Grid

Grid

Creates a new Grid, this contains columns, rows, and drag elements. Columns are vertical and can have n rows inside them. Rows are containers for divs, as the rows are resided, the div content will also be resided to fit.

Constructor

```
new Grid(width, height, columns, rows)
```

Source: grid/grid.js, line 35

The Grid class takes in the window width and height, passed in to establish a basic window size. The default grid has no rows or columns within it. To add content call the Grid#createColumn method.

Parameters:

Name	Type	Description
`width`	Number	This is the width of the grid in pixels.
`height`	Number	This is the height of the grid in pixels.
`columns`	Number	This is the number of columns that the grid should be initialized with.
`rows`	Number	This is the number of rows that each column should have.

Methods

```
addColumn(column)
```

Source: grid/grid.js, line 362

This method can be called to add a column to the current grid. The grid is flexible, and has a constants size of 100%, when a new column is added, the default width of the column is (1/n)% where n is the number of columns in the grid. This method can be called at any time to add a new column.

Parameters:

Name	Type	Description
`column`	Column	The Column that will be added.

addDrag(drag)

Source: grid/grid.js, line 392

This method lets the user register a new drag to appear on the window.

Parameters:

Name	Type	Description
`drag`	H_Drag	The Horizontal drag to add to the window.

addWidget(widget)

Source: grid/grid.js, line 707

This function adds a widget into the grid of widgets being displayed

Parameters:

Name	Type	Description
`widget`	Widget	This widget will be added into the grid at widget.col widget.row

createColumn(elements, properties) → {Column}

Source: grid/grid.js, line 133

Creates a Column. A column is a resizable container for data. Columns contain an index (left to right) starting at 0, representing which column they are numerically on the screen. Columns also contain a list of children

Parameters:

Name	Type	Description
`elements`	Array	This is an array or a single html element. For every dom element passed, a new unique row will be created in this column

Name	Type	Description
`properties`	Object	These are additional configuration parameters that can be passed into a column. *Properties* **Name** / **Type** / **Description** `color` / String / A hexadecimal color to apply to the background of this column. `id` / String / a string representing the ID that should be associated with this column

Returns:

A json object representing a Column.

Type Column

createDrag(col1, col2) → {H_Drag}

Source: grid/grid.js, line 288

Creates a horizontal Drag. A horizontal drag is a small element conjoining two Columns. Clicking and dragging on a Drag will change the relative scale of the linked columns.

Parameters:

Name	Type	Description
`col1`	Column	The first column
`col2`	Column	The second column

Returns:

Returns a json object representing a horizontal drag.

Type H_Drag

createVDrag(row1, row2) → {V_Drag}

Source: grid/grid.js, line 330

Creates a vertical Drag. A vertical drag is a small element conjoining two rows. Clicking and dragging on a vertical Drag will change the relative scale of the linked rows.

Parameters:

Name	Type	Description
`row1`	Element	
`row2`	Element	

Returns:

Returns a json object representing a Column.

Type `V_Drag`

`executeCallbacks()`

Source: grid/grid.js, line 468

This function executes all registered callback functions inside of this grid.

`generateSaveObject() → {Object}`

Source: grid/grid.js, line 536

This method is called when the editor is saving the active configuration. All columns and rows report their widths and heights respectivly. A multi-dimensional array object is constructed. This array has all of the height data needed to recreacte the current editor spacing on the next editor load.

Returns:

sizes - The widths and heights of the current window configuration.

Type `Object`

`getCOLUMNS() → {Array.<Column>}`

Source: grid/grid.js, line 585

Get all columns in this grid.

Returns:

COLUMNS - The columns that make up this grid.

Type `Array.<Column>`

`getHEIGHT() → {Integer}`

Source: grid/grid.js, line 577

Get the grid height.

Returns:

HEIGHT - The height of the window.

Type `Integer`

`getWidget(col, row) → {Widget|null}`

Source: grid/grid.js, line 678

This function returns the widget stored in grid cell (col, row). If there is no widget in that cell null is returned.

Parameters:

Name	Type	Description
`col`	`Integer`	This the column or X of the grid that you are trying to access.
`row`	`Integer`	This the row or Y of the grid that you are trying to access.

Returns:

widget - This is the widget in grid cell (col, row) if there is no widget in that cell, null is returned.

Type `Widget | null`

`getWIDTH() → {Integer}`

Source: grid/grid.js, line 569

Get the grid width.

Returns:

WIDTH - The width of the window.

Type `Integer`

`init(widgets)`

Source: grid/grid.js, line 667

Initializes the grid with Widgets. A Widget is a synchronously spawned promise. This function will iterate through the widgets array and initialize one widget after another. This way later widgets can have earlier widgets passed into them. Example [Document(requires:[]), Tab(requires:[Document])] In this example the Widget Document requires nothing and is the first widget to be initialized in the program. Tab is the second widget to be initialized, and it requires Document. When Tab's initialize method is called, the value of Document will be stable and usable. This chaining method can be propagated forwards to allow widgets to require previously initialized widgets.

`See Widget for information on built in widgets and how to create your own.`

Parameters:

Name	Type	Description
`widgets`	Array	This is an array of Widget elements that will be initialized in array order.

(async) initalize(widgets, COLUMNS, saveData) → {Promise.<void>}

Source: grid/grid.js, line 614

Asynchronous loop which will Synchronous populates a cell with an intialized widget untill all widgets are initialized, at this point it will then return.

Parameters:

Name	Type	Description
`widgets`	Array.<Widget>	Uninitialized widgets to be initialized and loaded into the grid.
`COLUMNS`	Array.<Column>	The Columns that make up this EGAD grid.
`saveData`	Object	A save object representing the state of all widgets the last time the application was closed.

Returns:

- This promise resolves once all widgets are initialized.

Type `Promise.<void>`

initializeWidgit(widget, saveData) → {Promise.<any>}

Source: grid/grid.js, line 596

This call wraps the widget promise constructor inside of another promise. The return value of this function is awaited upon inside of the initialize method.

Parameters:

Name	Type	Description
`widget`	Widget	An uninitialized widget to initialize.
`saveData`	Object	An object containing information about the state of widgets the last time the application was closed.s

Returns:

This returns a promise wrapped around the widget's init function. The promise resolves when the widget finishes initializing.

Type `Promise.<any>`

`loadGridSizes(sizes)`

Source: grid/grid.js, line 505

This method is called on editor load. It takes in an array of size configuration data. This data is used to position all grid elements such that they retain the positions they were in the last time the editor was closed.

Parameters:

Name	Type	Description
`sizes`	`Array`	The sizes of the grid elements.

`onDrag(event, index1, index2)`

Source: grid/grid.js, line 421

This function is called to whenever a horizontal drag event is triggered.

Parameters:

Name	Type	Description
`event`	`Event`	This is the drag event that was detected.
`index1`	`Integer`	This is the index of the first column to be offset by this drag event.
`index2`	`Integer`	This is the index of the second column to be offset by this drag event.

`onDragEnd()`

Source: grid/grid.js, line 481

This event is triggered after a horizontal drag has finished moving, it is used to set the absolute position of the 2 columns referenced by the drag event.

`onVDrag(event, row1, row2)`

Source: grid/grid.js, line 443

This function is called to whenever a vertical drag event is triggered.

Parameters:

Name	Type	Description

Name	Type	Description
`event`	Event	This is the drag event that was detected.
`row1`	Integer	This is the index of the first row to be offset by this drag event.
`row2`	Integer	This is the index of the second row to be offset by this drag event.

`refresh()`

Source: grid/grid.js, line 404

This function is called to reposition all of the drags and grid elements to proper positions after a drag has occurred, or any externalmovementt actions.

`removeWidget(widget)`

Source: grid/grid.js, line 715

This function adds a widget into the grid of widgets being displayed

Parameters:

Name	Type	Description
`widget`	Widget	This widget will be added into the grid at widget.col widget.row

`resize()`

Source: grid/grid.js, line 496

This function is used to sync the values of WIDTH and HEIGHT after the document window has been resided.

`setWidget(col, row, widget)`

Source: grid/grid.js, line 693

This function changes the contents of cell (col, row) to widget. Null can be passed in to remove a widget from the grid, the row that that widget was in will persist however

Parameters:

Name	Type	Description
`col`	Integer	This the column or X of the grid that you are trying to access.

Name	Type	Description
`row`	Integer	This the row or Y of the grid that you are trying to access.
`widget`	Widget	This is the widget that cell (col, row) should be set to.

Widget

Widget

There is a collection of built in widgets tailored to making editor applications.

```
    FileBrowser for information on built in widgets and how to create your own.
```

Constructor

`new Widget(configData, dependencies)`

Source: widgets/widget.js, line 21

The Widget class takes in configData about where to position this widget in the grid, as well as a list of dependencies.

Parameters:

Name	Type	Description
`configData`	`Object`	This is an object of config data, It will have the following fields as well as any custom fields that child classes require. *Properties* (see below)
`dependencies`	`Object`	This is a map of String Names to Widgets which this object relies on.

Properties of `configData`:

Name	Type	Description
`name`	`String`	This is the name of this widget.
`col`	`Integer`	This is the Column that this widget should be added to.
`row`	`Integer`	This is the row of of the Column that this widget should be added to.

Methods

`getCol() → {Integer}`

Source: widgets/widget.js, line 89

Get the position in columns(Left = 0 to Right = n) of this widget.

Returns:

colIndex - The column index of this Widget.

Type `Integer`

`getElement() → {Element}`

Source: widgets/widget.js, line 81

Get the dom element that represents this widget.

Returns:

element - The dom object for this widget.

Type `Element`

`getRow() → {Integer}`

Source: widgets/widget.js, line 97

Get the position in rows(Top = 0 to Bottom = n) of this widget.

Returns:

rowIndex - The row index of this Widget.

Type `Integer`

`(abstract) init() → {Promise}`

Source: widgets/widget.js, line 54

This function is implemented by every child class of widget. Any initialization that needs to happen will be put inside of this function.

Returns:

Returns a promise that will be executed when an instance of this widget is generated.

Type `Promise`

`setElement(element)`

Source: widgets/widget.js, line 70

Set the dom element that represents this widget.

Parameters:

Name	Type	Description
`element`	`Element`	The dom object for this widget.

WebviewWidget

WebviewWidget

`new WebviewWidget(x, y, url) → {WebviewWidget}`

Source: widgets/webviewWidget.js, line 13

Parameters:

Name	Type	Description
`x`	`Integer`	the Column to add this widget to.
`y`	`Integer`	the Row to add this widget to.
`url`	`String`	This is the URL of the document to display in a webview. It can be remote or a local path.

Returns:

Returns a WebviewWidget, an instance of the Widget class which allows a user to display a remote webpage, or a local html file.

Type `WebviewWidget`

Extends

- Widget

Methods

`getCol() → {Integer}`

Source: widgets/widget.js, line 89

Inherited From: Widget#getCol

Get the position in columns(Left = 0 to Right = n) of this widget.

Returns:

colIndex - The column index of this Widget.

Type `Integer`

`getElement() → {Element}`

Source:	widgets/widget.js, line 81
Inherited From:	Widget#getElement

Get the dom element that represents this widget.

Returns:

element - The dom object for this widget.

Type `Element`

`getRow() → {Integer}`

Source:	widgets/widget.js, line 97
Inherited From:	Widget#getRow

Get the position in rows(Top = 0 to Bottom = n) of this widget.

Returns:

rowIndex - The row index of this Widget.

Type `Integer`

`(async) init() → {Promise.<any>}`

Source:	widgets/webviewWidget.js, line 21
Overrides:	Widget#init

This function overrides the parent widget initialization call and creates a webview element with the desired document displayed inside.

Returns:

Type `Promise.<any>`

`postinit()`

Source:	widgets/webviewWidget.js, line 40

This function is called after initialization has occurred on this widget, by this time all fields this widget references should be initialized.

`setElement(element)`

Source:	widgets/widget.js, line 70

> Inherited From: Widget#setElement

Set the dom element that represents this widget.

Parameters:

Name	Type	Description
`element`	`Element`	The dom object for this widget.

FileTreeWidget

FileTreeWidget

```
new FileTreeWidget(x, y, path, fileManager) → {FileTreeWidget}
```

Source: widgets/fileTreeWidget.js, line 16

Parameters:

Name	Type	Description
`x`	Integer	This is the Column that this widget should be added to.
`y`	Integer	This is the row of of the Column that this widget should be added to.
`path`	String	This is a string representing the path to the directory this file tree should display. Paths relative to the root folder are denoted with '~/folderName'.
`fileManager`	FileManager	This is a reference to the fileManager class which provides this widget with access to the computers File System.

Returns:

Returns a FileTreeWidget, an instance of the Widget class.

Type `FileTreeWidget`

Extends

- Widget

Methods

```
doubleClick(event, data)
```

Source: widgets/fileTreeWidget.js, line 84

This function is the callback method injected into this fileTreeWidget. Whenever a user double clicks on a file inside of the file tree, this function is called. This function should be modified to a specific developers needs.

Parameters:

Name	Type	Description
`event`	`Object`	The event object contains information about what node was clicked as well as the specific DOM element that was interacted with.
`data`	`Object`	Data holds all of the data for that FileTree node.

`getCol() → {Integer}`

Source: widgets/widget.js, line 89

Inherited From: Widget#getCol

Get the position in columns(Left = 0 to Right = n) of this widget.

Returns:

colIndex - The column index of this Widget.

Type `Integer`

`getElement() → {Element}`

Source: widgets/widget.js, line 81

Inherited From: Widget#getElement

Get the dom element that represents this widget.

Returns:

element - The dom object for this widget.

Type `Element`

`getRow() → {Integer}`

Source: widgets/widget.js, line 97

Inherited From: Widget#getRow

Get the position in rows(Top = 0 to Bottom = n) of this widget.

Returns:

rowIndex - The row index of this Widget.

Type `Integer`

`(async) init() → {Promise}`

Source: widgets/fileTreeWidget.js, line 33

Overrides: Widget#init

This function recursively opens subdirectories from the given path, and then produces a file-tree object to be displayed within this file browser widget.

Returns:

Returns an asynchronous promise that will resolve on tree generation.

Type `Promise`

`setElement(element)`

Source: widgets/widget.js, line 70

Inherited From: Widget#setElement

Set the dom element that represents this widget.

Parameters:

Name	Type	Description
`element`	`Element`	The dom object for this widget.

`subscribe(observer)`

Source: widgets/fileTreeWidget.js, line 95

This function allows any class to pass functions into the observers object, when a file is clicked on, every observer will have their callback functions trigger.

Parameters:

Name	Type	Description
`observer`	`function`	Observer is a callback function to execute when a file is double clicked on.

`translateRelative() → {String}`

Source: widgets/fileTreeWidget.js, line 71

This function converts the '~' character into a path to the fileManger.PATH value. This value can be configured in the fileManager configuration file, therefore ~ will always point to that variable. This allows users to use realtive pathing by simply putting '~' in front of their path.

Returns:

Returns the path to a folder relative to root.

Type `String`

codeEditorWidget

codeEditorWidget

```
new codeEditorWidget(x, y, language, themeopt) → {WebviewWidget}
```

Source: widgets/codeEditorWidget.js, line 14

Parameters:

Name	Type	Attributes	Default	Description
`x`	`Integer`			This is the Column that this widget should be added to.
`y`	`Integer`			This is the row of of the Column that this widget should be added to.
`language`	`String`			This is the language that CodeMirror should target when formatting the contents of this widget.
`theme`	`String`	<optional>	`darcula`	CodeMirror theme to use when formatting this editor. Any theme in the 'node_modules/codemirror/theme/' directory will work. Just pass the name of the css file and this does the rest.

Returns:

Returns a WebviewWidget, an instance of the Widget class which allows a user to display a remote webpage, or a local html file.

Type `WebviewWidget`

Extends

- Widget

Methods

```
getCol() → {Integer}
```

Source: widgets/widget.js, line 89

Inherited From: Widget#getCol

Get the position in columns(Left = 0 to Right = n) of this widget.

Returns:

colIndex - The column index of this Widget.

Type `Integer`

`getElement() → {Element}`

Source: widgets/widget.js, line 81

Inherited From: Widget#getElement

Get the dom element that represents this widget.

Returns:

element - The dom object for this widget.

Type `Element`

`getRow() → {Integer}`

Source: widgets/widget.js, line 97

Inherited From: Widget#getRow

Get the position in rows(Top = 0 to Bottom = n) of this widget.

Returns:

rowIndex - The row index of this Widget.

Type `Integer`

`(async) init(configData) → {Promise.<any>}`

Source: widgets/codeEditorWidget.js, line 30

Overrides: Widget#init

This function overrides the parent widget initialize function and creates a code editor to be displayed within this widget.

Parameters:

Name	Type	Description
`configData`	`Object`	This object contains important information about the state that this widget was in the last time the application closed. The exact value inside of the widget is passed back into it here. This object also contains information about what language this editor is editing.

Returns:

Type `Promise.<any>`

`postinit()`

Source: widgets/codeEditorWidget.js, line 78

This function is called after initialization has occurred on this widget, by this time all fields this widget references should be initialized.

`save() → {Object}`

Source: widgets/codeEditorWidget.js, line 87

This function overrides the parent widget save function. The save object returned contians the language, theme, and content of the code editor.

Returns:

Type `Object`

`setElement(element)`

Source: widgets/widget.js, line 70

Inherited From: Widget#setElement

Set the dom element that represents this widget.

Parameters:

Name	Type	Description
`element`	`Element`	The dom object for this widget.

consoleWidget

consoleWidget

`new consoleWidget(x, y, textSize_opt) → {canvasWidget}`

Source: widgets/consoleWidget.js, line 20

Parameters:

Name	Type	Attributes	Default	Description
`x`	`Integer`			`This is the Column that this widget should be added to.`
`y`	`Integer`			`This is the row of of the Column that this widget should be added to.`
`textSize`	`Integer`	`<optional>`	`18`	`The font size to use in the console.`

Returns:

Returns a canvasWidget, an instance of the Widget class which allows a user to draw on an html canvas.

Type `canvasWidget`

Extends

- Widget

Methods

`getCol() → {Integer}`

Source: widgets/widget.js, line 89

Inherited From: Widget#getCol

Get the position in columns(Left = 0 to Right = n) of this widget.

Returns:

colIndex - The column index of this Widget.

Type `Integer`

`getElement() → {Element}`

Source:	widgets/widget.js, line 81
Inherited From:	Widget#getElement

Get the dom element that represents this widget.

Returns:

element - The dom object for this widget.

Type `Element`

`getRow() → {Integer}`

Source:	widgets/widget.js, line 97
Inherited From:	Widget#getRow

Get the position in rows(Top = 0 to Bottom = n) of this widget.

Returns:

rowIndex - The row index of this Widget.

Type `Integer`

`(async) init() → {Promise.<any>}`

Source:	widgets/consoleWidget.js, line 32
Overrides:	Widget#init

This function overrides the parent widget initialize function and creates a console to be displayed within this widget.

Returns:

Type `Promise.<any>`

`log(message)`

Source:	widgets/consoleWidget.js, line 119

This function allows a string to be passed in to then be printed to the consoles output.

Parameters:

Name	Type	Description
`message`	`String`	This is the message to print to this consoles output area.

setElement(element)

Source: widgets/widget.js, line 70

Inherited From: Widget#setElement

Set the dom element that represents this widget.

Parameters:

Name	Type	Description
`element`	`Element`	The dom object for this widget.

subscribe(observer)

Source: widgets/consoleWidget.js, line 132

This function allows developers to register function callbacks to be excuted whenever enter is pressed when text is inside the input field of this console. The text is passed into all callback functions.

Parameters:

Name	Type	Description
`observer`	`function`	Function to be called whenever enter is pressed from the consoles input field.

tabWidget

tabWidget

`new tabWidget(x, y, fileTreeWidget) → {canvasWidget}`

Source: widgets/tabWidget.js, line 19

Parameters:

Name	Type	Description
`x`	`Integer`	`This is the Column that this widget should be added to.`
`y`	`Integer`	`This is the row of of the Column that this widget should be added to.`
`fileTreeWidget`	`FileTreeWidget`	`Reference to a FileTree.`

Returns:

Returns a tabWidget, an instance of the Widget class which creates a new tab every time a file in the file tree is double clicked on.

Type `canvasWidget`

Extends

- Widget

Methods

`getCol() → {Integer}`

Source: widgets/widget.js, line 89

Inherited From: Widget#getCol

Get the position in columns(Left = 0 to Right = n) of this widget.

Returns:

colIndex - The column index of this Widget.

Type `Integer`

getElement() → {HTMLElement|*}

Source:	widgets/tabWidget.js, line 106
Overrides:	Widget#getElement

Getter for this widgets tab element. Used to append child nodes to the base tab bar element.

Returns:

Type `HTMLElement | *`

getPath(node) → {String}

Source:	widgets/tabWidget.js, line 147

This function is used to get an absolute path for a specific file from a FancyTree node element.

Parameters:

Name	Type	Description
`node`	`FancytreeNode`	This is a fancy tree node, a path is generated from this call.

Returns:

The file path to the root directory of this node.

Type `String`

getRow() → {Integer}

Source:	widgets/widget.js, line 97
Inherited From:	Widget#getRow

Get the position in rows(Top = 0 to Bottom = n) of this widget.

Returns:

rowIndex - The row index of this Widget.

Type `Integer`

(async) init() → {Promise.<any>}

Source:	widgets/tabWidget.js, line 39
Overrides:	Widget#init

This function overrides the parent widget initialize function and creates a tab bar to be displayed within this widget.

Returns:

Type `Promise.<any>`

`openFile(filePath)`

Source: widgets/tabWidget.js, line 68

This function takes a filePath, and adds a new tab to the tab bar, as well as calls the callback function defined for this file type if it is known.

Parameters:

Name	Type	Description
`filePath`	String	The Path to a file to open. Perform the callback function registered for this file type.

`performCallbackForFileType(title)`

Source: widgets/tabWidget.js, line 126

This function determines if any callbacks for the passed file extension are known. If they are known they will be performed.

Parameters:

Name	Type	Description
`title`	String	This is the name of a file. The file extension is stripped from the file.

`registerFiletype(extension, callback)`

Source: widgets/tabWidget.js, line 57

This lets you register a callback to trigger when a file of a specifc type is opened Callback must contain {extension:"the file extension", callback:function()}

Parameters:

Name	Type	Description
`extension`	String	This is the file extension that you want to register a callback for, ie '.png'
`callback`	function	This is the function you want to execute when a file of type 'extension' is clicked on.

`resizeTabs()`

Source: widgets/tabWidget.js, line 113

This function is called whenever this widget is resized, it will automatically set each widget to be the correct size.

`save() → {Object}`

Source: widgets/tabWidget.js, line 162

This is simply a wrapper to the parent widget save function.

Returns:

Type `Object`

`setElement(element)`

Source: widgets/widget.js, line 70

Inherited From: Widget#setElement

Set the dom element that represents this widget.

Parameters:

Name	Type	Description
`element`	`Element`	The dom object for this widget.

canvasWidget

canvasWidget

`new canvasWidget(x, y, width, height) → {canvasWidget}`

Source: widgets/canvasWidget.js, line 16

Parameters:

Name	Type	Description
`x`	`Integer`	`This is the Column that this widget should be added to.`
`y`	`Integer`	`This is the row of of the Column that this widget should be added to.`
`width`	`Integer`	`This is the width in pixels that this canvas element should take up.`
`height`	`Integer`	`This is the height in pixels that this canvas element should take up.`

Returns:

Returns a canvasWidget, an instance of the Widget class which allows a user to draw on an html canvas.

Type `canvasWidget`

Extends

- Widget

Methods

`draw()`

Source: widgets/canvasWidget.js, line 93

`getCol() → {Integer}`

Source: widgets/widget.js, line 89

Inherited From: Widget#getCol

Get the position in columns(Left = 0 to Right = n) of this widget.

Returns:

colIndex - The column index of this Widget.

Type `Integer`

`getElement() → {Element}`

Source: widgets/widget.js, line 81

Inherited From: Widget#getElement

Get the dom element that represents this widget.

Returns:

element - The dom object for this widget.

Type `Element`

`getRow() → {Integer}`

Source: widgets/widget.js, line 97

Inherited From: Widget#getRow

Get the position in rows(Top = 0 to Bottom = n) of this widget.

Returns:

rowIndex - The row index of this Widget.

Type `Integer`

`(async) init(configData)`

Source: widgets/canvasWidget.js, line 33

Overrides: Widget#init

This function overrides the parent widgets init function to create a new canvas widget.

Parameters:

Name	Type	Description
`configData`	`Object`	This is the save object passed back into the function, the only important field on this object is 'fps' which determines the target framerate of the canvas. * @return {Promise} - This promise resolves once this widget has initialized.

`postinit()`

Source: widgets/canvasWidget.js, line 59

This function triggers after the widget has initialized, at this point all fields should be able to be referenced. In the canvas widget this function registers a callback function to run 'fps' times per second.

`save() → {Object}`

Source: widgets/canvasWidget.js, line 122

This function generates a save object so that this widget can initialize to the state which it is in the next time the application starts.

Returns:

Type `Object`

`setElement(element)`

Source: widgets/widget.js, line 70

Inherited From: Widget#setElement

Set the dom element that represents this widget.

Parameters:

Name	Type	Description
`element`	`Element`	The dom object for this widget.

`setFameRate(fps)`

Source: widgets/canvasWidget.js, line 105

This function allows a user to adjust the rate at which the screen refreshes. The parameter fps specifies the new target frame-rate.

Parameters:

Name	Type	Description
`fps`	`Integeт`	The target frame rate for this canvas.

`subscribeToDraw(observer)`

Source: widgets/canvasWidget.js, line 82

This function allows a user to subscribe to this widgets draw call, The passed function will have gl passed to it, and will be called 'fps' times per second.

Parameters:

Name	Type	Description
`observer`	`function`	`This is a callback function to execute fps times per second.`

FileManager

FileManager

```
new FileManager()
```

Source: util/fileManager.js, line 22

Creates a file manager. This class can read and write files.

Methods

```
convertFileToFolderObject(subdir, fileName) → {Promise.<any>}
```

Source: util/fileManager.js, line 260

Helper function for the 'getProjectFiles' function, This function converts directories to directory endpoints, then recursively calls the getProjectFiles function to add children endpoints to itself.

Parameters:

Name	Type	Description
`subdir`		Directory name to convert to an object.
`fileName`		File name to convert to an object.

Returns:

Returns an object representing this directory and all children of this directory.

Type `Promise.<any>`

```
convertFileToObject(fileName) → {Object}
```

Source: util/fileManager.js, line 247

Helper function for the 'getProjectFiles' function, simply converts a string name, into an object indicating that this file is an endpoint, not a directory.

Parameters:

Name	Type	Description

Name	Type	Description
`fileName`		File name to convert to an object.

Returns:

Type `Object`

`getProjectFiles(subdir, ignore) → {Promise.<any>}`

Source: util/fileManager.js, line 178

This function is a recursive call, it will propogate through all subdirectories of 'subdir' until all child directories have been traversed.

Parameters:

Name	Type	Description
`subdir`	`String`	The directory to open and convert to a JSON object.
`ignore`	`Ignore`	This is the blacklist information to reference when generating this object.

Returns:

This promise will resolve once all subdirectories have been traversed and a valid save object is generated.

Type `Promise.<any>`

`initialize() → {Promise.<any>}`

Source: util/fileManager.js, line 36

This function is used to initialize this file manager. When this function is called, a promise is returned. The promise will resolve once the file defined by 'SAVE_PATH/CONFIG_FILE' has been read and parsed into a JSON object.

Returns:

Type `Promise.<any>`

`loadFile(fileName) → {Promise.<any>}`

Source: util/fileManager.js, line 61

This function allows a user to load a file relative to the 'SAVE_PATH' for example, if the user were to pass 'sampleLanguage.json' to this file, the framework would try to load the file 'root/sampleLanguage.json' Once the file has been found, the contents will be read as utf8 text and returned as a promise. This promise resolves once all lines of the file have been read and are contained within the 'data' object.

Parameters:

Name	Type	Description
`fileName`		

Returns:

Type `Promise.<any>`

`(async) readFromProperties(fieldName) → {Promise.<any>}`

Source: util/fileManager.js, line 106

This function is the inverse of 'writeToProperties' It allows a user to read the field 'fieldName' off of the properties object.

Parameters:

Name	Type	Description
`fieldName`		`The field to read.`

Returns:

This function returns a promise that resolves when the data is read off of the field, and rejects when their is an error reading that specific field.

Type `Promise.<any>`

`writeToFile(fileName, data) → {Promise.<any>}`

Source: util/fileManager.js, line 155

This function allows a user to write data to an arbitrary file. The user can specify the file in 'fileName' and the contents of that file in the 'data' object. The file refrenced by file name will be in the path defined by 'SAVE_PATH/fileName'.

Parameters:

Name	Type	Description
`fileName`		`The name of the file to write to.`
`data`		`The data to write to the file.`

Returns:

Type `Promise.<any>`

(async) writeToProperties(field, data) → {Promise.<any>}

Source: util/fileManager.js, line 82

This function allows a developer to esaily write to the config json object. This object is persisted between instances of the application running.

Parameters:

Name	Type	Description
`field`	String	This is the field on the config object that you want to set.
`data`	Object	This is the value that 'field' should be set to.

Returns:

This function returns a promise which resolves if the write was successful, or rejects if there was an error.

Type `Promise.<any>`

languageParser

languageParser

new languageParser(languageInformation) → {languageParser}

Source: util/languageParser.js, line 39

The language parser class allows for quick indexing and predictive searching of user defined functions.

Parameters:

Name	Type	Description
`languageInformation`	Object	This is the key to a language, all aspects of the language such as functions, primitive types, and commenting information are defined within this object.

Returns:

Returns a new language parser ready to parse the language defined by

Type `languageParser`

Methods

addFunction(l_function)

Source: util/languageParser.js, line 241

This function is used when reading the language description. This function generates a hashmap of functions defined within this languages domain. These functions are looked up when generating IntelliSence

Parameters:

Name	Type	Description
`l_function`	l_function	The language specific function to add to this languages registered functions.

cursorInScope(cursor, scope) → {boolean}

Source: util/languageParser.js, line 175

This function takes a cursor and a scope and returns true if the cursor is inside of the scope.

Parameters:

Name	Type	Description
`cursor`		`cursor object`
`scope`		`scope to check cursor against.`

Returns:

If cursor 'cursor' is inside of scope 'scope'

Type `boolean`

`cursorToScope(cursor) → {*}`

Source: util/languageParser.js, line 158

This function returns which scope the cursor is currently in. A cursor object contains a line number and a character.

Parameters:

Name	Type	Description
`cursor`	`Cursor`	`An object representing a cursors position inside of this file.`

Returns:

Returs the scope that the cursor is inside of.

Type *

`getLastToken(tokenArrayg) → {*}`

Source: util/languageParser.js, line 326

Returns the last token of a line

Parameters:

Name	Type	Description
`tokenArrayg`		

Returns:

Type *

getSubScope(scope) → {Array.<Scope>}

Source: util/languageParser.js, line 209

This function returns all child scopes of a desired scope.

Parameters:

Name	Type	Description
`scope`	`Scope`	`The scope to get the children of.`

Returns:

Returns the array of child scopes parented to 'scope'.

Type `Array.<Scope>`

getSuggestion(string, cursor) → {Array.<l_function>}

Source: util/languageParser.js, line 257

This function returs an array of l_functions which the user could be typing. This function performs a fuzzy search through the list of defined functions and generates a subset of functions that are similar to 'string'

Parameters:

Name	Type	Description
`string`	`String`	`String to search`
`cursor`	`Cursor`	`position in the file`

Returns:

Returns an array of l_functions which the user could be trying to type.

Type `Array.<l_function>`

loadFileSpecificData(fileData)

Source: util/languageParser.js, line 69

This function takes in data from a file and generate a tree structure for this file representing the scopes of this file. This scoping information is used to determine local variables.

Parameters:

Name	Type	Description

Name	Type	Description
`fileData`	String	This is a string of all lines of a file, with each line delimited with the \n character.

`offsetScopes(delta, cursor)`

Source: util/languageParser.js, line 222

When code is added to or removed from a document at a position, scopes need to be offset by that ammount. Example, if there is a scope which starts on line 42, but lines 5-10 are deleted, then the scope starting on 42 will now start on 36. This function updates all scopes so they are still in the correct place.

Parameters:

Name	Type	Description
`delta`	Integer	Number of lines to offset scopes by, either positive for new lines, or negative for deletions.
`cursor`	Cursor	The position of the cursor, so we know where these lines were inserted or deleted relative to.

`tokeniseString(string) → {Array.<String>}`

Source: util/languageParser.js, line 297

This function takes in a string and turns it into an array of string tokens.

Parameters:

Name	Type	Description
`string`	String	String to tokenise

Returns:

- Array of string tokens

Type `Array.<String>`

menuBuilder

menuBuilder

`new menuBuilder()`

Source: util/menuBuilder.js, line 10

Creates a menuBuilder. This class provides utilites which helps a user build a menu bar for their application

Methods

`findMenuDropDown(name) → {*}`

Source: util/menuBuilder.js, line 97

This function creates a new drop down tab on the menu. Example, 'file' will create a new tab called 'file'

Parameters:

Name	Type	Description
`name`		The name of the tab to add to the menyu.

Returns:

Returns a reference to the menu object [name] this object is passed into the registerCallback functions to correctly add functionality to tabs.

Type *

`getMenu()`

Source: util/menuBuilder.js, line 18

This function is simply a getter for this classes MENU object. The MENU object hold all configuration data needed to create the menu at the top of the window.

`registerAppCallback(menu, name, character, function_name)`

Source: util/menuBuilder.js, line 65

This function allows a user to create menu elements to trigger events on the main electron app object.

Parameters:

Name	Type	Description
`menu`		The tab of the menu to add this functionality to.
`name`		The name of this event
`character`		The character to associate this functionality with, Example S for save would map the hotkey ctrl+S to this function.
`function_name`		The textual name of the function to call on the app. Example, 'quit' will call app.quit() to close the application.

`registerFunctionCallback(menu, name, character, function_name)`

Source: util/menuBuilder.js, line 82

This function allows a user to create menu elements to trigger events on the editor.js class.

Parameters:

Name	Type	Description
`menu`		The tab of the menu to add this functionality to.
`name`		The name of this event
`character`		The character to associate this functionality with, Example S for save would map the hotkey ctrl+S to this function.
`function_name`		The textual name of the function to call on the editor.js object. Example, 'save' will call the editor.save() function.

`registerWindowCallback(menu, name, character, function_name)`

Source: util/menuBuilder.js, line 48

This function allows a user to create menu elements to trigger events on the Electron BrowserWindow object defined in app.js.

Parameters:

Name	Type	Description
`menu`		The tab of the menu to add this functionality to.
`name`		The name of this event

Name	Type	Description
`character`		The character to associate this functionality with, Example S for save would map the hotkey ctrl+S to this function.
`function_name`		The textual name of the function to call on the BrowserWindow object. Example, 'toggleDevTools' will open the devTools.

processSpawner

processSpawner

```
new processSpawner()
```

Source: util/processSpawner.js, line 9

Creates a processSpawner. This class allows a user to spawn a native process and acces the stdin and stdout streams spawned from the process.

Methods

```
spawn(cmd, args, stdIN, stdOUT, onCLOSE) → {Promise.<any>}
```

Source: util/processSpawner.js, line 22

This function allows a developer to spawn an arbatrary process on the host pc, and subscribe to various events the spawned process emits.

Parameters:

Name	Type	Description
`cmd`	`String`	The command to execute.
`args`	`Array.<String>`	Command line arguments to pass into the command.
`stdIN`	`function`	Function to execute whenever data is written to stdin of the process.
`stdOUT`	`function`	Function to execute whenever the process sends data to stdOut
`onCLOSE`	`function`	Function to execute when the process terminates.

Returns:

Returns a promise that will resolve once the process has spawned and is running.

Type `Promise.<any>`