**A JavaScript Logging Library for Productive Programmers**

The most widely used method of logging events for debugging in JavaScript is by calling “console.log(message)” which will show the message in the developer console. To log warnings or errors, developers can also use “console.warn(message)” or “console.error(message)”.

The “console” object is a *host* object which means that it is provided by the JavaScript runtime environment. Because it is a host object there is no specification describing its behavior, and as a result of that, the browser implementations differ, or can even be missing like in Internet Explorer 7.

If most browser already implement a logging functionality out of the box, why should there by a library for the same purpose? In order to extend the functionality offered by the built-in logging mechanism with features such as searching, filtering and formatting of messages.

I'm working on a web based project with the server side written in Java and the client side written in JavaScript. The client for whom we develop this web application couldn't provide us with the server logs in a timely manner. In order to solve the incoming defects as fast as possible, we implemented on the server side a mechanism which in the case of an exception, it would collect information about all the tables involved which along with the Java stacktrace were archived and sent to the browser so they could be downloaded on the fly by the client. He would then send this archive back to us, so that we could investigate and fix the issue.

On the front-end side, when the client submitted an issue which we couldn't reproduce, we had to remote connect to their desktops, install Firebug and start watching its console.

In order to improve this process and make issue reporting easier, we thought to replicate on the client side, the mechanism we implemented on the server side. This was the birth of “*Logger*”, a JavaScript logging library. What makes it stand apart from the rest of the logging libraries is its ability to save the events (logs) on the developer console, on the browser's local storage but also export them to a text file.

A few words about the usage of the library. Including the library is very easy and it requires only one line of code: “<script type="text/javascript" src="Logger.js"></script>”.

There two ways for logging events:

1. Using Logger.log(level, message, location):

Logger.log(Logger.level.Error, "error message", Logger.location.All);

[Tue Mar 04 2014 17:40:16 GMT+0200 (GTB Standard Time)] [ERROR]> error message

Because we used “All” as location, the message will also be saved in the local storage.

localStorage;

Storage {1392801830432\_ERROR: "error message", length; 1}

2. Using specific methods for each event:

Logger.error(message, location), Logger.warn(message, location), Logger.info(message, location)

Logger.error("second error message", Logger.location.All);

[Tue Mar 04 2014 17:40:28 GMT+0200 (GTB Standard Time)] [ERROR]> second error message

Logger.warn("a warning message", Logger.location.All);

[Tue Mar 04 2014 17:40:38 GMT+0200 (GTB Standard Time)] [WARN]> a warning message

Logger.info("an information message", Logger.location.All);

[Tue Mar 04 2014 17:40:48 GMT+0200 (GTB Standard Time)] [INFO]> an information message

Each event is saved in the browser's local storage as a key-value pair. The key is made up of the “timestamp\_level”, and the value contains the event message.

To list all of the events we can use “Logger.getEvents(level)”:

Logger.getEvents(Logger.level.All);

[Tue Mar 04 2014 17:40:16 GMT+0200 (GTB Standard Time)] [ERROR]> error message

[Tue Mar 04 2014 17:40:28 GMT+0200 (GTB Standard Time)] [ERROR]> second error message

[Tue Mar 04 2014 17:40:38 GMT+0200 (GTB Standard Time)] [WARN]> a warning message

[Tue Mar 04 2014 17:40:48 GMT+0200 (GTB Standard Time)] [INFO]> an information message

I'll continue with presenting the functionality which, in my opinion, will make programmers more productive: exporting all events to a text file. By exporting to a text file, programmers can have faster access to the events which occurred during the application's run time.

Exporting the events can be done by calling “Logger.exportLog()” or by setting a callback function in the event of an error occurring in the application:

“Logger.onError(exportLog, suppressErrorAlerts, errorCallback)”

If “exportLog” is set on **true** then the error which occurred and also all of the events present in the local storage will be exported to a text file. If ”suppressErrorAlerts” is set on **true** then the error will not be written also to the brwser's console. ”errorCallback” is the callback function which will get called when the error occurs.

Logger.onError(true, true, function(errorMsg, url, lineNumber) {

var errorMessage = errorMsg + ' at line ' + lineNumber + ' in ' + url,

response = '';

Logger.error(errorMessage, Logger.location.LocalStorage);

response = confirm("An error occurred. Would you like to export the log events to a text file?");

if (response === true) {

Logger.exportLog();

}

return true;//suppress errors on console

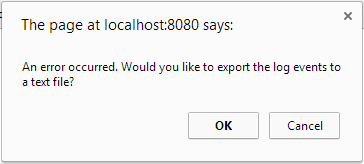
});

The following code sequence will display a dialog message in the event of a JavaScript error.

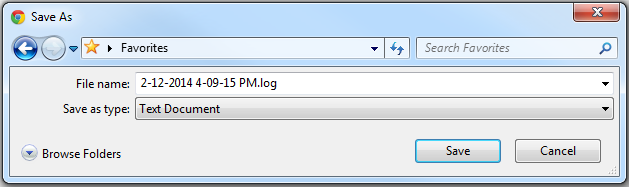
We can test if this code works by throwing an error when clicking a button:

”<button onclick="throw new Error('eroare.')">Buton</button>”

When clicking the button we will get the following messages



Pressing the “OK” button will show the browser's save dialog or save the file directly, depending on the browser settings. The filename is made up of the current date and time.



If we open up the file we will see the logged events and also the JavaScript error.

[Tue Mar 04 2014 17:40:16 GMT+0200 (GTB Standard Time)] [ERROR]> error message

[Tue Mar 04 2014 17:40:28 GMT+0200 (GTB Standard Time)] [ERROR]> second error message

[Tue Mar 04 2014 17:40:38 GMT+0200 (GTB Standard Time)] [WARN]> a warning message

[Tue Mar 04 2014 17:40:48 GMT+0200 (GTB Standard Time)] [INFO]> an information message

[Tue Mar 04 2014 17:48:40 GMT+0200 (GTB Standard Time)] [ERROR]> Uncaught Error: JS ERROR. at line 19 in http://localhost:8080/logger/main.html

We can see the error occurred in file “main.html” at line 19, which is where we created the button whose click event throws a JavaScript error:

<button onclick="throw new Error('JS ERROR.')">Buton</button>

The library uses the “*Revealing module*” pattern to expose the functions which can be called by the developers.

The easiest way to write data from the local storage to a file would have been to use a third party JavaScript library called “FileSaver.js” which uses the *FileSystem* API or when this is unavailable it provides an alternative implementation. To avoid using any third party libraries I found a solution using a *Blob* object:

saveToDisk: function(content, filename) {

var a = document.createElement('a'),

blob = new Blob([content], {'type' : 'application/octet-stream'});

a.href = window.URL.createObjectURL(blob);

a.download = filename;

a.click();

}

I created an anchor element and attached to its *href* attribute a URL object created from a blob. A Blob is an immutable object which contains raw data. The name of the file comprised from the current date and time is associated to the *download* attribute. The *click()* function is then called on the anchor element which will show the browser's save dialog or save the file directly to disk according to the browser settings.

In order to intercept the JavaScript errors I created a variable called *initialWindowErrorHandler* which contains a reference to window.onerror event handler. Initially, window.onerror will write all error messages to the console. If the *errorCallback* parameter is not undefined in the *onError()* function, then the function to which the *errorCallback* parameter points to will get called on each JavaScript error.

onError: function(exportLog, suppressErrorAlerts, errorCallback) {

var me = this,

errorMessage = '';

if (typeof errorCallback !== 'undefined') {

window.onerror = errorCallback;

} else if (exportLog && typeof errorCallback === 'undefined') {

window.onerror = function (errorMsg, url, lineNumber) {

errorMessage = errorMsg + ' at line: ' + lineNumber + ' in ' + url;

me.log(me.level.Error, errorMessage, me.location.All);

me.exportLog();

return suppressErrorAlerts;

}

} else {

me.resetWindowErrorHandler();

}

}

At any moment we can revert to the original window.onerror behavior by calling *Logger.resetWindowErrorHandler()*.

This is the initial version of the Logger library, without any advanced features. In the future it could be improved by adding the possibility to set a maximum number of recorded events, the ability to delete events by their event type, filtering the exported events by type, creating a graphical interface to browse through the existing records and generate statistics to determine the stability of the application in a given time frame and also the ability to customize the generated filename according to the programmer's needs.

If you have any questions or comments I can be reached at the following email address: [bogdan.cornianu@3pillarglobal.com](mailto:bogdan.cornianu@3pillarglobal.com)

References:

Mozilla Developer Network:

Create Object URL: <https://developer.mozilla.org/en-US/docs/Web/API/URL.createObjectURL>

Blob: <https://developer.mozilla.org/en-US/docs/Web/API/Blob>

File System API: <https://developer.mozilla.org/en-US/docs/WebGuide/API/File_System/Introduction>

Window.onerror: <https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers.onerror>

Github

FileSaver.js: <https://github.com/eligrey/FileSaver.js>

JSFiddle

Save to disk example: <http://jsfiddle.net/koldev/cW7W5/>

Revealing Module Pattern: <http://carldanley.com/js-revealing-module-pattern/>