Web GUI Handler

Version:	1.0
Date:	March 22, 2020
Author:	Fred Dijkstra
Doc. id:	rtaoqbv

Document history

Version	Date	Author(s)	Description
1.0	March 22, 2020	Fred Dijkstra (fred@oryxmovementsolutions.nl)	Initial version

Table of contents

Introduction

This document describes the *Web GUI Handler*, a JavaScript class that can be used to create a web interface to a Node.js application.

This is done by creating a simple web server by which the user can use a browser, to navigate to the local IP address and interface with the program using a web interface.

The class is implemented in the Node.js module `webGuiHandler.js`.

Setup

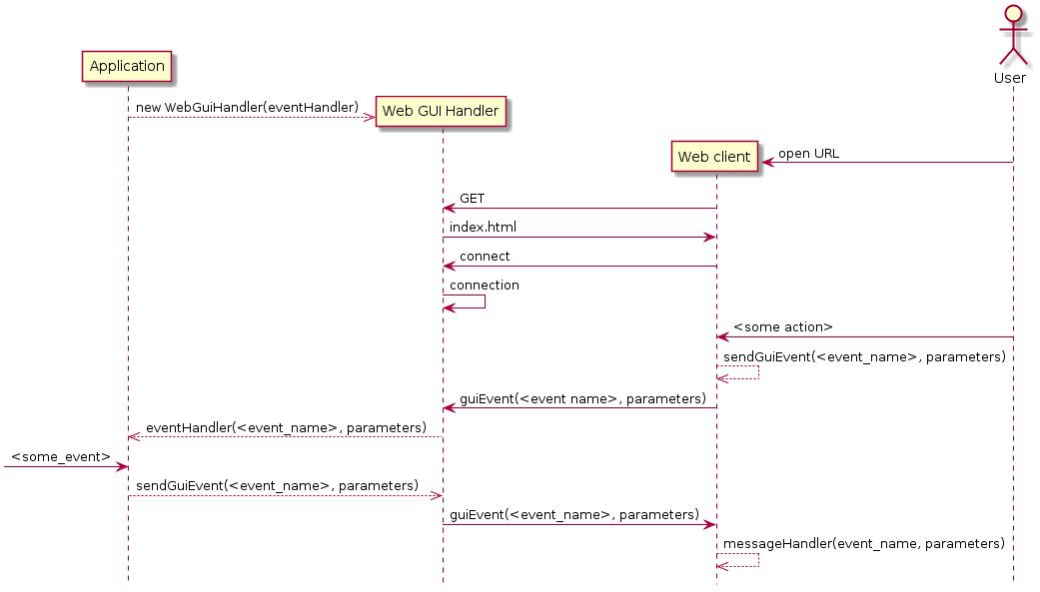
The Application creates a new instance of the Web GUI Handler. When doing so, the Application will specify the reference to the **`eventHandler()`** function. The creation of the Web GUI Handler will start the web server to which a web client can connect. This happens when the user navigates its browser to the URL (IP address and port number) of the web server. As a result the web client sends a GET request to the web server that responds by sending the **`index.html`** file to the web client which will render the file.

In the web page a script is included that opens a web socket to the web server, which will result in a "*connection*" event in the Web GUI Handler.

At some point in time the user will perform some action (e.g. press a button), which will result in a "*guiEvent*" being emitted on the web socket. This event will have two parameters: the indication of the action and the accompanying data. The event will be received by the Web GUI Handler. In the event handler the Web GUI Handler will call the indicated **`eventHandler()`** method with the event indication and the data as arguments.

When some asynchronous event occurs at the side of the Application which needs to result in an update of the web interface, the Application calls the **`sendGuiEvent()`** method of the Web GUI Handler, with the event name and its parameters as arguments. In this method the "*guiEvent*" is emitted with the event name and parameters. At the Web Client this results in the **`eventHandler()`** function being called.

The description above is illustrated in the figure below.



Constants

As will be described in more detail in "Static assets", the web page requires static files such as images, CSS files and JavaScript files. The directories in which these assets are located are listed in the **STATIC_ASSETS_DIRECTORIES** constant.

```
const STATIC_ASSETS_DIRECTORIES =
[
    '/images',
    '/data'
];
```

As will be described also in "Start web server", the web server will listen on a certain port. This port number is stored in the **PORT** constant.

```
const PORT = 8080;
```

Note that the port-number is part of the URL that is filled in the address bar of the web browser, e.g. http://192.168.10.1:8000.

The files generated by the application are stored in the data directory.

```
const APP_DATA_DIR = '/data';
```

To access the files, also the filesystem is required.

```
var fs = require('fs');
```

Constructor

A Web GUI Handler is constructed by supplying the reference to the delegate which is required to have an **`eventHandler()`** callback function implemented.

```
constructor( delegate )
{
    ....
}
```

The supplied **`eventHandler`** argument will be stored in the corresponding property

```
this.delegate = delegate;
```

Next the relevant properties must be initialized to be able to have the Web GUI Handler function as a HTTP server with a web socket.

```
this.express = require('express');
this.app     = this.express();
this.http    = require('http').Server(this.app);
this.io      = require('socket.io')(this.http);
```

Finally, a number of initialization functions are called.

```
staticAssets(this, STATIC_ASSETS_DIRECTORIES);
startWebserver(this);
setHttpGetHandler(this);
setWebsocketHandler(this);
```

These initialization functions are discussed in the next chapter.

Local functions

Overview

Function	Description
staticAssets	Make directories with assets statically available.
startWebServer	Start the web server.
setWebsocketHandler	Set all the handlers for the web socket.
setHttpGetHandler	Set the handler for the HTTP GET request (i.e. router).
sendFileList	Reads the application data directory and sends the files to the connected web client.

Static assets

A loading web page requires static files such as images, CSS files and JavaScript files. To make these *asset files* available, the directories on the server side must be *statically available*. This can be done by calling the **`staticAssets()`** function which takes the reference to the Web GUI Handler and the list of directories as arguments

```
function staticAssets( guiHandler, directories )
{
    ...
}
```

To serve static files, the **`express.static`** built-in middleware function in Express is to be used[1]. The function has an argument specifying the root directory from which to serve static assets.

For multiple directories, the function must be called on each directory.

```
directories.forEach( function( directory )
{
    guiHandler.app.use(guiHandler.express.static( process.cwd() + directory ));
});
```

Express looks up the files relative to the static directory, so the name of the static directory is not part of the URL.

Note that the **`node_modules`** directory is already made static by default.

[1] https://expressjs.com/en/starter/static-files.html

Start web server

The web server can be started by calling the `startWebserver()` function which takes the instantiated Web GUI Handler as the argument.

```
function startWebserver( guiHandler )
{
    ...
};
```

The web server must start listening for clients that want to connect. This is done by calling the `listen()` method of the instantiated HTTP object. This function takes the port number and the callback function as arguments.

```
guiHandler.http.listen(PORT, function ()
{
    ...
});
```

The callback function is called when the web server is listening. For convenience, the IP address of the server is retrieved to be logged onto the console.

```
var ip = require("ip");
```

This can then be logged on the console.

```
console.log("Web server listening on port " + PORT + ", IP address: " + ip );
```

Set Websocket handler

As was described in “Setup”, the web server uses web sockets to serve as an intermediate between the application and the web client. The web socket was instantiated as the server by the initialization of the **`io`** property. Since the web socket uses events, the proper handler must be set. This is done by calling the **`setWebsocketHandler()`** function.

```
function setWebsocketHandler( guiHandler )
{
    ...
}
```

As with normal client-server socket communication, when a client connects a new socket is created for the client-server communication. As such, the first event handler that needs to be set is for the “connection” event which has the created socket as its parameter.

```
guiHandler.io.on('connection', function( socket )
{
    ...
}
```

Inside the event handler, the connection of the client can be logged onto the console for convenience.

```
console.log( "Client connected!" );
```

Also over the new socket the communication will be done using events. As was described in “Setup”, a single event “clientEvent” was defined which has an arbitrary **`data`** object as its parameter. When this event is received, it is ‘forwarded’ to the application by calling the **`executeTransition()`** method of the delegate.

```
socket.on('guiEvent', function( eventName, parameters )
{
    guiHandler.delegate.eventHandler( eventName, parameters );
});
```

For convenience the event for the disconnection is also logged.

```
socket.on('disconnect', function()
{
    console.log('Client disconnected');
});
```

Next to GUI events, also the file access events must be handled in the Web GUI Handler itself.

```
socket.on('getFileList', function()
{
    sendFileList(guiHandler);
});
```

Next to the request for the file list, the Web Client can also request the deletion of a number of files. The list of files is retrieved and sent in the `sendFileList()` function.

```
socket.on('deleteFiles', function(files)
{
    files.forEach( function (filename)
    {
        fs.unlinkSync( process + filename );
    });

    sendFileList();
});
```

Set HTTP GET handler

The handler of the HTTP GET command is set by calling the `setHttpGetHandler()` method.

```
function setHttpGetHandler( guiHandler )
{
    ...
};
```

The method will need to return the `index.html` file located in the current working directory (cwd) of the process.

```
app.get('/', function (req, res)
{
    res.sendFile( process.cwd() + '/index.html' );
}
```

Send file list

To send the list of files in the application data directory to the connected Web Client, the **`sendFiles()`** function must be called.

```
function sendFileList( guiHandler )
{
    ...
};
```

The file list is retrieved using the file system **`readdir()`** method and the list is sent in the callback function.

```
fs.readdir( process.cwd() + APP_DATA_DIR, function (err, files)
{
    guiHandler.io.emit('fileList', files );
});
```

Send GUI event

As described in "Setup", when the Application wants to send a GUI event to the connected Web Client, the **`sendGuiEvent()`** method of the Web GUI Server is called with the event name and parameters as arguments.

```
sendGuiEvent( eventName, parameters )
{
    ...
}
```

The only thing that the Web GUI Handler needs to do is emit a "*guiEvent*" event on the created web socket with the received event name and parameters.

```
this.io.emit( 'guiEvent', eventName, parameters );
```

Client side scripting

To send and receive messages to the web server, a script is created that adds this functionality to a web page.

Inclusions

The website must include the socket.io package.

```
<script src="socket.io/socket.io.js"></script>
```

The functionality that is described in this chapter will be implemented in the `communication.js` file and therefore this script must be included as well.

```
<script src="communication.js"></script>
```

In the following sections the functionality that is added to the web page by including this script is described in more detail.

Web socket creation

The first thing that needs to be done in the script is declaring and creating the web socket. This is done by using the `io()` function that comes with the inclusion of the `socket.io.js` script as described in the previous section.

```
var socket = io();
```

Sending a GUI event

When the user performs a certain action on the web page, this could mean that the server needs to be informed. To do this, the `sendGuiEvent()` function is implemented that takes an event name and parameters as arguments. This function can be called in the code in the web page where the user action is handled.

```
function sendGuiEvent( eventName, parameters )
{
    ...
}
```

The only thing that this function needs to do is to forward the action as an "*guiEvent*" to the server using the web socket.

```
socket.emit("guiEvent", eventName, parameters );
```

Handling server event

As was described in "Setup", as a result of the Application calling the **`sendGuiEvent()`** method of the Web GUI Server, the "*guiEvent*" event is emitted with an event name and data as parameters.

To handle this event, an event handler must be set for the created socket.

```
socket.on( 'guiEvent', function( eventName, parameters )
{
    ...
});
```

If in the web page a function **`eventHandler()`** is defined, this function can now be called after it is checked whether the function is present.

```
if( typeof eventHandler === 'undefined' )
{
    console.log( "WARNING 'eventHandler()' function not defined on page!" )
}
else eventHandler( eventName, parameters );
```