expression tree.
2077 * @name OperatorNode
2078 * @kind function
2079 * @param { string } op - The operator symbol.
2080 * @param { string } fn - The function name.
2081 * @param { Node[] } args - An array of argument nodes.
2082 * @param { boolean } [implicit=false] - Is the operator implicit?
2083 * @returns { OperatorNode } A new OperatorNode instance.
2084 * @memberof PRDC_JSLAB_MATHJS_DOC
2085 */
2086
2087 /**
2088 * Logical OR operation.
2089 * @name or
2090 * @kind function
2091 * @param { boolean | Array | Matrix } x - First value.
2092 * @param { boolean | Array | Matrix } y - Second value.
2093 * @returns { boolean | Array | Matrix } The logical OR of x and y.
2094 * @memberof PRDC_JSLAB_MATHJS_DOC
```



```
2095      */
2096
2097  /**
2098   * A node representing parentheses in the expression tree.
2099   * @name ParenthesisNode
2100   * @kind function
2101   * @param { Node } content - The node encapsulated by the parentheses.
2102   * @returns { ParenthesisNode } A new ParenthesisNode instance.
2103   * @memberof PRDC_JSLAB_MATHJS_DOC
2104   */
2105
2106 /**
2107  * Parse and evaluate an expression.
2108  * @name parse
2109  * @kind function
2110  * @param { string | Object } expr - The expression to parse.
2111  * @returns { Node | Node[] } The parsed expression node(s).
2112  * @memberof PRDC_JSLAB_MATHJS_DOC
2113  */
2114
2115 /**
2116  * Create a parser with its own scope and functions.
2117  * @name parser
2118  * @kind function
2119  * @returns { Parser } A new Parser instance.
2120  * @memberof PRDC_JSLAB_MATHJS_DOC
2121  */
2122
2123 /**
2124  * Parser constructor.
2125  * @name Parser
2126  * @kind function
2127  * @returns { Parser } A new Parser instance.
2128  * @memberof PRDC_JSLAB_MATHJS_DOC
2129  */
2130
2131 /**
2132  * Select an element from a matrix or array based on its sorted position.
2133  * @name partitionSelect
2134  * @kind function
2135  * @param { Array | Matrix } x - The input array or matrix.
2136  * @param { number } k - The kth smallest value to select.
2137  * @param { function } [compare] - Optional comparison function.
2138  * @returns { number | BigNumber | Fraction | Complex } The selected
2139  * element.
2140  * @memberof PRDC_JSLAB_MATHJS_DOC
2141  */
2142 /**
2143  * Calculate the number of permutations of n items taken k at a time.
2144  * @name permutations
2145  * @kind function
2146  * @param { number | BigNumber } n - Total number of items.
2147  * @param { number | BigNumber } [k] - Number of items to choose.
2148  * @returns { number | BigNumber } Number of possible permutations.
```

```

2149   * @memberof PRDC_JSLAB_MATHJS_DOC
2150   */
2151
2152 /**
2153 * The golden ratio, approximately 1.618.
2154 * @name phi
2155 * @kind member
2156 * @memberof PRDC_JSLAB_MATHJS_DOC
2157 * @type { number }
2158 */
2159
2160 /**
2161 * The mathematical constant pi.
2162 * @name pi
2163 * @kind member
2164 * @memberof PRDC_JSLAB_MATHJS_DOC
2165 * @type { number }
2166 */
2167
2168 /**
2169 * Randomly pick one or more values from a list.
2170 * @name pickRandom
2171 * @kind function
2172 * @param { Array } array - Array to pick values from.
2173 * @param { number } [number] - Number of values to pick.
2174 * @returns { * | Array } Picked value(s).
2175 * @memberof PRDC_JSLAB_MATHJS_DOC
2176 */
2177
2178 /**
2179 * Compute the pseudoinverse of a matrix.
2180 * @name pinv
2181 * @kind function
2182 * @param { Array | Matrix } x - A matrix.
2183 * @returns { Matrix } The pseudoinverse of x.
2184 * @memberof PRDC_JSLAB_MATHJS_DOC
2185 */
2186
2187 /**
2188 * Planck charge, in coulombs.
2189 * @name planckCharge
2190 * @kind member
2191 * @memberof PRDC_JSLAB_MATHJS_DOC
2192 * @type { number }
2193 */
2194
2195 /**
2196 * Planck constant, in joule seconds.
2197 * @name planckConstant
2198 * @kind member
2199 * @memberof PRDC_JSLAB_MATHJS_DOC
2200 * @type { number }
2201 */
2202
2203 /**

```

```

2204     * Planck length , in meters .
2205     * @name planckLength
2206     * @kind member
2207     * @memberof PRDC_JSLAB_MATHJS_DOC
2208     * @type { number }
2209     */
2210
2211 /**
2212     * Planck mass , in kilograms .
2213     * @name planckMass
2214     * @kind member
2215     * @memberof PRDC_JSLAB_MATHJS_DOC
2216     * @type { number }
2217     */
2218
2219 /**
2220     * Planck temperature , in kelvin .
2221     * @name planckTemperature
2222     * @kind member
2223     * @memberof PRDC_JSLAB_MATHJS_DOC
2224     * @type { number }
2225     */
2226
2227 /**
2228     * Planck time , in seconds .
2229     * @name planckTime
2230     * @kind member
2231     * @memberof PRDC_JSLAB_MATHJS_DOC
2232     * @type { number }
2233     */
2234
2235 /**
2236     * Find roots of a univariate polynomial .
2237     * @name polynomialRoot
2238     * @kind function
2239     * @param { Array } coefficients - Coefficients of the polynomial .
2240     * @returns { Array } An array of roots .
2241     * @memberof PRDC_JSLAB_MATHJS_DOC
2242     */
2243
2244 /**
2245     * Calculate the power of x to y , x^y .
2246     * @name pow
2247     * @kind function
2248     * @param { number | BigNumber | Complex | Array | Matrix } x - Base .
2249     * @param { number | BigNumber | Complex | Array | Matrix } y - Exponent .
2250     * @returns { number | BigNumber | Complex | Array | Matrix } x raised to
2251     *          the power y .
2252     * @memberof PRDC_JSLAB_MATHJS_DOC
2253     */
2254 /**
2255     * Compute the product of a set of values .
2256     * @name prod
2257     * @kind function

```

```

2258 * @param { Array | Matrix } array - Input array.
2259 * @param { number | BigNumber } [dim] - Dimension along which to compute
2260     the product.
2261 * @returns { number | BigNumber | Array | Matrix } The product of all
2262     values.
2263 * @memberof PRDC_JSLAB_MATHJS_DOC
2264 */
2265
2266 /**
2267 * Proton mass, in kilograms.
2268 * @name protonMass
2269 * @kind member
2270 * @memberof PRDC_JSLAB_MATHJS_DOC
2271 * @type { number }
2272 */
2273
2274 /**
2275 * Compute the QR decomposition of a matrix.
2276 * @name qr
2277 * @kind function
2278 * @param { Matrix | Array } x - A matrix.
2279 * @returns { Object } An object containing Q and R matrices.
2280 * @memberof PRDC_JSLAB_MATHJS_DOC
2281 */
2282
2283 /**
2284 * Compute the quantile of a sequence.
2285 * @name quantileSeq
2286 * @kind function
2287 * @param { Array | Matrix } data - Input data.
2288 * @param { number | Array } prob - Probability or array of probabilities.
2289 * @param { boolean } [sorted=false] - Is data sorted?
2290 * @returns { number | Array } The quantile(s).
2291 * @memberof PRDC_JSLAB_MATHJS_DOC
2292 */
2293
2294 /**
2295 * Quantum of circulation, in m^2/s.
2296 * @name quantumOfCirculation
2297 * @kind member
2298 * @memberof PRDC_JSLAB_MATHJS_DOC
2299 * @type { number }
2300 */
2301
2302 /**
2303 * Generate a random integer between min and max.
2304 * @name randomInt
2305 * @kind function
2306 * @param { number | BigNumber } [min] - Minimum value, inclusive.
2307 * @param { number | BigNumber } max - Maximum value, inclusive.
2308 * @returns { number | BigNumber } A random integer.
2309 * @memberof PRDC_JSLAB_MATHJS_DOC
2310 */
2311

```

```

2311  * Create a range.
2312  * @name Range
2313  * @kind function
2314  * @param { * } start - Start of the range.
2315  * @param { * } end - End of the range.
2316  * @param { * } [step=1] - Step size.
2317  * @returns { Range } A new Range instance.
2318  * @memberof PRDC_JSLAB_MATHJS_DOC
2319  */
2320
2321 /**
2322  * A node representing a range in the expression tree.
2323  * @name RangeNode
2324  * @kind function
2325  * @param { Node } start - Start node.
2326  * @param { Node } end - End node.
2327  * @param { Node } [step] - Step node.
2328  * @returns { RangeNode } A new RangeNode instance.
2329  * @memberof PRDC_JSLAB_MATHJS_DOC
2330  */
2331
2332 /**
2333  * Rationalize an expression.
2334  * @name rationalize
2335  * @kind function
2336  * @param { string | Node } expr - The expression to rationalize.
2337  * @param { Object } [scope] - Scope of variables.
2338  * @param { Object } [options] - Options object.
2339  * @returns { Object } An object with expression and denominator.
2340  * @memberof PRDC_JSLAB_MATHJS_DOC
2341  */
2342
2343 /**
2344  * Get the real part of a complex number.
2345  * @name re
2346  * @kind function
2347  * @param { number | BigNumber | Complex | Array | Matrix } x - Input
2348  * value.
2349  * @returns { number | BigNumber | Array | Matrix } The real part of x.
2350  * @memberof PRDC_JSLAB_MATHJS_DOC
2351  */
2352 /**
2353  * Reduced Planck constant, in joule seconds.
2354  * @name reducedPlanckConstant
2355  * @kind member
2356  * @memberof PRDC_JSLAB_MATHJS_DOC
2357  * @type { number }
2358  */
2359
2360 /**
2361  * A node representing a relational operation.
2362  * @name RelationalNode
2363  * @kind function
2364  * @param { string } condition - Relational condition.

```

```
2365 * @param { Node } params - Parameters.
2366 * @returns { RelationalNode } A new RelationalNode instance.
2367 * @memberof PRDC_JSLAB_MATHJS_DOC
2368 */
2369
2370 /**
2371 * Replacer function for JSON serialization.
2372 * @name replacer
2373 * @kind function
2374 * @param { string } key - Property name.
2375 * @param { * } value - Property value.
2376 * @returns { * } The transformed value.
2377 * @memberof PRDC_JSLAB_MATHJS_DOC
2378 */
2379
2380 /**
2381 * Resize a matrix.
2382 * @name resize
2383 * @kind function
2384 * @param { Array | Matrix } x - The input matrix.
2385 * @param { Array } size - The new size.
2386 * @param { * } [defaultValue=0] - Default value for new entries.
2387 * @returns { Array | Matrix } The resized matrix.
2388 * @memberof PRDC_JSLAB_MATHJS_DOC
2389 */
2390
2391 /**
2392 * Resolve the value of a symbol or function.
2393 * @name resolve
2394 * @kind function
2395 * @param { string } name - The name to resolve.
2396 * @returns { * } The resolved value.
2397 * @memberof PRDC_JSLAB_MATHJS_DOC
2398 */
2399
2400 /**
2401 * ResultSet constructor.
2402 * @name ResultSet
2403 * @kind function
2404 * @param { Array } entries - The entries in the result set.
2405 * @returns { ResultSet } A new ResultSet instance.
2406 * @memberof PRDC_JSLAB_MATHJS_DOC
2407 */
2408
2409 /**
2410 * Reviver function for JSON deserialization.
2411 * @name reviver
2412 * @kind function
2413 * @param { string } key - Property name.
2414 * @param { * } value - Property value.
2415 * @returns { * } The transformed value.
2416 * @memberof PRDC_JSLAB_MATHJS_DOC
2417 */
2418
2419 /**
```

```
2420 * Bitwise right arithmetic shift operation.
2421 * @name rightArithShift
2422 * @kind function
2423 * @param { number | BigNumber | Array | Matrix } x - The value to shift.
2424 * @param { number | BigNumber | Array | Matrix } y - The amount to shift.
2425 * @returns { number | BigNumber | Array | Matrix } The shifted value.
2426 * @memberof PRDC_JSLAB_MATHJS_DOC
2427 */
2428
2429 /**
2430 * Bitwise right logical shift operation.
2431 * @name rightLogShift
2432 * @kind function
2433 * @param { number | BigNumber | Array | Matrix } x - The value to shift.
2434 * @param { number | BigNumber | Array | Matrix } y - The amount to shift.
2435 * @returns { number | BigNumber | Array | Matrix } The shifted value.
2436 * @memberof PRDC_JSLAB_MATHJS_DOC
2437 */
2438
2439 /**
2440 * Rotate the elements of a matrix.
2441 * @name rotate
2442 * @kind function
2443 * @param { Array | Matrix } x - The input matrix.
2444 * @param { number | BigNumber } [turns=1] - Number of 90-degree rotations
2445 * @returns { Array | Matrix } The rotated matrix.
2446 * @memberof PRDC_JSLAB_MATHJS_DOC
2447 */
2448
2449 /**
2450 * Create a 2D rotation matrix.
2451 * @name rotationMatrix
2452 * @kind function
2453 * @param { number | BigNumber } theta - Rotation angle in radians.
2454 * @returns { Matrix } The rotation matrix.
2455 * @memberof PRDC_JSLAB_MATHJS_DOC
2456 */
2457
2458 /**
2459 * Rydberg constant, in m^-1.
2460 * @name rydberg
2461 * @kind member
2462 * @memberof PRDC_JSLAB_MATHJS_DOC
2463 * @type { number }
2464 */
2465
2466 /**
2467 * Square root of 1/2.
2468 * @name SQRT1_2
2469 * @kind member
2470 * @memberof PRDC_JSLAB_MATHJS_DOC
2471 * @type { number }
2472 */
2473
```

```

2474 /**
2475 * Square root of 2.
2476 * @name SQRT2
2477 * @kind member
2478 * @memberof PRDC_JSLAB_MATHJS_DOC
2479 * @type { number }
2480 */
2481
2482 /**
2483 * Sackur-Tetrode constant at 1 atm, in J/K.
2484 * @name sackurTetrode
2485 * @kind member
2486 * @memberof PRDC_JSLAB_MATHJS_DOC
2487 * @type { number }
2488 */
2489
2490 /**
2491 * Compute the Schur decomposition of a matrix.
2492 * @name schur
2493 * @kind function
2494 * @param { Matrix | Array } A - A square matrix.
2495 * @returns { Object } An object containing matrices U and T.
2496 * @memberof PRDC_JSLAB_MATHJS_DOC
2497 */
2498
2499 /**
2500 * Calculate the secant of a value.
2501 * @name sec
2502 * @kind function
2503 * @param { number | Complex | Unit | Array | Matrix } x - Function input.
2504 * @returns { number | Complex | Array | Matrix } The secant of x.
2505 * @memberof PRDC_JSLAB_MATHJS_DOC
2506 */
2507
2508 /**
2509 * Calculate the hyperbolic secant of a value.
2510 * @name sech
2511 * @kind function
2512 * @param { number | Complex | Array | Matrix } x - Function input.
2513 * @returns { number | Complex | Array | Matrix } The hyperbolic secant of
2514 * x.
2515 * @memberof PRDC_JSLAB_MATHJS_DOC
2516 */
2517 /**
2518 * Second radiation constant, in m K.
2519 * @name secondRadiation
2520 * @kind member
2521 * @memberof PRDC_JSLAB_MATHJS_DOC
2522 * @type { number }
2523 */
2524
2525 /**
2526 * Create the Cartesian product of two or more sets.
2527 * @name setCartesian

```

```

2528     * @kind function
2529     * @param { ...Array | ...Matrix } sets - The sets to compute the product
2530     * of .
2531     * @returns { Array | Matrix } The Cartesian product.
2532     * @memberof PRDC_JSLAB_MATHJS_DOC
2533     */
2534
2535 /**
2536 * Compute the difference between two sets .
2537 * @name setDifference
2538 * @kind function
2539 * @param { Array | Matrix } a - First set .
2540 * @param { Array | Matrix } b - Second set .
2541 * @returns { Array | Matrix } The difference a \ b.
2542 * @memberof PRDC_JSLAB_MATHJS_DOC
2543 */
2544
2545 /**
2546 * Remove duplicate elements from a set .
2547 * @name setDistinct
2548 * @kind function
2549 * @param { Array | Matrix } a - Input set .
2550 * @returns { Array | Matrix } A set with distinct elements .
2551 * @memberof PRDC_JSLAB_MATHJS_DOC
2552 */
2553
2554 /**
2555 * Compute the intersection of two or more sets .
2556 * @name setIntersect
2557 * @kind function
2558 * @param { Array | Matrix } a - First set .
2559 * @param { Array | Matrix } b - Second set .
2560 * @param { ...Array | ...Matrix } [others] - Additional sets .
2561 * @returns { Array | Matrix } The intersection of the sets .
2562 * @memberof PRDC_JSLAB_MATHJS_DOC
2563 */
2564
2565 /**
2566 * Test whether a set is a subset of another set .
2567 * @name setIsSubset
2568 * @kind function
2569 * @param { Array | Matrix } a - Potential subset .
2570 * @param { Array | Matrix } b - Superset .
2571 * @returns { boolean } Returns true if a is a subset of b .
2572 * @memberof PRDC_JSLAB_MATHJS_DOC
2573 */
2574
2575 /**
2576 * Count the multiplicity of an element in a multiset .
2577 * @name setMultiplicity
2578 * @kind function
2579 * @param { * } e - Element to count .
2580 * @param { Array | Matrix } multiset - The multiset .
2581 * @returns { number } The multiplicity of e .
2582 * @memberof PRDC_JSLAB_MATHJS_DOC

```

```
2582      */
2583
2584  /**
2585   * Compute the power set of a set.
2586   * @name setPowerset
2587   * @kind function
2588   * @param { Array | Matrix } set - The input set.
2589   * @returns { Array | Matrix } The power set.
2590   * @memberof PRDC_JSLAB_MATHJS_DOC
2591   */
2592
2593 /**
2594  * Get the size of a set.
2595  * @name setSize
2596  * @kind function
2597  * @param { Array | Matrix } set - The input set.
2598  * @returns { number } The number of elements.
2599  * @memberof PRDC_JSLAB_MATHJS_DOC
2600  */
2601
2602 /**
2603  * Compute the symmetric difference of two sets.
2604  * @name setSymDifference
2605  * @kind function
2606  * @param { Array | Matrix } a - First set.
2607  * @param { Array | Matrix } b - Second set.
2608  * @returns { Array | Matrix } The symmetric difference.
2609  * @memberof PRDC_JSLAB_MATHJS_DOC
2610  */
2611
2612 /**
2613  * Compute the union of two or more sets.
2614  * @name setUnion
2615  * @kind function
2616  * @param { Array | Matrix } a - First set.
2617  * @param { Array | Matrix } b - Second set.
2618  * @param { ...Array | ...Matrix } [others] - Additional sets.
2619  * @returns { Array | Matrix } The union of the sets.
2620  * @memberof PRDC_JSLAB_MATHJS_DOC
2621  */
2622
2623 /**
2624  * Compute the sign of a number.
2625  * @name sign
2626  * @kind function
2627  * @param { number | BigNumber | Fraction | Complex | Array | Matrix } x -
2628  *     Input value.
2629  * @returns { number | BigNumber | Fraction | Complex | Array | Matrix }
2630  *     The sign of x.
2631  * @memberof PRDC_JSLAB_MATHJS_DOC
2632  */
2633
2634 /**
2635  * Simplify an expression.
2636  * @name simplify
```

```

2635   * @kind function
2636   * @param { string | Node } expr - The expression to simplify.
2637   * @param { Object | Array | Function } [rules] - Simplification rules.
2638   * @param { Object } [scope] - Scope for variables.
2639   * @returns { Node } The simplified expression.
2640   * @memberof PRDC_JSLAB_MATHJS_DOC
2641   */
2642
2643 /**
2644  * Simplify an expression containing constants.
2645  * @name simplifyConstant
2646  * @kind function
2647  * @param { Node } expr - The expression to simplify.
2648  * @param { Object } [options] - Simplification options.
2649  * @returns { Node } The simplified expression.
2650  * @memberof PRDC_JSLAB_MATHJS_DOC
2651  */
2652
2653 /**
2654  * Perform core simplifications on an expression.
2655  * @name simplifyCore
2656  * @kind function
2657  * @param { Node } expr - The expression to simplify.
2658  * @returns { Node } The simplified expression.
2659  * @memberof PRDC_JSLAB_MATHJS_DOC
2660  */
2661
2662 /**
2663  * Calculate the sine of a value.
2664  * @name sin
2665  * @kind function
2666  * @param { number | Complex | Unit | Array | Matrix } x - Function input.
2667  * @returns { number | Complex | Array | Matrix } The sine of x.
2668  * @memberof PRDC_JSLAB_MATHJS_DOC
2669  */
2670
2671 /**
2672  * Calculate the hyperbolic sine of a value.
2673  * @name sinh
2674  * @kind function
2675  * @param { number | Complex | Array | Matrix } x - Function input.
2676  * @returns { number | Complex | Array | Matrix } The hyperbolic sine of x
2677
2678 /**
2679  * @memberof PRDC_JSLAB_MATHJS_DOC
2680  */
2681
2682 /**
2683  * Sparse Linear Solver using LU decomposition.
2684  * @name slu
2685  * @kind function
2686  * @param { Matrix | Array } A - A sparse matrix.
2687  * @param { number } order - Ordering and analysis control parameter.
2688  * @param { number } threshold - Partial pivoting threshold.
2689  * @returns { Object } The LU decomposition.
2690  * @memberof PRDC_JSLAB_MATHJS_DOC

```

```

2689     */
2690
2691 /**
2692 * Test whether value x is smaller than y.
2693 * @name smaller
2694 * @kind function
2695 * @param { number | BigNumber | Fraction | Complex | Unit | string |
2696     Array | Matrix } x - First value.
2696 * @param { number | BigNumber | Fraction | Complex | Unit | string |
2697     Array | Matrix } y - Second value.
2697 * @returns { boolean | Array | Matrix } Returns true if x < y.
2698 * @memberof PRDC_JSLAB_MATHJS_DOC
2699 */
2700
2701 /**
2702 * Test whether value x is smaller than or equal to y.
2703 * @name smallerEq
2704 * @kind function
2705 * @param { number | BigNumber | Fraction | Complex | Unit | string |
2706     Array | Matrix } x - First value.
2706 * @param { number | BigNumber | Fraction | Complex | Unit | string |
2707     Array | Matrix } y - Second value.
2707 * @returns { boolean | Array | Matrix } Returns true if x <= y.
2708 * @memberof PRDC_JSLAB_MATHJS_DOC
2709 */
2710
2711 /**
2712 * Compute the Sparse LU decomposition of a matrix.
2713 * @name Spa
2714 * @kind function
2715 * @param { Matrix | Array } A - A sparse matrix.
2716 * @returns { Object } The LU decomposition.
2717 * @memberof PRDC_JSLAB_MATHJS_DOC
2718 */
2719
2720 /**
2721 * Create a sparse matrix.
2722 * @name sparse
2723 * @kind function
2724 * @param { Array | Matrix } [data] - The data for the matrix.
2725 * @param { string } [datatype] - The data type.
2726 * @returns { SparseMatrix } A new SparseMatrix instance.
2727 * @memberof PRDC_JSLAB_MATHJS_DOC
2728 */
2729
2730 /**
2731 * Sparse matrix constructor.
2732 * @name SparseMatrix
2733 * @kind function
2734 * @param { Object } [data] - The data for the matrix.
2735 * @returns { SparseMatrix } A new SparseMatrix instance.
2736 * @memberof PRDC_JSLAB_MATHJS_DOC
2737 */
2738
2739 /**

```

```
2740 * Speed of light in vacuum, in m/s.
2741 * @name speedOfLight
2742 * @kind member
2743 * @memberof PRDC_JSLAB_MATHJS_DOC
2744 * @type { number }
2745 */
2746
2747 /**
2748 * Split a unit into its numeric value and unit string.
2749 * @name splitUnit
2750 * @kind function
2751 * @param { Unit } unit - The unit to split.
2752 * @param { Array } parts - Array of units to split into.
2753 * @returns { Array } Array of units.
2754 * @memberof PRDC_JSLAB_MATHJS_DOC
2755 */
2756
2757 /**
2758 * Calculate the square root of a value.
2759 * @name sqrt
2760 * @kind function
2761 * @param { number | BigNumber | Complex | Unit | Array | Matrix } x -
2762 * Value for which to calculate the square root.
2763 * @returns { number | BigNumber | Complex | Unit | Array | Matrix } The
2764 * square root of x.
2765 * @memberof PRDC_JSLAB_MATHJS_DOC
2766 */
2767
2768 /**
2769 * Calculate the principal square root of a matrix.
2770 * @name sqrtm
2771 * @kind function
2772 * @param { Array | Matrix } x - A square matrix.
2773 * @returns { Matrix } The principal square root of x.
2774 * @memberof PRDC_JSLAB_MATHJS_DOC
2775 */
2776
2777 /**
2778 * Compute the square of a value.
2779 * @name square
2780 * @kind function
2781 * @param { number | BigNumber | Complex | Unit | Array | Matrix } x -
2782 * Input value.
2783 * @returns { number | BigNumber | Complex | Unit | Array | Matrix } The
2784 * square of x.
2785 * @memberof PRDC_JSLAB_MATHJS_DOC
2786 */
2787
2788 /**
2789 * Remove singleton dimensions from an array or matrix.
2790 * @name squeeze
2791 * @kind function
2792 * @param { Array | Matrix } x - The input array or matrix.
2793 * @returns { Array | Matrix } The squeezed array or matrix.
2794 * @memberof PRDC_JSLAB_MATHJS_DOC
```

```

2791     */
2792
2793 /**
2794 * Compute the standard deviation of a set of values.
2795 * @name std
2796 * @kind function
2797 * @param { Array | Matrix } array - Input array.
2798 * @param { string } [normalization='unbiased'] - Normalization mode.
2799 * @param { number | BigNumber } [dim] - Dimension along which to compute.
2800 * @returns { number | BigNumber | Array | Matrix } The standard deviation
2801
2802 */
2803
2804 /**
2805 * Stefan-Boltzmann constant , in W/(m^2K^4) .
2806 * @name stefanBoltzmann
2807 * @kind member
2808 * @memberof PRDC_JSLAB_MATHJS_DOC
2809 * @type { number }
2810 */
2811
2812 /**
2813 * Stirling numbers of the second kind .
2814 * @name stirlingS2
2815 * @kind function
2816 * @param { number | BigNumber } n - Total number of objects .
2817 * @param { number | BigNumber } k - Number of non-empty subsets .
2818 * @returns { number | BigNumber } The Stirling number .
2819 * @memberof PRDC_JSLAB_MATHJS_DOC
2820 */
2821
2822 /**
2823 * Parse a value into a string .
2824 * @name string
2825 * @kind function
2826 * @param { * } value - The value to convert .
2827 * @returns { string } The string representation of value .
2828 * @memberof PRDC_JSLAB_MATHJS_DOC
2829 */
2830
2831 /**
2832 * Get or set a subset of a matrix or array .
2833 * @name subset
2834 * @kind function
2835 * @param { Array | Matrix } x - The input matrix .
2836 * @param { Index } index - The index .
2837 * @param { * } [replacement] - The replacement value .
2838 * @param { boolean } [defaultValue] - Default value for missing entries .
2839 * @returns { * } The subset or updated matrix .
2840 * @memberof PRDC_JSLAB_MATHJS_DOC
2841 */
2842
2843 /**
2844 * Solve the Sylvester equation A*X + X*B = C .

```

```

2845  * @name sylvester
2846  * @kind function
2847  * @param { Matrix | Array } A - A square matrix.
2848  * @param { Matrix | Array } B - A square matrix.
2849  * @param { Matrix | Array } C - A matrix.
2850  * @returns { Matrix } Solution X of the Sylvester equation.
2851  * @memberof PRDC_JSLAB_MATHJS_DOC
2852  */
2853
2854 /**
2855  * A node representing a symbol in the expression tree.
2856  * @name SymbolNode
2857  * @kind function
2858  * @param { string } name - The symbol name.
2859  * @returns { SymbolNode } A new SymbolNode instance.
2860  * @memberof PRDC_JSLAB_MATHJS_DOC
2861  */
2862
2863 /**
2864  * Test whether two expressions are symbolically equal.
2865  * @name symbolicEqual
2866  * @kind function
2867  * @param { Node } expr1 - First expression.
2868  * @param { Node } expr2 - Second expression.
2869  * @param { Object } [options] - Comparison options.
2870  * @returns { boolean } Returns true if expressions are symbolically equal
2871
2872  * @memberof PRDC_JSLAB_MATHJS_DOC
2873 */
2874 /**
2875  * Calculate the tangent of a value.
2876  * @name tan
2877  * @kind function
2878  * @param { number | Complex | Unit | Array | Matrix } x - Function input.
2879  * @returns { number | Complex | Array | Matrix } The tangent of x.
2880  * @memberof PRDC_JSLAB_MATHJS_DOC
2881 */
2882
2883 /**
2884  * Calculate the hyperbolic tangent of a value.
2885  * @name tanh
2886  * @kind function
2887  * @param { number | Complex | Array | Matrix } x - Function input.
2888  * @returns { number | Complex | Array | Matrix } The hyperbolic tangent
2889  * of x.
2890  * @memberof PRDC_JSLAB_MATHJS_DOC
2891 */
2892 /**
2893  * The constant tau, equal to 2pi.
2894  * @name tau
2895  * @kind member
2896  * @memberof PRDC_JSLAB_MATHJS_DOC
2897  * @type { number }

```

```
2898 */  
2899  
2900 /**  
2901 * Thomson cross section, in m^2.  
2902 * @name thomsonCrossSection  
2903 * @kind member  
2904 * @memberof PRDC_JSLAB_MATHJS_DOC  
2905 * @type { number }  
2906 */  
2907  
2908 /**  
2909 * Convert a unit to another unit.  
2910 * @name to  
2911 * @kind function  
2912 * @param { Unit } x - The unit to be converted.  
2913 * @param { Unit | string } unit - Target unit.  
2914 * @returns { Unit } The converted unit.  
2915 * @memberof PRDC_JSLAB_MATHJS_DOC  
2916 */  
2917  
2918 /**  
2919 * Boolean value true.  
2920 * @name true  
2921 * @kind member  
2922 * @memberof PRDC_JSLAB_MATHJS_DOC  
2923 * @type { boolean }  
2924 */  
2925  
2926 /**  
2927 * Get the type of a variable.  
2928 * @name typeOf  
2929 * @kind function  
2930 * @param { * } x - The variable.  
2931 * @returns { string } The type of x.  
2932 * @memberof PRDC_JSLAB_MATHJS_DOC  
2933 */  
2934  
2935 /**  
2936 * Create a typed-function.  
2937 * @name typed  
2938 * @kind function  
2939 * @param { string } name - Function name.  
2940 * @param { Object } signatures - Function signatures.  
2941 * @returns { function } The typed function.  
2942 * @memberof PRDC_JSLAB_MATHJS_DOC  
2943 */  
2944  
2945 /**  
2946 * Invert the sign of a value.  
2947 * @name unaryMinus  
2948 * @kind function  
2949 * @param { number | BigNumber | Fraction | Complex | Unit | Array |  
* Matrix } x - Input value.  
* @returns { number | BigNumber | Fraction | Complex | Unit | Array |  
* Matrix } The negated value.
```

```
2951     * @memberof PRDC_JSLAB_MATHJS_DOC
2952     */
2953
2954 /**
2955 * Unary plus operation.
2956 * @name unaryPlus
2957 * @kind function
2958 * @param { number | BigNumber | Fraction | Complex | Unit | Array |
2959 *         Matrix } x - Input value.
2960 * @returns { number | BigNumber | Fraction | Complex | Unit | Array |
2961 *           Matrix } The input value.
2962 * @memberof PRDC_JSLAB_MATHJS_DOC
2963 */
2964
2965 /**
2966 * Test whether two values are unequal.
2967 * @name unequal
2968 * @kind function
2969 * @param { * } x - First value.
2970 * @param { * } y - Second value.
2971 * @returns { boolean | Array | Matrix } Returns true if x is not equal to
2972 *         y.
2973 * @memberof PRDC_JSLAB_MATHJS_DOC
2974 */
2975
2976 /**
2977 * Unit constructor.
2978 * @name Unit
2979 * @kind function
2980 * @param { number | BigNumber } value - The numeric value.
2981 * @param { string | Unit } unit - The unit string or Unit object.
2982 * @returns { Unit } A new Unit instance.
2983 * @memberof PRDC_JSLAB_MATHJS_DOC
2984 */
2985
2986 /**
2987 * Create a unit.
2988 * @name unit
2989 * @kind function
2990 * @param { number | BigNumber } value - The numeric value.
2991 * @param { string | Unit } unit - The unit string or Unit object.
2992 * @returns { Unit } A new Unit instance.
2993 * @memberof PRDC_JSLAB_MATHJS_DOC
2994 */
2995
2996 /**
2997 * The base of natural logarithms, e.
2998 * @name E
2999 * @kind member
3000 * @memberof PRDC_JSLAB_MATHJS_DOC
3001 * @type { number }
3002 */
3003
3004 /**
3005 * The mathematical constant pi.
```

```

3003   * @name PI
3004   * @kind member
3005   * @memberof PRDC_JSLAB_MATHJS_DOC
3006   * @type { number }
3007   */
3008
3009 /**
3010  * Solve a linear system  $A * x = b$  where  $A$  is an upper triangular matrix.
3011  * @name usolve
3012  * @kind function
3013  * @param { Matrix | Array } U - An upper triangular matrix.
3014  * @param { Matrix | Array } b - A column vector.
3015  * @returns { Matrix | Array } The solution vector  $x$ .
3016  * @memberof PRDC_JSLAB_MATHJS_DOC
3017  */
3018
3019 /**
3020  * Find all solutions of a linear system  $A * x = b$  where  $A$  is an upper
3021  * triangular matrix.
3022  * @name usolveAll
3023  * @kind function
3024  * @param { Matrix | Array } U - An upper triangular matrix.
3025  * @param { Matrix | Array } b - A column vector.
3026  * @returns { Array } An array of solutions.
3027  * @memberof PRDC_JSLAB_MATHJS_DOC
3028  */
3029 /**
3030  * Vacuum impedance, in ohms.
3031  * @name vacuumImpedance
3032  * @kind member
3033  * @memberof PRDC_JSLAB_MATHJS_DOC
3034  * @type { number }
3035  */
3036
3037 /**
3038  * Compute the variance of a set of values.
3039  * @name variance
3040  * @kind function
3041  * @param { Array | Matrix } array - Input array.
3042  * @param { string } [normalization='unbiased'] - Normalization mode.
3043  * @param { number | BigNumber } [dim] - Dimension along which to compute.
3044  * @returns { number | BigNumber | Array | Matrix } The variance.
3045  * @memberof PRDC_JSLAB_MATHJS_DOC
3046  */
3047
3048 /**
3049  * Weak mixing angle.
3050  * @name weakMixingAngle
3051  * @kind member
3052  * @memberof PRDC_JSLAB_MATHJS_DOC
3053  * @type { number }
3054  */
3055
3056 /**

```

```

3057   * Wien displacement constant , in m K .
3058   * @name wienDisplacement
3059   * @kind member
3060   * @memberof PRDC_JSLAB_MATHJS_DOC
3061   * @type { number }
3062   */
3063
3064 /**
3065   * Extended greatest common divisor for integers .
3066   * @name xgcd
3067   * @kind function
3068   * @param { number | BigNumber } a - An integer .
3069   * @param { number | BigNumber } b - An integer .
3070   * @returns { Array } An array [ gcd , x , y ] satisfying gcd = a*x + b*y .
3071   * @memberof PRDC_JSLAB_MATHJS_DOC
3072   */
3073
3074 /**
3075   * Logical XOR operation .
3076   * @name xor
3077   * @kind function
3078   * @param { boolean | Array | Matrix } x - First value .
3079   * @param { boolean | Array | Matrix } y - Second value .
3080   * @returns { boolean | Array | Matrix } The logical XOR of x and y .
3081   * @memberof PRDC_JSLAB_MATHJS_DOC
3082   */
3083
3084 /**
3085   * Error thrown when an incorrect number of arguments is passed .
3086   * @name ArgumentsError
3087   * @kind function
3088   * @param { string } fn - Function name .
3089   * @param { number } count - Actual argument count .
3090   * @param { number } min - Minimum required arguments .
3091   * @param { number } [max] - Maximum allowed arguments .
3092   * @returns { ArgumentsError } A new ArgumentsError instance .
3093   * @memberof PRDC_JSLAB_MATHJS_DOC
3094   */
3095
3096 /**
3097   * Error thrown when matrix dimensions mismatch .
3098   * @name DimensionError
3099   * @kind function
3100   * @param { number } actual - Actual dimension .
3101   * @param { number } expected - Expected dimension .
3102   * @param { string } [relation] - Relation between dimensions .
3103   * @returns { DimensionError } A new DimensionError instance .
3104   * @memberof PRDC_JSLAB_MATHJS_DOC
3105   */
3106
3107 /**
3108   * Error thrown when an index is out of range .
3109   * @name IndexError
3110   * @kind function
3111   * @param { number } index - The invalid index .

```

```

3112     * @param { number } min - Minimum allowed index .
3113     * @param { number } max - Maximum allowed index .
3114     * @returns { IndexError } A new IndexError instance .
3115     * @memberof PRDC_JSLAB_MATHJS_DOC
3116     */
3117 }
3118 }
```

Listing 97 - mathjs-doc.js

```

1  /**
2   * @file JSLAB library matrix math submodule
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   */
7
8  /**
9   * Class for JSLAB matrix math submodule .
10  */
11 class PRDC_JSLAB_MATRIX_MATH {
12
13 /**
14  * Constructs a matrix math submodule object with access to JSLAB's math
15  * functions .
16  * @constructor
17  * @param {Object} jsl - Reference to the main JSLAB object .
18  */
19 constructor(jsl) {
20     this.jsl = jsl;
21 }
22 /**
23  * Creates a new matrix .
24  * @param {Array} A - The matrix data .
25  * @param {number} rows - Number of rows .
26  * @param {number} cols - Number of columns .
27  * @returns {PRDC_JSLAB_MATRIX} A new matrix instance .
28  */
29 new(A, rows, cols) {
30     return new PRDC_JSLAB_MATRIX(this.jsl, A, rows, cols);
31 }
32 /**
33  * Creates a matrix filled with a specific value .
34  * @param {number} v - The value to fill the matrix with .
35  * @param {number} rows - Number of rows .
36  * @param {number} [cols=rows] - Number of columns .
37  * @returns {PRDC_JSLAB_MATRIX} The filled matrix .
38  */
39 fill(v, rows, cols) {
40     if(!cols) {
41         cols = rows;
42     }
43     return new PRDC_JSLAB_MATRIX(this.jsl,
44         this.jsl.array.createFilledArray(rows * cols, v), rows, cols);
45 }
```

```
46     }
47
48     /**
49      * Creates a matrix filled with NaN values.
50      * @param {number} rows - Number of rows.
51      * @param {number} [cols=rows] - Number of columns.
52      * @returns {PRDC_JSLAB_MATRIX} The NaN-filled matrix.
53      */
54     NaNs(rows, cols) {
55       if(!cols) {
56         cols = rows;
57       }
58       return this.fill(NaN, rows, cols);
59     }
60
61     /**
62      * Creates a matrix filled with ones.
63      * @param {number} rows - Number of rows.
64      * @param {number} [cols=rows] - Number of columns.
65      * @returns {PRDC_JSLAB_MATRIX} The ones-filled matrix.
66      */
67     ones(rows, cols) {
68       if(!cols) {
69         cols = rows;
70       }
71       return this.fill(1, rows, cols);
72     }
73
74     /**
75      * Creates a matrix filled with zeros.
76      * @param {number} rows - Number of rows.
77      * @param {number} [cols=rows] - Number of columns.
78      * @returns {PRDC_JSLAB_MATRIX} The zeros-filled matrix.
79      */
80     zeros(rows, cols) {
81       if(!cols) {
82         cols = rows;
83       }
84       return this.fill(0, rows, cols);
85     }
86
87     /**
88      * Creates a diagonal matrix from an array.
89      * @param {Array} A - The array to create the diagonal matrix from.
90      * @returns {PRDC_JSLAB_MATRIX} The diagonal matrix.
91      */
92     diag(A) {
93       return new PRDC_JSLAB_MATRIX(this.jsl,
94         this.jsl.array.diag(A, A.length), A.length, A.length);
95     }
96
97     /**
98      * Creates an identity matrix of a given size.
99      * @param {number} size - The size of the identity matrix.
100     * @returns {PRDC_JSLAB_MATRIX} The identity matrix.
```

```

101  /*
102   * eye(size) {
103     return new PRDC_JSLAB_MATRIX(this.jsl,
104       this.jsl.array.diag(this.jsl.array.ones(size), size), size, size);
105   }
106
107 /**
108  * Concatenates multiple matrices vertically (row-wise).
109 * @param {...PRDC_JSLAB_MATRIX} args - Matrices to concatenate.
110 * @returns {PRDC_JSLAB_MATRIX} The concatenated matrix.
111 */
112 concatRow(...args) {
113   var N = args.length;
114   var cols = args[0].cols;
115   var rows = args.reduce((a, e) => a += e.rows, 0);
116   var A = new Array(cols * rows).fill(0);
117
118   var p = 0;
119   for(var j = 0; j < cols; j++) {
120     for(var k = 0; k < N; k++) {
121       var P = args[k].data.length / cols;
122       for(var i = 0; i < P; i++) {
123         A[p++] = args[k].data[j * P + i];
124       }
125     }
126   }
127   return this.new(A, rows, cols);
128 }
129
130 /**
131  * Concatenates multiple matrices horizontally (column-wise).
132 * @param {...PRDC_JSLAB_MATRIX} args - Matrices to concatenate.
133 * @returns {PRDC_JSLAB_MATRIX} The concatenated matrix.
134 */
135 concatCol(...args) {
136   var rows = args[0].rows;
137   var A = args.map((a) => a.data).flat();
138   return this.new(A, rows, A.length / rows);
139 }
140
141 /**
142  * Checks if the provided object is a matrix.
143  * @param {Object} A - The object to check.
144  * @returns {boolean} True if A is a matrix, else false.
145 */
146 isMatrix(A) {
147   return A instanceof PRDC_JSLAB_MATRIX;
148 }
149 }
150
151 exports.PRDC_JSLAB_MATRIX_MATH = PRDC_JSLAB_MATRIX_MATH;
152
153 /**
154  * Class for JSLAB matrix.
155 */

```

```
156 class PRDC_JSLAB_MATRIX {
157
158     #jsl;
159     #rows;
160     #cols;
161
162     /**
163      * Constructs a JSLAB matrix.
164      * @constructor
165      * @param {Object} js1 - Reference to the main JSLAB object .
166      * @param {Array} A - The matrix data.
167      * @param {number} rows - Number of rows.
168      * @param {number} cols - Number of columns.
169     */
170     constructor(js1 , A, rows, cols) {
171         this.#jsl = js1;
172         this._set(A, rows, cols);
173     }
174
175     /**
176      * Sets the matrix data and dimensions.
177      * @param {Array} A - The matrix data.
178      * @param {number} rows - Number of rows.
179      * @param {number} cols - Number of columns.
180     */
181     _set(A, rows, cols) {
182         this.rows = rows;
183         this.cols = cols;
184         if(!rows) {
185             this.rows = A.length;
186         }
187         if(!cols) {
188             this.cols = A[0].length;
189         }
190         if(Array.isArray(A[0])) {
191             this.data = this.#jsl.array.transpose(A.flat() , this.rows , this.cols);
192         } else {
193             this.data = [ . . . A];
194         }
195     }
196
197     /**
198      * Extracts a specific column from a matrix.
199      * @param {number} index - The index of the column to extract.
200      * @returns {Array} The extracted column as an array.
201     */
202     column(index) {
203         return this.#jsl.array.column(this.toArray() , index);
204     }
205
206     /**
207      * Extracts a specific row from a matrix.
208      * @param {number} index - The index of the row to extract.
209      * @returns {Array} The extracted row as an array.
210     */
211 }
```

```

211  row(index) {
212    return this.#jsl.array.row(this.toArray(), index);
213  }
214
215  /**
216   * Gets the length of the matrix along a specified dimension.
217   * @param {number} dim - The dimension (0 for rows, 1 for columns).
218   * @returns {number} The length along the specified dimension.
219   */
220  length(dim) {
221    return this.size(dim);
222  }
223
224  /**
225   * Gets the number of elements in the matrix.
226   * @returns {number} The number of elements.
227   */
228  numel() {
229    return this.rows * this.cols;
230  }
231
232  /**
233   * Gets the size of the matrix along a specified dimension.
234   * @param {number} dim - The dimension (0 for rows, 1 for columns).
235   * @returns {number} The size along the specified dimension.
236   */
237  size(dim) {
238    var s = [this.rows, this.cols];
239    if(typeof dim == 'undefined') {
240      return s;
241    }
242    return s[dim];
243  }
244
245  /**
246   * Reshapes the matrix to the specified dimensions.
247   * @param {number} rows - New number of rows.
248   * @param {number} cols - New number of columns.
249   * @returns {PRDC_JSLAB_MATRIX} The reshaped matrix.
250   */
251  reshape(rows, cols) {
252    return this.#jsl.mat.new(this.#jsl.array.reshape(this.data,
253      rows, cols), rows, cols);
254  }
255
256  /**
257   * Replicates the matrix a specified number of times.
258   * @param {number} rowReps - Number of row repetitions.
259   * @param {number} colReps - Number of column repetitions.
260   * @returns {PRDC_JSLAB_MATRIX} The replicated matrix.
261   */
262  repmat(rowReps, colReps) {
263    var cols;
264    if(colReps > 1) {
265      cols = this.#jsl.array.createFilledArray(colReps, this);

```

```

266     cols = this.#jsl.mat.concatCol(...cols);
267 } else {
268     cols = this;
269 }
270
271 var A;
272 if (rowReps > 1) {
273     var rows = this.#jsl.array.createFilledArray(rowReps, cols);
274     A = this.#jsl.mat.concatRow(...rows);
275 } else {
276     A = cols;
277 }
278
279 return A;
280 }
281
282 /**
283 * Transposes the matrix.
284 * @returns {PRDC_JSLAB_MATRIX} The transposed matrix.
285 */
286 transpose() {
287     return this.#jsl.mat.new(this.#jsl.array.transpose(this.data, this.rows,
288         this.cols),
289         this.cols, this.rows);
290 }
291
292 /**
293 * Computes the inverse of the matrix.
294 * @returns {PRDC_JSLAB_MATRIX} The inverse matrix.
295 */
296 inv() {
297     return this.#jsl.mat.new(this.#jsl.env.math.inv(this.toArray()));
298 }
299
300 /**
301 * Computes the determinant of the matrix.
302 * @returns {number} The determinant.
303 */
304 det() {
305     return this.#jsl.env.math.det(this.toArray());
306 }
307
308 /**
309 * Computes the trace of the matrix (sum of diagonal elements).
310 * @returns {number} The trace of the matrix.
311 */
312 trace() {
313     return this.#jsl.env.math.trace(this.toArray());
314 }
315
316 /**
317 * Computes the Frobenius norm of the matrix.
318 * @returns {number} The Frobenius norm.
319 */
320 norm(p = 2) {

```

```

320     return this.#jsl.env.math.norm(this.toArray(), p);
321 }
322
323 /**
324 * Raises the matrix to a power.
325 * @param {number} p - The exponent.
326 * @returns {PRDC_JSLAB_MATRIX} The resulting matrix.
327 */
328 powm(p) {
329   return this.#jsl.mat.new(this.#jsl.env.math.pow(this.toArray(), p));
330 }
331
332 /**
333 * Computes the matrix exponential.
334 * @returns {PRDC_JSLAB_MATRIX} The exponential matrix.
335 */
336 expm() {
337   return this.#jsl.mat.new(this.#jsl.env.math.expm(this.toArray())._data);
338 }
339
340 /**
341 * Adds two matrices.
342 * @param {PRDC_JSLAB_MATRIX} A - The matrix to add.
343 * @returns {PRDC_JSLAB_MATRIX} The resulting matrix.
344 */
345 add(A) {
346   return this.#jsl.mat.new(this.#jsl.array.plus(this.data, A.data),
347     this.rows, this.cols);
348 }
349
350 /**
351 * Adds two matrices (alias for add).
352 * @param {PRDC_JSLAB_MATRIX} A - The matrix to add.
353 * @returns {PRDC_JSLAB_MATRIX} The resulting matrix.
354 */
355 plus(A) {
356   return this.add(A);
357 }
358
359 /**
360 * Subtracts matrix A from the current matrix.
361 * @param {PRDC_JSLAB_MATRIX} A - The matrix to subtract.
362 * @returns {PRDC_JSLAB_MATRIX} The resulting matrix.
363 */
364 subtract(A) {
365   return this.#jsl.mat.new(this.#jsl.array.minus(this.data, A.data),
366     this.rows, this.cols);
367 }
368
369 /**
370 * Subtracts matrix A from the current matrix (alias for subtract).
371 * @param {PRDC_JSLAB_MATRIX} A - The matrix to subtract.
372 * @returns {PRDC_JSLAB_MATRIX} The resulting matrix.
373 */
374 minus(A) {

```

```

375     return this.subtract(A);
376 }
377
378 /**
379 * Multiplies two matrices.
380 * @param {PRDC_JSLAB_MATRIX} A - The matrix to multiply with.
381 * @returns {PRDC_JSLAB_MATRIX} The resulting matrix.
382 */
383 multiply(A) {
384   return this.#jsl.mat.new(this.#jsl.array.multiply(this.data, A.data, this.
385     rows, this.cols, A.cols), this.rows, A.cols);
386 }
387
388 /**
389 * Solves a linear system.
390 * @param {PRDC_JSLAB_MATRIX} B - The right-hand side matrix.
391 * @returns {PRDC_JSLAB_MATRIX} The solution matrix.
392 */
393 linsolve(B) {
394   return this.#jsl.mat.new(this.#jsl.array.linsolve(this.data, B.data, this.
395     cols), this.rows, 1);
396 }
397
398 /**
399 * Divides each element by another matrix or scalar.
400 * @param {PRDC_JSLAB_MATRIX|number} A - The matrix or scalar to divide by.
401 * @returns {PRDC_JSLAB_MATRIX} The resulting matrix.
402 */
403 divideEl(A) {
404   if(this.#jsl.mat.isMatrix(A)) {
405     return this.#jsl.mat.new(this.#jsl.array.divideEl(this.data, A.data),
406       this.rows, this.cols);
407   } else {
408     return this.#jsl.mat.new(this.#jsl.array.scale(this.data, 1 / A),
409       this.rows, this.cols);
410   }
411 }
412
413 /**
414 * Multiplies each element by another matrix or scalar.
415 * @param {PRDC_JSLAB_MATRIX|number} A - The matrix or scalar to multiply by
416 * .
417 * @returns {PRDC_JSLAB_MATRIX} The resulting matrix.
418 */
419 multiplyEl(A) {
420   if(this.#jsl.mat.isMatrix(A)) {
421     return this.#jsl.mat.new(this.#jsl.array.multiplyEl(this.data, A.data),
422       this.rows, this.cols);
423   } else {
424     return this.#jsl.mat.new(this.#jsl.array.scale(this.data, A),
425       this.rows, this.cols);
426   }
427 }
428
429 /**

```

```
427     * Raises each element to a power.  
428     * @param {number} p - The exponent.  
429     * @returns {PRDC_JSLAB_MATRIX} The resulting matrix.  
430     */  
431   powEl(p) {  
432     return this.#jsl.mat.new(this.#jsl.array.powEl(this.data, p),  
433       this.rows, this.cols);  
434   }  
435  
436   /**  
437    * Applies a function to each element of the matrix.  
438    * @param {function} func - The function to apply.  
439    * @returns {PRDC_JSLAB_MATRIX} The resulting matrix.  
440    */  
441   elementWise(func) {  
442     return this.#jsl.mat.new(this.#jsl.array.elementWise((a) => func(a), this.  
443       data),  
444       this.rows, this.cols);  
445   }  
446  
447   /**  
448    * Computes the reciprocal of each element in the matrix.  
449    * @returns {PRDC_JSLAB_MATRIX} The matrix with reciprocals.  
450    */  
451   reciprocal() {  
452     return this.#jsl.mat.new(this.#jsl.array.reciprocal(this.data, this.rows *  
453       this.cols),  
454       this.rows, this.cols);  
455   }  
456  
457   /**  
458    * Computes the sum of all elements in the matrix.  
459    * @returns {number} The sum of all elements.  
460    */  
461   sum() {  
462     return this.#jsl.env.math.sum(this.toArray());  
463   }  
464  
465   /**  
466    * Sorts the elements of the matrix.  
467    * @param {string} [order='asc'] - The order of sorting ('asc' or 'desc').  
468    * @returns {PRDC_JSLAB_MATRIX} The sorted matrix.  
469    */  
470   sort() {  
471     return this.#jsl.env.math.sort(this.data);  
472   }  
473  
474   /**  
475    * Finds the minimum element in the matrix.  
476    * @returns {number} The minimum value.  
477    */  
478   min() {  
479     return this.#jsl.env.math.min(this.toArray());  
480   }
```

```

480  /**
481   * Finds the maximum element in the matrix.
482   * @returns {number} The maximum value.
483   */
484  max() {
485    return this.#jsl.env.math.max(this.toArray());
486  }
487
488 /**
489  * Creates a clone of the current matrix.
490  * @returns {PRDC_JSLAB_MATRIX} A cloned matrix instance.
491  */
492  clone() {
493    return this.#jsl.mat.new(this.data, this.rows, this.cols);
494  }
495
496 /**
497  * Retrieves elements based on row and column indices.
498  * @param {...(number|_)} args - Row and column indices.
499  * @returns {Array} The selected elements.
500  */
501 index(rows_in, cols_in) {
502   var rows;
503   var cols;
504   if(rows_in === _) {
505     rows = range(0, this.rows - 1);
506   } else {
507     rows = rows_in;
508   }
509   if(cols_in === _) {
510     cols = range(0, this.cols - 1);
511   } else {
512     cols = cols_in;
513   }
514   if(!Array.isArray(rows)) {
515     rows = [rows];
516   }
517   if(!Array.isArray(cols)) {
518     cols = [cols];
519   }
520   var A = zeros(rows.length * cols.length);
521
522   var k = 0;
523   for(var j = 0; j < cols.length; j++) {
524     for(var i = 0; i < rows.length; i++) {
525       A[k++] = rows[i] + cols[j] * this.rows;
526     }
527   }
528
529   return A;
530 }
531
532 /**
533  * Sets a subset of the matrix elements.
534  * @param {...*} args - Indices and values to set.

```

```

535  */
536 setSub (... args) {
537   var A = args.map(function(a) {
538     if (!Array.isArray(a) && a !== _) {
539       a = [a];
540     }
541     return a;
542   });
543
544   var indices;
545   var B;
546   if (A.length === 3) {
547     indices = this.index(A[0], A[1]);
548     B = A[2];
549   } else {
550     indices = A[0];
551     if (indices === _) {
552       indices = range(0, this.rows * this.cols - 1);
553     }
554     B = A[1];
555   }
556
557   var j = 0;
558   for (var i = 0; i < indices.length; i++) {
559     if (this.#jsl.mat.isMatrix(B[0])) {
560       this.data[indices[i]] = B[0].data[j++];
561     } else if (Array.isArray(B) && B.length === indices.length) {
562       this.data[indices[i]] = B[j++];
563     } else {
564       this.data[indices[i]] = B[0];
565     }
566   }
567 }
568
569 /**
570  * Gets a subset of the matrix elements.
571  * @param {...*} args - Indices to retrieve.
572  * @returns {PRDC_JSLAB_MATRIX} The subset matrix.
573 */
574 getSub (... args) {
575   var indices;
576   var rows;
577   var cols;
578   var A = args.map(function(a) {
579     if (!Array.isArray(a) && a !== _) {
580       a = [a];
581     }
582     return a;
583   });
584
585   if (A.length === 1) {
586     indices = A[0];
587     rows = indices.length;
588     cols = 1;
589   } else {

```

```

590     if(A[0] === _) {
591         rows = this.rows;
592     } else {
593         rows = A[0].length;
594     }
595     if(A[1] === _) {
596         cols = this.cols;
597     } else {
598         cols = A[1].length;
599     }
600     indices = this.index(A[0], A[1]);
601 }
602
603 var B = this.#jsl.array.createFilledArray(indices.length, 0);
604 var j = 0;
605 for(var i = 0; i < indices.length; i++) {
606     B[j++] = this.data[indices[i]];
607 }
608 return this.#jsl.mat.new(B, rows, cols);
609 }
610
611 /**
612 * Converts the matrix to a two-dimensional array.
613 * @returns {Array} The matrix data as a two-dimensional array.
614 */
615 toArray() {
616     return this.#jsl.array.reshape(this.data, this.rows, this.cols);
617 }
618
619 /**
620 * Converts the matrix to a one-dimensional array.
621 * @returns {Array} The matrix data as a one-dimensional array.
622 */
623 toFlatArray() {
624     return this.data;
625 }
626
627 /**
628 * Returns a string representation of the matrix.
629 * @returns {string} The string representation of the matrix.
630 */
631 toString() {
632     var str = 'Matrix([ \n';
633     for(var i = 0; i < this.rows; i++) {
634         str += '[';
635         for(var j = 0; j < this.cols; j++) {
636             str += this.data[j * this.rows + i] + ', ';
637         }
638         str = str.slice(0, -2);
639         str += '], \n';
640     }
641     str = str.slice(0, -2);
642     str += '\n])';
643     return str;
644 }

```

```

645
646  /**
647   * Returns a JSON representation of the matrix.
648   * @returns {Array} The matrix data as a two-dimensional array.
649   */
650  toJSON() {
651    return this.toArray();
652  }
653
654  /**
655   * Returns a safe JSON representation of the matrix.
656   * @returns {Array} The matrix data as a two-dimensional array.
657   */
658  toSafeJSON() {
659    return this.toJSON();
660  }
661
662  /**
663   * Returns a pretty string representation of the matrix.
664   * @returns {string} The pretty string representation of the matrix.
665   */
666  toPrettyString() {
667    return this.toString();
668  }
669 }
```

Listing 98 - matrix-math.js

```

1 /**
2  * @file JSLAB library networking TCP submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 /**
9  * Class representing a TCP client for handling network communications.
10 */
11 class PRDC_JSLAB_TCP_CLIENT {
12
13 /**
14  * Creates an instance of the TCP client.
15  * @param {Object} jsl Reference to the main JSLAB object.
16  * @param {string} host - The host IP address or hostname to connect to.
17  * @param {number} port - The port number to connect to.
18  * @param {function} onConnectCallback - A callback to execute upon
19  *         successful connection.
20  */
21 constructor(jsl, host, port, onConnectCallback) {
22   var obj = this;
23   this.jsl = jsl;
24   this.host = host;
25   this.port = port;
26   this.onConnectCallback = onConnectCallback;
27   this.onDataCallback;
28   this.onErrorCallback;
```

```

28
29     this.buffer = [];
30     this.active = false;
31
32     this._data_callback = false;
33
34     this.com = this.jsl.env.net.createConnection(port, host);
35     this.com.setTimeout(0);
36     this.com.on('connect', function() {
37         obj._onConnect();
38     });
39     this.com.on('data', function(data) {
40         obj._onData(data);
41     });
42     this.com.on('error', function(err) {
43         obj._onError(err);
44     });
45     this.com.on('close', function(err) {
46         if(err) {
47             obj._onError(err);
48         }
49     });
50     this.com.on('end', function() {
51         obj.active = false;
52     });
53
54     this.jsl.addForCleanup(this, function() {
55         obj.close();
56     });
57 }
58
59 /**
60 * Handles successful connection establishment by setting the client's
61 * active status to true.
62 */
63 _onConnect() {
64     this.active = true;
65     if(this.jsl.format.isFunction(this.onConnectCallback)) {
66         this.onConnectCallback();
67     }
68 }
69 /**
70 * Handles errors by setting the client's active status to false and
71 * possibly logging the error.
72 * @param {Error} err - The error object that was thrown.
73 */
74 _onError(err) {
75     this.active = false;
76     if(this.jsl.format.isFunction(this.onErrorCallback)) {
77         this.onErrorCallback(err);
78     }
79 }
80 /**

```

```
81  * Handles incoming data by appending it to the buffer.
82  * @param {Buffer} data - The incoming data buffer.
83  */
84  _onData(data) {
85      if(this._data_callback) {
86          this.onDataCallback(data);
87      } else {
88          this.buffer.push(...data);
89      }
90  }
91
92 /**
93  * Sets the callback function to handle incoming data events.
94  * @param {Function} callback - The function to be called when data is
95  * received.
96  */
97 setOnData(callback) {
98     if(this.jsl.format.isFunction(callback)) {
99         this.buffer = [];
100        this.onDataCallback = callback;
101        this._data_callback = true;
102    }
103
104 /**
105  * Sets the callback function to handle error events.
106  * @param {Function} callback - The function to be called when an error
107  * occurs.
108  */
109 setError(callback) {
110     if(this.jsl.format.isFunction(callback)) {
111         this.onErrorCallback = callback;
112     }
113
114 /**
115  * Enables or disables keep-alive functionality on the underlying socket.
116  * @param {boolean} [enable=true] - Whether to enable keep-alive.
117  * @param {number} [initialDelay=0] - Delay in milliseconds before the first
118  * keep-alive probe is sent.
119  */
120 setKeepAlive(...args) {
121     this.com.setKeepAlive(...args);
122 }
123
124 /**
125  * Disables the Nagle algorithm, allowing data to be sent immediately.
126  * @param {boolean} [noDelay=true] - Whether to disable the Nagle algorithm.
127  */
128 setNoDelay(...args) {
129     this.com.setNoDelay(...args);
130 }
131
132 /**
133  * Sets the socket timeout for inactivity.
```



```
133     * @param {number} timeout - Timeout in milliseconds.
134     * @param {Function} [callback] - Optional callback triggered on timeout.
135     */
136 setTimeout (...args) {
137     this.com.setTimeout (...args);
138 }
139
140 /**
141 * Reads a specified number of bytes from the buffer.
142 * @param {number} [N=Infinity] - The number of bytes to read. Reads all
143 * available bytes if not specified.
144 * @returns {Buffer} The data read from the buffer.
145 */
146 read(N = Infinity) {
147     N = Math.min(N, this.buffer.length);
148     return this.buffer.splice(0, N);
149 }
150 /**
151 * Writes data to the TCP connection if the client is active.
152 * @param {Buffer|string} data - The data to send over the TCP connection.
153 */
154 write(data) {
155     if(this.active) {
156         this.com.write(data);
157     }
158 }
159
160 /**
161 * Returns the number of bytes available in the buffer.
162 * @returns {number} The number of available bytes.
163 */
164 availableBytes() {
165     return this.buffer.length;
166 }
167
168 /**
169 * Closes the TCP connection and cleans up resources.
170 */
171 close() {
172     this.active = false;
173     if(this.com) {
174         this.com.destroy();
175     }
176 }
177 }
178
179 exports.PRDC_JSLAB_TCP_CLIENT = PRDC_JSLAB_TCP_CLIENT;
180
181 /**
182 * Class representing a TCP server for handling network communications.
183 */
184 class PRDC_JSLAB_TCP_SERVER {
185
186     /**

```

```
187 * Creates an instance of the TCP server.
188 * @param {Object} jsl - Reference to the main JSLAB object.
189 * @param {string} host - The host IP address or hostname to connect to.
190 * @param {number} port - The port number to connect to.
191 * @param {Function} onConnectCallback - Callback executed upon successful
192 * connection.
193 */
194 constructor(jsl, host, port, onConnectCallback) {
195     const obj = this;
196     this.jsl = jsl;
197     this.host = host;
198     this.port = port;
199     this.sockets = {};
200     this.sid = 0;
201
202     this.onConnectCallback = onConnectCallback;
203     this.onDataCallback = null;
204     this.onErrorCallback = null;
205     this.onDisconnectCallback = null;
206
207     this.buffer = [];
208     this.active = false;
209
210     this._data_callback = false;
211
212     this.server = this.jsl.env.net.createServer(function(socket) {
213         obj._onConnect(socket);
214
215         socket.on('data', function(data) {
216             obj._onData(socket, data);
217         });
218
219         socket.on('end', function() {
220             obj._onDisconnect(socket);
221         });
222
223         socket.on('error', function(err) {
224             obj._onError(socket, err);
225         });
226     });
227
228     this.server.listen(this.port, this.host, function() {
229         obj.active = true;
230     });
231
232     this.jsl.addForCleanup(this, function() {
233         obj.close();
234     });
235 }
236 /**
237 * Handles new client connections.
238 * @param {Object} socket - The connected socket.
239 */
240 _onConnect(socket) {
```

```

241     this.sid += 1;
242     this.sockets[ this.sid ] = socket;
243     socket.sid = this.sid;
244
245     if(this.jsl.format.isFunction(this.onConnectCallback)) {
246         this.onConnectCallback(socket);
247     }
248 }
249
250 /**
251 * Handles socket errors.
252 * @param {Object} socket - The socket that encountered an error.
253 * @param {Error} err - The error object.
254 */
255 _onError(socket, err) {
256     if(this.jsl.format.isFunction(this.onErrorCallback)) {
257         this.onErrorCallback(socket, err);
258     }
259 }
260
261 /**
262 * Handles incoming data from sockets.
263 * @param {Object} socket - The socket that received data.
264 * @param {Buffer|string} data - The received data.
265 */
266 _onData(socket, data) {
267     if(this.jsl.format.isFunction(this.onDataCallback)) {
268         this.onDataCallback(socket, data);
269     }
270 }
271
272 /**
273 * Handles socket disconnections.
274 * @param {Object} socket - The socket that disconnected.
275 */
276 _onDisconnect(socket) {
277     if(this.jsl.format.isFunction(this.onDisconnectCallback)) {
278         this.onDisconnectCallback(socket);
279     }
280 }
281
282 /**
283 * Sets the callback function to handle incoming data events.
284 * @param {Function} callback - Function called when data is received.
285 */
286 setOnData(callback) {
287     if(this.jsl.format.isFunction(callback)) {
288         this.onDataCallback = callback;
289     }
290 }
291
292 /**
293 * Sets the callback function to handle error events.
294 * @param {Function} callback - Function called when an error occurs.
295 */

```

```

296   setOnError(callback) {
297     if(this.jsl.format.isFunction(callback)) {
298       this.onErrorCallback = callback;
299     }
300   }
301
302 /**
303 * Sets the callback function to handle disconnection events.
304 * @param {Function} callback - Function called when a socket disconnects.
305 */
306 setOnDisconnect(callback) {
307   if(this.jsl.format.isFunction(callback)) {
308     this.onDisconnectCallback = callback;
309   }
310 }
311
312 /**
313 * Writes data to a specific TCP connection.
314 * @param {Object} socket - The socket to write data to.
315 * @param {Buffer|string} data - The data to send over the TCP connection.
316 */
317 write(socket, data) {
318   if(this.active && this.sockets[socket.sid]) {
319     this.sockets[socket.sid].write(data);
320   }
321 }
322
323 /**
324 * Closes the TCP server and all active connections.
325 */
326 close() {
327   this.active = false;
328   if(this.server) {
329     this.server.close();
330   }
331   for(const sid in this.sockets) {
332     this.sockets[sid].destroy();
333   }
334   this.sockets = {};
335 }
336 }
337
338 exports.PRDC_JSLAB_TCP_SERVER = PRDC_JSLAB_TCP_SERVER;

```

Listing 99 - networking-tcp.js

```

1 /**
2 * @file JSLAB library networking UDP submodule
3 * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4 * PR-DC, Republic of Serbia
5 * info@pr-dc.com
6 */
7
8 /**
9 * Class representing a UDP client for handling network communications.
10 */

```

```

11 class PRDC_JSLAB_UDP {
12
13     /**
14      * Creates an instance of the UDP client.
15      * @param {Object} js1 Reference to the main JSLAB object.
16      * @param {string} host - The host IP address or hostname to connect to.
17      * @param {number} port - The port number to connect to.
18      */
19     constructor(js1, host, port) {
20         var obj = this;
21         this.js1 = js1;
22         this.host = host;
23         this.port = port;
24
25         this.active = false;
26
27         this.com = this.js1.env.udp.createSocket('udp4');
28         this.com.connect(port, host, function(err) {
29             if(err) {
30                 obj._onError(err);
31             } else {
32                 obj._onConnect();
33             }
34         });
35     }
36
37     /**
38      * Handles successful connection establishment by setting the client's
39      * active status to true.
40      */
41     _onConnect() {
42         this.active = true;
43     }
44
45     /**
46      * Handles errors by setting the client's active status to false and
47      * possibly logging the error.
48      * @param {Error} err - The error object that was thrown.
49      */
50     _onError() {
51         this.active = false;
52     }
53
54     /**
55      * Sends data over the UDP connection if the client is active.
56      * @param {Buffer|string} data - The data to send over the UDP connection.
57      */
58     write(data) {
59         if(this.active) {
60             this.com.write(data);
61         }
62     }
63     /**
64      * Closes the UDP connection and cleans up resources.

```



```
64      */
65      close() {
66          var obj = this;
67          this.active = false;
68          this.com.close(function() {
69              delete obj.com;
70          });
71      }
72  }
73
74 exports.PRDC_JSLAB_UDP = PRDC_JSLAB_UDP;
75
76 /**
77 * Represents a UDP server that listens on a specific port.
78 */
79 class PRDC_JSLAB_UDP_SERVER {
80
81 /**
82 * Initializes a UDP server that binds to the specified port and listens for
83 * incoming messages.
84 * @param {Object} js1 Reference to the main JSLAB object.
85 * @param {number} port - The port number on which the server will listen
86 * for incoming UDP packets.
87 */
88 constructor(js1, port) {
89     var obj = this;
90     this.js1 = js1;
91     this.port = port;
92
93     this._data_callback = false;
94
95     this.com = this.js1.env.udp.createSocket('udp4');
96
97     this.com.on('message', function(msg) {
98         obj._onData(msg);
99     });
100
101     this.com.bind(port);
102 }
103
104 /**
105 * Called when data is received. Buffers the incoming data for later
106 * retrieval.
107 * @param {Buffer} data - The received data buffer.
108 */
109 _onData(data) {
110     if(this._data_callback) {
111         this.onDataCallback(data);
112     } else {
113         this.buffer.push(...data);
114     }
115 }
```

```

116  /**
117   * Sets the callback function to handle incoming data events.
118   * @param {Function} callback - The function to be called when data is
119   * received.
120  */
121 setOnData(callback) {
122   if(this.jsl.format.isFunction(callback)) {
123     this.buffer = [];
124     this.onDataCallback = callback;
125     this._data_callback = true;
126   }
127 }
128 /**
129  * Reads a specified number of bytes from the buffer.
130  * @param {number} [N=Infinity] - The maximum number of bytes to read. Reads
131  * all available bytes by default.
132  * @returns {Array} An array containing the first N bytes of buffered data.
133 */
134 read(N = Infinity) {
135   N = Math.min(N, this.buffer.length);
136   return this.buffer.splice(0, N);
137 }
138 /**
139  * Returns the number of bytes available in the buffer.
140  * @returns {number} The number of bytes currently stored in the buffer.
141  */
142 availableBytes() {
143   return this.buffer.length;
144 }
145 /**
146  * Closes the UDP server and releases any resources.
147  */
148 close() {
149   var obj = this;
150   this.com.close(function() {
151     delete obj.com;
152   });
153 }
154 }
155 }
156 }
157 exports.PRDC_JSLAB_UDP_SERVER = PRDC_JSLAB_UDP_SERVER;
```

Listing 100 - networking-udp.js

```

1 /**
2  * @file JSLAB library networking video call submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 /**
9  * Class representing a video call.
```

```

10  /*
11  class PRDC_JSLAB_VIDEOCALL {
12
13  /**
14  * Creates a new video call instance.
15  * @param {string} type - The call type ('server' or 'client').
16  * @param {string} video_source_type - The video source type ('webcam' or
17  *   'desktop').
18  * @param {string} video_id - The video device or source ID.
19  * @param {string} mic_id - The microphone device ID.
20  * @param {string} tcp_host - The TCP host address.
21  * @param {number} tcp_port - The TCP port number.
22  * @param {object} opts - Additional configuration options.
23  */
24  constructor(jsl, type, video_source_type, video_id, mic_id, tcp_host,
25  tcp_port, opts) {
26    var obj = this;
27    this.jsl = jsl;
28    this.type = type;
29    this.video_source_type = video_source_type;
30    this.video_id = video_id;
31    this.mic_id = mic_id;
32    this.host = tcp_host;
33    this.port = tcp_port;
34    this.opts = opts || {};
35    this.timeout = this.opts.timeout || 15000;
36
37    this.is_initiator = (this.type === 'server');
38    this.peer_connection;
39    this.local_stream;
40    this.remote_stream;
41    this.incoming_buffer = '';
42
43    this.jsl.addForCleanup(this, function() {
44      obj.endCall();
45    });
46
47    this._init();
48
49  /**
50  * Attempt to connect the client socket
51  */
52  _connectClient() {
53    var obj = this;
54    this.tcp_client = this.jsl.networking.tcp(this.host, this.port, function()
55    {
56      obj._setupClientHandlers();
57    });
58
59  /**
60  * Start a timeout waiting for signaling messages.
61  */
62  _startConnectionTimeout() {

```

```

62     var obj = this;
63
64     clearTimeoutIf(this.connection_timeout);
65     this.connection_timeout = setTimeout(function() {
66       obj._reconnectClient();
67     }, this.timeout);
68   }
69
70 /**
71 * Attempt to reconnect the client by closing the current connection and
72 * starting a new one.
73 */
74 _reconnectClient() {
75   if(this.tcp_client) {
76     try {
77       this.tcp_client.close();
78     } catch {}}
79   this._connectClient();
80 }
81
82 /**
83 * Initializes the video call by opening a window, setting up the UI,
84 * obtaining media, and creating the peer connection.
85 */
86 async _init() {
87   var obj = this;
88
89   this.win = await openWindowBlank();
90   this.win.setTitle('Video call');
91   this.win.document.body.innerHTML += '<video id="video"></video>';
92   this.dom = this.win.document.getElementById('video');
93   Object.assign(this.dom.style, {
94     position: 'absolute',
95     top: '50%',
96     left: '50%',
97     transform: 'translate(-50%, -50%)',
98     width: '100%',
99     height: '100%',
100    objectFit: 'contain',
101    background: 'url("../img/no-video.svg") center / 30% no-repeat',
102  });
103  await this._getLocalMedia();
104  this._createPeerConnection();
105
106  if(this.is_initiator) {
107    this.tcp_server = this.jsl.networking.tcpServer(this.host, this.port,
108      function(socket) {
109        obj._setupServerSocketHandlers(socket);
110      });
111  } else {
112    this._connectClient();
113  }

```

```

114  /**
115   * Sets up handlers for the server-side socket once a client connects.
116   * @param {Object} socket - The connected socket.
117   */
118 _setupServerSocketHandlers(socket) {
119     var obj = this;
120     this.server_socket = socket;
121     this.tcp_server.setOnData(function(socket, data) {
122       if(socket.sid === obj.server_socket.sid) {
123         obj._processIncomingData(data);
124       }
125     });
126   }
127
128 /**
129  * Set up handlers for the client side once connected to server.
130  */
131 _setupClientHandlers() {
132   var obj = this;
133   this.tcp_client.setOnData(function(data) {
134     obj._processIncomingData(data);
135   });
136
137   this._sendSignalingMessage({ type: 'request-offer' });
138   this._startConnectionTimeout();
139 }
140
141 /**
142  * Processes incoming data over TCP.
143  * @param {string} data - The incoming data as a string.
144  */
145 _processIncomingData(data) {
146   this.incoming_buffer += data.toString('utf8');
147   let lines = this.incoming_buffer.split('\0');
148   while(lines.length > 1) {
149     let line = lines.shift().trim();
150     if(line) {
151       try {
152         let msg = JSON.parse(line);
153         this._handleSignalingMessage(msg);
154       } catch {}}
155     }
156   }
157
158   this.incoming_buffer = lines[0];
159 }
160
161 /**
162  * Sends a signaling message as JSON via TCP.
163  * @param {Object} msg - The signaling message to send.
164  */
165 _sendSignalingMessage(msg) {
166   const json_str = JSON.stringify(msg) + '\0';
167   if(this.is_initiator) {
168     if(this.server_socket) {

```



```

222         ... video_opts
223     }
224   };
225 }
226
227 if (!this.mic_id) {
228   constraints.audio = false;
229 } else {
230   constraints.audio = {
231     deviceId: { exact: this.mic_id },
232     ... mic_opts
233   };
234 }
235
236 try {
237   if (constraints.audio || constraints.video) {
238     this.local_stream = await this.jsl.env.navigator.mediaDevices.
239       getUserMedia(constraints);
240   } else {
241     this.local_stream = new MediaStream();
242   }
243 } catch {
244   this.jsl.env.error('@videocall: '+language.string(222));
245 }
246
247 /**
248 * Starts the call by creating and sending an offer if the instance is the
249 * initiator.
250 */
251 async _startCall() {
252   if (!this.is_initiator) return;
253   const offer = await this.peer_connection.createOffer({offerToReceiveVideo:
254     true, offerToReceiveAudio: true});
255   await this.peer_connection.setLocalDescription(offer);
256   this._sendSignalingMessage({ type: 'offer', sdp: this.peer_connection.
257     localDescription });
258 }
259
260 /**
261 * Answers an incoming offer by creating and sending an answer if the
262 * instance is not the initiator.
263 */
264 async _answerCall() {
265   const answer = await this.peer_connection.createAnswer();
266   await this.peer_connection.setLocalDescription(answer);
267   this._sendSignalingMessage({ type: 'answer', sdp: this.peer_connection.
268     localDescription });
269 }
270
271 /**
272 * Handles incoming signaling messages based on their type.
273 * @param {Object} message - The signaling message received.
274 */
275 async _handleSignalingMessage(message) {

```

```

271     if (!this.is_initiator) {
272         this.connection_timeout = clearTimeoutIf(this.connection_timeout);
273     }
274
275     switch (message.type) {
276         case 'request-offer':
277             if (this.is_initiator) {
278                 await this._startCall();
279             }
280             break;
281         case 'offer':
282             if (!this.is_initiator) {
283                 await this.peer_connection.setRemoteDescription(message.sdp);
284                 await this._answerCall();
285             }
286             break;
287         case 'answer':
288             if (this.is_initiator) {
289                 await this.peer_connection.setRemoteDescription(message.sdp);
290             }
291             break;
292         case 'candidate':
293             this.peer_connection.addIceCandidate(message.candidate);
294             break;
295         case 'message':
296             if (typeof this._onMessage === 'function') {
297                 this._onMessage(message.data);
298             } else {
299                 disp(message.data);
300             }
301             break;
302     }
303 }
304
305 /**
306 * Sets a callback function to handle incoming messages.
307 * @param {Function} callback - The function to call when a message is
308 * received.
309 */
310 setOnMessage(callback) {
311     if (typeof callback === 'function') {
312         this._onMessage = callback;
313     }
314 }
315 /**
316 * Sends a message to the connected peer.
317 * @param {any} data - The data to send.
318 */
319 sendMessage(data) {
320     this._sendSignalingMessage({ type: 'message', data: data });
321 }
322
323 /**
324 * Toggles the local audio track on or off.

```

```

325     * @param {boolean} mute - If true, mutes the audio; otherwise, unmutes.
326     */
327     toggleAudio(mute) {
328       if(this.local_stream) {
329         this.local_stream.getAudioTracks().forEach((track) => track.enabled = !mute);
330       }
331     }
332
333   /**
334    * Toggles the local video track on or off.
335    * @param {boolean} disable - If true, disables the video; otherwise,
336    * enables it.
337   */
338   toggleVideo(disable) {
339     if(this.local_stream) {
340       this.local_stream.getVideoTracks().forEach((track) => track.enabled = !disable);
341     }
342
343   /**
344    * Ends the call by closing peer connections and media streams.
345   */
346   endCall() {
347     if(!this.is_initiator) {
348       this.connection_timeout = clearTimeoutIf(this.connection_timeout);
349     }
350
351     if(this.peer_connection) {
352       this.peer_connection.close();
353     }
354     if(this.local_stream) {
355       this.local_stream.getTracks().forEach((track) => track.stop());
356     }
357     if(this.remote_stream) {
358       this.remote_stream.getTracks().forEach((track) => track.stop());
359     }
360
361     if(this.tcp_server) {
362       this.tcp_server.close();
363     }
364     if(this.tcp_client) {
365       this.tcp_client.close();
366     }
367   }
368 }
369
370 exports.PRDC_JSLAB_VIDEOCALL = PRDC_JSLAB_VIDEOCALL;

```

Listing 101 - networking-videocall.js

```

1 /**
2  * @file JSLAB library networking submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia

```

```

5   * info@pr-dc.com
6   */
7
8 var { PRDC_JSLAB_TCP_CLIENT, PRDC_JSLAB_TCP_SERVER } = require('./networking-
  tcp');
9 var { PRDC_JSLAB_UDP, PRDC_JSLAB_UDP_SERVER } = require('./networking-udp');
10 var { PRDC_JSLAB_VIDEOCALL } = require('./networking-video-call');

11 /**
12  * Class for JSLAB networking submodule.
13  */
14 class PRDC_JSLAB_LIB_NETWORKING {

15 /**
16  * Create submodule object.
17  * @param {Object} js1 Reference to the main JSLAB object.
18  */
19 constructor(js1) {
20   var obj = this;
21   this.js1 = js1;

22 /**
23  * XMODEM CRC table
24  * @type {Array}
25  */
26 this.CRC_TABLE_XMODEM = [
27   0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
28   0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
29   0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
30   0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
31   0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
32   0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
33   0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
34   0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
35   0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
36   0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
37   0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0xa50, 0x3a33, 0x2a12,
38   0xdbfd, 0xcbdc, 0xfbff, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
39   0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
40   0xedae, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
41   0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
42   0xff9f, 0xefbe, 0xdfdd, 0cfffc, 0xbff1b, 0xaf3a, 0x9f59, 0x8f78,
43   0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
44   0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
45   0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
46   0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
47   0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
48   0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
49   0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
50   0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
51   0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
52   0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
53   0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabb8, 0xbb9a,
54   0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ado, 0x2ab3, 0x3a92,
55   0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,
56
57
58

```

```

59      0x7c26 , 0x6c07 , 0x5c64 , 0x4c45 , 0x3ca2 , 0x2c83 , 0x1ce0 , 0x0cc1 ,
60      0xef1f , 0xff3e , 0xcf5d , 0xdf7c , 0xaf9b , 0xbfbfa , 0x8fd9 , 0x9ff8 ,
61      0x6e17 , 0x7e36 , 0x4e55 , 0x5e74 , 0x2e93 , 0x3eb2 , 0x0ed1 , 0x1ef0 ,
62    ];
63  }
64
65 /**
66 * Checks if the environment is currently online.
67 * @returns {boolean} ‘true’ if online, otherwise ‘false’.
68 */
69 isOnline() {
70   return this.jsl.env.online;
71 }
72
73 /**
74 * Calculates the CRC-16/XMODEM checksum of a byte array.
75 * @param {Uint8Array} byte_array - The input data as a byte array.
76 * @returns {number} The CRC-16/XMODEM checksum as a numeric value.
77 */
78 crc16xmodem(byte_array) {
79   let crc = 0x0000;
80   for(let i = 0; i < byte_array.length; i++) {
81     crc = (crc << 8) ^ this.CRC_TABLE_XMODEM[((crc >> 8) ^ byte_array[i]) &
82           0xFF];
83   }
84   return crc & 0xFFFF;
85 }
86 /**
87 * Retrieves the primary IPv4 address of the current machine.
88 * @returns {string} The IP address if found, otherwise an empty string.
89 */
90 getIP() {
91   return Object.values(this.jsl.env.os.networkInterfaces())
92     .reduce(function(r, list) { return r.concat(
93       list.reduce(function(rr, i) { return rr.concat(
94         i.family === 'IPv4' && !i.internal && i.address || []); }, []);
95     ), []];
96 }
97 /**
98 * Attempts to establish a TCP connection to the specified host and port to
99 * check reachability.
100 * @param {string} host - The IP address or hostname to ping.
101 * @param {number} port - The port number to use for the connection.
102 * @param {number} [timeout=1000] - The timeout in milliseconds before the
103 * attempt is considered failed.
104 * @returns {Promise<boolean>} A promise that resolves to ‘true’ if the
105 * connection is successful, ‘false’ otherwise.
106 */
107 async pingAddressTCP(host, port, timeout = 1000) {
108   var obj = this;
109   return new Promise(function(resolve) {
110     const socket = obj.jsl.env.net.createConnection(port, host);
111     socket.setTimeout(timeout);

```

```

109     socket.on('connect', function() {
110         socket.end();
111         resolve(true);
112     });
113     socket.on('timeout', function() {
114         socket.destroy();
115         resolve(false);
116     });
117     socket.on('error', function() {
118         socket.destroy();
119         resolve(false);
120     });
121 });
122 }
123
124 /**
125 * Executes a ping command to check if an IP address is reachable.
126 * @param {string} host - The IP address or hostname to ping.
127 * @param {number} timeout - The timeout in milliseconds for the ping
128   command.
129 */
130 async pingAddress(host, timeout) {
131     return new Promise((resolve) => {
132         exec('ping -n 1 -w '+timeout+' '+host, function(error, stdout, stderr) {
133             if(error || stderr) {
134                 resolve(false);
135             } else {
136                 resolve(stdout.includes('Reply from'));
137             }
138         });
139     })
140 }
141 /**
142 * Executes a ping command synchronized to check if an IP address is
143   reachable.
144 * @param {string} host - The IP address or hostname to ping.
145 * @param {number} timeout - The timeout in milliseconds for the ping
146   command.
147 */
148 pingAddressSync(host, timeout) {
149     try {
150         var stdout = execSync('ping -n 1 -w '+timeout+' '+host);
151         return stdout.includes('Reply from');
152     } catch {
153         return false;
154     }
155 }
156 /**
157 * Finds the first unused port within a specified range, checking
158   sequentially from 'port' to 'max_port'.
159 * @param {number} port - The starting port number to check.
160 * @param {number} min_port - The minimum port number in the range.
161 * @param {number} max_port - The maximum port number in the range.

```

```

160     */
161     async findFirstUnusedPort(port, min_port, max_port) {
162         let currentPort = port;
163
164         while(true) {
165             const inUse = await this.jsl.env.tcpPortUsed.check(port);
166             if(!inUse) {
167                 return currentPort;
168             }
169
170             currentPort++;
171             if(currentPort > max_port) {
172                 currentPort = min_port;
173             }
174         }
175     }
176
177 /**
178 * Converts an IPv4 address to its decimal equivalent.
179 * @param {string} ip - The IPv4 address in dot-decimal notation.
180 * @param {number} [subnets=4] - The number of subnets in the IP address,
181 *     default is 4.
182 * @returns {number} The decimal equivalent of the IPv4 address.
183 */
184 ip2dec(ip, subnets = 4) {
185     const octets = ip.split('.').map(Number).reverse();
186     let value = 0;
187     for(let i = 0; i < subnets; i++) {
188         value += octets[i]*Math.pow(256,i);
189     }
190     return value;
191 }
192 /**
193 * Creates a TCP client for communication with a specified host and port.
194 * @param {string} host - The hostname or IP address to connect to.
195 * @param {number} port - The port number on the host to connect to.
196 * @param {function} onConnectCallback - A callback function that is called
197 *     when the connection is successfully established.
198 * @returns {PRDC_JSLAB_TCP_CLIENT} An instance of the TCP client with event
199 *     handlers set up.
200 */
201 tcp(host, port, onConnectCallback) {
202     return new PRDC_JSLAB_TCP_CLIENT(this.jsl, host, port, onConnectCallback);
203 }
204
205 /**
206 * Creates a TCP server to listen on a specified port.
207 * @param {string} host - The hostname or IP address.
208 * @param {number} port - The port number.
209 * @param {function} onConnectCallback - A callback function that is called
210 *     when the connection is successfully established.
211 * @returns {PRDC_JSLAB_TCP_SERVER} An instance of the TCP server.
212 */
213 tcpServer(...args) {

```

```

211     return new PRDC_JSLAB_TCP_SERVER(this.jsl, ... args);
212 }
213
214 /**
215 * Creates a UDP client for sending data to a specified host and port.
216 * @param {string} host - The hostname or IP address to connect to.
217 * @param {number} port - The port number to connect to.
218 * @returns {PRDC_JSLAB_UDP} An instance of the UDP client.
219 */
220 udp(host, port) {
221   return new PRDC_JSLAB_UDP(this.jsl, host, port);
222 }
223
224 /**
225 * Creates a UDP server to listen on a specified port.
226 * @param {number} port - The port number to listen on.
227 * @returns {PRDC_JSLAB_UDP_SERVER} An instance of the UDP server.
228 */
229 udpServer(port) {
230   return new PRDC_JSLAB_UDP_SERVER(this.jsl, port);
231 }
232
233 /**
234 * Initializes and returns a new video call instance.
235 * @param {string} type - The call type ('server' or 'client').
236 * @param {string} video_source_type - The video source type ('webcam' or
237 *   'desktop').
238 * @param {string} video_id - The video device or source ID.
239 * @param {string} mic_id - The microphone device ID.
240 * @param {string} tcp_host - The TCP host address.
241 * @param {number} tcp_port - The TCP port number.
242 * @param {object} opts - Additional configuration options.
243 * @returns {PRDC_JSLAB_VIDEOCALL} The created video call object.
244 */
245 videoCall(type, video_source_type, video_id, mic_id, tcp_host, tcp_port,
246   opts) {
247   return new PRDC_JSLAB_VIDEOCALL(this.jsl, type, video_source_type,
248     video_id, mic_id, tcp_host, tcp_port, opts);
249 }
250 exports.PRDC_JSLAB_LIB_NETWORKING = PRDC_JSLAB_LIB_NETWORKING;

```

Listing 102 - networking.js

```

1 /**
2 * @file JSLAB library non blocking functions submodule
3 * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4 * PR-DC, Republic of Serbia
5 * info@pr-dc.com
6 */
7
8 /**
9 * Class for JSLAB non blocking functions submodule.
10 */
11 class PRDC_JSLAB_LIB_NON_BLOCKING {

```

```

12
13 /**
14 * Initializes a new instance of the non-blocking functions submodule.
15 * @param {Object} js1 Reference to the main JSLAB object.
16 */
17 constructor(js1) {
18     var obj = this;
19     this.js1 = js1;
20 }
21
22 /**
23 * Executes a given function in a non-blocking while loop.
24 * @param {Function} fn A function that returns a boolean value; when false,
25 *   the loop exits.
26 */
27 nbwhile(fn) {
28     var obj = this;
29     function fnw() {
30         if (!obj.js1.basic.checkStopLoop()) {
31             if (!fn()) {
32                 obj.js1.env.setImmediate(fnw);
33             }
34         } else {
35             obj.js1.env.error('@nbwhile: '+language.string(125));
36         }
37     }
38     this.js1.env.setImmediate(fnw);
39 }
40
41 /**
42 * Executes a given function once in a non-blocking manner.
43 * @param {Function} fn The function to be executed.
44 */
45 nbrun(fn) {
46     if (!this.js1.basic.checkStopLoop()) {
47         this.js1.env.setImmediate(fn);
48     } else {
49         this.js1.env.error('@nbrun: '+language.string(125));
50     }
51 }
52
53 /**
54 * Schedules the next block of code to be executed in a non-blocking manner.
55 * @param {Function} fn The function to execute next.
56 */
57 nbnext(fn) {
58     var obj = this;
59     function fnw() {
60         if (!obj.js1.basic.checkStopLoop()) {
61             fn();
62         } else {
63             obj.js1.env.error('@nbnext: '+language.string(125));
64         }
65     }
66     this.js1.env.setImmediate(fnw);

```

```

66   }
67
68  /**
69   * Waits for a specified number of milliseconds in a non-blocking manner.
70   * @param {Number} ms The number of milliseconds to wait.
71   * @returns {Promise<void>} A promise that resolves after the specified time
72   * has elapsed.
73   */
74  waitMSeconds(ms) {
75    if(!this.jsl.basic.checkStopLoop()) {
76      return new Promise(function(resolve, reject) { setTimeout(resolve, ms)
77        });
78    } else {
79      this.jsl.env.error('@waitMSeconds: '+language.string(125), true);
80    }
81    return false;
82  }
83
84  /**
85   * Waits for a specified number of seconds in a non-blocking manner.
86   * @param {Number} s The number of seconds to wait.
87   * @returns {Promise<void>} A promise that resolves after the specified time
88   * has elapsed.
89   */
90  waitSeconds(s) {
91    if(!this.jsl.basic.checkStopLoop()) {
92      return waitMSeconds(s*1000);
93    } else {
94      this.jsl.env.error('@waitSeconds: '+language.string(125), true);
95    }
96    return false;
97  }
98
99  /**
100   * Waits for a specified number of minutes in a non-blocking manner.
101   * @param {Number} min The number of minutes to wait.
102   * @returns {Promise<void>} A promise that resolves after the specified time
103   * has elapsed.
104   */
105  waitMinutes(min) {
106    if(!this.jsl.basic.checkStopLoop()) {
107      return waitMSeconds(min*60*1000);
108    } else {
109      this.jsl.env.error('@waitMinutes: '+language.string(125), true);
110    }
111    return false;
112  }
113
114  /**
115   * Clears the specified interval if it exists.
116   * @param {number|undefined} timeout - The interval ID to be cleared.
117   * @returns {boolean} Always returns false.
118   */
119  clearIntervalIf(timeout) {
120    if(timeout) {

```

```

117         clearInterval(timeout);
118     }
119     return false;
120 }
121
122 /**
123 * Clears the specified timeout if it exists.
124 * @param {number|undefined} timeout - The timeout ID to be cleared.
125 * @returns {boolean} Always returns false.
126 */
127 clearTimeoutIf(timeout) {
128   if(timeout) {
129     clearTimeout(timeout);
130   }
131   return false;
132 }
133
134 /**
135 * Initializes a new worker with the specified module path.
136 * @param {string} path - The path to the module to configure the worker.
137 * @returns {Worker} The initialized Worker instance.
138 */
139 initWorker(path) {
140   var worker = new Worker(app_path + "/js/sandbox/init-worker.js");
141   worker.postMessage({
142     type: 'configureWorker',
143     module_path: path
144   });
145   return worker;
146 }
147 }
148
149 exports.PRDC_JSLAB_LIB_NON_BLOCKING = PRDC_JSLAB_LIB_NON_BLOCKING;

```

Listing 103 - non-blocking.js

```

1 /**
2  * @file JSLAB_OpenModelicaLink submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 const zmq = require('zeromq');
9 const fs = require('fs');
10 const path = require('path');
11 const os = require('os');
12 const { exec, execSync } = require('child_process');
13 const { XMLParser } = require('fast-xml-parser');
14
15 /**
16  * Class for JSLAB_OpenModelicaLink.
17 */
18 class PRDC_JSLAB_OPENMODELICA_LINK {
19
20 /**

```

```

21  * Initializes a new instance of the OpenModelicaLink.
22  * @param {Object} js1 Reference to the main JSLAB object.
23  */
24 constructor(js1) {
25   this.js1 = js1;
26 }
27
28 /**
29  * Starts the interaction with an external executable by initializing the
30  * necessary environment and parameters.
31  * Launches the executable with the appropriate command-line arguments for
32  * interaction via ZMQ.
33  * Waits for a port file to be created to establish the ZMQ communication.
34  * @param {string} exe - Path to the executable to be run, defaults to the
35  * OpenModelica compiler if not provided.
36  */
37
38 async start(exe) {
39   var obj = this;
40   this.exe = exe;
41
42   this.pid = 0;
43   this.active = false;
44   this.requester = null;
45   this.portfile = '';
46
47   this.filename = '';
48   this.modelname = '';
49   this.xmlfile = '';
50   this.resultfile = '';
51   this.csvfile = '';
52   this.mat_temp_dir = '';
53   this.simulation_options = {};
54   this.quantities_list = [];
55   this.parameter_list = {};
56   this.continuous_list = {};
57   this.input_list = {};
58   this.output_list = {};
59   this.mapped_names = {};
60   this.override_variables = {};
61   this.sim_opt_override = {};
62   this.input_flag = false;
63   this.linear_options = {
64     startTime: '0.0',
65     stopTime: '1.0',
66     numberofIntervals: '500',
67     stepSize: '0.002',
68     tolerance: '1e-6'
69   };
70   this.linearfile = '';
71   this.linear_flag = false;
72   this.linear_modelname = '';
73   this.linear_inputs = '';
74   this.linear_outputs = '';
75   this.linear_states = '';
76   this.linear_quantity_list = []
77 }
```

```

73
74  const random_string = Math.random().toString(36).substring(7);
75  let cmd, portfile;
76
77  if(process.platform === 'win32') {
78    exe = exe || path.join(process.env.OPENMODELICAHOME, 'bin', 'omc.exe');
79    cmd = `START /b "" "${exe}" --interactive= zmq +z=jslab.${random_string}`;
80    portfile = path.join(os.tmpdir(), 'openmodelica.port.jslab.${random_string}`);
81  } else {
82    exe = exe || 'omc';
83    cmd = `${exe} --interactive= zmq -z=jslab.${random_string} &`;
84    portfile = path.join(os.tmpdir(), `openmodelica.${process.env.USER}.port.jslab.${random_string}`);
85  }
86
87  this.portfile = portfile;
88
89  var [flag1, pids1] = isProgramRunning('omc.exe');
90  var omc_process = exec(cmd);
91  await waitMSeconds(200);
92  var [flag2, pids2] = isProgramRunning('omc.exe');
93  this.pid = pids2.filter(function(e) {
94    return !pids1.includes(e)
95  });
96
97  while(true) {
98    await waitMSeconds(10);
99    if(fs.existsSync(this.portfile)) {
100      const filedata = fs.readFileSync(this.portfile, 'utf-8');
101      this.requester = new zmq.Request();
102      this.requester.connect(filedata);
103      this.active = true;
104      break;
105    }
106  }
107}
108
109 /**
110 * Sends an expression to be evaluated by the external executable through
111 * the ZMQ connection and waits for the result.
112 * Parses the response using a dedicated expression parser.
113 * @param {string} expr - The expression to be evaluated.
114 * @returns {Promise<any>} - A promise that resolves with the parsed result
115 * of the expression evaluation.
116 * @throws {Error} - Throws an error if there is no active connection.
117 */
118 async sendExpression(expr) {
119   if(this.active) {
120     await this.requester.send(expr);
121     const [result] = await this.requester.receive();
122     return this.parseExpression(result.toString());
123   } else {
124     this.jsl.env.error(`@sendExpression: ${language.string(201)})`);
125   }
126 }

```

```

123         return false;
124     }
125 }
126 /**
127 * Initializes and configures a Modelica system with the specified
128   parameters and libraries.
129 * Loads necessary files and prepares the environment for simulation.
130 * @param {string} filename - The path to the Modelica file.
131 * @param {string} modelName - The name of the Modelica model.
132 * @param {string[]} [libraries=[]] - An array of library paths to load.
133 * @param {string} [command_line_options=''] - Additional command-line
134   options for the simulation.
135 */
136 async ModelicaSystem(filename, modelName, libraries = [], command_line_options = '') {
137   if(!filename || !modelName) {
138     this.jsl.env.error('@ModelicaSystem: '+language.string(203));
139     return false;
140   }
141   if(command_line_options) {
142     const cmd_exp = await this.sendExpression(`setcommand_line_options("${{
143       command_line_options}}")`);
144     if(cmd_exp === 'false') {
145       this.jsl.env.error('@ModelicaSystem: '+ await this.sendExpression(`

146       getErrorString()`));
147       return false;
148     }
149   }
150   const filepath = path.normalize(filename).replace(/\\/g, '/');
151   const load_file_msg = await this.sendExpression(`loadFile("${{filepath}}")`)
152   ;
153   if(load_file_msg === 'false') {
154     this.jsl.env.error('@ModelicaSystem: '+ await this.sendExpression(`

155       getErrorString()`));
156     return false;
157   }
158   for(const lib of libraries) {
159     let libmsg;
160     if(fs.existsSync(lib)) {
161       libmsg = await this.sendExpression(`loadFile("${{lib}}")`);
162     } else {
163       libmsg = await this.sendExpression(`loadModel(${lib})`);
164     }
165     if(libmsg === 'false') {
166       this.jsl.env.error('@ModelicaSystem: '+ await this.sendExpression(`

167       getErrorString()`));
168       return false;
169     }
170   }
171   this.filename = filename;

```

```

170     this.modelname = modelname;
171     this.mat_temp_dir = fs.mkdtempSync(path.join(os.tmpdir() , ' 
172         OpenModelicaLink-));
173     await this.sendExpression(`cd(" + this.mat_temp_dir + ")`);
174     await this.BuildModelicaModel();
175     return true;
176 }
177 /**
178 * Builds the Modelica model by sending the appropriate build command and
179     parsing the resulting XML file.
180 */
181 async BuildModelicaModel() {
182     const build_model_result = await this.sendExpression(`buildModel(${this.
183         modelname})`);
184     if(!build_model_result) {
185         this.jsl.env.error(`@BuildModelicaModel: ${await this.sendExpression(` 
186             getErrorString()`)}`);
187         return false;
188     }
189     this.xmlfile = path.join(this.mat_temp_dir, build_model_result[1]);
190     this.xmlparse();
191     return true;
192 }
193 /**
194 * Retrieves the working directory used for temporary files and simulations.
195 * @returns {string} - The path to the working directory.
196 */
197 getWorkDirectory() {
198     return this.mat_temp_dir;
199 }
200 /**
201 * Parses the XML file generated by the Modelica compiler to extract
202     simulation parameters and variables.
203 */
204 xmlparse() {
205     if(fs.existsSync(this.xmlfile)) {
206         const xml_data = fs.readFileSync(this.xmlfile, 'utf-8');
207         const parser = new XMLParser({ ignoreAttributes: false ,
208             attributeNamePrefix: '' });
209         const xDoc = parser.parse(xml_data);
210
211         // default_experiment
212         const default_experiment = xDoc.fmiModelDescription.DefaultExperiment;
213         if(default_experiment) {
214             this.simulation_options.startTime = default_experiment.startTime || ' 
215                 0.0';
216             this.simulation_options.stopTime = default_experiment.stopTime || '1.0 
217                 ';
218             this.simulation_options.stepSize = default_experiment.stepSize || ' 
219                 0.002';
220             this.simulation_options.tolerance = default_experiment.tolerance || '1

```

```

216     e-6';
217     this.simulation_options.solver = default_experiment.solver || '';
218 }
219
220 // scalar_variables
221 const scalar_variables = xDoc.fmiModelDescription.ModelVariables.
222     ScalarVariable || [];
223 const fields = ['name', 'isValueChangeable', 'description', 'variability',
224     'causality', 'alias', 'aliasVariable'];
225 for(let k = 0; k < scalar_variables.length; k++) {
226     const item = scalar_variables[k];
227     let scalar = {};
228     fields.forEach(field => {
229         scalar[field] = item[field] || '';
230     });
231
232     if(item.Real) {
233         scalar.value = item.Real.start || '';
234     }
235 }
236 } else {
237     this.jsl.env.error('@xmlparse: '+language.string(204));
238     return false;
239 }
240 return true;
241 }

242 /**
243 * Processes a scalar variable from the XML file and categorizes it based on
244 * its properties.
245 * @param {Object} scalar - The scalar variable to process.
246 */
247 processVariable(scalar) {
248     const name = scalar.name;
249     const value = scalar.value;
250
251     if(!this.linear_flag) {
252         if(scalar.variability === 'parameter') {
253             this.parameter_list[name] = value;
254         } else if(scalar.variability === 'continuous') {
255             this.continuous_list[name] = value;
256         } else if(scalar.causality === 'input') {
257             this.input_list[name] = value;
258         } else if(scalar.causality === 'output') {
259             this.output_list[name] = value;
260         }
261     }
262
263     if(this.linear_flag) {
264         if(scalar.alias === 'alias') {
265             if(name.startsWith('x')) {
266                 this.linear_states.push(name.slice(3, -1));
267             } else if(name.startsWith('u')) {
268             this.linear_states.push(name);
269         }
270     }

```

```

267         this.linear_inputs.push(name.slice(3, -1));
268     } else if(name.startsWith('y')) {
269         this.linear_outputs.push(name.slice(3, -1));
270     }
271 }
272 this.linear_quantity_list.push(scalar);
273 } else {
274     this.quantities_list.push(scalar);
275 }
276 }
277
278 /**
279 * Retrieves a list of quantities based on the provided arguments.
280 * @param {string[]} [args] - An array of quantity names to retrieve. If
281 * omitted, returns all quantities.
282 * @returns {Object[]} - An array of quantity objects.
283 */
284 getQuantities(args) {
285     if(args && args.length > 0) {
286         const tmpresult = [];
287         for(let n = 0; n < args.length; n++) {
288             for(let q = 0; q < this.quantities_list.length; q++) {
289                 if(this.quantities_list[q].name === args[n]) {
290                     tmpresult.push(this.quantities_list[q]);
291                 }
292             }
293             return tmpresult;
294         } else {
295             return this.quantities_list;
296         }
297     }
298 }
299 /**
300 * Retrieves a list of linearized quantities based on the provided arguments
301 * @param {string[]} [args] - An array of linear quantity names to retrieve.
302 * If omitted, returns all linear quantities.
303 * @returns {Object[]} - An array of linear quantity objects.
304 */
305 getLinearQuantities(args) {
306     if(args && args.length > 0) {
307         const tmpresult = [];
308         for(let n = 0; n < args.length; n++) {
309             for(let q = 0; q < this.linear_quantity_list.length; q++) {
310                 if(this.linear_quantity_list[q].name === args[n]) {
311                     tmpresult.push(this.linear_quantity_list[q]);
312                 }
313             }
314             return tmpresult;
315         } else {
316             return this.linear_quantity_list;
317         }
318     }

```

```

319
320  /**
321   * Retrieves simulation parameters based on the provided arguments.
322   * @param {string|string[]} [args] - A single parameter name or an array of
323   * parameter names to retrieve. If omitted, returns all parameters.
324   * @returns {Object|any} - An object containing the requested parameters or
325   * a single parameter value.
326   */
327   getParameters(args) {
328     if(args && args.length > 0) {
329       if(Array.isArray(args)) {
330         const param = {};
331         for(let n = 0; n < args.length; n++) {
332           param[args[n]] = this.parameter_list[args[n]];
333         }
334         return param;
335       } else {
336         return this.parameter_list[args];
337       }
338     }
339   }
340
341 /**
342  * Retrieves input variables based on the provided arguments.
343  * @param {string|string[]} [args] - A single input name or an array of
344  * input names to retrieve. If omitted, returns all inputs.
345  * @returns {Object|any} - An object containing the requested inputs or a
346  * single input value.
347  */
348   getInputs(args) {
349     if(args && args.length > 0) {
350       if(Array.isArray(args)) {
351         const inputs = {};
352         for(let n = 0; n < args.length; n++) {
353           inputs[args[n]] = this.input_list[args[n]];
354         }
355         return inputs;
356       } else {
357         return this.input_list[args];
358       }
359     }
360   }
361
362 /**
363  * Retrieves output variables based on the provided arguments.
364  * @param {string|string[]} [args] - A single output name or an array of
365  * output names to retrieve. If omitted, returns all outputs.
366  * @returns {Object|any} - An object containing the requested outputs or a
367  * single output value.
368  */
369   getOutputs(args) {

```

```

368     if(args && args.length > 0) {
369       if(Array.isArray(args)) {
370         const outputs = {};
371         for(let n = 0; n < args.length; n++) {
372           outputs[args[n]] = this.output_list[args[n]];
373         }
374         return outputs;
375       } else {
376         return this.output_list[args];
377       }
378     } else {
379       return this.output_list;
380     }
381   }

383 /**
384 * Retrieves continuous variables based on the provided arguments.
385 * @param {string|string[]} [args] - A single continuous variable name or an
386 * array of names to retrieve. If omitted, returns all continuous
387 * variables.
388 * @returns {Object|any} - An object containing the requested continuous
389 * variables or a single value.
390 */
391 getContinuous(args) {
392   if(args && args.length > 0) {
393     if(Array.isArray(args)) {
394       const continuous = {};
395       for(let n = 0; n < args.length; n++) {
396         continuous[args[n]] = this.continuous_list[args[n]];
397       }
398       return continuous;
399     } else {
400       return this.continuous_list[args];
401     }
402   }

404 /**
405 * Retrieves simulation options based on the provided arguments.
406 * @param {string|string[]} [args] - A single simulation option name or an
407 * array of names to retrieve. If omitted, returns all simulation options.
408 * @returns {Object|any} - An object containing the requested simulation
409 * options or a single option value.
410 */
411 getSimulationOptions(args) {
412   if(args && args.length > 0) {
413     if(Array.isArray(args)) {
414       const simoptions = {};
415       for(let n = 0; n < args.length; n++) {
416         simoptions[args[n]] = this.simulation_options[args[n]];
417       }
418       return simoptions;
419     } else {
420       return this.simulation_options;
421     }
422   }

```

```

418         return this.simulation_options[args];
419     }
420 } else {
421     return this.simulation_options;
422 }
423 }

424 /**
425 * Retrieves linearization options based on the provided arguments.
426 * @param {string|string[]} [args] - A single linearization option name or
427 * an array of names to retrieve. If omitted, returns all linearization
428 * options.
429 * @returns {Object|any} - An object containing the requested linearization
430 * options or a single option value.
431 */
432 getLinearizationOptions(args) {
433     if(args && args.length > 0) {
434         if(Array.isArray(args)) {
435             const linoptions = {};
436             for(let n = 0; n < args.length; n++) {
437                 linoptions[args[n]] = this.linear_options[args[n]];
438             }
439             return linoptions;
440         } else {
441             return this.linear_options[args];
442         }
443     }
444 }

445 /**
446 * Sets simulation parameters based on the provided arguments.
447 * @param {string|string[]} args - A single parameter assignment (e.g., "param=5")
448 * or an array of assignments.
449 */
450 setParameters(args) {
451     if(args && args.length > 0) {
452         if(!Array.isArray(args)) {
453             args = [args];
454         }
455         args.forEach(arg => {
456             const val = arg.replace(/\s+/g, '');
457             const [key, value] = val.split("=");
458             if(key in this.parameter_list) {
459                 this.parameter_list[key] = value;
460                 this.override_variables[key] = value;
461             } else {
462                 this.jsl.env.error(`@setParameters: ${key} ${language.string(209)}`);
463             }
464         });
465     }
466 }

467 /**

```

```

469  * Sets simulation options based on the provided arguments.
470  * @param {string|string[]} args - A single simulation option assignment (e.
471  *     g., "stepSize=0.01") or an array of assignments.
472  */
473  setSimulationOptions(args) {
474      if(args && args.length > 0) {
475          if(!Array.isArray(args)) {
476              args = [args];
477          }
478          args.forEach(arg => {
479              const val = arg.replace(/\s+/g, '');
480              const [key, value] = val.split("=");
481              if(key in this.simulation_options) {
482                  this.simulation_options[key] = value;
483                  this.sim_opt_override[key] = value;
484              } else {
485                  this.jsl.env.error('@setSimulationOptions: ' + key + language.string(210));
486              }
487          });
488      }
489  }
490 /**
491  * Sets linearization options based on the provided arguments.
492  * @param {string|string[]} args - A single linearization option assignment
493  *     or an array of assignments.
494  */
495  setLinearizationOptions(args) {
496      if(args && args.length > 0) {
497          if(!Array.isArray(args)) {
498              args = [args];
499          }
500          args.forEach(arg => {
501              const val = arg.replace(/\s+/g, '');
502              const [key, value] = val.split("=");
503              if(key in this.linear_options) {
504                  this.linear_options[key] = value;
505              } else {
506                  this.jsl.env.error('@setLinearizationOptions: ' + key + language.
507                      string(211));
508              }
509          });
510      }
511 /**
512  * Sets input variables based on the provided arguments.
513  * @param {string|string[]} args - A single input assignment (e.g., "input1
514  *     =10") or an array of assignments.
515  */
516  setInputs(args) {
517      if(args && args.length > 0) {
518          if(!Array.isArray(args)) {

```

```

519     }
520     args.forEach(arg => {
521       const val = arg.replace(/\s+/g, " ");
522       const [key, value] = val.split("=");
523       if(key in this.input_list) {
524         this.input_list[key] = value;
525         this.input_flag = true;
526       } else {
527         this.jsl.env.error('@setInputs: ' + key + language.string(212));
528       }
529     });
530   }
531 }
532
533 /**
534 * Creates a CSV file containing input data for the simulation.
535 */
536 createcsvData() {
537   this.csvfile = path.join(this.mat_temp_dir, `${this.modelname}.csv`);
538   const file_id = fs.openSync(this.csvfile, 'w');
539   const fields = Object.keys(this.input_list);
540   let header = `time,${fields.join(",")},end\n`;
541   fs.writeFileSync(file_id, header);
542
543   let time = [];
544   let tmptmpcsvdata = {};
545
546   fields.forEach(field => {
547     let var_data = this.input_list[field] || "0";
548     try {
549       var_data = eval(var_data.replace(/\\[\\]/|\\(|\\)/g, match => {
550         return match === '[' || match === ']' ? '{' : '}';
551       }));
552     } catch {
553       var_data = [[0, 0]]; // Default to 0 if evaluation fails
554     }
555     tmptmpcsvdata[field] = var_data;
556
557     if(var_data.length > 1) {
558       var_data.forEach(entry => {
559         time.push(entry[0]);
560       });
561     }
562   });
563
564   if(time.length === 0) {
565     time = [parseFloat(this.simulation_options.startTime), parseFloat(this.simulation_options.stopTime)];
566   }
567
568   const sorted_time = time.sort((a, b) => a - b);
569   let previous_value = {};
570
571   sorted_time.forEach(t => {
572     let line = `${t},`;

```

```

573   fields.forEach((field) => {
574     let data = tmpcsvdata[field];
575     let value = previous_value[field] || 0;
576
577     if(Array.isArray(data)) {
578       for(let j = 0; j < data.length; j++) {
579         if(data[j][0] === t) {
580           value = data[j][1];
581           data.splice(j, 1);
582           tmpcsvdata[field] = data;
583           break;
584         }
585       }
586     previous_value[field] = value;
587   }
588   line += `${value},`;
589 });
590 line += "0\n";
591 fs.writeFileSync(file_id, line);
592 });
593
594 fs.closeSync(file_id);
595 }
596
597 /**
598 * Runs the simulation with optional result file and simulation flags.
599 * @param {string} [resultfile=''] - The name of the result file to generate
600 * @param {string} [sim_flags=''] - Additional simulation flags.
601 */
602 async simulate(resultfile = '', sim_flags = '') {
603   let r = resultfile ? ` -r=${resultfile}` : '';
604   this.resultfile = resultfile ? path.join(this.mat_temp_dir, resultfile) :
605     path.join(this.mat_temp_dir, `${this.modelname}_res.mat`);
606
607   if(fs.existsSync(this.xmlfile)) {
608     let getexefile;
609     if(process.platform === 'win32') {
610       getexefile = path.join(this.mat_temp_dir, `${this.modelname}.exe`);
611     } else {
612       getexefile = path.join(this.mat_temp_dir, this.modelname);
613     }
614
615     if(fs.existsSync(getexefile)) {
616       let overridevar = '';
617       if(Object.keys(this.override_variables).length ||
618         Object.keys(this.sim_opt_override).length) {
619         const allOverrides = { ...this.override_variables, ...this.
620           sim_opt_override };
621         const fields = Object.keys(allOverrides);
622         const tmpoverride1 = fields.map(field => `${field}=${allOverrides[
623           field]}`);
624         overridevar = ` -override=${tmpoverride1.join(' ', '')}`;
625       }
626     }
627   }
628 }

```

```

623     let csvinput = '';
624     if(this.input_flag) {
625       this.createcsvData();
626       csvinput = '-csvInput=${this.csvfile}';
627     }
628
629     const final_simulation_exe = "${getexefile}${overridevar}${csvinput}
630       ${r}${sim_flags}";
631     execSync(final_simulation_exe, { cwd: this.mat_temp_dir });
632   } else {
633     this.jsl.env.error('@simulate: '+language.string(205));
634   }
635 } else {
636   this.jsl.env.error('@simulate: '+language.string(206));
637 }
638
639 /**
640 * Performs linearization of the model and retrieves the linear matrices.
641 * @returns {Array<Object>} - An array containing the A, B, C, and D
642 * matrices.
643 */
644 async linearize() {
645   const linres = await this.sendExpression("setcommand_line_options(\"+
646     generateSymbolicLinearization\")");
647   if(linres && linres[0] === "false") {
648     this.jsl.env.error('@simulate: '+language.string(207)+ await this.
649       sendExpression("getErrorString()"));
650     return false;
651   }
652
653   const fields = Object.keys(this.override_variables);
654   const tmpoverride1 = fields.map(field => `${field}=${this.
655     override_variables[field]}`);
656   const tmpoverride2 = tmpoverride1.length ? ` -override=${tmpoverride1.join
657     (',')}` : "";
658
659   const lin_fields = Object.keys(this.linear_options);
660   const tmpoverride1lin = lin_fields.map(field => `${field}=${this.
661     linear_options[field]}`);
662   const overridelinear = tmpoverride1lin.join(',');
663
664   let csvinput = '';
665   if(this.input_flag) {
666     this.createcsvData();
667     csvinput = '-csvInput=${this.csvfile.replace(/\//g, '/')}`;
668   }
669
670   const linexpr = `linearize(${this.modelname},${overridelinear},sim_flags=
671     ${csvinput} ${tmpoverride2})`;
672   const res = await this.sendExpression(linexpr);
673   this.resultfile = res.resultFile;
674   this.linear_modelname = `linear_${this.modelname}`;
675   this.linearfile = path.join(this.mat_temp_dir, `${this.linear_modelname}.
676     mo`).replace(/\//g, '/');

```

```

669   if(fs.existsSync(this.linearfile)) {
670     const loadmsg = await this.sendExpression(`loadFile("${this.linearfile}"`));
671     if(loadmsg && loadmsg[0] === "false") {
672       this.jsl.env.error(`@linearize: ${await this.sendExpression("getErrorResponse()")}`);
673       return false;
674     }
675
676     const cNames = await this.sendExpression("getclassNames()");
677     const buildmodeexpr = `buildModel(${cNames[0]})`;
678     const buildModelmsg = await this.sendExpression(buildmodeexpr);
679     if(buildModelmsg && buildModelmsg.length > 0) {
680       this.linear_flag = true;
681       this.xmlfile = path.join(this.mat_temp_dir, buildModelmsg[1]);
682       this.linear_quantity_list = [];
683       await this.xmlparse();
684       return this.getLinearMatrix();
685     } else {
686       this.jsl.env.error(`@linearize: ${await this.sendExpression("getErrorResponse()")}`);
687       return false;
688     }
689   }
690   return true;
691 }
692
693 /**
694 * Retrieves the linear A, B, C, and D matrices.
695 * @returns {Array<Object>} - An array containing the A, B, C, and D
696 * matrices.
697 */
698 getLinearMatrix() {
699   const matrix_A = {};
700   const matrix_B = {};
701   const matrix_C = {};
702   const matrix_D = {};
703
704   this.linear_quantity_list.forEach(item => {
705     const name = item.name;
706     const value = item.value;
707
708     if(item.variability === "parameter") {
709       if(name.startsWith('A')) {
710         matrix_A[name] = value;
711       } else if(name.startsWith('B')) {
712         matrix_B[name] = value;
713       } else if(name.startsWith('C')) {
714         matrix_C[name] = value;
715       } else if(name.startsWith('D')) {
716         matrix_D[name] = value;
717       }
718     }
719   });
720 }

```

```

720     return [matrix_A, matrix_B, matrix_C, matrix_D];
721 }
722
723 /**
724 * Converts linear matrix data into a two-dimensional array format.
725 * @param {Object} matrix_name - The linear matrix object to convert.
726 * @returns {number[][]|number} - The converted matrix as a 2D array or 0 if
727   empty.
728 */
729 getLinearMatrixValues(matrix_name) {
730   if(Object.keys(matrix_name).length > 0) {
731     const fields = Object.keys(matrix_name);
732     const last_field = fields[fields.length - 1];
733     const rows = parseInt(last_field.charAt(2), 10);
734     const columns = parseInt(last_field.charAt(4), 10);
735     const tmp_matrix = Array.from({ length: rows }, () => Array(columns).
736       fill(0));
737
738     fields.forEach(field => {
739       const r = parseInt(field.charAt(2), 10) - 1;
740       const c = parseInt(field.charAt(4), 10) - 1;
741       const val = parseFloat(matrix_name[field]);
742       tmp_matrix[r][c] = val;
743     });
744
745     return tmp_matrix;
746   } else {
747     return 0;
748   }
749 }
750 /**
751 * Retrieves the linear input variables.
752 * @returns {string|boolean} - The linear input variables or false if the
753   model is not linearized.
754 */
755 getlinear_inputs() {
756   if(this.linear_flag) {
757     return this.linear_inputs;
758   } else {
759     this.jsl.env.error("@getlinear_inputs: "+language.string(202));
760     return false;
761   }
762 }
763 /**
764 * Retrieves the linear output variables.
765 * @returns {string|boolean} - The linear output variables or false if the
766   model is not linearized.
767 */
768 getlinear_outputs() {
769   if(this.linear_flag) {
770     return this.linear_outputs;
771   } else {
772     this.jsl.env.error("@getlinear_outputs: "+language.string(202));
773   }
774 }
```

```

771         return false;
772     }
773 }
774 /**
775 * Retrieves the linear state variables.
776 * @returns {string|boolean} - The linear state variables or false if the
777 * model is not linearized.
778 */
779 getlinear_states() {
780     if(this.linear_flag) {
781         return this.linear_states;
782     } else {
783         this.jsl.env.error("@getlinear_states: "+language.string(202));
784         return false;
785     }
786 }
787 /**
788 * Retrieves simulation solutions based on the provided arguments and result
789 * file.
790 * @param {string|string[]} [args] - A single variable name or an array of
791 * names to retrieve solutions for. If omitted, retrieves all variables.
792 * @param {string} [resultfile=this.resultfile] - The path to the result
793 * file.
794 * @returns {Promise<any>} - A promise that resolves with the simulation
795 * results or an error message.
796 */
797 async getSolutions(args, resultfile = this.resultfile) {
798     resultfile = resultfile.replace(/\\/g, '/');
799     if(fs.existsSync(resultfile)) {
800         if(args && args.length > 0) {
801             const variables = `${args.join(',')}`;
802             const simresult = await this.sendExpression('readSimulationResult("${{
803                 resultfile}", ${variables})');
804             await this.sendExpression("closeSimulationResultFile()");
805             return simresult;
806         } else {
807             const variables = await this.sendExpression('readSimulationResultVars(
808                 "${resultfile}")');
809             await this.sendExpression("closeSimulationResultFile()");
810             return variables;
811         }
812     } else {
813         this.jsl.env.error('@getSolutions: ' + language.string(208) + resultfile
814             );
815         return false;
816     }
817 /**
818 * Creates valid variable names by replacing invalid characters and
819 * categorizes them based on the structure name.
820 * @param {string} name - The original variable name.
821 * @param {any} value - The value of the variable.

```

```

817   * @param {string} structname - The structure name (e.g., 'continuous', 'parameter').
818   */
819   createValidNames(name, value, structname) {
820     const tmpname = name.replace(/[^a-zA-Z0-9]/g, '_'); // Replace invalid
821     characters with underscore
822     this.mapped_names[tmpname] = name;
823
824     if(structname === 'continuous') {
825       this.continuous_list[tmpname] = value;
826     } else if(structname === 'parameter') {
827       this.parameter_list[tmpname] = value;
828     } else if(structname === 'input') {
829       this.input_list[tmpname] = value;
830     } else if(structname === 'output') {
831       this.output_list[tmpname] = value;
832     }
833   }
834 /**
835  * Parses a given expression string into structured data based on predefined
836  * formats.
837  * Handles various formats including single and nested lists, records, and
838  * single elements.
839  * @param {string} args - The expression string to parse.
840  * @returns {Array|Object|string} - The parsed data which could be an array,
841  * an object, or a string.
842  */
843 parseExpression(args) {
844   // Use regular expressions to match strings and key parts of the
845   // expression
846   const final = args.match(/\((.*?)\)|[\{\}()]=|-?\d+(\.\d+)?([eE][+-]?\d+)?|[a-
847   zA-Z_][a-zA-Z0-9_.]* /g);
848
849   if(final.length > 1) {
850     if(final[0] === "{" && final[1] !== "(") {
851       // Handle single-level list
852       let buff = [];
853       for(let i = 0; i < final.length; i++) {
854         if(!["{", "}", ")", "(", ","].includes(final[i])) {
855           const value = final[i].replace(/\//g, "");
856           buff.push(value);
857         }
858       }
859       return buff;
860     }
861   } else if(final[0] === "{" && final[1] === "(") {
862     // Handle nested lists
863     let buff = [];
864     let tmp = [];
865     for(let i = 1; i < final.length - 1; i++) {
866       if(final[i] === "(") {
867         tmp = [];
868       } else if(final[i] === ")") {
869         buff.push(tmp);
870       }
871     }
872     return buff;
873   }
874 }

```

```

865         tmp = [];
866     } else {
867         tmp.push(final[i].replace(/"/g, ""));
868     }
869 }
870 return buff;
871
872 } else if(final[0] === "record") {
873     // Handle record structure
874     let result = {};
875     for(let i = 2; i < final.length - 2; i++) {
876         if(final[i] === "=") {
877             const key = final[i - 1];
878             const value = final[i + 1].replace(/"/g, "");
879             result[key] = value;
880         }
881     }
882     return result;
883
884 } else if(final[0] === "fail") {
885     // Handle failure case
886     return this.sendExpression("getErrorMessage()");
887 } else {
888     // Return as a simple string if no special cases match
889     return args.replace(/"/g, "");
890 }
891 } else if(final.length === 1) {
892     // Handle single element case
893     return final[0].replace(/"/g, "");
894 } else {
895     // Handle empty result
896     return args.replace(/"/g, "");
897 }
898 }
899
900 /**
901 * Closes the current session safely by cleaning up resources such as
902 * temporary files and network connections.
903 * Terminates any active processes and removes temporary port files.
904 */
905 async close() {
906     if(this.portfile && fs.existsSync(this.portfile)) {
907         fs.unlinkSync(this.portfile);
908     }
909     if(this.active) {
910         await this.requester.close();
911         this.active = false;
912     }
913     killProcess(this.pid);
914 }
915 }
916

```

```
918 exports.PRDC_JSLAB_OPENMODELICA_LINK = PRDC_JSLAB_OPENMODELICA_LINK;
```

Listing 104 - om-link.js

```

1  /**
2   * @file JSLAB library optim Real Coded Mixed Integer Genetic Algorithm
3   * submodule
4   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
5   * PR-DC, Republic of Serbia
6   * info@pr-dc.com
7   * @version 0.0.1
8   */
9 /**
10 * Class for Real Coded Mixed Integer Genetic Algorithm - RCMIGA.
11 */
12 class PRDC_JSLAB_OPTIM_RCMIGA {
13
14 /**
15 * Creates an instance of PRDC_JSLAB_LIB_OPTIM_RCMIGA.
16 * @param {Object} problem - The optimization problem definition.
17 * @param {Object} opts - Configuration options for the algorithm.
18 */
19 constructor(problem, opts) {
20   this.flag = 'preinit';
21   this.problem = problem;
22   this.opts = opts;
23   this.state = {};
24   this.selection = {};
25   this.solution = {};
26   this.constrained = problem.constrained;
27   this.stoped = 0;
28
29   this.checkInputs();
30   if(this.bounded) {
31     this.opts.lbm = repCol(this.opts.lb, opts.PopulationSize);
32     this.opts.ubm = repCol(this.opts.ub, opts.PopulationSize);
33   }
34
35   this.opts.a = opts.a ?? 0;
36   this.opts.b_real = opts.b_real ?? 0.15;
37   this.opts.b_integer = opts.b_integer ?? 0.35;
38   this.opts.p_real = opts.p_real ?? 10;
39   this.opts.p_integer = opts.p_integer ?? 4;
40   this.opts.UseVectorized = opts.UseVectorized ?? false;
41   this.opts.UseParallel = opts.UseParallel ?? false;
42   this.opts.Display = opts.Display ?? 'iter';
43
44 // Inicijalizacija
45 this.flag = 'init';
46 this.outputFcn();
47 this.state.StartTime = tic;
48 this.state.StallTime = tic;
49 this.state.StallGenerations = 1;
50 this.state.RandSeed = 'rcmiga';
51 this.state.Generation = 0;

```

```

52
53     this.rand = seedRandom(this.state.RandSeed);
54 }
55
56 /**
57 * Executes the optimization process.
58 */
59 async run(parallel_context, parallel_setup_fun) {
60   if(this.opts.UseParallel) {
61     this.parallel_context = parallel_context;
62     this.parallel_setup_fun = parallel_setup_fun;
63     parallel.terminate();
64   }
65
66   this.initState();
67   this.initPopulation();
68   await this.updateState();
69
70   this.flag = 'iter';
71   this.outputFcn();
72   this.createPlotFcn();
73   this.plotFcn();
74
75   // Optimizacija
76   var nvars = this.problem.nvars;
77   var newPopulation = zeros(nvars * this.opts.PopulationSize);
78   var stop = 0;
79   while(!stop) { // Petlja za optimizaciju
80     // Merenje vremena
81     this.state.GenTime = tic;
82
83     // Elitizam
84     if(this.state.ReproductionCount.elite > 0) {
85       var [S, I] = sorti(this.state.FunVal);
86       this.selection.current_elite = getSub(I, this.selection.elite);
87       setSub(newPopulation, index(range(0, nvars - 1),
88         this.selection.elite, nvars), getSub(this.state.Population,
89         index(range(0, nvars - 1), this.selection.current_elite, nvars)));
90     }
91
92     // Selekcija
93     this.state.parents = this.selectionFcn();
94
95     // Ukrstanje
96     setSub(newPopulation, index(range(0, nvars - 1),
97       this.selection.children, nvars), this.crossoverFcn());
98
99     // Mutacija
100    setSub(newPopulation, index(range(0, nvars - 1),
101      this.selection.mutants, nvars), this.mutationFcn());
102
103    // Provera granica
104    this.state.Population = [...newPopulation];
105    this.integerRestriction();
106    if(this.bounded) {

```

```

107         this.checkBounds();
108     }
109
110     // Odredjivanje vrednosti funkcije prilagodjenosti i ogranicenja
111     this.state.Generation = this.state.Generation + 1;
112     await this.updateState();
113
114     // Prikaz stanja
115     stop = this.stoppingCriteria(); // Kriterijumi zaustavljanja
116     this.outputFcn();
117     this.plotFcn();
118     await waitMSeconds(5);
119   }
120
121   this.solution.generations = this.state.Generation;
122   var [x_min, I] = mini(this.state.FunVal);
123   this.solution.feasible = 1;
124   if(this.constrained && this.state.ConSumVal[I] > 0) {
125     this.solution.feasible = 0;
126   }
127   this.solution.x = getSub(this.state.Population,
128     index(range(0, nvars - 1), I, nvars));
129   this.solution.fval = this.problem.fitnessfcn(this.solution.x);
130   this.solution.StoppingCriteria = stop;
131
132   this.flag = 'done';
133   this.outputFcn();
134 }
135
136 /**
137 * Validates the input parameters and options.
138 */
139 checkInputs() {
140   if(typeof this.problem.nonlconfcn === 'function') {
141     this.constrained = 1;
142   }
143   if(typeof this.problem.IntCon === 'undefined') {
144     this.problem.IntCon = [];
145   }
146
147   this.bounded = true;
148   if(typeof this.opts.lb === 'undefined' || typeof this.opts.ub === 'undefined') {
149     this.bounded = false;
150     if(typeof this.opts.InitialUnboundedRange === 'undefined') {
151       this.opts.InitialUnboundedRange = createFilledArray(this.problem.nvars,
152         [0, 1]);
153     }
154   } else if((this.problem.lb && this.problem.lb.length != this.problem.nvars
155             ) ||
156             (this.problem.ub && this.problem.ub.length != this.problem.nvars)) {
157     throw new Error('Problem bounds have invalid dimension!');
158   }
159
160   if(this.opts.UseVectorized) {

```

```

159     if (this.opts.UseParallel == true) {
160         disp('Option UseParallel is ignored while option UseVectorized is true
161             .');
162     } else if (this.opts.UseParallel != false) {
163         disp('Option UseParallel must be true or false.');
164     }
165     } else {
166         if (![true, false].includes(this.opts.UseParallel)) {
167             throw new Error('Option UseParallel must be true or false!');
168         }
169     }
170 }
171 /**
172 * Creates the initial population.
173 */
174 creationFcn() {
175     return this.creationMixedUniform();
176 }
177 /**
178 * Selects individuals from the population.
179 */
180 selectionFcn() {
181     return this.binaryTournamentSelection();
182 }
183 /**
184 * Performs crossover between selected parents.
185 */
186 crossoverFcn() {
187     return this.laplaceMixedCrossover();
188 }
189 /**
190 * Mutates individuals in the population.
191 */
192 mutationFcn() {
193     return this.powerMixedMutation();
194 }
195 /**
196 * Initializes the plotting function.
197 */
198 createPlotFcn() {
199 }
200 /**
201 * Updates the graphical plot with current optimization status.
202 */
203 plotFcn() {
204     this.displayPlot();
205 }
206 
```

```

213 /**
214 * Renders the optimization plot.
215 */
216 displayPlot() {
217 }
218 /**
219 * Handles button events to stop the optimization process.
220 */
221 buttonCallback() {
222     this.stoped = 1;
223 }
224 /**
225 * Manages the output display based on configuration.
226 */
227 outputFcn() {
228     if([ 'iter' , 'final' ].includes(this.opts.Display)) {
229         this.displayOutput();
230     }
231 }
232 /**
233 * Displays the current state of optimization.
234 */
235 displayOutput() {
236     if(this.opts.Display === 'iter') {
237         switch(this.flag) {
238             case 'init':
239                 disp(' Optimization is initialized!');
240                 if(this.opts.UseVectorized) {
241                     disp(' Vectorized functions evaluation in use.');
242                 } else if(this.opts.UseParallel){
243                     disp(' Parallel functions evaluation in use.');
244                 }
245                 break;
246             case 'iter':
247                 if(this.state.Generation === 0 || this.state.Generation % 20 === 0)
248                 {
249                     if(this.constrained) {
250                         dispMonospaced('\n' Best Max
251                                     Stall');
252                         dispMonospaced(' Generation f(x) Constraint
253                                     Generations');
254                     } else {
255                         dispMonospaced('\n' Best Stall');
256                         dispMonospaced(' Generation f(x) Generations');
257                     }
258                 }
259             }
260
261             let fval_s = num2str(this.state.Best[this.state.Generation] , 5);
262             if(this.constrained) {
263                 if(!this.state.Feasible[this.state.Generation]) {
264                     fval_s += '*';
265                 }
266             }
267         }
268     }
269 }
270 
```

```

265
266      }
267      dispMonospaced(
268          `    ${num2str(this.state.Generation, 0).padStart(10)}    ${}
269          fval_s.padStart(12)}    ${
270          num2str(max(this.state.ConSumVal), 2).padStart(12)}    ${
271          num2str(this.state.StallGenerations, 0).padStart(12)}`'
272      );
273  } else {
274      dispMonospaced(
275          `    ${num2str(this.state.Generation, 0).padStart(10)}    ${}
276          fval_s.padStart(12)}    ${
277          num2str(this.state.StallGenerations, 0).padStart(12)}`'
278      );
279  }
280  break;
281 case 'done':
282     var str = '\nOptimization is done';
283     if(this.constrained) {
284         if(this.state.Feasible[this.state.Generation]) {
285             str += ', solution found';
286         } else {
287             str += ', no feasible solution found';
288         }
289     }
290     disp(` ${str}, stopping criteria = ${this.solution.StoppingCriteria}
291           }!\n`);
292     break;
293 }
294 } else if(this.opts.Display === 'final') {
295     if(this.flag === 'done') {
296         var str = '\nOptimization is done';
297         if(this.constrained) {
298             if(this.state.Feasible[this.state.Generation]) {
299                 str += ', solution found';
300             } else {
301                 str += ', no feasible solution found';
302             }
303         }
304     }
305 }
306 /**
307 * Creates the initial population with mixed uniform distribution.
308 */
309 creationMixedUniform() {
310     var lb = this.opts.lb;
311     var ub = this.opts.ub;
312     if(!this.bounded) {
313         lb = this.opts.InitialUnboundedRange.map((v) => v[0]);
314         ub = this.opts.InitialUnboundedRange.map((v) => v[1]);
315     }
316     var N = this.problem.IntCon.length;

```

```

318   var population = arrayRand(lb , ub ,
319     this . problem . nvars , this . opts . PopulationSize , this . rand );
320   if (N) {
321     var r = arrayRandi([0 , 1] , N, this . opts . PopulationSize , this . rand );
322     var I = index(this . problem . IntCon ,
323       range(0 , this . opts . PopulationSize - 1) , this . problem . nvars );
324
325     setSub(population , I , plus(fix(getSub(population , I)) , r));
326   }
327   return population;
328 }
329
330 /**
331 * Performs binary tournament selection of parents .
332 */
333 binaryTournamentSelection() {
334   var parents = zeros(this . state . nParents );
335   var r1 = arrayRandi([0 , this . opts . PopulationSize - 1] , 2 ,
336     this . state . nParents , this . rand );
337   var r2 = arrayRandi([0 , this . opts . PopulationSize - 1] , 2 ,
338     this . state . nParents , this . rand );
339   var neq = elementWise((a , b) => a > b , r1 , r2 );
340   var I1 = indexOfAll(neq , true); // closer r2
341   var I2 = indexOfAll(neq , false); // closer r1
342   setSub(parents , I1 , getSub(r2 , I1));
343   setSub(parents , I2 , getSub(r1 , I2));
344   return parents;
345 }
346
347 /**
348 * Executes Laplace mixed crossover to generate offspring .
349 */
350 laplaceMixedCrossover() {
351   var N = this . problem . IntCon . length ;
352   var nvars = this . problem . nvars ;
353   var iParents = this . state . parents ;
354   var nChildren = this . state . ReproductionCount . children ;
355   var nParents = this . state . parentsCount . crossover ;
356   var sParents = this . state . parentsSelection . crossover ;
357
358   var b = scale(ones(nvars * nChildren / 2) , this . opts . b _ real );
359   if (N) {
360     setSub(b , index(this . problem . IntCon , range(0 , nChildren / 2 - 1) , nvars ) ,
361       scale(ones(N * nChildren / 2) , this . opts . b _ integer));
362   }
363   var u = arrayRand(zeros(nvars) , ones(nvars) , nvars , nChildren / 2 , this . rand );
364   var r = arrayRand(zeros(nvars) , ones(nvars) , nvars , nChildren / 2 , this . rand );
365   var beta = elementWise((x , y) => this . opts . a - x * log10(y) , b , u); // a - b *
366   log10(u);
367   var I = indexOfAll(r . map((ri) => ri > 0.5) , true);
368   setSub(beta , I , elementWise((x , y) => this . opts . a + x * log10(y) ,
369     getSub(b , I) , getSub(u , I))); // a + b * log10(u)

```

```

370     var mother = getSub(this.state.Population, index(range(0, nvars - 1),
371         getSub(iParents, getSub(sParents, range(0, nParents/2-1))), nvars));
372     var father = getSub(this.state.Population, index(range(0, nvars - 1),
373         getSub(iParents, getSub(sParents, range(nParents/2, nParents-1))), nvars
374         ));
375     var children = plus(concatRow(nvars, mother, father),
376         elementWise((a, b, c) => a*abs(b-c),
377             repCol(beta, 2), repCol(mother, 2), repCol(father, 2)));
378     return children;
379   }
380
381 /**
382 * Applies power mixed mutation to selected individuals.
383 */
384 powerMixedMutation() {
385   var N = this.problem.IntCon.length;
386   var nvars = this.problem.nvars;
387   var nMutants = this.state.ReproductionCount.mutants;
388   var sParents = this.state.parentsSelection.mutation;
389
390   var p = scale(ones(nvars * nMutants), this.opts.p_real);
391   if(N) {
392     setSub(p, index(this.problem.IntCon, range(0, nMutants - 1), nvars),
393         scale(ones(N * nMutants), this.opts.p_integer));
394   }
395   var s1 = arrayRand(zeros(nvars), ones(nvars), nvars, nMutants, this.rand);
396   var r = arrayRand(zeros(nvars), ones(nvars), nvars, nMutants, this.rand);
397   var s = elementWise((a, b) => pow(a, b), s1, p);
398
399   var parents = getSub(this.state.Population,
400     getSub(sParents, index(range(0, nvars - 1), sParents, nvars)));
401   var mutants;
402
403   if(!this.bounded) {
404     mutants = elementWise((a, b, c) => c < 0.5 ? a + b : a - b, parents, s,
405       r);
406   } else {
407     var ubm = repCol(this.opts.ub, nMutants);
408     var ldm = repCol(this.opts.lb, nMutants);
409     var t = elementWise((a, b, c) => (a - b)/(c - a), parents, this.opts.lb,
410       this.opts.ub);
411     mutants = elementWise((a, b, c) => a+b*(a-c), parents, s, ldm);
412
413     var I = indexOfAll(elementWise((a, b) => a < b, t, r), true);
414     setSub(mutants, I, elementWise((a, b, c) => a-b*(c-a),
415       getSub(parents, I), getSub(s, I), getSub(ubm, I)));
416   }
417
418   return mutants;
419 }
420 /**
421 * Applies Gaussian mutation to selected individuals.
422 */
423 gaussianMutation() {

```

```

422
423     }
424
425     /**
426      * Ensures integer constraints are met for specific variables.
427      */
428     integerRestriction() {
429         var N = this.problem.IntCon.length;
430         if(N) {
431             var I = index(this.problem.IntCon,
432                 [...this.selection.children, ...this.selection.mutants], this.problem.
433                 nvars);
434             var p = getSub(this.state.Population, I);
435
436             var r = arrayRandi([0, 1], N,
437                 this.state.ReproductionCount.children +
438                 this.state.ReproductionCount.mutants, this.rand);
439
440             var Is = indexOfAll(neg(isInteger(p)), true);
441             setSub(p, Is, plus(fix(getSub(p, Is)), getSub(r, Is)));
442             setSub(this.state.Population, I, p);
443         }
444
445     /**
446      * Checks and enforces variable bounds within the population.
447      */
448     checkBounds() {
449         this.state.Population = elementWise((a, b, c) => min([max([a, b]), c]),
450             this.state.Population, this.opts.lbm, this.opts.ubm);
451     }
452
453     /**
454      * Evaluates whether stopping criteria have been met.
455      * @returns {number} Code indicating the reason to stop or continue.
456      */
457     stoppingCriteria() {
458         var stop = 0;
459         var gen = this.state.Generation;
460
461         if(gen > 0) {
462             if(this.state.Best[gen] < this.state.Best[gen - 1] ||
463                 !this.state.Feasible[gen] ||
464                 this.state.Feasible[gen] > this.state.Feasible[gen - 1]) {
465                 this.state.StallTime = tic;
466                 this.state.StallGenerations = 1;
467             } else {
468                 this.state.StallGenerations = this.state.StallGenerations + 1;
469             }
470         }
471
472         if(gen >= (this.opts.MaxGenerations - 1)) {
473             // 1 - Maksimalni broj generacija ostvaren
474             stop = 1;
475         } else if(toc(this.state.StartTime) >= this.opts.MaxTime) {

```

```

476      // 2 - Proteklo maksimalno vreme trajanja optimizacije
477      stop = 2;
478    } else if(this.state.Feasible[gen] &&
479      this.state.Best[gen] <= this.opts.FitnessLimit) {
480      // 3 - Funkcija prilagodjenosti dostigla ciljanu vrednost
481      stop = 3;
482    } else if(gen > (this.opts.MaxStallGenerations - 2) &&
483      this.state.Feasible[gen - (this.opts.MaxStallGenerations - 2)] &&
484      this.state.Feasible[gen] &&
485      ((this.state.Best[gen - (this.opts.MaxStallGenerations - 2)] -
486      this.state.Best[gen]) <= this.opts.FunctionTolerance)) {
487      // 4 - Promena vrednosti funkcija manja od tolerancije
488      stop = 4;
489    } else if(this.state.StallGenerations >= this.opts.MaxStallGenerations) {
490      // 5 - Bez promene vrednosti funkcije kroz odredjeni broj generacija
491      stop = 5;
492    } else if(toc(this.state.StallTime) >= this.opts.MaxStallTime) {
493      // 6 - Bez promene vrednosti funkcije odredjeno vreme
494      stop = 6;
495    } else if(this.stopped) {
496      // 7 - Korisnik zaustavlja proces
497      stop = 7;
498    }
499    return stop;
500  }
501
502 /**
503 * Initializes the population for the optimization process.
504 */
505 initPopulation() {
506   this.state.Population = this.creationFcn();
507   this.integerRestriction();
508   if(this.bounded) {
509     this.checkBounds();
510   }
511 }
512
513 /**
514 * Initializes the internal state of the algorithm.
515 */
516 initState() {
517   // Merenje vremena
518   this.state.GenTime = tic;
519
520   this.state.FunVal = zeros(this.opts.PopulationSize);
521
522   this.state.ConSumVal = zeros(this.opts.PopulationSize);
523   this.state.Best = zeros(this.opts.MaxGenerations);
524   this.state.Feasible = ones(this.opts.MaxGenerations);
525   this.state.Time = zeros(this.opts.MaxGenerations);
526   this.state.x = zeros(this.opts.MaxGenerations * this.problem.nvars);
527
528   this.state.ReproductionCount = {};
529   this.state.ReproductionCount.elite = this.opts.EliteCount;
530   this.state.ReproductionCount.children = fix(this.opts.CrossoverFraction *

```

```

531   (this.opts.PopulationSize - this.opts.EliteCount));
532
533   if(mod(this.state.ReproductionCount.children, 2) != 0) {
534     this.state.ReproductionCount.children = this.state.ReproductionCount.
535       children - 1;
536   }
537
538   this.state.ReproductionCount.mutants = this.opts.PopulationSize -
539     (this.state.ReproductionCount.elite + this.state.ReproductionCount.
540       children);
541
542   this.state.parentsCount = {};
543   this.state.parentsCount.crossover = this.state.ReproductionCount.children;
544   this.state.parentsCount.mutation = this.state.ReproductionCount.mutants;
545   this.state.nParents = this.state.parentsCount.crossover +
546     this.state.parentsCount.mutation;
547
548   this.state.parentsSelection = {};
549   this.state.parentsSelection.crossover = range(0,
550     this.state.parentsCount.crossover - 1);
551   this.state.parentsSelection.mutation = plus(
552     this.state.parentsCount.crossover,
553     range(0, this.state.parentsCount.mutation - 1));
554
555   this.selection.elite = range(0, this.state.ReproductionCount.elite - 1);
556   this.selection.children = plus(this.state.ReproductionCount.elite,
557     range(0, this.state.ReproductionCount.children - 1));
558   this.selection.mutants = plus((this.state.ReproductionCount.elite +
559     this.state.ReproductionCount.children),
560     range(0, this.state.ReproductionCount.mutants - 1));
561
562   if(this.constrained) {
563     this.state.ConSize = [
564       this.problem.nonlconfcn(zeros(this.problem.nvars)).length, this.opts.
565         PopulationSize];
566
567     this.state.ConVal = zeros(this.state.ConSize[0] * this.state.ConSize[1])
568       ;
569     this.state.ConNormVal = zeros(this.state.ConSize[0] * this.state.ConSize
570       [1]);
571   }
572
573   /**
574    * Updates the current state of the algorithm based on evaluations.
575    */
576   async updateState() {
577     await this.evalFitnessFcn();
578     if(this.constrained) {
579       await this.evalConstraintsFcn();
580       this.updatePenalty();
581     }
582     this.state.Time[this.state.Generation] = toc(this.state.GenTime);
583     var i;
584     [this.state.Best[this.state.Generation], i] = mini(this.state.FunVal);
585   }

```

```

581     setSub(this.state.x, index(range(0, this.problem.nvars - 1),
582           this.state.Generation, this.problem.nvars),
583           getSub(this.state.Population,
584           index(range(0, this.problem.nvars - 1), i, this.problem.nvars)));
585
586     if(this.constrained && this.state.ConSumVal[i] > 0) {
587       this.state.Feasible[this.state.Generation] = 0;
588     }
589   }
590
591 /**
592 * Updates penalty values based on constraint violations.
593 */
594 updatePenalty() {
595   if(any(this.state.ConSumVal.map((a) => a == 0))) {
596     if(this.state.Generation == 0) {
597       var [s, i] = sorti(this.state.FunVal);
598       this.selection.current_elite = getSub(i, this.selection.elite);
599     }
600     var fmin;
601     if(this.state.ReproductionCount.elite > 0) {
602       // Ne moze da bude bolje od najgore elitne jedinke
603       fmin = max(getSub(this.state.FunVal, this.selection.current_elite));
604     } else {
605       // Ne moze da bude bolje od najbolje jedinike
606       fmin = min(getSub(this.state.FunVal, this.selection.current_elite));
607     }
608     var I1 = indexOfAll(elementWise((a, b) => a > 0 && b <= fmin,
609                           this.state.ConSumVal, this.state.FunVal), true);
610     var I2 = indexOfAll(elementWise((a, b) => a > 0 && b > fmin,
611                           this.state.ConSumVal, this.state.FunVal), true);
612     setSub(this.state.FunVal, I1, plus(fmin,
613                                         getSub(this.state.ConSumVal, I1)));
614     setSub(this.state.FunVal, I2, plus(getSub(this.state.FunVal, I2),
615                                         getSub(this.state.ConSumVal, I2)));
616   } else {
617     this.state.FunVal = this.state.ConSumVal;
618   }
619 }
620
621 /**
622 * Evaluates the fitness function for the current population.
623 */
624 async evalFitnessFcn() {
625   var obj = this;
626   var nvars = this.problem.nvars;
627
628   var p;
629   var n;
630   if(this.state.ReproductionCount.elite > 0 && this.state.Generation > 0) {
631     p = getSub(this.state.Population, index(range(0, nvars - 1),
632                                               [...this.selection.children, ...this.selection.mutants], nvars));
633     n = this.state.ReproductionCount.children + this.state.ReproductionCount
634       .mutants;
635   } else {

```

```

635     p = this.state.Population;
636     n = this.opts.PopulationSize;
637 }
638
639 var val;
640 if(this.opts.UseVectorized) {
641     val = this.problem.fitnessfcn(p);
642 } else {
643     if(this.opts.UseParallel) {
644         val = await parallel.parfor(0, n-1, 1, parallel.getProcessorsNum(),
645             this.parallel_context, this.parallel_setup_fun, 'function(i) {
646                 var fun = ${obj.problem.fitnessfcn.toString()};
647                 return fun(getSub(${JSON.stringify(p)}), index(range(0, ${nvars} -
648                     1), i, ${nvars})));
649             } );
650     } else {
651         val = zeros(n);
652         for(var i = 0; i < n; i++) {
653             val[i] = this.problem.fitnessfcn(getSub(p, index(range(0, nvars - 1)
654                 , i, nvars)));
655         }
656     }
657
658     if(this.state.ReproductionCount.elite > 0 && this.state.Generation > 0) {
659         this.state.FunVal = [... getSub(this.state.FunVal, this.selection.
660             current_elite), ... val];
661     } else {
662         this.state.FunVal = val;
663     }
664
665     /**
666     * Evaluates constraint functions for the current population.
667     */
668     async evalConstraintsFcn() {
669         var obj = this;
670         var nvars = this.problem.nvars;
671
672         var p;
673         var n;
674         var m = this.state.ConSize[0];
675         if(this.state.ReproductionCount.elite > 0 && this.state.Generation > 0) {
676             p = getSub(this.state.Population, index(range(0, nvars - 1),
677                 [... this.selection.children, ... this.selection.mutants], nvars));
678             n = this.state.ReproductionCount.children + this.state.ReproductionCount
679                 .mutants;
680         } else {
681             p = this.state.Population;
682             n = this.opts.PopulationSize;
683         }
684
685         var val;
686         if(this.opts.UseVectorized) {
687             val = obj.problem.nonlconfcn(p);

```

```

686 } else {
687   if(this.opts.UseParallel) {
688     val = await parallel.parfor(0, n-1, 1, parallel.getProcessorsNum(),
689       this.parallel_context, this.parallel_setup_fun, 'function(i) {
690         var fun = ${obj.problem.nonlconfcn.toString()};
691         return fun(getSub(${JSON.stringify(p)}, index(range(0, ${nvars} -
692           1), i, ${nvars})));
693       };
694       val = val.flat();
695     } else {
696       m = this.state.ConSize[0];
697       val = zeros(m * n);
698       for(var i = 0; i < n; i++) {
699         setSub(val, index(range(0, m - 1), i, m),
700           this.problem.nonlconfcn(getSub(p,
701             index(range(0, nvars - 1), i, nvars))));
702       }
703     }
704   if(this.state.ReproductionCount.elite > 0 && this.state.Generation > 0) {
705     this.state.ConVal = concatCol(m, getSub(this.state.ConVal,
706       index(range(0, m - 1), this.selection.current_elite, m)), val);
707   } else {
708     this.state.ConVal = val;
709   }
710   this.normConstraintsFcn();
711   this.state.ConSumVal = sumCol(this.state.ConNormVal, m, this.opts.
712     PopulationSize);
713 }
714 /**
715 * Normalizes constraint values to maintain consistent scaling.
716 */
717 normConstraintsFcn() {
718   var ConNormVal = this.state.ConVal;
719   var sNaN = indexOfAll(isNaN(ConNormVal), true);
720   var sInf = indexOfAll(isInfinity(ConNormVal), true);
721   var sNeg = indexOfAll(isNegative(ConNormVal), true);
722
723   setSub(ConNormVal, sNaN, zeros(sNaN.length));
724   setSub(ConNormVal, sInf, zeros(sInf.length));
725   setSub(ConNormVal, sNeg, zeros(sNeg.length));
726
727   var m = this.state.ConSize[0];
728   var n = this.opts.PopulationSize;
729
730   if(this.state.Generation == 0 ||
731     (this.state.Generation > 0 &&
732      this.state.Feasible[this.state.Generation-1])) {
733
734     // Use first values of the first generation for normalization of
735     // constraints until there is a feasible solution
736     var l = elementWise((a) => sqrt(a),
737       sumRow(elementWise((a) => pow(a, 2), ConNormVal), m, n));
738     var I = indexOfAll(l, 0);

```

```

739     setSub(l, I, ones(I.length));
740     if(isEmpty(this.l)) {
741         this.l = l;
742     } else {
743         var s = indexOfAll(elementWise((a, b) => a > b, this.l, l), true);
744         setSub(this.l, s, getSub(l, s));
745     }
746     this.lm = repCol(this.l, n);
747 }
748
749
750     setSub(ConNormVal, sNaN, getSub(this.lm, sNaN));
751     setSub(ConNormVal, sInf, getSub(this.lm, sInf));
752     this.state.ConNormVal = divideEl(ConNormVal, this.lm);
753 }
754 }
755
756 exports.PRDC_JSLAB_OPTIM_RCMIGA = PRDC_JSLAB_OPTIM_RCMIGA;

```

Listing 105 - optim-rcmiga.js

```

1  /**
2   * @file JSLAB library optim submodule
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   */
7
8  var { PRDC_JSLAB_OPTIM_RCMIGA } = require('./optim-rcmiga');
9
10 /**
11  * Class for JSLAB optim submodule.
12  */
13 class PRDC_JSLAB_LIB_OPTIM {
14
15 /**
16  * Initializes the optim submodule.
17  * @param {Object} js1 Reference to the main JSLAB object.
18  */
19 constructor(js1) {
20     var obj = this;
21     this.js1 = js1;
22 }
23
24 /**
25  * Minimizes an unconstrained function using a coordinate descent-like
26  * Powell algorithm.
27  * @param {function} fnc Function to be minimized. Accepts an array of size
28  * N and returns a scalar.
29  * @param {Array} x0 Initial guess for the parameters as an array of size N.
30  * @param {Object} [options] Optional parameters:
31  *   - eps: Convergence threshold (default: 1e-6)
32  *   - alpha: Initial step size scaling factor (default: 0.001)
33  *   - stepSize: Finite difference step size for gradient estimation (
34  *     default: 1e-6)
35  *   - maxIterations: Maximum number of iterations to prevent infinite loops

```

```

    ( default: 1000)
  * @return {Object} An object with two fields:
  *   - argument: The parameter array that minimizes the function.
  *   - fncvalue: The function value at the minimized parameters.
 */
37 optimPowell(fnc, x0, options = {}) {
38   const {
39     eps = 1e-6,
40     alpha = 0.001,
41     stepSize = 1e-6,
42     maxIterations = 1000
43   } = options;
44
45   let convergence = false;
46   let x = x0.slice(); // Create a copy of the initial guess
47   let currentAlpha = alpha; // Current step size scaling factor
48
49   let pfx = Infinity; // Previous function value initialized to Infinity
50   let fx = fnc(x); // Current function value
51
52   let iteration = 0;
53   let dx;
54
55   while(!convergence && iteration < maxIterations) {
56     iteration++;
57     const indices = shuffleIndices(x);
58     convergence = true; // Assume convergence until a significant update is
      found
59
60     // Iterate over each variable in shuffled order
61     for(let i = 0; i < indices.length; i++) {
62       const idx = indices[i];
63
64       // Estimate the derivative (gradient) using finite differences
65       x[idx] += stepSize;
66       const fxi = fnc(x);
67       x[idx] -= stepSize;
68       dx = (fxi - fx) / stepSize;
69
70       // Check convergence based on the derivative
71       if(Math.abs(dx) > eps) {
72         convergence = false;
73       }
74
75       // Update the parameter by moving against the gradient
76       x[idx] -= currentAlpha * dx;
77
78       // Update the function value after the parameter change
79       fx = fnc(x);
80     }
81
82     // Adaptive step size adjustment
83     if(fx < pfx) {
84       currentAlpha *= 1.1; // Increase step size if improvement
85     } else {

```

```

86         currentAlpha *= 0.7; // Decrease step size if no improvement
87     }
88     pfx = fx;
89
90     // Optional: Log progress every 100 iterations
91     if(options.disp && iteration % 100 === 0) {
92         this.jsl.env.disp(`Iteration ${iteration}: f(x) = ${fx}`);
93     }
94 }
95
96 return { x, fx };
97 }
98
99 /**
100 * Performs optimization using the Nelder-Mead algorithm.
101 * @param {Function} f - The objective function to minimize. It should
102 *   accept an array of numbers and return a scalar value.
103 * @param {number[]} x0 - An initial guess for the parameters as an array of
104 *   numbers.
105 * @param {Object} [parameters] - Optional parameters to control the
106 *   optimization process.
107 * @param {number} [parameters.maxIterations=x0.length * 200] - Maximum
108 *   number of iterations to perform.
109 * @param {number} [parameters.nonZeroDelta=1.05] - Scaling factor for non-
110 *   zero initial steps in the simplex.
111 * @param {number} [parameters.zeroDelta=0.001] - Initial step size for
112 *   parameters that are initially zero.
113 * @param {number} [parameters.minErrorDelta=1e-6] - Minimum change in
114 *   function value to continue iterations.
115 * @param {number} [parameters.minTolerance=1e-5] - Minimum change in
116 *   parameters to continue iterations.
117 * @param {number} [parameters.rho=1] - Reflection coefficient.
118 * @param {number} [parameters.chi=2] - Expansion coefficient.
119 * @param {number} [parameters.psi=-0.5] - Contraction coefficient.
120 * @param {number} [parameters.sigma=0.5] - Reduction coefficient.
121 * @param {Array<Object>} [parameters.history] - Optional array to store the
122 *   history of simplex states for analysis.
123 */
124 * @returns {{ fx: number, x: number[] }} An object containing:
125 *   - 'fx': The minimum function value found.
126 *   - 'x': The parameters corresponding to the minimum function value.
127 */
128 optimNelderMead(...args) {
129     return this.jsl.env.fmin.nelderMead(...args);
130 }
131
132 /**
133 * Performs optimization using the Conjugate Gradient method.
134 * @param {Function} f - The objective function to minimize. It should
135 *   accept an array of numbers and return a scalar value and its gradient.
136 * @param {number[]} initial - An initial guess for the parameters as an
137 *   array of numbers.
138 * @param {Object} [params] - Optional parameters to control the
139 *   optimization process.
140 * @param {number} [params.maxIterations=initial.length * 20] - Maximum

```

```

    number of iterations to perform.

129  * @param {Array<Object>} [params.history] - Optional array to store the
   *           history of optimization steps for analysis.
130  *
131  * @returns {{ fx: number, x: number[], fxprime: number[] }} An object
   *           containing:
132  *           - 'fx': The minimum function value found.
133  *           - 'x': The parameters corresponding to the minimum function value.
134  *           - 'fxprime': The gradient of the function at the minimum.
135  */
136 optimConjugateGradient(...args) {
137   return this.jsl.env.fmin.conjugateGradient(...args);
138 }

139 /**
140  * Performs optimization using the Gradient Descent method.
141  * @param {Function} f - The objective function to minimize. It should
   *           accept an array of numbers and return a scalar value and its gradient.
142  * @param {number[]} initial - An initial guess for the parameters as an
   *           array of numbers.
143  * @param {Object} [params] - Optional parameters to control the
   *           optimization process.
144  * @param {number} [params.maxIterations=initial.length * 100] - Maximum
   *           number of iterations to perform.
145  * @param {number} [params.learnRate=0.001] - Learning rate or step size for
   *           each iteration.
146  * @param {Array<Object>} [params.history] - Optional array to store the
   *           history of optimization steps for analysis.
147  *
148  * @returns {{ fx: number, x: number[], fxprime: number[] }} An object
   *           containing:
149  *           - 'fx': The minimum function value found.
150  *           - 'x': The parameters corresponding to the minimum function value.
151  *           - 'fxprime': The gradient of the function at the minimum.
152  */
153 optimGradientDescent(...args) {
154   return this.jsl.env.fmin.gradientDescent(...args);
155 }

156 /**
157  * Performs optimization using the Gradient Descent method with Wolfe Line
158  * Search.
159  * @param {Function} f - The objective function to minimize. It should
   *           accept an array of numbers and return a scalar value and its gradient.
160  * @param {number[]} initial - An initial guess for the parameters as an
   *           array of numbers.
161  * @param {Object} [params] - Optional parameters to control the
   *           optimization process.
162  * @param {number} [params.maxIterations=initial.length * 100] - Maximum
   *           number of iterations to perform.
163  * @param {number} [params.learnRate=1] - Initial learning rate or step size
   *           for the line search.
164  * @param {number} [params.c1=1e-3] - Parameter for the Armijo condition in
   *           Wolfe Line Search.
165  * @param {number} [params.c2=0.1] - Parameter for the curvature condition
166

```

```

    in Wolfe Line Search.

167  * @param {Array<Object>} [ params.history ] - Optional array to store the
168  *   history of optimization steps for analysis, including line search
169  *   details.
170  * @returns {{ fx: number, x: number[], fxprime: number[] }} An object
171  *   containing:
172  *     - 'fx': The minimum function value found.
173  *     - 'x': The parameters corresponding to the minimum function value.
174  *     - 'fxprime': The gradient of the function at the minimum.
175  */
176 optimGradientDescentLineSearch(... args) {
177   return this.jsl.env.fmin.gradientDescentLineSearch(... args);
178 }
179 /**
180  * Performs root finding using the Bisection method.
181  * @param {Function} f - The function for which to find a root. It should
182  *   accept a number and return a number.
183  * @param {number} a - The start of the interval. Must satisfy f(a) and f(b)
184  *   have opposite signs.
185  * @param {number} b - The end of the interval. Must satisfy f(a) and f(b)
186  *   have opposite signs.
187  * @param {Object} [ parameters ] - Optional parameters to control the root-
188  *   finding process.
189  * @param {number} [ parameters.maxIterations=100 ] - Maximum number of
190  *   iterations to perform.
191  * @param {number} [ parameters.tolerance=1e-10 ] - Tolerance for convergence.
192  *   The method stops when the interval width is below this value.
193  * @returns {number} The root found within the interval [a, b].
194  */
195 optimBisect(... args) {
196   return this.jsl.env.fmin.bisect(... args);
197 }
198 /**
199  * Performs search using the Nelder-Mead algorithm.
200  * @param {Function} f - The objective function to minimize. It should
201  *   accept an array of numbers and return a scalar value.
202  * @param {number[]} x0 - An initial guess for the parameters as an array of
203  *   numbers.
204  * @param {Object} [ parameters ] - Optional parameters to control the
205  *   optimization process.
206  * @param {number} [ parameters.maxIterations=x0.length * 200 ] - Maximum
207  *   number of iterations to perform.
208  * @param {number} [ parameters.nonZeroDelta=1.05 ] - Scaling factor for non-
209  *   zero initial steps in the simplex.
210  * @param {number} [ parameters.zeroDelta=0.001 ] - Initial step size for
211  *   parameters that are initially zero.
212  * @param {number} [ parameters.minErrorDelta=1e-6 ] - Minimum change in
213  *   function value to continue iterations.
214  * @param {number} [ parameters.minTolerance=1e-5 ] - Minimum change in
215  *   parameters to continue iterations.
216  * @param {number} [ parameters.rho=1 ] - Reflection coefficient.
217  * @param {number} [ parameters.chi=2 ] - Expansion coefficient.
218  * @param {number} [ parameters.psi=-0.5 ] - Contraction coefficient.

```

```

204  * @param {number} [ parameters.sigma=0.5] - Reduction coefficient .
205  * @param {Array<Object>} [ parameters.history] - Optional array to store the
206  *           history of simplex states for analysis .
207  *
208  * @returns {{ fx: number, x: number[] }} An object containing:
209  *           - 'fx ': The minimum function value found .
210  *           - 'x ': The parameters corresponding to the minimum function value .
211  */
212 fminsearch(... args) {
213   return this.jsl.env.fmin.nelderMead(... args);
214 }
215 /**
216  * Finds the minimum of a univariate function within a specified interval
217  * using a bracketing method .
218  * @param {function} func - The function to minimize . Should accept a single
219  *           number and return a number .
220  * @param {number} a - The lower bound of the interval .
221  * @param {number} b - The upper bound of the interval .
222  * @param {number} [ tol=1e-5] - The tolerance for convergence (optional) .
223  * @returns {{ fx: number, x: number[] }} An object containing:
224  *           - 'fx ': The minimum function value found .
225  *           - 'x ': The x-value where the function attains its minimum within [a, b
226  *           ] .
227  */
228 fminbnd(func, a, b, tol = 1e-5) {
229   const eps = 1e-10; // A small value to prevent division by zero or to
230   // avoid precision issues .
231   const golden_ratio = (3 - Math.sqrt(5)) / 2;
232
233   let x = a + golden_ratio * (b - a);
234   let w = x;
235   let v = w;
236   let fx = func(x);
237   let fw = fx;
238   let fv = fw;
239
240   let d = 0;
241   let e = 0;
242
243   while(Math.abs(b - a) > tol) {
244     const m = 0.5 * (a + b);
245     const toll1 = tol * Math.abs(x) + eps;
246     const toll2 = 2 * toll1;
247
248     // Check for convergence
249     if(Math.abs(x - m) <= toll2 - 0.5 * (b - a)) {
250       break;
251     }
252
253     let u;
254     let useGolden = true;
255
256     // Try parabolic interpolation
257     if(Math.abs(e) > toll1) {

```

```

254     const r = (x - w) * (fx - fv);
255     const q = (x - v) * (fx - fw);
256     const p = (x - v) * q - (x - w) * r;
257     const q2 = 2 * (q - r);
258     const q2abs = Math.abs(q2);
259     if(q2abs > eps) {
260         u = x - p / q2;
261         if(a + tol1 <= u && u <= b - tol1 && Math.abs(u - x) < 0.5 * Math.
262             abs(e)) {
263             useGolden = false;
264         }
265     }
266
267 // If parabolic interpolation is not used , fall back to golden section
268 if(useGolden) {
269     if(x < m) {
270         u = x + golden_ratio * (b - x);
271     } else {
272         u = x - golden_ratio * (x - a);
273     }
274     e = d;
275 } else {
276     e = d;
277 }
278
279 const fu = func(u);
280
281 // Update a, b, v, w, x
282 if(fu <= fx) {
283     if(u < x) b = x;
284     else a = x;
285     v = w; fv = fw;
286     w = x; fw = fx;
287     x = u; fx = fu;
288 } else {
289     if(u < x) a = u;
290     else b = u;
291     if(fu <= fw || w === x) {
292         v = w; fv = fw;
293         w = u; fw = fu;
294     } else if(fu <= fv || v === x || v === w) {
295         v = u; fv = fu;
296     }
297 }
298
299 return { x, fx };
300 }
301
302 /**
303 * Creates an instance of PRDC_JSLAB_LIB_OPTIM_RCMIGA.
304 * @param {Object} problem - The optimization problem definition.
305 * @param {Object} opts - Configuration options for the algorithm.
306 */

```

```

308   rcmiga (... args) {
309     return new PRDC_JSLAB_OPTIM_RCMIGA (... args);
310   }
311 }
312 }
313
314 exports.PRDC_JSLAB_LIB_OPTIM = PRDC_JSLAB_LIB_OPTIM;

```

Listing 106 - optim.js

```

1  /**
2   * @file JSLAB library parallel submodule
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   */
7
8  /**
9   * Class for JSLAB parallel submodule.
10  */
11 class PRDC_JSLAB_PARALLEL {
12
13  /**
14   * Constructs parallel submodule object with access to JSLAB's parallel
15   * functions.
16   * @constructor
17   * @param {Object} jsl - Reference to the main JSLAB object.
18   */
19  constructor(jsl) {
20    this.jsl = jsl;
21    this.worker_pool = [];
22    this.task_queue = [];
23    this.is_initialized = false;
24  }
25
26  /**
27   * Retrieves the number of logical processors available.
28   * @returns {number} Number of processors.
29   */
30  getProcessorsNum() {
31    return this.jsl.env.processors_number || 4;
32  }
33
34  /**
35   * Generates the worker's internal script.
36   * @param {Object} [context={}] - Optional context to pass to the
37   * work_function.
38   * @param {Function|String} work_function_str - The work function to execute
39   *
40   * @param {Function|String} [setup_function_str] - Optional setup function
41   * to execute on init.
42   */
43  workerFunction(context = {}, setup_function_str = "") {
44    return `
45      self.addEventListener("message", async function(e) {

```

```

43     if(e.data.type === 'execute') {
44       try {
45         const { work_fun_str, args } = e.data;
46
47         // Reconstruct the work function
48         const work_function = new Function('return ' + work_fun_str)();
49
50         // Execute the work function with provided arguments
51         const result = await work_function(...args);
52
53         // Send back the result
54         self.postMessage({ type: 'result', result });
55       } catch(err) {
56         self.postMessage({ type: 'error', error: err });
57       }
58     }
59   );
60
61   // Assign context variables
62   Object.assign(self, ${JSON.stringify(context)});
63
64   // Reconstruct and execute the setup function if provided
65   ${setup_function_str}.then(function() {
66     self.postMessage({ type: 'ready' });
67   });
68   ';
69 }
70
71 /**
72 * Initializes the worker pool with the specified number of workers.
73 * @param {number} num_workers - Number of workers to initialize.
74 * @param {Object} [context={}] - Optional context to pass to the
75 *   work_function.
76 * @param {Function|String} [setup_function_str] - Optional setup function
77 *   to execute on init.
78 */
79 init(num_workers, context = {}, setup_function_str = "") {
80   if(this.is_initialized) return;
81
82   if(!num_workers || num_workers <= 0) {
83     num_workers = this.getProcessorsNum();
84   }
85
86   const worker_script =
87     `${this.jsl.getWorkerInit()}${this.workerFunction(context, setup_function_str)}';
88
89   if(config.DEBUG_PARALLEL_WORKER_SETUP_FUN) {
90     this.jsl._console.log(worker_script);
91   }
92
93   const blob = new Blob([worker_script], { type: 'application/javascript' })
94   ;
95   const blobURL = URL.createObjectURL(blob);

```

```
95
96     for(let i = 0; i < num_workers; i++) {
97         const worker = new Worker(blobURL);
98         worker.busy = false;
99         worker.ready = false;
100        this.worker_pool.push(worker);
101    }
102
103    this.is_initialized = true;
104 }
105
106 /**
107 * Assigns tasks from the queue to available workers.
108 */
109 assignTasksToWorkers() {
110     for(const worker of this.worker_pool) {
111         if(!worker.busy && this.task_queue.length > 0) {
112             const task = this.task_queue.shift();
113             worker.busy = true;
114
115             if(config.DEBUG_PARALLEL_WORKER_WORK_FUN) {
116                 this.jsl._console.log(task);
117             }
118
119             function executeTask() {
120                 worker.postMessage({
121                     type: 'execute',
122                     work_fun_str: task.work_function_str,
123                     args: task.args,
124                 });
125             }
126
127             worker.onmessage = (e) => {
128                 if(e.data.type === 'ready') {
129                     worker.ready = true;
130                     executeTask();
131                 } else if(e.data.type === 'result') {
132                     task.resolve(e.data.result);
133                 } else if(e.data.type === 'error') {
134                     task.reject(new Error(e.data.error));
135                 }
136                 worker.busy = false;
137                 this.assignTasksToWorkers();
138             };
139
140             worker.onerror = (e) => {
141                 task.reject(new Error(e.message));
142                 worker.busy = false;
143                 this.assignTasksToWorkers();
144             };
145
146             if(worker.ready) {
147                 executeTask();
148             }
149         }
150     }
151 }
```

```

150     }
151   }
152
153  /**
154   * Enqueues a task to be executed by the worker pool.
155   * @param {Object} context - Context variables to assign in the worker.
156   * @param {Function|String} [setup_function] - Optional setup function to
157   *   execute on init.
158   * @param {Array} args - Arguments to pass to the work_function.
159   * @param {Function|String} work_function - The work function to execute.
160   * @param {boolean} reset_workers - Whether to reset all workers or not.
161   * @returns {Promise} - Resolves with the result of the work_function.
162   */
163   run(context = {}, setup_function = null, args = [] , work_function ,
164       reset_workers = false) {
165     var setup_function_str = setup_function;
166     if(typeof setup_function_str !== 'string') {
167       setup_function_str = this.jsl.eval.rewriteCode(this.jsl.eval.
168           getFunctionBody(setup_function)).code;
169     }
170     var work_function_str = work_function;
171     if(typeof work_function_str !== 'string') {
172       work_function_str = work_function.toString();
173     }
174     if(reset_workers) {
175       this.terminate();
176     }
177     if(!this.is_initialized) {
178       this.init(0, context, setup_function_str);
179     }
180     return new Promise((resolve , reject) => {
181       this.task_queue.push({
182         work_function_str ,
183         args ,
184         resolve ,
185         reject ,
186       });
187     }
188
189  /**
190   * Executes a parallel for loop by dividing the iteration range among
191   * workers.
192   * @param {number} start - The initial value of the loop counter.
193   * @param {number} end - The terminating value of the loop counter.
194   * @param {number} [step=1] - The amount by which to increment the loop
195   *   counter each iteration.
196   * @param {number} [num_workers=this.getProcessorsNum()] - The number of
197   *   workers to use.
198   * @param {Object} [context={}] - Optional context to pass to the
199   *   work_function.
200   * @param {Function} [setup_function=null] - Optional setup function to

```

```

    execute before work function.
198  * @param {Function} work_function - The function to execute on each
199   iteration.
200  * @param {boolean} reset_workers - Wether to reset all workers or not.
201  * @returns {Promise<Array>} - A promise that resolves to an array of
202   results.
203  */
204  async parfor(start, end, step = 1, num_workers, context,
205   setup_function, work_function, reset_workers = false) {
206   var setup_function_str = this.jsl.eval.rewriteCode(this.jsl.eval.
207     getFunctionBody(setup_function)).code;
208   if(reset_workers) {
209     this.terminate();
210   }
211   if(!this.is_initialized) {
212     this.init(num_workers, context, setup_function_str);
213   }
214
215   const isAscending = (end - start) * step > 0;
216
217   if(!isAscending) {
218     this.jsl.env.error('@parfor: '+language.string(197));
219   }
220
221   const total_items = Math.ceil(Math.abs((end - start) / step)) + 1;
222   const chunk_size = Math.ceil(total_items / num_workers);
223   const tasks = [];
224
225   const task_function = async function(chunk_start, chunk_end, step,
226     work_fun_str) {
227     const work_function = new Function('return ' + work_fun_str)();
228     const results = [];
229     for(let i = chunk_start; i <= chunk_end; i += step) {
230       const result = await work_function(i);
231       results.push(result);
232     }
233     return results;
234   };
235   var task_function_str = task_function.toString();
236
237   for(let worker_index = 0; worker_index < num_workers; worker_index++) {
238     const chunk_start_index = worker_index * chunk_size;
239     const chunk_end_index = Math.min(chunk_start_index + chunk_size,
240       total_items);
241
242     if(chunk_start_index >= chunk_end_index) {
243       break;
244     }
245
246     const chunk_start = start + chunk_start_index * step;
247     let chunk_end = start + (chunk_end_index * step) - step;

```

```

247
248     if(chunk_end > end) {
249         chunk_end = end;
250     }
251
252     var work_function_str = work_function;
253     if(typeof work_function === 'function') {
254         work_function_str = work_function.toString();
255     }
256
257     const task = this.run(
258         context,
259         setup_function_str,
260         [chunk_start, chunk_end, step, work_function_str],
261         task_function_str
262     );
263
264     tasks.push(task);
265 }
266
267     const nested_results = await Promise.all(tasks);
268     return nested_results.flat();
269 }
270
271 /**
272 * Terminates all workers and resets the worker pool.
273 */
274 terminate() {
275     for(const worker of this.worker_pool) {
276         worker.terminate();
277     }
278     this.worker_pool = [];
279     this.task_queue = [];
280     this.is_initialized = false;
281 }
282 }
283
284 exports.PRDC_JSLAB_PARALLEL = PRDC_JSLAB_PARALLEL;

```

Listing 107 - parallel.js

```

1 /**
2  * @file JSLAB library path submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 /**
9  * Class for JSLAB path submodule.
10 */
11 class PRDC_JSLAB_LIB_PATH {
12
13 /**
14  * Initializes a new instance of the path submodule, providing access to
15  * path manipulation utilities.

```

```

15   * @param {Object} js1 Reference to the main JSLAB object .
16   */
17 constructor(js1) {
18   var obj = this;
19   this.js1 = js1;
20 }
21
22 /**
23 * Extracts the directory of file .
24 * @param {String} path The filesystem path from which to extract the
25 *   directory .
26 * @returns {String} The directory from the given path .
27 */
28 getDir(path) {
29   return this.js1.env.pathDirName(path);
30 }
31
32 /**
33 * Extracts the name of the directory from a given filesystem path .
34 * @param {String} path The filesystem path from which to extract the
35 *   directory name .
36 * @returns {String} The name of the directory from the given path .
37 */
38 getDirName(path) {
39   return path.replaceAll('\\', '/').match(/(^[/]* )/*$/) [1];
40 }
41
42 /**
43 * Retrieves the platform-specific path separator character .
44 * @returns {String} The path separator character used by the system .
45 */
46 pathSep() {
47   return this.js1.env.pathGetSep();
48 }
49
50 /**
51 * Determines if the current path is absolute .
52 * @returns {boolean} True if the current path is absolute , false otherwise .
53 */
54 isAbsolutePath() {
55   return this.js1.env.pathIsAbsolute();
56 }
57
58 /**
59 * Retrieves the file name from the provided file path .
60 * @param {string} path - The complete file path .
61 * @returns {string} The file name extracted from the path .
62 */
63 pathFileName(path) {
64   return this.js1.env.pathFileName(path);
65 }
66
67 /**
68 * Returns the last portion of a path , similar to the Unix ‘basename’
69 * command .

```

```
67     * @param {string} path - The file path to process.
68     * @returns {string} The last segment of the path.
69     */
70   pathBaseName(path) {
71     return this.jsl.env.pathBaseName(path);
72   }
73
74   /**
75    * Retrieves the file extension from the provided file path.
76    * @param {string} path - The complete file path.
77    * @returns {string} The file extension extracted from the path.
78    */
79   pathFileExt(path) {
80     return this.jsl.env.pathExtName(path);
81   }
82
83   /**
84    * Resolves a sequence of path segments into an absolute path using the
85    * environment's path resolver.
86    * @param {string} path - The path or sequence of paths to resolve.
87    * @returns {string} - The resolved absolute path.
88    */
89   pathResolve(path) {
90     return this.jsl.env.pathResolve(path);
91   }
92
93   /**
94    * Normalizes a given path, resolving '..' and '.' segments using the
95    * environment's path normalizer.
96    * @param {string} path - The path to normalize.
97    * @returns {string} - The normalized path.
98    */
99   pathNormalize(path) {
100     return this.jsl.env.pathNormalize(path);
101   }
102
103   /**
104    * Compares two file paths after resolving them to their absolute forms to
105    * check if they refer to the same location.
106    * @param {string} path1 - The first file path to compare.
107    * @param {string} path2 - The second file path to compare.
108    * @returns {boolean} Returns true if both paths resolve to the same
109    * absolute path, otherwise false.
110    */
111   comparePaths(path1, path2) {
112     return this.jsl.env.pathResolve(path1) === this.jsl.env.pathResolve(path2)
113     ;
114   }
115
116   /**
117    * Generates a unique filesystem path by appending a number to the input
118    * path if the original path exists.
119    * @param {String} path The base path for which a unique version is required
120    *
121    * @returns {String} A unique filesystem path based on the input path.
```

```

115   */
116   getUniquePath(path) {
117     var i = 0;
118     var unique_path = path;
119     while(fs.existsSync(unique_path)) {
120       i = i+1;
121       unique_path = path+i;
122     }
123     return unique_path;
124   }
125
126 /**
127 * Generates a unique filename by appending a number to the original path if
128 * it already exists.
129 * @param {string} path - The original file path.
130 * @param {string} ext - The original file extension.
131 * @returns {string} A unique folder path.
132 */
133 getUniqueFilename(filename, ext) {
134   var i = 0;
135   var unique_filename = filename+'.'+ext;
136   while(fs.existsSync(unique_filename)) {
137     i = i+1;
138     unique_filename = filename+i+'.'+ext;
139   }
140   return unique_filename;
141 }
142
143 exports.PRDC_JSLAB_LIB_PATH = PRDC_JSLAB_LIB_PATH;

```

Listing 108 - path.js

```

1 /**
2  * @file JSLAB library render submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 /**
9  * Class for JSLAB render submodule.
10 */
11 class PRDC_JSLAB_LIB_RENDER {
12
13 /**
14  * Initializes a new instance of the render submodule.
15  * @param {Object} js1 Reference to the main JSLAB object.
16 */
17 constructor(js1) {
18   var obj = this;
19   this.js1 = js1;
20 }
21
22 /**
23  * Debounces a function, ensuring it's only invoked once at the beginning of

```

```
        consecutive calls during the wait period.
24     * @param {Function} func - The function to debounce.
25     * @param {number} wait - The period to wait before allowing another call ,
26         in milliseconds.
27     * @returns {Function} The debounced function.
28     */
29     debounceIn(func, wait) {
30         var timeout;
31         let last_args;
32
33         return function (...args) {
34             var context = this;
35             last_args = args;
36             var later = function () {
37                 timeout = null;
38             };
39             if (!timeout) {
40                 timeout = setTimeout(later, wait);
41                 func.apply(context, last_args);
42             }
43         };
44
45     /**
46      * Debounces a function, calling it at the first and last of consecutive
47      * calls during the wait period.
48      * @param {Function} func - The function to debounce.
49      * @param {number} wait - The period to wait before allowing another call ,
50          in milliseconds.
51      * @returns {Function} The debounced function.
52      */
53     debounceInOut(func, wait) {
54         var timeout;
55         var hit = false;
56         let last_args;
57
58         return function (...args) {
59             var context = this;
60             last_args = args;
61             var later = function () {
62                 if (hit) {
63                     hit = false;
64                     func.apply(context, last_args);
65                     setTimeout(later, wait);
66                 } else {
67                     timeout = null;
68                 }
69             };
70             if (!timeout) {
71                 timeout = setTimeout(later, wait);
72                 func.apply(context, last_args);
73             } else {
74                 hit = true;
75             }
76         };
77     };
78 }
```

```

75   }
76
77  /**
78   * Debounces a function, ensuring it's only invoked once at the end of
79   * consecutive calls during the wait period.
80   * @param {Function} func - The function to debounce.
81   * @param {number} wait - The period to wait before allowing another call,
82   *           in milliseconds.
83   * @returns {Function} The debounced function.
84   */
85   debounceOut(func, wait) {
86     var timeout;
87     let last_args;
88
89     return function(...args) {
90       var context = this;
91       last_args = args;
92       var later = function() {
93         timeout = null;
94         func.apply(context, last_args);
95       };
96       if (!timeout) {
97         timeout = setTimeout(later, wait);
98       }
99     };
100 }
101 exports.PRDC_JSLAB_LIB_RENDER = PRDC_JSLAB_LIB_RENDER;

```

Listing 109 - render.js

```

1 /**
2  * @file JSLAB library symbolic submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 /**
9  * Class for JSLAB symbolic submodule.
10 */
11 class PRDC_JSLAB_SYMBOLIC_MATH {
12
13 /**
14  * Constructs a symbolic submodule object with access to JSLAB's symbolic
15  * functions.
16  * @constructor
17  * @param {Object} js1 - Reference to the main JSLAB object.
18  */
19 constructor(js1) {
20   var obj = this;
21   this.js1 = js1;
22
23   this.loaded = false;

```

```

24     this._var_counter = 0;
25     this._symbols = [];
26   }
27
28 /**
29 * Loads the symbolic math libraries (SymPy and NumPy) using Pyodide.
30 * Initializes the Python environment for symbolic computations.
31 * @returns {Promise<void>} A promise that resolves when the libraries are
32 * loaded.
33 */
34 async load() {
35   if(!this.loaded) {
36     this.pyodide = await loadPyodide({
37       indexURL: app_path+'lib/sympy-0.26.2/',
38     });
39     await this.pyodide.loadPackage(['sympy', 'numpy'], { messageCallback: () => {}});
40     this.loaded = true;
41     this.pyodide.runPython('globals().clear()');
42     this.pyodide.runPython(`
43       from sympy import *
44       import numpy as np
45     `);
46   }
47
48 /**
49 * Generates the next unique variable name for symbolic expressions.
50 * @returns {string} The next unique variable name (e.g., 'jslabVar1').
51 */
52 _nextVar() {
53   this._var_counter += 1;
54   return 'jslabVar'+this._var_counter;
55 }
56
57 /**
58 * Creates a new symbolic variable with an optional name and value.
59 * @param {string} [name] - The name of the symbolic variable. If undefined,
60 *   a unique name is generated.
61 * @param {*} [value] - The initial value of the symbolic variable.
62 * @returns {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL} The newly created symbolic
63 *   variable.
64 */
65 _newSymbol(name, value) {
66   if(typeof name === 'undefined') {
67     name = this._nextVar();
68   }
69   var symbol = new PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL(name, value);
70   this._symbols.push(symbol);
71   return symbol;
72 }
73 /**
74 * Retrieves the name of a symbolic variable.
75 * @param {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL|string} symbol - The symbolic

```

```

    variable or its name.
  * @returns {string} The name of the symbolic variable.
  */
getSymbolName(symbol) {
  if(typeof symbol === 'object' && symbol.constructor.name === 'PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL') {
    return symbol.name;
  } else {
    return symbol;
  }
}

/**
 * Checks if the symbolic libraries are loaded. Throws an error if not.
 * @throws {Error} If the symbolic libraries are not loaded.
 */
checkLoaded() {
  if(!this.loaded) {
    this.jsl.env.error('@sym: '+language.string(175));
  }
}

/**
 * Evaluates a Python code string within the symbolic math environment.
 * @param {string} code - The Python code to evaluate.
 * @returns {*} The result of the evaluated code.
 * @throws {Error} If there is an error during code evaluation.
 */
eval(code) {
  this.checkLoaded();
  if(this.jsl.config.DEBUG_SYM_PYTHON_EVAL_CODE) {
    this.jsl._console.log('@sym: eval: ' + code);
  }
  try {
    return this.pyodide.runPython(code);
  } catch(err) {
    this.jsl.env.error('@sym: ' + err);
  }
  return false;
}

/**
 * Creates a single symbolic variable.
 * @param {string} name - The name of the symbolic variable.
 * @returns {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL} The created symbolic variable.
 */
sym(name) {
  this.checkLoaded();

  this.eval(`$name = Symbol(`'$name`')`);
  return this._newSymbol(name, name);
}

/**
 * Creates multiple symbolic variables.
 */

```

```

128  * @param {string[]} names - An array of names for the symbolic variables.
129  * @returns {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL[]} An array of created symbolic
130  * variables.
131  */
132  syms(names) {
133    var obj = this;
134    this.checkLoaded();
135
136    this.eval(` ${names.join(' ', ' )} = symbols(` ${names.join(' ', ' )} `)` );
137
138    var symbols = [];
139    names.forEach(function(name) {
140      symbols.push(obj._newSymbol(name, name));
141    });
142    return symbols;
143  }
144
145 /**
146  * Creates a symbolic matrix from a nested array expression.
147  * @param {Array<Array<PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL| string | number>>} expr
148  *   - The nested array representing the matrix.
149  * @returns {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL} The symbolic matrix.
150  */
151 mat(expr) {
152   this.checkLoaded();
153
154   expr = JSON.stringify(expr.map(row => row.map(el => this.getSymbolName(el))));
155
156   var symbol = this._newSymbol();
157   symbol.setValue(this.eval(`
158     ${symbol.name} = Matrix(${expr})
159     ${symbol.name}
160   `));
161   return symbol;
162 }
163
164 /**
165  * Multiplies multiple symbolic expressions.
166  * @param {...PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL| string | number} args - The
167  *   symbolic expressions to multiply.
168  * @returns {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL} The resulting symbolic
169  *   expression after multiplication.
170  */
171 mul(...args) {
172   var obj = this;
173   this.checkLoaded();
174
175   var expr = args
176     .map(function(item) {
177       return obj.getSymbolName(item);
178     }).join(' * ');
179
180   var symbol = this._newSymbol();
181   symbol.setValue(this.eval(`
```

```

178     ${symbol.name} = ${expr}
179     ${symbol.name}
180   ') );
181   return symbol;
182 }
183
184 /**
185 * Divides multiple symbolic expressions.
186 * @param {...PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL|string|number} args - The
187 *   symbolic expressions to divide.
188 * @returns {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL} The resulting symbolic
189 *   expression after division.
190 */
191 div(...args) {
192   var obj = this;
193   this.checkLoaded();
194
195   var expr = args
196     .map(function(item) {
197       return obj.getSymbolName(item);
198     }).join(' / ');
199
200   var symbol = this._newSymbol();
201   symbol.setValue(this.eval(`
202     ${symbol.name} = ${expr}
203     ${symbol.name}
204   ') );
205   return symbol;
206 }
207
208 /**
209 * Adds multiple symbolic expressions.
210 * @param {...PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL|string|number} args - The
211 *   symbolic expressions to add.
212 * @returns {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL} The resulting symbolic
213 *   expression after addition.
214 */
215 plus(...args) {
216   var obj = this;
217   this.checkLoaded();
218
219   var expr = args
220     .map(function(item) {
221       return obj.getSymbolName(item);
222     }).join(' + ');
223
224   var symbol = this._newSymbol();
225   symbol.setValue(this.eval(`
226     ${symbol.name} = ${expr}
227     ${symbol.name}
228   ') );
229   return symbol;
230 }
231
232 /**

```

```

229     * Subtracts multiple symbolic expressions.
230     * @param {...PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL| string | number} args - The
231         symbolic expressions to subtract.
232     * @returns {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL} The resulting symbolic
233         expression after subtraction.
234     */
235     minus (... args) {
236         var obj = this;
237         this.checkLoaded ();
238
239         var expr = args
240             .map(function(item) {
241                 return obj.getSymbolName(item);
242             }).join(' - ');
243
244         var symbol = this._newSymbol ();
245         symbol.setValue(this.eval(
246             `${symbol.name} = ${expr}
247             ${symbol.name}
248         ));
249         return symbol;
250     }
251
252     /**
253      * Raises a symbolic expression to a power.
254      * @param {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL| string | number} expr - The base
255          expression.
256      * @param {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL| string | number} n - The exponent .
257      * @returns {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL} The resulting symbolic
258          expression after exponentiation.
259      */
260     pow(expr , n) {
261         this.checkLoaded ();
262
263         expr = this.getSymbolName(expr);
264         n = this.getSymbolName(n);
265
266         var symbol = this._newSymbol ();
267         symbol.setValue(this.eval(
268             `${symbol.name} = ${expr}**${n}
269             ${symbol.name}
270         ));
271         return symbol;
272     }
273
274     /**
275      * Transposes a symbolic matrix expression.
276      * @param {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL| string} expr - The matrix
277          expression to transpose.
278      * @returns {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL} The transposed matrix
279          expression.
280      */
281     transp(expr) {
282         this.checkLoaded ();
283     }

```

```
278     expr = this.getSymbolName(expr);
279
280     var symbol = this._newSymbol();
281     symbol.setValue(this.eval(
282         ${symbol.name} = ${expr}.T
283         ${symbol.name}
284     ));
285     return symbol;
286 }
287
288 /**
289 * Computes the inverse of a symbolic matrix expression.
290 * @param {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL|string} expr - The matrix
291 *   expression to invert.
292 * @returns {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL} The inverse of the matrix
293 *   expression.
294 */
295 inv(expr) {
296     this.checkLoaded();
297
298     expr = this.getSymbolName(expr);
299
300     var symbol = this._newSymbol();
301     symbol.setValue(this.eval(
302         ${symbol.name} = ${expr}.inv()
303         ${symbol.name}
304     ));
305     return symbol;
306 }
307
308 /**
309 * Computes the determinant of a symbolic matrix expression.
310 * @param {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL|string} expr - The matrix
311 *   expression whose determinant is to be computed.
312 * @returns {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL} The determinant of the matrix
313 *   expression.
314 */
315 det(expr) {
316     this.checkLoaded();
317
318     expr = this.getSymbolName(expr);
319
320     var symbol = this._newSymbol();
321     symbol.setValue(this.eval(
322         ${symbol.name} = ${expr}.det()
323         ${symbol.name}
324     ));
325     return symbol;
326 }
327
328 /**
329 * Differentiates a symbolic expression with respect to a variable.
330 * @param {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL|string} expr - The expression to
331 *   differentiate.
332 * @param {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL|string} x - The variable with
```

```

    respect to which differentiation is performed.
328  * @param {number} [n=1] - The order of differentiation.
329  * @returns {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL} The differentiated expression.
330  */
331 diff(expr, x, n = 1) {
332   this.checkLoaded();
333
334   expr = this.getSymbolName(expr);
335
336   var symbol = this._newSymbol();
337   symbol.setValue(this.eval(`
338     ${symbol.name} = diff(${expr}, ${x}, ${n})
339     ${symbol.name}
340   `));
341   return symbol;
342 }
343
344 /**
345  * Integrates a symbolic expression with respect to a variable.
346  * @param {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL|string} expr - The expression to
347  * integrate.
348  * @param {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL|string} x - The variable with
349  * respect to which integration is performed.
350  * @param {Array<PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL|string>|undefined} lims -
351  * The limits of integration as [lower, upper]. If undefined, indefinite
352  * integration is performed.
353  * @returns {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL} The integrated expression.
354  */
355 intg(expr, x, lims) {
356   this.checkLoaded();
357
358   expr = this.getSymbolName(expr);
359   x = this.getSymbolName(x);
360
361   var symbol = this._newSymbol();
362   if(lims) {
363     lims[0] = this.getSymbolName(lims[0]);
364     lims[1] = this.getSymbolName(lims[1]);
365     symbol.setValue(this.eval(`
366       ${symbol.name} = integrate(${expr}, (${x}, ${lims[0]}, ${lims[1]}))
367       ${symbol.name}
368     `));
369   } else {
370     symbol.setValue(this.eval(`
371       ${symbol.name} = integrate(${expr}, ${x})
372       ${symbol.name}
373     `));
374   }
375   return symbol;
376 }
377
378 /**
379  * Substitutes a value into a symbolic expression.
380  * @param {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL|string} expr - The expression in
381  * which to substitute.

```



```
377 * @param {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL|string} x - The variable to
378 * substitute.
379 * @param {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL|string|number|Array} val - The
380 * value or array of values to substitute.
381 * @returns {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL} The expression after
382 * substitution.
383 */
384 subs(expr, x, val) {
385     this.checkLoaded();
386
387     var symbol = this._newSymbol();
388     if(Array.isArray(val)) {
389         val = JSON.stringify(val);
390         symbol.setValue(this.eval(
391             `${symbol.name} = ${expr}.subs(${x}, ${val}).evalf() for ${val} in ${val}`)
392             `${symbol.name}`);
393     } else {
394         val = this.getSymbolName(val);
395         if(isNumber(val)) {
396             symbol.setValue(this.eval(
397                 `${symbol.name} = ${expr}.subs(${x}, ${val}).evalf()`)
398                     `${symbol.name}`);
399         } else {
400             symbol.setValue(this.eval(
401                 `${symbol.name} = ${expr}.subs(${x}, ${val})`)
402                     `${symbol.name}`);
403         }
404     }
405     return symbol;
406 }
407 */
408 /**
409 * Simplifies a symbolic expression.
410 * @param {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL|string} expr - The expression to
411 * simplify.
412 * @returns {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL} The simplified expression.
413 */
414 simplify(expr) {
415     this.checkLoaded();
416
417     expr = this.getSymbolName(expr);
418
419     var symbol = this._newSymbol();
420     symbol.setValue(this.eval(
421         `${symbol.name} = simplify(${expr})`)
422             `${symbol.name}`);
423     return symbol;
424 }
425 */
426 /**
427 */
```

```

428
429 /**
430 * Displays the LaTeX representation of a symbolic expression.
431 * @param {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL|string} expr - The expression to
432 *   display in LaTeX format.
433 */
434 showLatex(expr) {
435     expr = this.getSymbolName(expr);
436     this.jsl.env.dispLatex(this.eval(`\\$` + latex(`\\${expr}`)));
437 }
438 /**
439 * Displays the LaTeX string of a symbolic expression.
440 * @param {PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL|string} expr - The expression to
441 *   convert to a LaTeX string.
442 */
443 dispLatex(expr) {
444     expr = this.getSymbolName(expr);
445     this.jsl.env.disp(this.eval(`\\$` + latex(`\\${expr}`)));
446 }
447 /**
448 * Clears all symbolic variables and resets the symbolic math environment.
449 */
450 clear() {
451     if(this.loaded) {
452         this.pyodide.runPython(`global().clear()`);
453         this.pyodide.runPython(`from sympy import *
454         import numpy as np
455         `);
456         this._var_counter = 0;
457         this._symbols = [];
458     }
459 }
460 }
461 }

462 exports.PRDC_JSLAB_SYMBOLIC_MATH = PRDC_JSLAB_SYMBOLIC_MATH;
463

464 /**
465 * Class for JSLAB symbolic math symbol.
466 */
467 class PRDC_JSLAB_SYMBOLIC_MATH_SYMBOL {
468
469 /**
470 * Constructs a symbolic math symbol with a name and initial value.
471 * @constructor
472 * @param {string} name - The name of the symbolic variable.
473 * @param {*} value - The initial value of the symbolic variable.
474 */
475 constructor(name, value) {
476     this.name = name;
477     this.value = value;
478 }
479
480 }
```



```
481  /**
482   * Sets the value of the symbolic variable.
483   * @param {*} value - The new value to assign to the symbolic variable.
484   */
485 setValue(value) {
486     this.value = value;
487     this.parsed_value = parseString(value.toString());
488 }
489 /**
490  * Returns a string representation of the vector.
491  * @returns {string} The string representation in the format 'Vector(x:, y:, z:)'.
492  */
493 toString() {
494   return `Symbolic(${JSON.stringify(this.toJSON(), null, 2)})`;
495 }
496 /**
497  * Converts the symbolic value to a numeric value.
498  * @returns {*} The numeric representation of the symbolic value.
499  */
500 toNumeric() {
501   return this.parsed_value;
502 }
503 /**
504  * Converts the symbolic value to a JSON string.
505  * @returns {string} The JSON string representation of the symbolic value.
506  */
507 toJSON() {
508   if(typeof this.parsed_value === undefined) {
509     return this.name;
510   }
511   return this.parsed_value;
512 }
513 /**
514  * Converts the object to a safe JSON representation.
515  * @returns {Object} The safe JSON representation of the object.
516  */
517 toSafeJSON() {
518   return this.toJSON();
519 }
520 /**
521  * Converts the object to a pretty string representation.
522  * @returns {string} The pretty string representation of the object.
523  */
524 toPrettyString() {
525   return this.toString();
526 }
527 }
```

```

535  * Parses a string representation of Python objects into JavaScript objects.
536  * Specifically handles conversion of SymPy Matrix objects to nested arrays.
537  * @param {string} str - The string to parse.
538  * @returns {Array<Array<*>>|Array<Array<Array<*>>>} The parsed JavaScript
539  * representation of the input string.
540  */
541 function parseString(str) {
542   // Remove any leading or trailing square brackets if the input is a list
543   const trimmed_str = str.trim().replace(/^\[\|\]$/g, '');
544
545   // Find all occurrences of "Matrix ([[...]])" using regex
546   const matrix_regex = /Matrix\(\[\[\(.+\?)\]\]\)/g;
547   const matrices = [];
548   let match;
549
550   // Iterate over all matches of the regex
551   while((match = matrix_regex.exec(trimmed_str)) !== null) {
552     // Each match's first capture group contains the matrix content
553     const matrix_content = match[1];
554
555     // Split rows by detecting "],[ " pattern
556     const rows = matrix_content.split(/\],\s*/);
557
558     // For each row, split by commas while ignoring any spaces
559     const parsed_rows = rows.map(row => row.split(',')).map(entry => {
560       const trimmed_entry = entry.trim();
561       // Convert numeric strings to numbers
562       return isNaN(trimmed_entry) ? trimmed_entry : Number(trimmed_entry);
563     });
564
565     // Push the parsed rows (matrix) into the matrices array
566     matrices.push(parsed_rows);
567   }
568
569   // If only one Matrix(...) was found, return the array directly instead of
570   // wrapping in another array
571   return matrices.length === 1 ? matrices[0] : matrices;
572 }
```

Listing 110 - sym-math.js

```

1 /**
2  * @file JSLAB library system submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 /**
9  * Class for JSLAB system submodule.
10 */
11 class PRDC_JSLAB_LIB_SYSTEM {
12
13 /**
14  * Initializes a new instance of the system submodule.
15  * @param {Object} js1 Reference to the main JSLAB object.

```

```

16     */
17     constructor(jsl) {
18         var obj = this;
19         this.jsl = jsl;
20     }
21
22 /**
23 * Executes a system command and returns the output.
24 * @param {...*} args Command arguments.
25 * @returns {string|boolean} The output of the command as a string, or false
26 if an error occurred.
27 */
28 system(...args) {
29     try {
30         return this.jsl.env.execSync(...args).toString();
31     } catch(err) {
32         this.jsl.env.error('@system: '+err);
33         return false;
34     }
35
36 /**
37 * Executes a system command.
38 * @param {...*} args Command arguments.
39 */
40 exec(...args) {
41     try {
42         return this.jsl.env.exec(...args);
43     } catch(err) {
44         this.jsl.env.error('@exec: '+err);
45         return false;
46     }
47 }
48
49 /**
50 * Executes a system command.
51 * @param {...*} args Command arguments.
52 */
53 spawn(...args) {
54     try {
55         return this.jsl.env.spawn(...args);
56     } catch(err) {
57         this.jsl.env.error('@spawn: '+err);
58         return false;
59     }
60 }
61
62 /**
63 * Retrieves a list of all running tasks on the system.
64 * @returns {Array} An array containing the tasklist output or [false, []]
65 if an error occurred.
66 */
67 getTaskList() {
68     try {
69         return this.jsl.env.execSync('tasklist').toString();

```

```

69     } catch(err) {
70       this.jsl.env.error('@getTaskList: '+err);
71       return false;
72     }
73   }
74
75 /**
76 * Checks if a specific program is running and retrieves its process IDs.
77 * @param {string} program_name - The name of the program to check.
78 * @returns {Array} An array where the first element is a boolean indicating
79   if the program is running,
80   and the second element is an array of process IDs if the
81   program is running.
82 */
83 isProgramRunning(program_name) {
84   try {
85     // Execute the tasklist command and get the output
86     const output = this.jsl.env.execSync('tasklist').toString();
87
88     // Convert the output to lowercase for case-insensitive comparison
89     const output_lower = output.toLowerCase();
90     const program_nameLower = program_name.toLowerCase();
91
92     // Split the output into lines and filter the ones containing the
93     // program name
94     const lines = output_lower.split('\n');
95     const matching_lines = lines.filter(function(line) {
96       return line.includes(program_nameLower);
97     });
98
99     // Extract PIDs from the matching lines
100    const pids = matching_lines.map(line => {
101      // Split the line by spaces and filter out empty elements
102      const columns = line.trim().split(/\s+/);
103      return parseInt(columns[1], 10); // PID is the second column
104    }).filter(function(pid) {
105      return !isNaN(pid);
106    });
107    return [is_running, pids];
108  } catch(err) {
109    this.jsl.env.error('@isProgramRunning: '+err);
110    return [false, []];
111  }
112}
113 /**
114 * Attempts to kill a process by its process ID.
115 * @param {number} pid - The process ID of the process to kill.
116 * @returns {string|boolean} The output of the kill command as a string, or
117   false if an error occurred.
118 */
119 killProcess(pid) {
120   try {

```

```

120      // Execute the tasklist command and get the output
121      return this.jsl.env.execSync('taskkill /pid '+pid+' /T /F').toString();
122    } catch(err) {
123      this.jsl.error('@killProcess: '+err);
124      return false;
125    }
126  }
127 }
128
129 exports.PRDC_JSLAB_LIB_SYSTEM = PRDC_JSLAB_LIB_SYSTEM;

```

Listing 111 - system.js

```

1 /**
2  * @file JSLAB library time submodule
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6 */
7
8 /**
9  * Class for JSLAB time submodule.
10 */
11 class PRDC_JSLAB_LIB_TIME {
12
13 /**
14  * Initializes the time submodule, setting up properties for time tracking
15  * and exposing time measurement utilities.
16  * @param {Object} jsl Reference to the main JSLAB object.
17 */
18 constructor(jsl) {
19   var obj = this;
20   this.jsl = jsl;
21
22 /**
23  * Current timezone string.
24  * @type {string}
25  */
26   this.timezone = 'Europe/Belgrade';
27
28 /**
29  * Last tic timestamp.
30  * @type {number}
31  */
32   this.last_tic = 0;
33
34 /**
35  * Starts a timer for measuring elapsed time. To be used with 'toc' to
36  * measure time intervals.
37  * @name tic
38  * @kind function
39  * @memberof PRDC_JSLAB_LIB_TIME
40  */
41 Object.defineProperty(this.jsl.context, 'tic', {configurable: true, get:
42   this._tic});
43 }

```

```

41
42  /**
43   * Records the current wall-clock time to measure elapsed time.
44   * @returns {number} The current wall-clock time in milliseconds.
45   */
46 _tic() {
47   this.jsl.context.last_tic = performance.now() / 1000; // [s]
48   return this.jsl.context.last_tic;
49 }
50
51 /**
52  * Calculates the wall-clock time elapsed since a specified start time or
53  * the last call to 'tic()', in seconds.
54  * @param {number} [tic] - The start time in milliseconds from which to
55  * calculate elapsed time. If omitted, uses the last time recorded by 'tic()
56  * ()'.
57  * @returns {number} The elapsed time in seconds.
58  */
59 toc(tic) {
60   var dt = performance.now() / 1000;
61   if(arguments.length < 1) {
62     return dt - this.jsl.context.last_tic;
63   } else {
64     return dt - tic;
65   }
66 }
67 /**
68  * Calculates the wall-clock time elapsed since a specified start time or
69  * the last call to 'tic()', in seconds.
70  * @param {number} [tic] - The start time in milliseconds from which to
71  * calculate elapsed time. If omitted, uses the last time recorded by 'tic()
72  * ()'.
73  * @returns {number} The elapsed time in milliseconds.
74  */
75 tocms(tic) {
76   if(!tic) {
77     tic = global.last_tic;
78   }
79   return (performance.now() - tic * 1000);
80 }
81 /**
82  * Gets the current Unix timestamp adjusted for a specified timezone.
83  * @returns {number} The current Unix timestamp as an integer.
84  */
85 getTimestamp() {
86   return luxon.DateTime.now().setZone(this.timezone).toMillis();
87 }
88 /**
89  * Gets the current time as a string adjusted for a specified timezone.
90  * @returns {string} The current time in 'HH:mm:ss' format.
91  */
92 getTime() {

```

```
90     return luxon.DateTime.now().setZone(this.timezone).toFormat('HH:mm:ss'));
91 }
92
93 /**
94 * Gets the current time with milliseconds as a string adjusted for a
95 * specified timezone.
96 * @returns {string} The current time in 'HH:mm:ss.SSS' format.
97 */
98 getFullTime() {
99     return luxon.DateTime.now().setZone(this.timezone).toFormat('HH:mm:ss.SSS');
100 }
101
102 /**
103 * Gets the current date as a string adjusted for a specified timezone.
104 * @returns {string} The current date in 'dd.MM.yyyy.' format.
105 */
106 getDate() {
107     return luxon.DateTime.now().setZone(this.timezone).toFormat('dd.MM.yyyy.');
108 }
109
110 /**
111 * Gets the current date and time as a string adjusted for a specified
112 * timezone.
113 * @returns {string} The current date and time in 'dd.MM.yyyy. HH:mm:ss'
114 */
115 getDateTime() {
116     return luxon.DateTime.now().setZone(this.timezone).toFormat('dd.MM.yyyy.
117         HH:mm:ss');
118 }
119
120 /**
121 * Gets the current date and time with milliseconds as a string adjusted for
122 * a specified timezone.
123 * @returns {string} The current date and time in 'dd.MM.yyyy. HH:mm:ss.SSS'
124 */
125 getDateTimeFull() {
126     return luxon.DateTime.now().setZone(this.timezone).toFormat('dd.MM.yyyy.
127         HH:mm:ss.SSS');
128 }
129
130 /**
131 * Gets the current date and time as a string suitable for filenames,
132 * adjusted for a specified timezone.
133 * @returns {string} The current date and time in 'ddMMyyyy_HHmmss' format
134 * for use in filenames.
135 */
136 getDateTimeStr() {
137     return luxon.DateTime.now().setZone(this.timezone).toFormat(
138         'ddMMyyyy_HHmmss');
139 }
140
141 /**
142 * Gets the current date and time as a string suitable for filenames,
143 * adjusted for a specified timezone.
144 * @returns {string} The current date and time in 'ddMMyyyy_HHmmss' format
145 * for use in filenames.
146 */
147 getDateTimeStrFull() {
148     return luxon.DateTime.now().setZone(this.timezone).toFormat(
149         'ddMMyyyy_HHmmss.SSS');
150 }
```

```

133  /**
134   * Sets the timezone to be used for time calculations and formatting. This
135   * method allows the application to adjust displayed times according to a
136   * specific timezone.
137   * @param {String} tz The timezone identifier (e.g., "America/New_York", "Europe/Paris") to be set for all time-related operations.
138   */
139  setTimezone(tz) {
140      this.timezone = tz;
141  }
142 exports.PRDC_JSLAB_LIB_TIME = PRDC_JSLAB_LIB_TIME;

```

Listing 112 - time.js

```

1  /**
2   * @file JSLAB library vector math submodule
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   */
7
8  /**
9   * Class for JSLAB vector math submodule.
10  */
11 class PRDC_JSLAB_VECTOR_MATH {
12
13  /**
14   * Constructs vector math submodule object with access to JSLAB's vector
15   * functions.
16   * @constructor
17   * @param {Object} js1 - Reference to the main JSLAB object.
18   */
19  constructor(js1) {
20      this.js1 = js1;
21  }
22
23  /**
24   * Creates a new vector with specified x, y, z components.
25   * @param {number} x - The x-component of the vector.
26   * @param {number} y - The y-component of the vector.
27   * @param {number} z - The z-component of the vector.
28   * @returns {PRDC_JSLAB_VECTOR} A new vector instance.
29   */
30  new(x, y, z) {
31      return new PRDC_JSLAB_VECTOR(this.js1, x, y, z);
32  }
33
34  /**
35   * Creates a vector from polar coordinates.
36   * @param {number} length - The length of the vector.
37   * @param {number} radian - The angle in radians.
38   * @returns {PRDC_JSLAB_VECTOR} The resulting vector.
39   */
40  polar(length, radian) {

```

```

40     const x = length * Math.cos(radian);
41     const y = length * Math.sin(radian);
42     const z = 0;
43     return new PRDC_JSLAB_VECTOR(this.jsl, x, y, z);
44   }
45
46   /**
47    * Creates a vector from spherical coordinates.
48    * @param {number} length - The length (magnitude) of the vector.
49    * @param {number} azimuth - The azimuth angle in radians (angle from the X-
50      axis in the XY plane).
51    * @param {number} elevation - The elevation angle in radians (angle from
52      the XY plane towards Z).
53    * @returns {PRDC_JSLAB_VECTOR} The resulting vector.
54    */
55   spherical(length, azimuth, elevation) {
56     const x = length * Math.cos(elevation) * Math.cos(azimuth);
57     const y = length * Math.cos(elevation) * Math.sin(azimuth);
58     const z = length * Math.sin(elevation);
59     return new PRDC_JSLAB_VECTOR(this.jsl, x, y, z);
60   }
61
62   exports.PRDC_JSLAB_VECTOR_MATH = PRDC_JSLAB_VECTOR_MATH;
63
64   /**
65    * Class for JSLAB vector.
66   */
67   class PRDC_JSLAB_VECTOR {
68
69     #jsl;
70
71     /**
72      * Creates a new vector instance.
73      * @constructor
74      * @param {Object} jsl - Reference to the main JSLAB object.
75      * @param {number|Object} x - The x-component or an object with x, y, z
76        properties.
77      * @param {number} [y=0] - The y-component of the vector.
78      * @param {number} [z=0] - The z-component of the vector.
79      */
80     constructor(jsl, x, y, z) {
81       this.#jsl = jsl;
82       this._set(x, y, z);
83     }
84
85     /**
86      * Sets the components of the vector.
87      * @param {number|Object} x - The x-component or an object with x, y, z
88        properties.
89      * @param {number} [y=0] - The y-component of the vector.
90      * @param {number} [z=0] - The z-component of the vector.
91      * @returns {PRDC_JSLAB_VECTOR} The updated vector instance.
92      */
93     _set(x, y, z) {

```

```

91      if (Array.isArray(x)) {
92          z = x[2];
93          y = x[1];
94          x = x[0];
95      } else if (isObject(x)) {
96          z = x.z;
97          y = x.y;
98          x = x.x;
99      }
100
101      this.x = x || 0;
102      this.y = y || 0;
103      this.z = z || 0;
104  }
105
106  /**
107   * Calculates the length (magnitude) of the vector.
108   * @returns {number} The length of the vector.
109   */
110  length() {
111      return Math.sqrt(this.x * this.x + this.y * this.y + this.z * this.z);
112  }
113
114  /**
115   * Calculates the length (magnitude) of the vector.
116   * @returns {number} The length of the vector.
117   */
118  norm() {
119      return this.length();
120  }
121
122  /**
123   * Adds two vectors and returns the result.
124   * @param {PRDC_JSLAB_VECTOR} v - The second vector.
125   * @returns {PRDC_JSLAB_VECTOR} The resulting vector after addition.
126   */
127  add(v) {
128      return this.#jsl.vec.new(this.x + v.x, this.y + v.y, this.z + v.z);
129  }
130
131  /**
132   * Adds two vectors and returns the result.
133   * @param {PRDC_JSLAB_VECTOR} v - The second vector.
134   * @returns {PRDC_JSLAB_VECTOR} The resulting vector after addition.
135   */
136  plus(v) {
137      return this.add(v);
138  }
139
140  /**
141   * Subtracts the second vector from the first and returns the result.
142   * @param {PRDC_JSLAB_VECTOR} v - The vector to subtract.
143   * @returns {PRDC_JSLAB_VECTOR} The resulting vector after subtraction.
144   */
145  subtract(v) {

```

```

146     return this.#jsl.vec.new(this.x - v.x, this.y - v.y, this.z - v.z);
147 }
148
149 /**
150 * Subtracts the second vector from the first and returns the result.
151 * @param {PRDC_JSLAB_VECTOR} v - The vector to subtract.
152 * @returns {PRDC_JSLAB_VECTOR} The resulting vector after subtraction.
153 */
154 minus(v) {
155   return this.subtract(v);
156 }
157
158 /**
159 * Scales a vector by the given factors.
160 * @param {number|Object} scale_x - The scale factor for the x-component or
161 * an object with x, y, z properties.
162 * @param {number} [scale_y] - The scale factor for the y-component.
163 * @param {number} [scale_z] - The scale factor for the z-component.
164 * @returns {PRDC_JSLAB_VECTOR} The scaled vector.
165 */
166 scale(scale_x, scale_y, scale_z) {
167   if(isObject(scale_x)) {
168     scale_x = scale_x.x;
169     scale_y = scale_x.y;
170     scale_z = scale_x.z;
171   } else if(!isNumber(scale_y)) {
172     scale_y = scale_x;
173     scale_z = scale_x;
174   }
175   return this.#jsl.vec.new(this.x * scale_x, this.y * scale_y, this.z *
176     scale_z);
177 }
178
179 /**
180 * Scales a vector by the given factor.
181 * @param {number} s - The scale factor.
182 * @returns {PRDC_JSLAB_VECTOR} The scaled vector.
183 */
184 multiply(s) {
185   return this.scale(s);
186 }
187
188 /**
189 * Scales a vector by dividing each element by given factor.
190 * @param {number} s - The scale factor.
191 * @returns {PRDC_JSLAB_VECTOR} The scaled vector.
192 */
193 divide(s) {
194   return this.scale(1 / s);
195 }
196
197 /**
198 * Checks if two vectors are equal.
199 * @param {PRDC_JSLAB_VECTOR} v - The second vector.
200 * @returns {boolean} True if vectors are equal, false otherwise.

```

```
199  */
200 equals(v) {
201     return this.x == v.x && this.y == v.y && this.z == v.z;
202 }
203
204 /**
205 * Calculates the angle of a vector.
206 * @param {PRDC_JSLAB_VECTOR} v - The vector.
207 * @returns {number} The angle in degrees.
208 */
209 angleTo(v) {
210     let cos_theta = this.dot(v) / (this.norm() * v.norm());
211     return Math.acos(cos_theta) * (180 / Math.PI);
212 }
213
214 /**
215 * Projects vector to given vector.
216 * @param {PRDC_JSLAB_VECTOR} v - The vector.
217 * @returns {PRDC_JSLAB_VECTOR} The projected vector.
218 */
219 projectTo(v) {
220     return v.multiply(this.dot(v) / Math.pow(v.norm(), 2));
221 }
222
223 /**
224 * Calculates the angles of a vector.
225 * @param {PRDC_JSLAB_VECTOR} v - The vector.
226 * @returns {Array} The angles azimuth and elevation in degrees.
227 */
228 angles() {
229     const length_xy = Math.sqrt(this.x * this.x + this.y * this.y);
230     const azimuth = Math.atan2(this.y, this.x) * (180 / Math.PI);
231     const elevation = Math.atan2(this.z, length_xy) * (180 / Math.PI);
232     return [azimuth, elevation];
233 }
234
235 /**
236 * Calculates the distance between two vectors.
237 * @param {PRDC_JSLAB_VECTOR} v - The second vector.
238 * @returns {number} The distance between the two vectors.
239 */
240 distance(v) {
241     var a = this.x - v.x;
242     var b = this.y - v.y;
243     var c = this.z - v.z;
244     return Math.sqrt(a * a + b * b + c * c);
245 }
246
247 /**
248 * Calculates the dot product of two vectors.
249 * @param {PRDC_JSLAB_VECTOR} v - The second vector.
250 * @returns {number} The dot product.
251 */
252 dot(v) {
253     return this.x * v.x + this.y * v.y + this.z * v.z;
```

```

254     }
255
256     /**
257      * Calculates the cross product of two vectors.
258      * @param {PRDC_JSLAB_VECTOR} v - The second vector.
259      * @returns {PRDC_JSLAB_VECTOR} The cross product vector.
260      */
261     cross(v) {
262       var cross_x = this.y * v.z - this.z * v.y;
263       var cross_y = this.z * v.x - this.x * v.z;
264       var cross_z = this.x * v.y - this.y * v.x;
265       return this.#jsl.vec.new(cross_x, cross_y, cross_z);
266     }
267
268     /**
269      * Interpolates between two vectors by a factor.
270      * @param {PRDC_JSLAB_VECTOR} v - The ending vector.
271      * @param {number} f - The interpolation factor.
272      * @returns {PRDC_JSLAB_VECTOR} The interpolated vector.
273      */
274     interpolate(v, f) {
275       var dx = v.x - this.x;
276       var dy = v.y - this.y;
277       var dz = v.z - this.z;
278       return this.#jsl.vec.new(this.x + dx * f, this.y + dy * f, this.z + dz * f
279         );
280     }
281     /**
282      * Offsets the vector by the given amounts.
283      * @param {number|Object} x - The amount to offset in the x-direction or an
284      * object with x, y, z properties.
285      * @param {number} [y=0] - The amount to offset in the y-direction.
286      * @param {number} [z=0] - The amount to offset in the z-direction.
287      * @returns {PRDC_JSLAB_VECTOR} The updated vector instance.
288      */
289     offset(x, y, z) {
290       if(isObject(x)) {
291         x = x.x;
292         z = x.z;
293         y = x.y;
294       }
295       x = this.x + x || 0;
296       y = this.y + y || 0;
297       z = this.z + z || 0;
298
299       return this.#jsl.vec.new(x, y, z);
300     }
301
302     /**
303      * Normalizes the vector to have a length of 1.
304      * @returns {PRDC_JSLAB_VECTOR} The normalized vector.
305      */
306     normalize() {

```



```
307     var length = this.length();
308
309     if(length > 0) {
310         var x = this.x / length;
311         var y = this.y / length;
312         var z = this.z / length;
313         return this.#jsl.vec.new(x, y, z);
314     }
315     return this.clone();
316 }
317
318 /**
319 * Negates the vector components.
320 * @returns {PRDC_JSLAB_VECTOR} The negated vector.
321 */
322 negate() {
323     var x = this.x * -1;
324     var y = this.y * -1;
325     var z = this.z * -1;
326
327     return this.#jsl.vec.new(x, y, z);
328 }
329
330 /**
331 * Creates a clone of this vector.
332 * @returns {PRDC_JSLAB_VECTOR} A new vector instance with the same
333 * components.
334 */
335 clone() {
336     return this.#jsl.vec.new(this.x, this.y, this.z);
337 }
338
339 /**
340 * Converts the vector to an array.
341 * @returns {number[]} An array containing the x, y, z components of the
342 * vector.
343 */
344 toArray() {
345     return [this.x, this.y, this.z];
346 }
347
348 /**
349 * Converts the vector to a matrix.
350 * @returns {Object} A matrix representation of the vector.
351 */
352 toMatrix() {
353     return this.#jsl.mat.new([this.x, this.y, this.z], 3, 1);
354 }
355
356 /**
357 * Converts the vector to a column matrix.
358 * @returns {Object} A column matrix representation of the vector.
359 */
360 toColMatrix() {
361     return this.#jsl.mat.new([this.x, this.y, this.z], 1, 3);
```

```

360     }
361
362     /**
363      * Converts the vector to a row matrix.
364      * @returns {Object} A row matrix representation of the vector.
365      */
366     toRowMatrix() {
367       return this.toMatrix();
368     }
369
370     /**
371      * Returns a string representation of the vector.
372      * @returns {string} The string representation in the format 'Vector(x:, y:, z:)'.
373      */
374     toString() {
375       return 'Vector([' + this.x + ', ' + this.y + ', ' + this.z + '])';
376     }
377
378     /**
379      * Returns a string representation of the vector.
380      * @returns {string} The string representation in the format 'Vector(x:, y:, z:)'.
381      */
382     toJSON() {
383       return { x: this.x, y: this.y, z: this.z };
384     }
385
386     /**
387      * Returns a string representation of the vector.
388      * @returns {string} The string representation in the format 'Vector(x:, y:, z:)'.
389      */
390     toSafeJSON() {
391       return this.toJSON();
392     }
393
394     /**
395      * Converts the object to a pretty string representation.
396      * @returns {string} The pretty string representation of the object.
397      */
398     toPrettyString() {
399       return this.toString();
400     }
401   }

```

Listing 113 - vector-math.js

```

1  /**
2   * @file JSLAB library windows submodule
3   * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4   * PR-DC, Republic of Serbia
5   * info@pr-dc.com
6   */
7
8  /**

```

```

9  * Class for JSLAB windows submodule.
10 */
11 class PRDC_JSLAB_LIB_WINDOWS {
12
13 /**
14  * Initializes a new instance of the windows submodule.
15  * @param {Object} js1 Reference to the main JSLAB object.
16  */
17 constructor(js1) {
18   var obj = this;
19   this.js1 = js1;
20   this._wid = 0;
21
22 /**
23  * Current active window ID.
24  * @type {Number}
25  */
26 this.active_window;
27
28 /**
29  * Array of open windows.
30  * @type {Array}
31  */
32 this.open_windows = {};
33 }
34
35 /**
36  * Opens a new window with the specified file.
37  * @param {string} file - The path to the HTML file to open in the new
38  * window.
39  * @returns {number} The identifier (wid) of the newly opened window.
40  */
41 openWindow(file) {
42   if(!this.js1.env.pathIsAbsolute(file)) {
43     file = app_path + '/html/' + file;
44   }
45
46   if(!this.js1.env.checkFile(file)) {
47     this.js1.env.error('@openWindow: '+language.string(199));
48   }
49
50   this._wid += 1;
51   var wid = this._wid;
52   this.open_windows[wid] = new PRDC_JSLAB_WINDOW(this.js1, wid);
53   this.open_windows[wid].open(file);
54   this._setActiveWindow(wid);
55   this.js1.no_ans = true;
56   this.js1.ignore_output = true;
57   return wid;
58 }
59
60 /**
61  * Opens the developer tools for a specified window by ID if it exists.
62  * @param {string} wid - The window ID.
63  * @returns {boolean} True if the developer tools were opened; otherwise,

```

```
        false.  
63     */  
64     openWindowDevTools(wid) {  
65         if(this.open_windows.hasOwnProperty(wid)) {  
66             return this.open_windows[wid].openDevTools();  
67         } else {  
68             return false;  
69         }  
70     }  
71  
72     /**  
73      * Closes the specified window.  
74      * @param {number} wid - Identifier for the window to close.  
75      */  
76     closeWindows(wid) {  
77         this.jsl.env.closeWindow(wid);  
78         this.jsl.no_ans = true;  
79         this.jsl.ignore_output = true;  
80     }  
81  
82     /**  
83      * Closes the specified window.  
84      * @param {number} wid - Identifier for the window to close.  
85      */  
86     closeWindow(wid) {  
87         if(this.open_windows.hasOwnProperty(wid)) {  
88             return this.open_windows[wid].close();  
89         } else {  
90             return false;  
91         }  
92     }  
93  
94     /**  
95      * Retrieves the window object with the specified ID.  
96      * @param {number} wid - The ID of the window.  
97      * @returns {Object|boolean} - The window object if found, otherwise false.  
98      */  
99     getWindow(wid) {  
100         if(this.open_windows.hasOwnProperty(wid)) {  
101             return this.open_windows[wid];  
102         } else {  
103             return false;  
104         }  
105     }  
106  
107     /**  
108      * Retrieves current active window object.  
109      * @returns {Object|boolean} - The window object if found, otherwise false.  
110      */  
111     getCurrentWindow() {  
112         if(this.open_windows.hasOwnProperty(this.active_window)) {  
113             return this.open_windows[this.active_window];  
114         } else {  
115             return false;  
116         }  
117     }
```

```
117     }
118
119     /**
120      * Retrieves current active window object .
121      * @returns { Object | boolean } - The window object if found , otherwise false .
122      */
123     gcw() {
124       return this.getCurrentWindow();
125     }
126
127     /**
128      * Shows the specified window .
129      * @param { number } wid - The ID of the window to show .
130      * @returns { boolean | undefined } - Returns false if the window ID is invalid ,
131      *          otherwise the result of the show() method .
132      */
133     showWindow(wid) {
134       if (this.open_windows.hasOwnProperty(wid)) {
135         return this.open_windows[wid].show();
136       } else {
137         return false;
138       }
139     }
140     /**
141      * Hides the specified window .
142      * @param { number } wid - The ID of the window to hide .
143      * @returns { boolean | undefined } - Returns false if the window ID is invalid ,
144      *          otherwise the result of the hide() method .
145      */
146     hideWindow(wid) {
147       if (this.open_windows.hasOwnProperty(wid)) {
148         return this.open_windows[wid].hide();
149       } else {
150         return false;
151       }
152     }
153     /**
154      * Brings the specified window to the foreground .
155      * @param { number } wid - The ID of the window to focus .
156      * @returns { boolean | undefined } - Returns false if the window ID is invalid .
157      */
158     focusWindow(wid) {
159       if (this.open_windows.hasOwnProperty(wid)) {
160         return this.open_windows[wid].focus();
161       } else {
162         return false;
163       }
164     }
165
166     /**
167      * Minimizes the specified window .
168      * @param { number } wid - The ID of the window to minimize .
169      * @returns { boolean | undefined } - Returns false if the window ID is invalid ,
```

```
otherwise the result of the minimize() method.  
170  */  
171  minimizeWindow(wid) {  
172      if(this.open_windows.hasOwnProperty(wid)) {  
173          return this.open_windows[wid].minimize();  
174      } else {  
175          return false;  
176      }  
177  }  
178  
179  /**  
180   * Centers the specified window on the screen.  
181   * @param {number} wid - The ID of the window to center.  
182   * @returns {boolean|undefined} - Returns false if the window ID is invalid,  
183   * otherwise the result of the center() method.  
184   */  
185  centerWindow(wid) {  
186      if(this.open_windows.hasOwnProperty(wid)) {  
187          return this.open_windows[wid].center();  
188      } else {  
189          return false;  
190      }  
191  }  
192  
193  /**  
194   * Moves the specified window to the top of the window stack.  
195   * @param {number} wid - The ID of the window to move to the top.  
196   * @returns {boolean|undefined} - Returns false if the window ID is invalid,  
197   * otherwise the result of the moveTop() method.  
198   */  
199  moveTopWindow(wid) {  
200      if(this.open_windows.hasOwnProperty(wid)) {  
201          return this.open_windows[wid].moveTop();  
202      } else {  
203          return false;  
204      }  
205  }  
206  
207  /**  
208   * Sets the size of the specified window.  
209   * @param {number} wid - The ID of the window.  
210   * @param {number} width - The new width of the window.  
211   * @param {number} height - The new height of the window.  
212   * @returns {boolean|undefined} - Returns false if the window ID is invalid.  
213   */  
214  setWindowSize(wid, width, height) {  
215      if(this.open_windows.hasOwnProperty(wid)) {  
216          return this.open_windows[wid].setSize(width, height);  
217      } else {  
218          return false;  
219      }  
220  }  
221  
222  /**  
223   * Sets the position of the specified window.
```

```
222     * @param {number} wid - The ID of the window.
223     * @param {number} left - The new left position of the window.
224     * @param {number} top - The new top position of the window.
225     * @returns {boolean|undefined} - Returns false if the window ID is invalid.
226     */
227 setWindowPos(wid, left, top) {
228     if(this.open_windows.hasOwnProperty(wid)) {
229         return this.open_windows[wid].setPos(left, top);
230     } else {
231         return false;
232     }
233 }
234
235 /**
236     * Sets the resizable state of the specified window.
237     * @param {number} wid - The ID of the window.
238     * @param {boolean} state - Whether the window should be resizable.
239     * @returns {boolean|undefined} - Returns false if the window ID is invalid,
240     * otherwise the result of the setResizable() method.
241     */
242 setWindowResizable(wid, state) {
243     if(this.open_windows.hasOwnProperty(wid)) {
244         return this.open_windows[wid].setResizable(state);
245     } else {
246         return false;
247     }
248
249 /**
250     * Sets the movable state of the specified window.
251     * @param {number} wid - The ID of the window.
252     * @param {boolean} state - Whether the window should be movable.
253     * @returns {boolean|undefined} - Returns false if the window ID is invalid,
254     * otherwise the result of the setMovable() method.
255     */
256 setWindowMovable(wid, state) {
257     if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
258         return this.open_windows[wid].setMovable(state);
259     } else {
260         return false;
261     }
262
263 /**
264     * Sets the aspect ratio of the specified window.
265     * @param {number} wid - The ID of the window.
266     * @param {number} aspect_ratio - The desired aspect ratio of the window.
267     * @returns {boolean|undefined} - Returns false if the window ID is invalid,
268     * otherwise the result of the setAspectRatio() method.
269     */
270 setWindowAspectRatio(wid, aspect_ratio) {
271     if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
272         return this.open_windows[wid].setAspectRatio(aspect_ratio);
273     } else {
274         return false;
275     }
276 }
```

```

274     }
275 }
276
277 /**
278 * Sets the opacity of the specified window.
279 * @param {number} wid - The ID of the window.
280 * @param {number} opacity - The desired opacity level (0 to 1).
281 * @returns {boolean|undefined} - Returns false if the window ID is invalid,
282 * otherwise the result of the setOpacity() method.
283 */
284 setWindowOpacity(wid, opacity) {
285     if(this.jsl.windows.open_windows.hasOwnProperty(wid)) {
286         return this.open_windows[wid].setOpacity(opacity);
287     } else {
288         return false;
289     }
290 }
291 /**
292 * Sets the position of the specified window.
293 * @param {number} wid - The ID of the window.
294 * @param {number} left - The new left position of the window.
295 * @param {number} top - The new top position of the window.
296 * @returns {boolean|undefined} - Returns false if the window ID is invalid.
297 */
298 setTitle(wid, title) {
299     if(this.open_windows.hasOwnProperty(wid)) {
300         return this.open_windows[wid].setTitle(title);
301     } else {
302         return false;
303     }
304 }
305 /**
306 * Retrieves the size of the specified window.
307 * @param {number} wid - The ID of the window.
308 * @returns {Array|boolean} - An array [width, height] if the window exists,
309 * otherwise false.
310 */
311 getSize(wid) {
312     if(this.open_windows.hasOwnProperty(wid)) {
313         return this.open_windows[wid].getSize();
314     } else {
315         return false;
316     }
317 }
318 /**
319 * Retrieves the position of the specified window.
320 * @param {number} wid - The ID of the window.
321 * @returns {Array|boolean} - An array [left, top] if the window exists,
322 * otherwise false.
323 */
324 getPos(wid) {
325     if(this.open_windows.hasOwnProperty(wid)) {

```

```

326     return this.open_windows[wid].getPos();
327 } else {
328     return false;
329 }
330 }
331 /**
332 * Opens window with documentation
333 */
334 async openDocumentation(query) {
335     var wid = this.openWindow('../docs/documentation.html');
336     var win = this.open_windows[wid];
337     await win.ready;
338     if(query) {
339         win.context.location.href = '../docs/documentation.html#' + encodeURI(
340             query);
341         win.context.location.reload(true);
342     }
343     win.setTitle('JSLAB / DOCUMENTATION');
344 }
345 /**
346 * Opens window with documentation
347 */
348 async openDoc(query) {
349     await this.openDocumentation(query);
350 }
351 /**
352 * Opens a new 3D window and imports specified modules.
353 * @param {Array<Object>} [imports=[]] - An array of import objects
354 * specifying modules to import.
355 * @returns {Promise<Object>} A promise that resolves to the window object
356 * once imports are ready.
357 */
358 async openWindow3D(imports = []) {
359     var wid = this.openWindow('three.html');
360     await this.open_windows[wid].ready;
361     var context = this.open_windows[wid].context;
362     var script = context.document.createElement('script');
363     script.type = 'module';
364
365     const import_statements = [];
366     const window_assignments = [];
367     var imports_array = [{import: '*', as: 'THREE', from: 'three'}];
368     imports_array.push(...imports);
369
370     imports_array.forEach((item) => {
371         const { import: imported, as, from } = item;
372
373         if(imported === '*') {
374             if(!as) {
375                 this.jsl.env.error('@openWindow3D: '+language.string(200)+from+'.');
376             }
377             // Namespace import

```

```

378     import_statements.push(`import * as ${as} from '${from}';`);
379     window_assignments.push(`window.${as} = ${as};`);
380   } else if(Array.isArray(imported)) {
381     // Named imports with multiple specifiers
382     const specifiers = imported.join(', ');
383     import_statements.push(`import { ${specifiers} } from '${from}';`);
384     imported.forEach((spec) => {
385       window_assignments.push(`window.${spec} = ${spec};`);
386     });
387   } else {
388     // Single named import
389     import_statements.push(`import { ${imported} } from '${from}';`);
390     window_assignments.push(`window.${imported} = ${imported};`);
391   }
392 });
393
394 script.textContent = `${import_statements.join('\n')}\n\n${window_assignments.join('\n')}\n` window.imports_ready = true;;
395
396 context.imports_ready = false;
397 context.document.body.appendChild(script);
398 while(!context.imports_ready) {
399   await this.jsl.non_blocking.waitMSeconds(1);
400 }
401 return context;
402 }

403 /**
404 * Opens a Plotly.js window and initializes the plot container.
405 * @returns {Promise<Window>} The window object where Plotly.js is loaded
406 * and the plot container is available.
407 */
408 async openPlotlyjs() {
409   var wid = this.openWindow('plotlyjs.html');
410   await this.open_windows[wid].ready;
411   var context = this.open_windows[wid].context;
412   context.imports_ready = false;
413   while(!context.imports_ready) {
414     if(typeof context.Plotly != 'undefined') {
415       context.imports_ready = true;
416     }
417     await this.jsl.non_blocking.waitMSeconds(1);
418   }
419   context.plot_cont = context.document.getElementById('plot-cont');
420   context.plot_cont.style = 'position: absolute; top:0; left:0; right:0; bottom
421   :0;';
422   return context;
423 }

424 /**
425 * Opens a window with canvas and D3 and initializes the canvas element.
426 * @returns {Promise<Window>} The window object where D3 is loaded and the
427 * canvas element is available.
428 */
429 async openCanvas() {

```

```

429     var wid = this.openWindow('d3.html');
430     await this.open_windows[wid].ready;
431     var context = this.open_windows[wid].context;
432     context.imports_ready = false;
433     while (!context.imports_ready) {
434       if(typeof context.d3 != 'undefined') {
435         context.imports_ready = true;
436       }
437       await this.jsl.non_blocking.waitMSeconds(1);
438     }
439     context.svg = context.document.getElementById('d3-svg');
440     context.canvas = context.document.getElementById('d3-canvas');
441     context.svg.style = 'position: absolute; top:0; left:0; right:0; bottom:0;';
442     context.canvas.style = 'position: absolute; top:0; left:0; right:0; bottom:0;';
443     return context;
444   }
445
446 /**
447 * Opens a new blank window.
448 * @returns {Promise<Object>} A promise that resolves to the window object
449 * once it is ready.
450 */
451 async openWindowBlank() {
452   var wid = this.openWindow('blank.html');
453   await this.open_windows[wid].ready;
454   return this.open_windows[wid].context;
455 }
456 /**
457 * Sets the specified window as the active window.
458 * @param {number} wid - The identifier of the window to set as active.
459 */
460 _ setActiveWindow(wid) {
461   if(this.open_windows.hasOwnProperty(wid)) {
462     this.active_window = wid;
463   } else {
464     this.active_window = -1;
465   }
466 }
467
468 /**
469 * Updates the language of the text elements within all open windows.
470 */
471 _ updateLanguage() {
472   Object.values(this.open_windows).forEach(function(win) {
473     win._updateLanguage(false);
474   });
475 }
476
477 /**
478 * Closes a window identified by the given ID and updates the active window
479 * if necessary.
480 * @param {number} wid - The identifier of the window to close.
481 */

```

```

481     _closedWindow(wid) {
482       if(this.open_windows.hasOwnProperty(wid)) {
483         var new_wid = -1;
484         if(this.active_window == wid) {
485           var wids = Object.keys(this.open_windows);
486           var N = wids.length;
487           if(N > 1) {
488             if(wids[N-1] !== wid) {
489               new_wid = wids[N-1];
490             } else {
491               new_wid = wids[N-2];
492             }
493           }
494           this._ setActiveWindow(new_wid);
495         }
496         this.open_windows[wid].onClosed();
497         delete this.open_windows[wid];
498       }
499     }
500   }
501
502 exports.PRDC_JSLAB_LIB_WINDOWS = PRDC_JSLAB_LIB_WINDOWS;
503
504 /**
505 * Class for JSLAB window.
506 */
507 class PRDC_JSLAB_WINDOW {
508
509   #jsl;
510
511   /**
512    * Initializes a new instance of the JSLAB window.
513    * @param {Object} jsl - Reference to the main JSLAB object.
514    * @param {number} wid - Identifier for the window.
515    */
516   constructor(jsl, wid) {
517     var obj = this;
518
519     this.#jsl = jsl;
520     this.wid = wid;
521
522     this.context;
523     this.dom;
524
525     this.opened = false;
526     this.ready = new Promise((resolve) => {
527       obj._readyResolve = resolve;
528     });
529
530     this.onClosed = function() {};
531   }
532
533   /**
534    * Opens the window with the specified file.
535    * @param {string} file - The path to the HTML file to open in the window.
536  
```

```

536   * @returns {Promise<void>} A promise that resolves when the window is
537   * opened and ready.
538   */
539   async open(file) {
540     if(!this.opened) {
541       this.opened = true;
542       var [context, ready] = this.#jsl.env.openWindow(this.wid, file);
543       this.context = context;
544       await ready;
545       this.dom = this.context.document;
546       this._onReady();
547     }
548
549   /**
550   * Shows the window.
551   * @returns {Promise<boolean|undefined>} - Resolves to 'true' if the window
552   * was shown successfully, or 'false' if the window ID is invalid.
553   */
554   async show() {
555     await this.#jsl.promiseOrStoped(this.ready);
556     return this.#jsl.env.showWindow(this.wid);
557   }
558
559   /**
560   * Hides the window.
561   * @returns {Promise<boolean|undefined>} - Resolves to 'true' if the window
562   * was hidden successfully, or 'false' if the window ID is invalid.
563   */
564   async hide() {
565     await this.#jsl.promiseOrStoped(this.ready);
566     return this.#jsl.env.hideWindow(this.wid);
567   }
568
569   /**
570   * Brings focus to the window.
571   * @returns {Promise} - Resolves when the window size is focused.
572   */
573   async focus() {
574     await this.#jsl.promiseOrStoped(this.ready);
575     return this.#jsl.env.focusWindow(this.wid);
576   }
577
578   /**
579   * Minimizes the window.
580   * @returns {Promise<boolean|undefined>} - Resolves to 'true' if the window
581   * was minimized successfully, or 'false' if the window ID is invalid.
582   */
583   async minimize() {
584     await this.#jsl.promiseOrStoped(this.ready);
585     return this.#jsl.env.minimizeWindow(this.wid);
586   }
587
588   /**
589   * Centers the window on the screen.

```

```
587 * @returns {Promise<boolean|undefined>} - Resolves to 'true' if the window
588   was centered successfully , or 'false' if the window ID is invalid .
589 */
590 async center() {
591   await this.#jsl.promiseOrStoped(this.ready);
592   return this.#jsl.env.centerWindow(this.wid);
593 }
594 /**
595 * Moves the window to the top of the window stack.
596 * @returns {Promise<boolean|undefined>} - Resolves to 'true' if the window
597   was moved to the top successfully , or 'false' if the window ID is
598   invalid .
599 */
600 async moveTop() {
601   await this.#jsl.promiseOrStoped(this.ready);
602   return this.#jsl.env.moveTopWindow(this.wid);
603 }
604 /**
605 * Sets the size of the current window.
606 * @param {number} width - The desired width of the window.
607 * @param {number} height - The desired height of the window.
608 * @returns {Promise} - Resolves when the window size is set .
609 */
610 async setSize(width, height) {
611   await this.#jsl.promiseOrStoped(this.ready);
612   return this.#jsl.env.setWindowSize(this.wid, width, height);
613 }
614 /**
615 * Sets the position of the current window.
616 * @param {number} left - The desired left position of the window.
617 * @param {number} top - The desired top position of the window.
618 * @returns {Promise} - Resolves when the window position is set .
619 */
620 async setPos(left, top) {
621   await this.#jsl.promiseOrStoped(this.ready);
622   return this.#jsl.env.setWindowPos(this.wid, left, top);
623 }
624 /**
625 * Sets the resizable state of the window.
626 * @param {boolean} state - Whether the window should be resizable ('true')
627   or not ('false') .
628 * @returns {Promise<boolean|undefined>} - Resolves to 'true' if the
629   resizable state was set successfully , or 'false' if the window ID is
630   invalid .
631 */
632 async setResizable(state) {
633   await this.#jsl.promiseOrStoped(this.ready);
634   return this.#jsl.env.setWindowResizable(this.wid, state);
635 }
```




```
685     * Retrieves the position of the current window.  
686     * @returns {Promise<Array>} - Resolves with an array [ left , top ].  
687     */  
688     async getPos() {  
689         await this.#jsl.promiseOrStoped(this.ready);  
690         return this.#jsl.env.getWindowPos(this.wid);  
691     }  
692  
693     /**
694      * Closes the current window.  
695      * @returns {Promise} - Resolves when the window is closed.  
696      */  
697     async close() {  
698         await this.#jsl.promiseOrStoped(this.ready);  
699         return this.#jsl.env.closeWindows(this.wid);  
700     }  
701  
702     /**
703      * Opens the developer tools for the current window asynchronously.  
704      * @returns {Promise<boolean>} A promise that resolves to true when dev  
705      * tools are opened.  
706      */  
707     async openDevTools() {  
708         await this.#jsl.promiseOrStoped(this.ready);  
709         return this.#jsl.env.openWindowDevTools(this.wid);  
710     }  
711  
712     /**
713      * Handles actions to perform when the window is ready.  
714      * @returns {void}  
715      */  
716     _onReady() {  
717         var style = this.dom.createElement('style');  
718         this.dom.head.appendChild(style);  
719         this.lang_styles = style.sheet;  
720         this._updateLanguage();  
721         this._readyResolve(true);  
722     }  
723  
724     /**
725      * Updates the language of the text elements within the DOM.  
726      * @param {boolean} [flag=true] - Whether to update the language.  
727      * @returns {void}  
728      */  
729     _updateLanguage(flag = true) {  
730         if(flag) {  
731             this.dom.querySelectorAll('str').forEach(function(el) {  
732                 var id = el.getAttribute('sid');  
733                 el.innerHTML = language.string(id);  
734             });  
735         }  
736  
737         if(this.lang_styles.cssRules.length > 1) {  
738             this.lang_styles.deleteRule(1);  
739         }  
740     }  
741  
742     /**
743      * Deletes the language styles from the DOM.  
744      * @returns {void}  
745      */  
746     _deleteLanguage() {  
747         this.lang_styles.cssRules.forEach(function(rule) {  
748             rule.deleteRule();  
749         });  
750     }  
751  
752     /**
753      * Sets the language for the current window.  
754      * @param {string} lang - The language code to set.  
755      * @returns {void}  
756      */  
757     _setLanguage(lang) {  
758         this.lang = lang;  
759         this._updateLanguage();  
760     }  
761  
762     /**
763      * Gets the current language code.  
764      * @returns {string} The current language code.  
765      */  
766     _getLanguage() {  
767         return this.lang;  
768     }  
769  
770     /**
771      * Checks if the current language is supported.  
772      * @param {string} lang - The language code to check.  
773      * @returns {boolean} Whether the language is supported.  
774      */  
775     _isSupported(lang) {  
776         return language.isSupported(lang);  
777     }  
778  
779     /**
780      * Updates the language of the text elements within the DOM.  
781      * @param {boolean} [flag=true] - Whether to update the language.  
782      * @returns {void}  
783      */  
784     _updateLanguage(flag = true) {  
785         if(flag) {  
786             this.dom.querySelectorAll('str').forEach(function(el) {  
787                 var id = el.getAttribute('sid');  
788                 el.innerHTML = language.string(id);  
789             });  
790         }  
791     }  
792  
793     /**
794      * Deletes the language styles from the DOM.  
795      * @returns {void}  
796      */  
797     _deleteLanguage() {  
798         this.lang_styles.cssRules.forEach(function(rule) {  
799             rule.deleteRule();  
800         });  
801     }  
802  
803     /**
804      * Sets the language for the current window.  
805      * @param {string} lang - The language code to set.  
806      * @returns {void}  
807      */  
808     _setLanguage(lang) {  
809         this.lang = lang;  
810         this._updateLanguage();  
811     }  
812  
813     /**
814      * Gets the current language code.  
815      * @returns {string} The current language code.  
816      */  
817     _getLanguage() {  
818         return this.lang;  
819     }  
820  
821     /**
822      * Checks if the current language is supported.  
823      * @param {string} lang - The language code to check.  
824      * @returns {boolean} Whether the language is supported.  
825      */  
826     _isSupported(lang) {  
827         return language.isSupported(lang);  
828     }  
829  
830     /**
831      * Deletes the language styles from the DOM.  
832      * @returns {void}  
833      */  
834     _deleteLanguage() {  
835         this.lang_styles.cssRules.forEach(function(rule) {  
836             rule.deleteRule();  
837         });  
838     }  
839  
840     /**
841      * Sets the language for the current window.  
842      * @param {string} lang - The language code to set.  
843      * @returns {void}  
844      */  
845     _setLanguage(lang) {  
846         this.lang = lang;  
847         this._updateLanguage();  
848     }  
849  
850     /**
851      * Gets the current language code.  
852      * @returns {string} The current language code.  
853      */  
854     _getLanguage() {  
855         return this.lang;  
856     }  
857  
858     /**
859      * Checks if the current language is supported.  
860      * @param {string} lang - The language code to check.  
861      * @returns {boolean} Whether the language is supported.  
862      */  
863     _isSupported(lang) {  
864         return language.isSupported(lang);  
865     }  
866  
867     /**
868      * Deletes the language styles from the DOM.  
869      * @returns {void}  
870      */  
871     _deleteLanguage() {  
872         this.lang_styles.cssRules.forEach(function(rule) {  
873             rule.deleteRule();  
874         });  
875     }  
876  
877     /**
878      * Sets the language for the current window.  
879      * @param {string} lang - The language code to set.  
880      * @returns {void}  
881      */  
882     _setLanguage(lang) {  
883         this.lang = lang;  
884         this._updateLanguage();  
885     }  
886  
887     /**
888      * Gets the current language code.  
889      * @returns {string} The current language code.  
890      */  
891     _getLanguage() {  
892         return this.lang;  
893     }  
894  
895     /**
896      * Checks if the current language is supported.  
897      * @param {string} lang - The language code to check.  
898      * @returns {boolean} Whether the language is supported.  
899      */  
900     _isSupported(lang) {  
901         return language.isSupported(lang);  
902     }  
903  
904     /**
905      * Deletes the language styles from the DOM.  
906      * @returns {void}  
907      */  
908     _deleteLanguage() {  
909         this.lang_styles.cssRules.forEach(function(rule) {  
910             rule.deleteRule();  
911         });  
912     }  
913  
914     /**
915      * Sets the language for the current window.  
916      * @param {string} lang - The language code to set.  
917      * @returns {void}  
918      */  
919     _setLanguage(lang) {  
920         this.lang = lang;  
921         this._updateLanguage();  
922     }  
923  
924     /**
925      * Gets the current language code.  
926      * @returns {string} The current language code.  
927      */  
928     _getLanguage() {  
929         return this.lang;  
930     }  
931  
932     /**
933      * Checks if the current language is supported.  
934      * @param {string} lang - The language code to check.  
935      * @returns {boolean} Whether the language is supported.  
936      */  
937     _isSupported(lang) {  
938         return language.isSupported(lang);  
939     }  
940  
941     /**
942      * Deletes the language styles from the DOM.  
943      * @returns {void}  
944      */  
945     _deleteLanguage() {  
946         this.lang_styles.cssRules.forEach(function(rule) {  
947             rule.deleteRule();  
948         });  
949     }  
950  
951     /**
952      * Sets the language for the current window.  
953      * @param {string} lang - The language code to set.  
954      * @returns {void}  
955      */  
956     _setLanguage(lang) {  
957         this.lang = lang;  
958         this._updateLanguage();  
959     }  
960  
961     /**
962      * Gets the current language code.  
963      * @returns {string} The current language code.  
964      */  
965     _getLanguage() {  
966         return this.lang;  
967     }  
968  
969     /**
970      * Checks if the current language is supported.  
971      * @param {string} lang - The language code to check.  
972      * @returns {boolean} Whether the language is supported.  
973      */  
974     _isSupported(lang) {  
975         return language.isSupported(lang);  
976     }  
977  
978     /**
979      * Deletes the language styles from the DOM.  
980      * @returns {void}  
981      */  
982     _deleteLanguage() {  
983         this.lang_styles.cssRules.forEach(function(rule) {  
984             rule.deleteRule();  
985         });  
986     }  
987  
988     /**
989      * Sets the language for the current window.  
990      * @param {string} lang - The language code to set.  
991      * @returns {void}  
992      */  
993     _setLanguage(lang) {  
994         this.lang = lang;  
995         this._updateLanguage();  
996     }  
997  
998     /**
999      * Gets the current language code.  
1000     * @returns {string} The current language code.  
1001    */  
1002    _getLanguage() {  
1003        return this.lang;  
1004    }  
1005  
1006    /**
1007      * Checks if the current language is supported.  
1008      * @param {string} lang - The language code to check.  
1009      * @returns {boolean} Whether the language is supported.  
1010      */  
1011     _isSupported(lang) {  
1012         return language.isSupported(lang);  
1013     }  
1014  
1015     /**
1016      * Deletes the language styles from the DOM.  
1017      * @returns {void}  
1018      */  
1019     _deleteLanguage() {  
1020         this.lang_styles.cssRules.forEach(function(rule) {  
1021             rule.deleteRule();  
1022         });  
1023     }  
1024  
1025     /**
1026      * Sets the language for the current window.  
1027      * @param {string} lang - The language code to set.  
1028      * @returns {void}  
1029      */  
1030     _setLanguage(lang) {  
1031         this.lang = lang;  
1032         this._updateLanguage();  
1033     }  
1034  
1035     /**
1036      * Gets the current language code.  
1037      * @returns {string} The current language code.  
1038      */  
1039     _getLanguage() {  
1040         return this.lang;  
1041     }  
1042  
1043     /**
1044      * Checks if the current language is supported.  
1045      * @param {string} lang - The language code to check.  
1046      * @returns {boolean} Whether the language is supported.  
1047      */  
1048     _isSupported(lang) {  
1049         return language.isSupported(lang);  
1050     }  
1051  
1052     /**
1053      * Deletes the language styles from the DOM.  
1054      * @returns {void}  
1055      */  
1056     _deleteLanguage() {  
1057         this.lang_styles.cssRules.forEach(function(rule) {  
1058             rule.deleteRule();  
1059         });  
1060     }  
1061  
1062     /**
1063      * Sets the language for the current window.  
1064      * @param {string} lang - The language code to set.  
1065      * @returns {void}  
1066      */  
1067     _setLanguage(lang) {  
1068         this.lang = lang;  
1069         this._updateLanguage();  
1070     }  
1071  
1072     /**
1073      * Gets the current language code.  
1074      * @returns {string} The current language code.  
1075      */  
1076     _getLanguage() {  
1077         return this.lang;  
1078     }  
1079  
1080     /**
1081      * Checks if the current language is supported.  
1082      * @param {string} lang - The language code to check.  
1083      * @returns {boolean} Whether the language is supported.  
1084      */  
1085     _isSupported(lang) {  
1086         return language.isSupported(lang);  
1087     }  
1088  
1089     /**
1090      * Deletes the language styles from the DOM.  
1091      * @returns {void}  
1092      */  
1093     _deleteLanguage() {  
1094         this.lang_styles.cssRules.forEach(function(rule) {  
1095             rule.deleteRule();  
1096         });  
1097     }  
1098  
1099     /**
1100      * Sets the language for the current window.  
1101      * @param {string} lang - The language code to set.  
1102      * @returns {void}  
1103      */  
1104     _setLanguage(lang) {  
1105         this.lang = lang;  
1106         this._updateLanguage();  
1107     }  
1108  
1109     /**
1110      * Gets the current language code.  
1111      * @returns {string} The current language code.  
1112      */  
1113     _getLanguage() {  
1114         return this.lang;  
1115     }  
1116  
1117     /**
1118      * Checks if the current language is supported.  
1119      * @param {string} lang - The language code to check.  
1120      * @returns {boolean} Whether the language is supported.  
1121      */  
1122     _isSupported(lang) {  
1123         return language.isSupported(lang);  
1124     }  
1125  
1126     /**
1127      * Deletes the language styles from the DOM.  
1128      * @returns {void}  
1129      */  
1130     _deleteLanguage() {  
1131         this.lang_styles.cssRules.forEach(function(rule) {  
1132             rule.deleteRule();  
1133         });  
1134     }  
1135  
1136     /**
1137      * Sets the language for the current window.  
1138      * @param {string} lang - The language code to set.  
1139      * @returns {void}  
1140      */  
1141     _setLanguage(lang) {  
1142         this.lang = lang;  
1143         this._updateLanguage();  
1144     }  
1145  
1146     /**
1147      * Gets the current language code.  
1148      * @returns {string} The current language code.  
1149      */  
1150     _getLanguage() {  
1151         return this.lang;  
1152     }  
1153  
1154     /**
1155      * Checks if the current language is supported.  
1156      * @param {string} lang - The language code to check.  
1157      * @returns {boolean} Whether the language is supported.  
1158      */  
1159     _isSupported(lang) {  
1160         return language.isSupported(lang);  
1161     }  
1162  
1163     /**
1164      * Deletes the language styles from the DOM.  
1165      * @returns {void}  
1166      */  
1167     _deleteLanguage() {  
1168         this.lang_styles.cssRules.forEach(function(rule) {  
1169             rule.deleteRule();  
1170         });  
1171     }  
1172  
1173     /**
1174      * Sets the language for the current window.  
1175      * @param {string} lang - The language code to set.  
1176      * @returns {void}  
1177      */  
1178     _setLanguage(lang) {  
1179         this.lang = lang;  
1180         this._updateLanguage();  
1181     }  
1182  
1183     /**
1184      * Gets the current language code.  
1185      * @returns {string} The current language code.  
1186      */  
1187     _getLanguage() {  
1188         return this.lang;  
1189     }  
1190  
1191     /**
1192      * Checks if the current language is supported.  
1193      * @param {string} lang - The language code to check.  
1194      * @returns {boolean} Whether the language is supported.  
1195      */  
1196     _isSupported(lang) {  
1197         return language.isSupported(lang);  
1198     }  
1199  
1200     /**
1201      * Deletes the language styles from the DOM.  
1202      * @returns {void}  
1203      */  
1204     _deleteLanguage() {  
1205         this.lang_styles.cssRules.forEach(function(rule) {  
1206             rule.deleteRule();  
1207         });  
1208     }  
1209  
1210     /**
1211      * Sets the language for the current window.  
1212      * @param {string} lang - The language code to set.  
1213      * @returns {void}  
1214      */  
1215     _setLanguage(lang) {  
1216         this.lang = lang;  
1217         this._updateLanguage();  
1218     }  
1219  
1220     /**
1221      * Gets the current language code.  
1222      * @returns {string} The current language code.  
1223      */  
1224     _getLanguage() {  
1225         return this.lang;  
1226     }  
1227  
1228     /**
1229      * Checks if the current language is supported.  
1230      * @param {string} lang - The language code to check.  
1231      * @returns {boolean} Whether the language is supported.  
1232      */  
1233     _isSupported(lang) {  
1234         return language.isSupported(lang);  
1235     }  
1236  
1237     /**
1238      * Deletes the language styles from the DOM.  
1239      * @returns {void}  
1240      */  
1241     _deleteLanguage() {  
1242         this.lang_styles.cssRules.forEach(function(rule) {  
1243             rule.deleteRule();  
1244         });  
1245     }  
1246  
1247     /**
1248      * Sets the language for the current window.  
1249      * @param {string} lang - The language code to set.  
1250      * @returns {void}  
1251      */  
1252     _setLanguage(lang) {  
1253         this.lang = lang;  
1254         this._updateLanguage();  
1255     }  
1256  
1257     /**
1258      * Gets the current language code.  
1259      * @returns {string} The current language code.  
1260      */  
1261     _getLanguage() {  
1262         return this.lang;  
1263     }  
1264  
1265     /**
1266      * Checks if the current language is supported.  
1267      * @param {string} lang - The language code to check.  
1268      * @returns {boolean} Whether the language is supported.  
1269      */  
1270     _isSupported(lang) {  
1271         return language.isSupported(lang);  
1272     }  
1273  
1274     /**
1275      * Deletes the language styles from the DOM.  
1276      * @returns {void}  
1277      */  
1278     _deleteLanguage() {  
1279         this.lang_styles.cssRules.forEach(function(rule) {  
1280             rule.deleteRule();  
1281         });  
1282     }  
1283  
1284     /**
1285      * Sets the language for the current window.  
1286      * @param {string} lang - The language code to set.  
1287      * @returns {void}  
1288      */  
1289     _setLanguage(lang) {  
1290         this.lang = lang;  
1291         this._updateLanguage();  
1292     }  
1293  
1294     /**
1295      * Gets the current language code.  
1296      * @returns {string} The current language code.  
1297      */  
1298     _getLanguage() {  
1299         return this.lang;  
1300     }  
1301  
1302     /**
1303      * Checks if the current language is supported.  
1304      * @param {string} lang - The language code to check.  
1305      * @returns {boolean} Whether the language is supported.  
1306      */  
1307     _isSupported(lang) {  
1308         return language.isSupported(lang);  
1309     }  
1310  
1311     /**
1312      * Deletes the language styles from the DOM.  
1313      * @returns {void}  
1314      */  
1315     _deleteLanguage() {  
1316         this.lang_styles.cssRules.forEach(function(rule) {  
1317             rule.deleteRule();  
1318         });  
1319     }  
1320  
1321     /**
1322      * Sets the language for the current window.  
1323      * @param {string} lang - The language code to set.  
1324      * @returns {void}  
1325      */  
1326     _setLanguage(lang) {  
1327         this.lang = lang;  
1328         this._updateLanguage();  
1329     }  
1330  
1331     /**
1332      * Gets the current language code.  
1333      * @returns {string} The current language code.  
1334      */  
1335     _getLanguage() {  
1336         return this.lang;  
1337     }  
1338  
1339     /**
1340      * Checks if the current language is supported.  
1341      * @param {string} lang - The language code to check.  
1342      * @returns {boolean} Whether the language is supported.  
1343      */  
1344     _isSupported(lang) {  
1345         return language.isSupported(lang);  
1346     }  
1347  
1348     /**
1349      * Deletes the language styles from the DOM.  
1350      * @returns {void}  
1351      */  
1352     _deleteLanguage() {  
1353         this.lang_styles.cssRules.forEach(function(rule) {  
1354             rule.deleteRule();  
1355         });  
1356     }  
1357  
1358     /**
1359      * Sets the language for the current window.  
1360      * @param {string} lang - The language code to set.  
1361      * @returns {void}  
1362      */  
1363     _setLanguage(lang) {  
1364         this.lang = lang;  
1365         this._updateLanguage();  
1366     }  
1367  
1368     /**
1369      * Gets the current language code.  
1370      * @returns {string} The current language code.  
1371      */  
1372     _getLanguage() {  
1373         return this.lang;  
1374     }  
1375  
1376     /**
1377      * Checks if the current language is supported.  
1378      * @param {string} lang - The language code to check.  
1379      * @returns {boolean} Whether the language is supported.  
1380      */  
1381     _isSupported(lang) {  
1382         return language.isSupported(lang);  
1383     }  
1384  
1385     /**
1386      * Deletes the language styles from the DOM.  
1387      * @returns {void}  
1388      */  
1389     _deleteLanguage() {  
1390         this.lang_styles.cssRules.forEach(function(rule) {  
1391             rule.deleteRule();  
1392         });  
1393     }  
1394  
1395     /**
1396      * Sets the language for the current window.  
1397      * @param {string} lang - The language code to set.  
1398      * @returns {void}  
1399      */  
1400     _setLanguage(lang) {  
1401         this.lang = lang;  
1402         this._updateLanguage();  
1403     }  
1404  
1405     /**
1406      * Gets the current language code.  
1407      * @returns {string} The current language code.  
1408      */  
1409     _getLanguage() {  
1410         return this.lang;  
1411     }  
1412  
1413     /**
1414      * Checks if the current language is supported.  
1415      * @param {string} lang - The language code to check.  
1416      * @returns {boolean} Whether the language is supported.  
1417      */  
1418     _isSupported(lang) {  
1419         return language.isSupported(lang);  
1420     }  
1421  
1422     /**
1423      * Deletes the language styles from the DOM.  
1424      * @returns {void}  
1425      */  
1426     _deleteLanguage() {  
1427         this.lang_styles.cssRules.forEach(function(rule) {  
1428             rule.deleteRule();  
1429         });  
1430     }  
1431  
1432     /**
1433      * Sets the language for the current window.  
1434      * @param {string} lang - The language code to set.  
1435      * @returns {void}  
1436      */  
1437     _setLanguage(lang) {  
1438         this.lang = lang;  
1439         this._updateLanguage();  
1440     }  
1441  
1442     /**
1443      * Gets the current language code.  
1444      * @returns {string} The current language code.  
1445      */  
1446     _getLanguage() {  
1447         return this.lang;  
1448     }  
1449  
1450     /**
1451      * Checks if the current language is supported.  
1452      * @param {string} lang - The language code to check.  
1453      * @returns {boolean} Whether the language is supported.  
1454      */  
1455     _isSupported(lang) {  
1456         return language.isSupported(lang);  
1457     }  
1458  
1459     /**
1460      * Deletes the language styles from the DOM.  
1461      * @returns {void}  
1462      */  
1463     _deleteLanguage() {  
1464         this.lang_styles.cssRules.forEach(function(rule) {  
1465             rule.deleteRule();  
1466         });  
1467     }  
1468  
1469     /**
1470      * Sets the language for the current window.  
1471      * @param {string} lang - The language code to set.  
1472      * @returns {void}  
1473      */  
1474     _setLanguage(lang) {  
1475         this.lang = lang;  
1476         this._updateLanguage();  
1477     }  
1478  
1479     /**
1480      * Gets the current language code.  
1481      * @returns {string} The current language code.  
1482      */  
1483     _getLanguage() {  
1484         return this.lang;  
1485     }  
1486  
1487     /**
1488      * Checks if the current language is supported.  
1489      * @param {string} lang - The language code to check.  
1490      * @returns {boolean} Whether the language is supported.  
1491      */  
1492     _isSupported(lang) {  
1493         return language.isSupported(lang);  
1494     }  
1495  
1496     /**
1497      * Deletes the language styles from the DOM.  
1498      * @returns {void}  
1499      */  
1500     _deleteLanguage() {  
1501         this.lang_styles.cssRules.forEach(function(rule) {  
1502             rule.deleteRule();  
1503         });  
1504     }  
1505  
1506     /**
1507      * Sets the language for the current window.  
1508      * @param {string} lang - The language code to set.  
1509      * @returns {void}  
1510      */  
1511     _setLanguage(lang) {  
1512         this.lang = lang;  
1513         this._updateLanguage();  
1514     }  
1515  
1516     /**
1517      * Gets the current language code.  
1518      * @returns {string} The current language code.  
1519      */  
1520     _getLanguage() {  
1521         return this.lang;  
1522     }  
1523  
1524     /**
1525      * Checks if the current language is supported.  
1526      * @param {string} lang - The language code to check.  
1527      * @returns {boolean} Whether the language is supported.  
1528      */  
1529     _isSupported(lang) {  
1530         return language.isSupported(lang);  
1531     }  
1532  
1533     /**
1534      * Deletes the language styles from the DOM.  
1535      * @returns {void}  
1536      */  
1537     _deleteLanguage() {  
1538         this.lang_styles.cssRules.forEach(function(rule) {  
1539             rule.deleteRule();  
1540         });  
1541     }  
1542  
1543     /**
1544      * Sets the language for the current window.  
1545      * @param {string} lang - The language code to set.  
1546      * @returns {void}  
1547      */  
1548     _setLanguage(lang) {  
1549         this.lang = lang;  
1550         this._updateLanguage();  
1551     }  
1552  
1553     /**
1554      * Gets the current language code.  
1555      * @returns {string} The current language code.  
1556      */  
1557     _getLanguage() {  
1558         return this.lang;  
1559     }  
1560  
1561     /**
1562      * Checks if the current language is supported.  
1563      * @param {string} lang - The language code to check.  
1564      * @returns {boolean} Whether the language is supported.  
1565      */  
1566     _isSupported(lang) {  
1567         return language.isSupported(lang);  
1568     }  
1569  
1570     /**
1571      * Deletes the language styles from the DOM.  
1572      * @returns {void}  
1573      */  
1574     _deleteLanguage() {  
1575         this.lang_styles.cssRules.forEach(function(rule) {  
1576             rule.deleteRule();  
1577         });  
1578     }  
1579  
1580     /**
1581      * Sets the language for the current window.  
1582      * @param {string} lang - The language code to set.  
1583      * @returns {void}  
1584      */  
1585     _setLanguage(lang) {  
1586         this.lang = lang;  
1587         this._updateLanguage();  
1588     }  
1589  
1590     /**
1591      * Gets the current language code.  
1592      * @returns {string} The current language code.  
1593      */  
1594     _getLanguage() {  
1595         return this.lang;  
1596     }  
1597  
1598     /**
1599      * Checks if the current language is supported.  
1600      * @param {string} lang - The language code to check.  
1601      * @returns {boolean} Whether the language is supported.  
1602      */  
1603     _isSupported(lang) {  
1604         return language.isSupported(lang);  
1605     }  
1606  
1607     /**
1608      * Deletes the language styles from the DOM.  
1609      * @returns {void}  
1610      */  
1611     _deleteLanguage() {  
1612         this.lang_styles.cssRules.forEach(function(rule) {  
1613             rule.deleteRule();  
1614         });  
1615     }  
1616  
1617     /**
1618      * Sets the language for the current window.  
1619      * @param {string} lang - The language code to set.  
1620      * @returns {void}  
1621      */  
1622     _setLanguage(lang) {  
1623         this.lang = lang;  
1624         this._updateLanguage();  
1625     }  
1626  
1627     /**
1628      * Gets the current language code.  
1629      * @returns {string} The current language code.  
1630      */  
1631     _getLanguage() {  
1632         return this.lang;  
1633     }  
1634  
1635     /**
1636      * Checks if the current language is supported.  
1637      * @param {string} lang - The language code to check.  
1638      * @returns {boolean} Whether the language is supported.  
1639      */  
1640     _isSupported(lang) {  
1641         return language.isSupported(lang);  
1642     }  
1643  
1644     /**
1645      * Deletes the language styles from the DOM.  
1646      * @returns {void}  
1647      */  

```

```

739     }
740     this.lang_styles.insertRule("lang."+language.lang+" { display: initial }",
741         1);
742
743     this.dom.querySelectorAll('[title-str]').forEach(function(el) {
744         var id = el.getAttribute('title-str');
745         if(id in language.s) {
746             el.setAttribute('title', language.s[id][language.lang]);
747         }
748     });
749 }
```

Listing 114 - windows.js

6.6 windows

```

1  /**
2  * @file JSLAB library plot script
3  * @author Milos Petrasinovic <mpetrasinovic@pr-dc.com>
4  * PR-DC, Republic of Serbia
5  * info@pr-dc.com
6  */
7
8 /**
9 * Class for JSLAB plot.
10 */
11 class PRDC_JSLAB_PLOT {
12
13 /**
14 * Initializes an instance of the PRDC_JSLAB_PLOT class.
15 */
16 constructor() {
17     var obj = this;
18     this.figure_content = document.getElementById('figure-content');
19     this.plot_cont;
20 }
21
22 /**
23 * Set a new plot container to the figure content.
24 */
25 setCont() {
26     this.plot_cont = document.createElement('div');
27     this.plot_cont.className = 'plot-cont';
28     this.figure_content.appendChild(this.plot_cont);
29 }
30
31 /**
32 * Creates a new plot using Plotly in the plot container.
33 * @param {Object} plot_in - An object containing plot initial settings.
34 * @param {Array} traces - Data traces for the plot.
35 * @param {Object} layout - Layout configuration for the plot.
36 * @param {Object} config - Additional configuration settings for the plot.
37 */

```

```

38     async newPlot(plot_in, traces, layout, config) {
39         await Plotly.newPlot(this.plot_cont, traces, layout, config);
40     }
41
42     /**
43      * Updates the layout of the existing plot.
44      * @param {Object} layout - Layout configuration to apply to the plot.
45      */
46     relayout(layout) {
47         Plotly.relayout(this.plot_cont, layout);
48     }
49
50     /**
51      * Restyles the plot with updated trace data.
52      * @param {Object} traces - The trace data to apply styling changes to.
53      * @param {number} N - The index or length of data for styling adjustments.
54      */
55     restyle(traces, N) {
56         Plotly.restyle(this.plot_cont, traces, N);
57     }
58
59     /**
60      * Resizes the plot to fit its container.
61      */
62     resize() {
63         Plotly.Plots.resize(this.plot_cont);
64     }
65
66     /**
67      * Converts the plot to an image.
68      * @param {string} ext - The image format extension (e.g., 'png', 'jpeg').
69      * @param {Function} callback - Callback function that handles the image URL
70      */
71     async toImage(ext, size) {
72         var width = this.plot_cont.clientWidth;
73         var height = this.plot_cont.clientHeight;
74         if(typeof size != 'undefined') {
75             width = size[0];
76             height = size[1];
77         }
78         return await Plotly.toImage(this.plot_cont, {format: ext,
79             width: width,
80             height: height
81         });
82     }
83 }
84
85 var plot = new PRDC_JSLAB_PLOT();

```

Listing 115 - plot.js