

Tracing

API Reference

© 2018 All rights reserved by Metrological

This document contains information which is proprietary and confidential to Metrological. It is provided with the expressed understanding that the recipient will not divulge its content to other parties or otherwise misappropriate the information contained herein. This information is furnished for guidance; specifications and availability of goods mentioned in it are subject to change without notice. No part of this publication may be reproduced, stored in a database, retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the written prior permission of Metrological.

History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Description** |
| 0.1 | 17-04-2018 | P. Wielders | Initial version |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 4

1.1 Scope 4

1.2 Case sensitivity 4

1.3 Acronyms, Abbreviations and Terms 4

1.4 Standards 4

1.5 References 5

1.6 Open Issues 5

1.7 Limitations 5

2. Rationale behind tracing 6

2.1 Definition, logging versus tracing 6

2.2 Tracing in practice 6

2.3 Configuration 6

2.4 Flow of trace information. 6

3. TraceControl Plugin 8

3.1 Configuration 8

3.2 Application Programming Interface (API) 8

3.2.1 General information 8

3.2.2 State changes 8

3.3 Events 8

3.4 JSON definitions 9

3.4.1 General information (tracing\_info) 9

# Introduction

## Scope

This document describes the Plugin Tracing API interface. This plugin can be configured to be loaded and executed to target trace information for categories or modules (or any given combination) to be turned on and off runtime in the WPEFramework. For details on the WPEFramework API, refer to: [WPEF]

## Case sensitivity

All identifiers on the interface described here are case-sensitive. E.g. an id known in the plugin as 'C0FFEE' is not the same as 'c0ffee'.

All keywords, entities, properties, relations and actions should be treated as case-sensitive.

## Acronyms, Abbreviations and Terms

The next list provides an overview of acronyms and abbreviations used in this document and their definitions.

|  |  |
| --- | --- |
| **Acronym** | **Definitions** |
| API | Application Programming Interface |
| JSON | JavaScript Object Notation |
| UTC | Coordinated Universal Time |

Below terms are listed with their definitions, as used in this document.

|  |  |
| --- | --- |
| **Term** | **Definitions** |
| Callsign | The callsign is the name given to an instance of a plugin. One plugin can be instantiated multiple times, but each instance the instance name, callsign, must be unique. |
| Proxy | An object in one process space representing the “real” object in another process space. The Proxy takes care of marshalling the parameters. |
| Stub | An object in the process space that contains the actual object. The stub takes care of un-marshalling the request from the Proxy and executes the call, on behave of the Proxy object, on the real object |

## Standards

Date time formats between the systems shall be in UTC time and W3C (ISO 8601 profile) formatting [ISO 8601], e.g.: 2004-11-05T13:15:30Z. This way time discontinuities can be avoided due to daylight savings. Note that all interfacing systems must decode/encode the date time to the correct local time.

Languages used in the WPEFramework will be conform [ISO 639-1] using two letter language codes. If WPEFramework encounters a language code it does not recognize, it will use ‘xx’ instead. For a list of available two letter ISO language codes, please visit:  
<http://www.loc.gov/standards/iso639-2/php/code_list.php>

## References

This section lists the references made in this document:

|  |  |
| --- | --- |
| [WPEF] | WPEFramework API Reference  <https://github.com/WebPlatformForEmbedded/WPEFramework> |
| [HTTP] | Hypertext Transfer Protocol  <http://www.w3.org/Protocols> |
| [ISO 8601] | Date and time format  http://www.iso.org/iso/date\_and\_time\_format |
| [ISO-3166] | Country code specification  <http://www.iso.org/iso/country_codes.htm> |
| [ISO-639-1] | Language code specification (Alpha-2 code)  <http://www.loc.gov/standards/iso639-2/php/code_list.php> |
| [JSON] | JavaScript Object Notation  [http://www.json.org](http://www.json.org/) |
| [URLENC] | URL Encoding  <http://www.w3schools.com/tags/ref_urlencode.asp> |

## Open Issues

This is a list of open issues that needs to be resolved:

* This document is still a work in progress.

## Limitations

The information described in this document is preliminary and subject to change in the future.

Legend:



**Be aware of:** implementation choice is needed or side-effect needs to be handled.



**Implementation advice:** Guide line for implementation mostly related to performance.

# Rationale behind tracing

## Definition, logging versus tracing

The WPEFramework makes a distinction between logging and tracing. The distinction can be found in the targeted audience.   
Logging in WPEFramework context is targeted for Testers and Operators. Logging messages should be high levels and descriptive in the context of the end-to-end system.

Tracing in WPEFramework context is targeted for Developers. Tracing message might contain details about the internals of the software (state) and are useful in the context of the developed software. This document focusses on Tracing only!!!

## Tracing in practice

Tracing is by definition intrusive to the execution of the software. For this reason, all tracing is compiled into the binaries but by default turned off. This way the application has the best performance.

In the unlikely case of an emergency, it is good to have tooling in place with which the developer can “see” what is going on in the code he created. In the case of the emergency, it would be preferable to limit the intrusiveness of the tracing as much as possible.

The WPEFramework has realized this by allowing a very fine-grained selection possibility of tracing. Traces are characterized by category and plugin. Trace selection, runtime on or off, can be triggered, based on these two properties. This way the developer can select to turn tracing on for a certain plugin and only a certain category.

Category/Plugin properties are created compile time. Traces are active at the moment the plugin is loaded. The plugin does not necessarily need to be in an activated state.

## Configuration

Default, a trace is disabled and the impact is limited to comparing a single boolean (no dereference required) with true or false. Sometimes it is useful to actually have traces from the startup and thus a developer might require the tracing to be enabled by default.

This can be achieved by adding a trace configuration line to the main WPEFramework config. See [WPEF] for details.

The default state of a category is by default disabled or is set by the configuration. During the lifetime of the application, the state of the trace line can still be changed, using the RESTFull API described in §3.2 Application Programming Interface (API).

## Flow of trace information.

The WPEFramework potentially comprises of multiple processes. All these processes might generate tracing. If all these processes start tracing at the same time, the traces might be intertwined due to the multitasking/multithreaded character of the operating system. This could potentially render the tracoutput useless for analyzing.

An option could be to implement a system wide lock to serialize the tracing output. However this solution would implicitly synchronize processes and in case of a crash of a process requires complex recovery code to release the lock.

To avoid this complexity, the WPEFramework employs the use of a memory mapped file per process. All trace lines will be written to this process wide trace buffer. This trace buffer is implemented as a cyclic buffer, with the option to override, if there is no more storage space. An overwrite will be signaled but trace lines can be lost, if the trace buffer is not read fast enough. However, if information is lost, it will be signaled.

One process is responsible for reading and reconstructing the traces from the available tracebuffers and feed them to the proper output. This re-assembling of the trace information is done by the TraceControl and the TraceControl plugin has the ability to output the re-assembled traces to console, syslog or send it out over a UDP socket connection.

# TraceControl Plugin

## Configuration

|  |  |
| --- | --- |
| callsign | [string] the instance name for the plugin e.g. Tracing. Default: TraceControl. |
| classname | [string] TraceControl. |
| locator | [string] libTraceControl.so |
| autostart | [bool] should the browser plugin be instantiated at the moment the WPEFramework is starts up. |
| configuration | [JSON] JSON object specifying the exact configuration for this plugin. See the next paragraph for details. |

Configuration of the TraceControl Plugin:

|  |  |
| --- | --- |
| console | [bool] enable trace output in console (stdout/stderr) |
| syslog | [bool] enable trace output directed into syslog-file |

## Application Programming Interface (API)

### General information

Using this method, actual trace status information can be retrieved from the WPEFramework.

|  |  |
| --- | --- |
| Request: | GET /Service/TraceControl |
| Success: | HTTP/1.1 200 OK  { tracing\_info } |

|  |  |
| --- | --- |
| Request: | GET /Service/TraceControl/<ModuleName> |
| Success: | HTTP/1.1 200 OK  { tracing\_info } |

### State changes

Using these methods, the browser state (suspend/resume, hidden/visible) can be toggled.

|  |  |
| --- | --- |
| Request: | PUT /Service/TraceControl/<ModuleName>/<Category>/<on|off> |
| Success: | HTTP/1.1 200 OK |

## Events

Events are autonomous events, triggered by the internals of the plugin. These events will be broadcasted as JSON to all the connected web socket connections that where opened to this plugin.

## JSON definitions

### General information (tracing\_info)

|  |  |
| --- | --- |
| url | [string] The currently set/about to be loaded URL in the browser. |
| fps | [uint32] the number of frames per second the browser is painting on the screen. |
| hidden | [bool] true in case the browser is not visible (hidden) on screen. |
| suspended | [bool] true, in case the client is in a suspended mode. All system critical resources have been relinquished. |