Log4js - Users Guide

1.0

by Stephan Strittmatter

NOTICE: This document is still under construction and improvements and suggestions are welcome.

This is the users guide for Log4js. Use this guide to learn how to use Log4js efficient in your projects.

Users Guide of Log4js - The Logging API for JavaScript

Table of contents

1. Introduction

Log4js is a very small but usefull JavaScript library to log events in your scripts. Often it is not useful to use `alert('debug message');` spreaded through your code. On the other hand a debugger like venkman can help you also only if you debug on your own computer and when you can use FireFox as browser.

But the most problems will occure if you publish your web application. Probably you have first a team of testers which tests your software, but they will not have experiences with debugging JavaScript.

Log4js will solve these problems and probably some more. You simply add Log4js to your scripts and the library loggs all your events like you configured. The installation and configuration will be described in the next sections.

Using *Log4js* you need not to modify your complete scripts if you are going productive as you have if you use the alert-dialogs. You just configure if you still need logs and how they should be logged on productive system. Nothing more.

2. Download and Installation

To use *Log4js* you have to download the library first from the download section (http://developer.berlios.de/project/showfiles.php?group_id=5368) of the project. The library is available in two different archives. It is up to you, if you prefere zip archive or tar.gz archive. The content is the same.

Preserve Directory Structure

Extract the archive to a directory of your choice, but preserve the directory structure of the archive!

After extracting the archive, you will find following directory structure:

- log4js-{version}
  - docs
    - api
      - index.html *(the API documentation)*
    - users-guide.pdf *(this users guide)*
  - examples
  - lib
    - log4js-lib.js *(compressed version of Log4js library)*
  - src
    - js
      - log4js.js *(uncompressed version of Log4js library)*

Two versions of Log4js

There are two versions of Log4js. One file in the lib-directory called `log4js-lib.js`, which is a compressed version of Log4js. This can be used in productive systems to reduce bandwidth.
The `log4js.js` file in the `src/js` directory is the origin log4js library. Uncompressed and much more readable. Use this, if you want to understand, what log4js is doing internal.

3. Usage

Create a new JavaScript project or take one of yours existing projects and copy the `log4js.js` file to it. No more files are required for logging.

After adding the `log4js.js` file to your project, you have to include it in your scripts using following line.
`<script src="js/log4js.js" type="text/javascript"></script>`
See the following example HTML file:

```
<html>
<head>
 <title>Log4js Example</title>

 <script src="js/log4js.js" type="text/javascript"></script>
</head>
<body>

</body>
</html>
```

Now the logger has to be initialized before it can be used. Log4js provides a method for this, where you pass a category for your logs and you get an instance of a logger:
`var myLogger = new Log4js.getLogger("myCategory");`

> **Note:**
> The loggers are cached and if you request a second time a logger for the same category the old one is returned.

Now the logger can be used like following code snippets are showing:

```
myLogger.info('an info');
myLogger.warn('a warning');
myLogger.error('an error');
```

> **Warning:**
> If you test these snippets nothing will happen and you will probaby wounder why. The problem is that the logger is not configured currently. Logging is by default disabled that the behaviour of your scripts is not different to the scripts before using *Log4js*.

4. Configuration

Currently there are already logs in the scripts but the events are not shown. The logger has to be configured for that now.

4.1. Define Logging Level

First the logging level has to be defined. There are several levels for logging. They are the same as the JavaTM implementation log4j is using.

Log4js.Level	Description
OFF	nothing is logged
FATAL	fatal errors are logged
ERROR	errors are logged
WARN	warnings are logged
INFO	infos are logged
DEBUG	debug infos are logged
TRACE	traces are logged
ALL	everything is logged

Table 1: Log4js Log Levels

The levels are cumulative. If you for example set the logging level to `WARN` all warnings, errors and fatals are logged:

```
myLogger.setLevel(Log4js.Level.WARN);
```

If everything should be logged, use:

```
myLogger.setLevel(Log4js.Level.ALL);
```

4.2. Define Appender

After defining the log level, we have to set up an appender. An appender is required to define, how the logging events are processed. Where they should be shown or stored etc.

For the first example the `ConsoleAppender` will be used. The other apenders are described later. The `ConsoleAppender` can be used in two modes:

- inline console within the current page
- separate console window

The inline console is hidden by default and can be activated by pressing `ALT+D`. The `ConsoleAppender` can be added by following line as inline console:

```
myLogger.addAppender( new ConsoleAppender(true) );
```

Now the logger is ready to be used. Try again some calls like

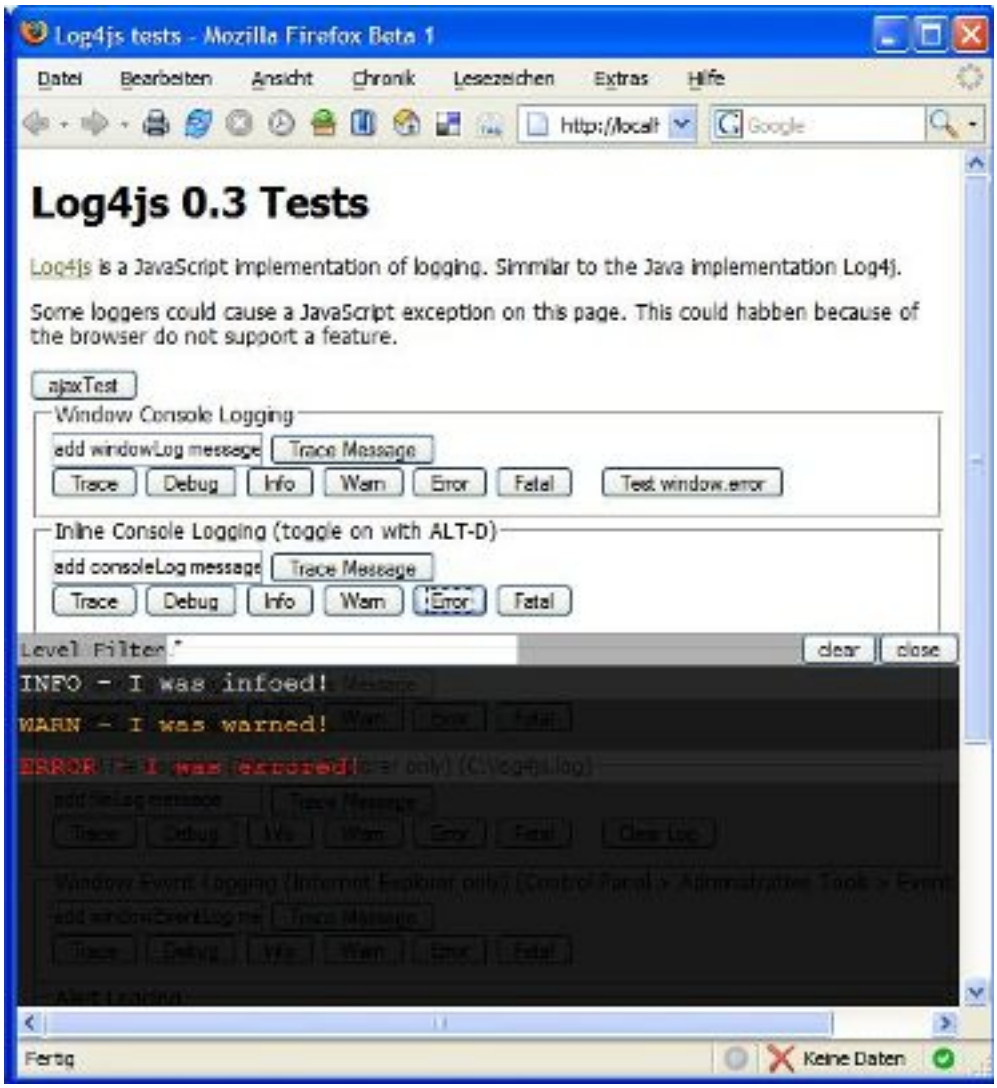
```
myLogger.info('an info');
myLogger.warn('a warning');
myLogger.error('an error');
```

The `example.html` should look like following lines:

```
<html>
<head>
 <title>Log4js Example</title>

 <script src="js/log4js.js" type="text/javascript"></script>
 <script><!--
     myLogger.setLevel(Log4js.Level.ALL);
     myLogger.addAppender(new ConsoleAppender(true);
 //--></script>
</head>
<body>
 <script><!--
    myLogger.info('an info');
    myLogger.warn('a warning');
    myLogger.error('an error');
 //--></script>
</body>
</html>
```

If you open the console by pressing ALT + D (in Firefox > 2.0 press SHIFT+ALT+D you will find the logged events like in the image below.



Congratulations, now you are ready to use *Log4js* in your project! The further sections you need only to read if you require more details how to configure and use *Log4js*.

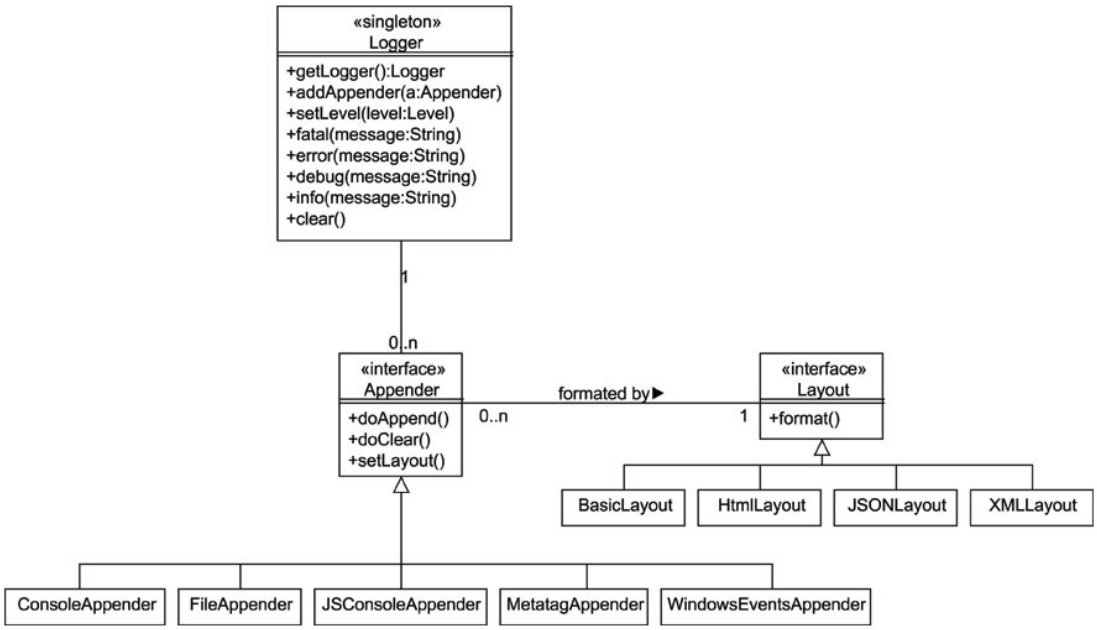
5. Advanced

Log4js is a flexible API where you can define several different `Appenders` as it is required in your project. As the `addAppender`-method implies, it is possible to set more than one appender for a logger. It is possible to set for example an appender which logst to the shown console and one which sends the logging events via AJAX' XmlHttpRequest to your server.

The Layout of the logged messages is also configurable, but this depends also on the appender which is taken. For example if you are using the AjaxAppender, you can send the events JSON or XML formatted to process them on the server. For more details have a look at the API documentation (http://log4js.berlios.de/apidocs/index.html) and the next sections.

5.1. Class Diagram

The class diagram below, will provide a short overview about the logger, its appenders and the layout classes and their dependencies.



5.2. Usage of AjaxAppender

Log4js supports also sending the logging events via `XmlHttpRequest` asynchronous to the server. This appender give you the most flexibility in productive systems to log

JavaScript methods. As you have no access to the browsers in live systems, you can enable the AjaxAppender to send the logs asynchronous to the server. On the server the logs can be processed for example using a simple JSP file like in the examples to store them via log4js.

The AjaxAppender requires an URL to send the logs to. Because of security it can be only a relative URL on the same server:

```
   var ajaxLog = new Log4js.getLogger("ajaxTest");
   ajaxLog.setLevel(Log4js.Level.ALL);
   var ajaxAppender = new AjaxAppender("./log4j.jsp");
   ajaxAppender.setThreshold(5);
   ajaxLog.addAppender(ajaxAppender);
```

In the source example, the AjaxAppender sends the logging events to the relative URL `./log4j.jsp`. Defining a threshold, the appender collects the events until this threshold is reached to send then all the collected events in one request.

By default the AjaxAppender uses the XMLLayout, which transforms the logging events to XML. But it is also possible to set another layout like JSONLayout or HTMLLayout if you prefer.

The XMLLayout uses a format like following example:

```
  <log4js:eventSet version="0.3"
xmlns:log4js="http://log4js.berlios.de/log4js/">
      <log4js:event logger="ajaxTest" level="TRACE"
         client="Mozilla/5.0 (Windows; U; Windows NT 5.1; de;
rv:1.8.1b1) Gecko/20060710 Firefox/2.0b1"
         referer="http://localhost:8080/log4js-example-0.3/"
         timestamp="2006-08-02T17:40:32+02:00">
              <log4js:message><![CDATA[I was traced!]]></log4js:message>
      </log4js:event>
      <log4js:event logger="ajaxTest" level="DEBUG"
         client="Mozilla/5.0 (Windows; U; Windows NT 5.1; de;
rv:1.8.1b1) Gecko/20060710 Firefox/2.0b1"
         referer="http://localhost:8080/log4js-example/"
         timestamp="2006-08-02T17:40:32+02:00">
              <log4js:message><![CDATA[I was
debuged!]]></log4js:message>
      </log4js:event>
      <log4js:event logger="ajaxTest" level="INFO"
         client="Mozilla/5.0 (Windows; U; Windows NT 5.1; de;
rv:1.8.1b1) Gecko/20060710 Firefox/2.0b1"
         referer="http://log4js.berlios.de/log4js/"
         timestamp="2006-08-02T17:40:33+02:00">
              <log4js:message><![CDATA[I was
informed!]]></log4js:message>
      </log4js:event>
  </log4js:eventSet>
```

Note:

This format is very similar to the XML format of log4j. The log4js.xsd (/log4js/log4js.xsd) defines the `eventSet` and

the server `response`.

In contrast to the XML format of log4j *Log4js* provides also the id of the client browser and the referer URL to identify the sources. On the other hand some advanced features like `NDC` are not supported currently in *Log4js*.

5.3. Performance

To improve the performance of your software, it is a good idea to surround your logging statements with appropriate isXXXEnabled() methods to evaluate before logging if the required logging level is enabled:

```
if( myLogger.isDebugEnabled()) {
      myLogger.debug(...);
}
```

If debug is disabled, it is much faster to evaluate only if it is disabled than constructing a possible complex message before finding debug level disabled. But this makes only sense when the messages are a little bit more complex than just log a simple string. Otherwise the sources are blown up too much with condition statements which is not much readable.

5.4. Create new Appenders and Layouts

There is already a small set of appenders and layouting classes, but probably you need another feature, which is not implemented in *Log4js*. For this it is possible to enhance the API for special features.

Share your Extentions

If it is a feature which is usable also by other developers, please share your extensions by providing patches. See also the project development (/project/index.html) section.

6. License

7. References

- Log4js Website (http://log4js.berlios.de)
- Log4js WIKI (http://scratchpad.wikia.com/wiki/Log4js)
- Apache Logging Website (http://logging.apache.org)