

fswatch

1.11.1

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Changelog . . . . .	1
1.3	Available Bindings . . . . .	1
1.4	libtool's versioning scheme . . . . .	2
1.5	The C and the C++ API . . . . .	2
1.6	Thread Safety . . . . .	2
1.7	C++11 . . . . .	3
1.8	Reporting Bugs and Suggestions . . . . .	3
<b>2</b>	<b>C++ API</b>	<b>5</b>
2.1	Usage . . . . .	5
2.2	Example . . . . .	6
<b>3</b>	<b>C API</b>	<b>7</b>
3.1	Translating the C++ API to C . . . . .	7
3.2	Thread Safety . . . . .	7
3.3	Library Initialization . . . . .	7
3.4	Status Codes and Errors . . . . .	8
3.5	Example . . . . .	8

<b>4</b>	<b>History</b>	<b>9</b>
4.1	9:0:0 . . . . .	9
4.2	8:0:2 . . . . .	9
4.3	8:0:2 . . . . .	9
4.4	8:0:2 . . . . .	9
4.5	5:0:2 . . . . .	10
4.6	4:0:1 . . . . .	10
4.7	3:0:0 . . . . .	10
<b>5</b>	<b>Path Filtering</b>	<b>11</b>
<b>6</b>	<b>Namespace Index</b>	<b>13</b>
6.1	Namespace List . . . . .	13
<b>7</b>	<b>Hierarchical Index</b>	<b>15</b>
7.1	Class Hierarchy . . . . .	15
<b>8</b>	<b>Class Index</b>	<b>17</b>
8.1	Class List . . . . .	17
<b>9</b>	<b>File Index</b>	<b>19</b>
9.1	File List . . . . .	19
<b>10</b>	<b>Namespace Documentation</b>	<b>21</b>
10.1	fsw Namespace Reference . . . . .	21
10.1.1	Detailed Description . . . . .	23
10.1.2	Typedef Documentation . . . . .	23
10.1.2.1	FSW_EVENT_CALLBACK . . . . .	23
10.1.2.2	fsw_hash_map . . . . .	24
10.1.2.3	fsw_hash_set . . . . .	24
10.1.2.4	monitor_filter . . . . .	24
10.1.3	Function Documentation . . . . .	24
10.1.3.1	get_directory_children() . . . . .	24
10.1.3.2	lstat_path() . . . . .	25

10.1.3.3	<a href="#">operator&lt;&lt;()</a>	25
10.1.3.4	<a href="#">read_link_path()</a>	26
10.1.3.5	<a href="#">stat_path()</a>	26
10.2	<a href="#">fsw::string_utils Namespace Reference</a>	26
10.2.1	<a href="#">Detailed Description</a>	27
10.2.2	<a href="#">Function Documentation</a>	27
10.2.2.1	<a href="#">string_from_format()</a>	27
10.2.2.2	<a href="#">vstring_from_format()</a>	27
10.3	<a href="#">fsw::win_paths Namespace Reference</a>	27
10.3.1	<a href="#">Detailed Description</a>	28
10.3.2	<a href="#">Function Documentation</a>	28
10.3.2.1	<a href="#">posix_to_win_w()</a>	28
10.3.2.2	<a href="#">win_w_to_posix()</a>	28
10.4	<a href="#">fsw::win_strings Namespace Reference</a>	29
10.4.1	<a href="#">Detailed Description</a>	29
10.4.2	<a href="#">Function Documentation</a>	29
10.4.2.1	<a href="#">wstring_to_string()</a> [1/2]	29
10.4.2.2	<a href="#">wstring_to_string()</a> [2/2]	29
<b>11</b>	<b><a href="#">Class Documentation</a></b>	<b>31</b>
11.1	<a href="#">fsw::compiled_monitor_filter Struct Reference</a>	31
11.2	<a href="#">fsw::directory_change_event Class Reference</a>	31
11.2.1	<a href="#">Detailed Description</a>	32
11.3	<a href="#">fsw::event Class Reference</a>	32
11.3.1	<a href="#">Detailed Description</a>	33
11.3.2	<a href="#">Constructor &amp; Destructor Documentation</a>	33
11.3.2.1	<a href="#">event()</a>	33
11.3.2.2	<a href="#">~event()</a>	33
11.3.3	<a href="#">Member Function Documentation</a>	33
11.3.3.1	<a href="#">get_event_flag_by_name()</a>	33
11.3.3.2	<a href="#">get_event_flag_name()</a>	34

11.3.3.3	<a href="#">get_flags()</a>	34
11.3.3.4	<a href="#">get_path()</a>	35
11.3.3.5	<a href="#">get_time()</a>	35
11.4	<a href="#">fsw::fen_monitor Class Reference</a>	35
11.4.1	<a href="#">Detailed Description</a>	36
11.4.2	<a href="#">Member Function Documentation</a>	36
11.4.2.1	<a href="#">run()</a>	36
11.5	<a href="#">fsw::FSEventFlagType Struct Reference</a>	36
11.6	<a href="#">fsw::fsevents_monitor Class Reference</a>	37
11.6.1	<a href="#">Detailed Description</a>	37
11.6.2	<a href="#">Member Function Documentation</a>	37
11.6.2.1	<a href="#">run()</a>	38
11.7	<a href="#">fsw_callback_context Struct Reference</a>	38
11.8	<a href="#">fsw_cevent Struct Reference</a>	38
11.8.1	<a href="#">Detailed Description</a>	39
11.9	<a href="#">fsw_cmonitor_filter Struct Reference</a>	39
11.10	<a href="#">fsw_event_type_filter Struct Reference</a>	39
11.10.1	<a href="#">Detailed Description</a>	39
11.11	<a href="#">FSW_SESSION Struct Reference</a>	40
11.12	<a href="#">fsw::inotify_monitor Class Reference</a>	40
11.12.1	<a href="#">Detailed Description</a>	41
11.12.2	<a href="#">Member Function Documentation</a>	41
11.12.2.1	<a href="#">run()</a>	41
11.13	<a href="#">fsw::inotify_monitor_impl Struct Reference</a>	41
11.14	<a href="#">fsw::kqueue_monitor Class Reference</a>	42
11.14.1	<a href="#">Detailed Description</a>	42
11.14.2	<a href="#">Member Function Documentation</a>	42
11.14.2.1	<a href="#">run()</a>	42
11.15	<a href="#">fsw::libfsw_exception Class Reference</a>	43
11.15.1	<a href="#">Detailed Description</a>	43

11.15.2 Constructor & Destructor Documentation . . . . .	43
11.15.2.1 libfsw_exception() . . . . .	43
11.15.3 Member Function Documentation . . . . .	44
11.15.3.1 error_code() . . . . .	44
11.15.3.2 what() . . . . .	44
11.16fsw::monitor Class Reference . . . . .	44
11.16.1 Detailed Description . . . . .	47
11.16.2 Constructor & Destructor Documentation . . . . .	48
11.16.2.1 monitor() . . . . .	48
11.16.2.2 ~monitor() . . . . .	48
11.16.3 Member Function Documentation . . . . .	49
11.16.3.1 accept_event_type() . . . . .	49
11.16.3.2 accept_path() [1/2] . . . . .	49
11.16.3.3 accept_path() [2/2] . . . . .	50
11.16.3.4 add_event_type_filter() . . . . .	50
11.16.3.5 add_filter() . . . . .	50
11.16.3.6 filter_flags() . . . . .	51
11.16.3.7 get_context() . . . . .	51
11.16.3.8 get_property() . . . . .	51
11.16.3.9 is_running() . . . . .	52
11.16.3.10notify_events() . . . . .	52
11.16.3.11notify_overflow() . . . . .	52
11.16.3.12on_stop() . . . . .	53
11.16.3.13run() . . . . .	53
11.16.3.14set_allow_overflow() . . . . .	53
11.16.3.15set_context() . . . . .	54
11.16.3.16set_directory_only() . . . . .	54
11.16.3.17set_event_type_filters() . . . . .	55
11.16.3.18set_filters() . . . . .	55
11.16.3.19set_fire_idle_event() . . . . .	55

11.16.3.20	<a href="#">set_follow_symlinks()</a>	55
11.16.3.21	<a href="#">set_latency()</a>	56
11.16.3.22	<a href="#">set_properties()</a>	56
11.16.3.23	<a href="#">set_property()</a>	57
11.16.3.24	<a href="#">set_recursive()</a>	57
11.16.3.25	<a href="#">set_watch_access()</a>	57
11.16.3.26	<a href="#">start()</a>	58
11.16.3.27	<a href="#">stop()</a>	58
11.16.4	<a href="#">Member Data Documentation</a>	58
11.16.4.1	<a href="#">callback</a>	59
11.16.4.2	<a href="#">fire_idle_event</a>	59
11.16.4.3	<a href="#">paths</a>	59
11.16.4.4	<a href="#">properties</a>	59
11.17	<a href="#">fsw::monitor_factory Class Reference</a>	60
11.17.1	<a href="#">Detailed Description</a>	60
11.17.2	<a href="#">Member Function Documentation</a>	61
11.17.2.1	<a href="#">create_monitor()</a> [1/2]	61
11.17.2.2	<a href="#">create_monitor()</a> [2/2]	61
11.17.2.3	<a href="#">exists_type()</a> [1/2]	62
11.17.2.4	<a href="#">exists_type()</a> [2/2]	62
11.17.2.5	<a href="#">get_types()</a>	63
11.17.2.6	<a href="#">register_creator()</a>	63
11.17.2.7	<a href="#">register_creator_by_type()</a>	63
11.18	<a href="#">fsw::monitor_filter Struct Reference</a>	64
11.18.1	<a href="#">Detailed Description</a>	64
11.18.2	<a href="#">Member Function Documentation</a>	64
11.18.2.1	<a href="#">read_from_file()</a>	65
11.18.3	<a href="#">Member Data Documentation</a>	65
11.18.3.1	<a href="#">extended</a>	65
11.18.3.2	<a href="#">text</a>	66



11.19fsw::monitor_registrant< M > Class Template Reference . . . . .	66
11.19.1 Detailed Description . . . . .	66
11.19.2 Constructor & Destructor Documentation . . . . .	66
11.19.2.1 monitor_registrant() . . . . .	66
11.20fsw::poll_monitor Class Reference . . . . .	67
11.20.1 Detailed Description . . . . .	67
11.20.2 Member Function Documentation . . . . .	68
11.20.2.1 run() . . . . .	68
11.21fsw::poll_monitor::poll_monitor_data Struct Reference . . . . .	68
11.22fsw::win_error_message Class Reference . . . . .	68
11.22.1 Detailed Description . . . . .	69
11.22.2 Constructor & Destructor Documentation . . . . .	69
11.22.2.1 win_error_message() [1/2] . . . . .	69
11.22.2.2 win_error_message() [2/2] . . . . .	69
11.22.3 Member Function Documentation . . . . .	70
11.22.3.1 current() . . . . .	70
11.22.3.2 get_error_code() . . . . .	70
11.22.3.3 get_message() . . . . .	70
11.22.3.4 operator std::wstring() . . . . .	71
11.23fsw::win_flag_type Struct Reference . . . . .	71
11.24fsw::win_handle Class Reference . . . . .	71
11.24.1 Detailed Description . . . . .	72
11.24.2 Constructor & Destructor Documentation . . . . .	72
11.24.2.1 ~win_handle() . . . . .	72
11.24.2.2 win_handle() . . . . .	72
11.24.3 Member Function Documentation . . . . .	72
11.24.3.1 is_valid() [1/2] . . . . .	73
11.24.3.2 is_valid() [2/2] . . . . .	73
11.24.3.3 operator=() [1/2] . . . . .	73
11.24.3.4 operator=() [2/2] . . . . .	74
11.25fsw::windows_monitor Class Reference . . . . .	74
11.25.1 Detailed Description . . . . .	75
11.25.2 Member Function Documentation . . . . .	75
11.25.2.1 run() . . . . .	75

<b>12 File Documentation</b>	<b>77</b>
12.1 libswatch/c++/event.hpp File Reference	77
12.1.1 Detailed Description	78
12.2 libswatch/c++/fen_monitor.hpp File Reference	78
12.2.1 Detailed Description	78
12.3 libswatch/c++/filter.hpp File Reference	79
12.3.1 Detailed Description	79
12.4 libswatch/c++/fsevents_monitor.hpp File Reference	80
12.4.1 Detailed Description	80
12.5 libswatch/c++/inotify_monitor.hpp File Reference	80
12.5.1 Detailed Description	81
12.6 libswatch/c++/kqueue_monitor.hpp File Reference	81
12.6.1 Detailed Description	82
12.7 libswatch/c++/libswatch_exception.hpp File Reference	82
12.7.1 Detailed Description	83
12.8 libswatch/c++/libswatch_map.hpp File Reference	83
12.8.1 Detailed Description	83
12.9 libswatch/c++/libswatch_set.hpp File Reference	84
12.9.1 Detailed Description	84
12.10 libswatch/c++/monitor.hpp File Reference	84
12.10.1 Detailed Description	85
12.10.2 Macro Definition Documentation	86
12.10.2.1 REGISTER_MONITOR	86
12.10.2.2 REGISTER_MONITOR_IMPL	86
12.11 libswatch/c++/path_utils.hpp File Reference	87
12.11.1 Detailed Description	87
12.12 libswatch/c++/poll_monitor.hpp File Reference	88
12.12.1 Detailed Description	88
12.13 libswatch/c++/string/string_utils.hpp File Reference	88
12.13.1 Detailed Description	89

12.14libfswatch/c++/windows/win_directory_change_event.hpp File Reference . . . . .	89
12.14.1 Detailed Description . . . . .	90
12.15libfswatch/c++/windows/win_error_message.hpp File Reference . . . . .	90
12.15.1 Detailed Description . . . . .	91
12.16libfswatch/c++/windows/win_handle.hpp File Reference . . . . .	91
12.16.1 Detailed Description . . . . .	91
12.17libfswatch/c++/windows/win_paths.hpp File Reference . . . . .	92
12.17.1 Detailed Description . . . . .	92
12.18libfswatch/c++/windows/win_strings.hpp File Reference . . . . .	92
12.18.1 Detailed Description . . . . .	93
12.19libfswatch/c++/windows_monitor.hpp File Reference . . . . .	93
12.19.1 Detailed Description . . . . .	94
12.20libfswatch/c/cevent.h File Reference . . . . .	94
12.20.1 Detailed Description . . . . .	95
12.20.2 Typedef Documentation . . . . .	95
12.20.2.1 fsw_cevent . . . . .	96
12.20.2.2 FSW_CEVENT_CALLBACK . . . . .	96
12.20.3 Enumeration Type Documentation . . . . .	96
12.20.3.1 fsw_event_flag . . . . .	96
12.20.4 Function Documentation . . . . .	97
12.20.4.1 fsw_get_event_flag_by_name() . . . . .	97
12.20.4.2 fsw_get_event_flag_name() . . . . .	97
12.21libfswatch/c/cfilter.h File Reference . . . . .	98
12.21.1 Detailed Description . . . . .	98
12.22libfswatch/c/cmonitor.h File Reference . . . . .	99
12.22.1 Detailed Description . . . . .	99
12.22.2 Enumeration Type Documentation . . . . .	99
12.22.2.1 fsw_monitor_type . . . . .	99
12.23libfswatch/c/error.h File Reference . . . . .	100
12.23.1 Detailed Description . . . . .	100

12.23.2 Macro Definition Documentation . . . . .	101
12.23.2.1 FSW_ERR_CALLBACK_NOT_SET . . . . .	101
12.23.2.2 FSW_ERR_INVALID_CALLBACK . . . . .	101
12.23.2.3 FSW_ERR_INVALID_LATENCY . . . . .	101
12.23.2.4 FSW_ERR_INVALID_PATH . . . . .	101
12.23.2.5 FSW_ERR_INVALID_PROPERTY . . . . .	101
12.23.2.6 FSW_ERR_INVALID_REGEX . . . . .	101
12.23.2.7 FSW_ERR_MEMORY . . . . .	101
12.23.2.8 FSW_ERR_MISSING_CONTEXT . . . . .	102
12.23.2.9 FSW_ERR_MONITOR_ALREADY_EXISTS . . . . .	102
12.23.2.10 FSW_ERR_MONITOR_ALREADY_RUNNING . . . . .	102
12.23.2.11 FSW_ERR_PATHS_NOT_SET . . . . .	102
12.23.2.12 FSW_ERR_SESSION_UNKNOWN . . . . .	102
12.23.2.13 FSW_ERR_UNKNOWN_ERROR . . . . .	102
12.23.2.14 FSW_ERR_UNKNOWN_MONITOR_TYPE . . . . .	102
12.23.2.15 FSW_ERR_UNKNOWN_VALUE . . . . .	102
12.23.2.16 FSW_OK . . . . .	103
12.24 libfswatch/c/libfswatch.cpp File Reference . . . . .	103
12.24.1 Detailed Description . . . . .	104
12.24.2 Function Documentation . . . . .	104
12.24.2.1 fsw_add_event_type_filter() . . . . .	104
12.24.2.2 fsw_add_filter() . . . . .	105
12.24.2.3 fsw_add_path() . . . . .	105
12.24.2.4 fsw_add_property() . . . . .	105
12.24.2.5 fsw_destroy_session() . . . . .	105
12.24.2.6 fsw_init_library() . . . . .	105
12.24.2.7 fsw_init_session() . . . . .	106
12.24.2.8 fsw_is_verbose() . . . . .	106
12.24.2.9 fsw_last_error() . . . . .	106
12.24.2.10 fsw_set_allow_overflow() . . . . .	107

12.24.2.11	<code>fsw_set_callback()</code>	107
12.24.2.12	<code>fsw_set_directory_only()</code>	107
12.24.2.13	<code>fsw_set_follow_symlinks()</code>	107
12.24.2.14	<code>fsw_set_latency()</code>	107
12.24.2.15	<code>fsw_set_recursive()</code>	108
12.24.2.16	<code>fsw_set_verbose()</code>	108
12.24.2.17	<code>fsw_start_monitor()</code>	108
12.24.2.18	<code>fsw_stop_monitor()</code>	108
12.25	<code>libfswatch/c/libfswatch.h</code> File Reference	108
12.25.1	Detailed Description	109
12.25.2	Function Documentation	109
12.25.2.1	<code>fsw_add_event_type_filter()</code>	110
12.25.2.2	<code>fsw_add_filter()</code>	110
12.25.2.3	<code>fsw_add_path()</code>	110
12.25.2.4	<code>fsw_add_property()</code>	110
12.25.2.5	<code>fsw_destroy_session()</code>	110
12.25.2.6	<code>fsw_init_library()</code>	111
12.25.2.7	<code>fsw_init_session()</code>	111
12.25.2.8	<code>fsw_is_verbose()</code>	112
12.25.2.9	<code>fsw_last_error()</code>	112
12.25.2.10	<code>fsw_set_allow_overflow()</code>	112
12.25.2.11	<code>fsw_set_callback()</code>	112
12.25.2.12	<code>fsw_set_directory_only()</code>	112
12.25.2.13	<code>fsw_set_follow_symlinks()</code>	112
12.25.2.14	<code>fsw_set_latency()</code>	113
12.25.2.15	<code>fsw_set_recursive()</code>	113
12.25.2.16	<code>fsw_set_verbose()</code>	113
12.25.2.17	<code>fsw_start_monitor()</code>	113
12.25.2.18	<code>fsw_stop_monitor()</code>	113
12.26	<code>libfswatch/c/libfswatch_log.h</code> File Reference	113
12.26.1	Detailed Description	114
12.26.2	Function Documentation	114
12.26.2.1	<code>fsw_flog()</code>	115
12.26.2.2	<code>fsw_flogf()</code>	115
12.26.2.3	<code>fsw_log()</code>	115
12.26.2.4	<code>fsw_log_perror()</code>	115
12.26.2.5	<code>fsw_logf()</code>	115
12.26.2.6	<code>fsw_logf_perror()</code>	115
12.27	<code>libfswatch/c/libfswatch_types.h</code> File Reference	116
12.27.1	Detailed Description	116



# Chapter 1

## Main Page

### 1.1 Introduction

`fswatch` is a cross-platform file change monitor currently supporting the following backends:

- A monitor based on the *FSEvents* API of Apple OS X.
- A monitor based on *kqueue*, an event notification interface introduced in FreeBSD 4.1 and supported on most \*BSD systems (including OS X).
- A monitor based on *File Events Notification*, an event notification API of the Solaris/Illumos kernel.
- A monitor based on *inotify*, a Linux kernel subsystem that reports file system changes to applications.
- A monitor based on the Microsoft Windows' `ReadDirectoryChangesW` function and reads change events asynchronously.
- A monitor which periodically stats the file system, saves file modification times in memory and manually calculates file system changes, which can work on any operating system where `stat` can be used.

Instead of using different APIs, a programmer can use just one: the API of `libfswatch`. The advantages of using `libfswatch` are many:

- *Portability*: `libfswatch` supports many backends, effectively giving support to a great number of operating systems, including Solaris, \*BSD Unix and Linux.
- *Ease of use*: using `libfswatch` should be easier than using any of the APIs it supports.

### 1.2 Changelog

See the [History](#) page.

### 1.3 Available Bindings

`libfswatch` is a C++ library with C bindings which makes it available to a wide range of programming languages. If a programming language has C bindings, then `libfswatch` can be used from it. The C binding provides all the functionality provided by the C++ implementation and it can be used as a fallback solution when the C++ API cannot be used.

## 1.4 libtool's versioning scheme

libtool's versioning scheme is described by three integers: `current:revision:age` where:

- `current` is the most recent interface number implemented by the library.
- `revision` is the implementation number of the current interface.
- `age` is the difference between the newest and the oldest interface that the library implements.

## 1.5 The C and the C++ API

The C API is built on top of the C++ API but the two are very different, to reflect the fundamental differences between the two languages.

The [C++ API](#) centres on the concept of *monitor*, a class of objects modelling the functionality of the file monitoring API. Different monitor types are modelled as different classes inheriting from the `fsw::monitor` abstract class, that is the type that defines the core monitoring API. API clients can pick the current platform's default monitor, or choose a specific implementation amongst the available ones, configure it and *run* it. When running, a monitor gathers file system change events and communicates them back to the caller using a *callback*.

The [C API](#), on the other hand, centres on the concept of *monitoring session*. A session internally wraps a monitor instance and represents an opaque C bridge to the C++ monitor API. Sessions are identified by a *session handle* and they can be thought as a sort of C facade of the C++ monitor class. In fact there is an evident similarity between the C library functions operating on a monitoring session and the methods of the `monitor` class.

## 1.6 Thread Safety

The C++ API does not deal with thread safety explicitly. Rather, it leaves the responsibility of implementing a thread-safe use of the library to the callers. The C++ implementation has been designed in order to:

- Encapsulate all the state of a monitor into its class fields.
- Perform no concurrent access control in methods or class fields.
- Guarantee that functions and *static* methods are thread safe.

As a consequence, it is *not* thread-safe to access a monitor's member, be it a method or a field, from different threads concurrently. The easiest way to implement thread-safety when using `libfswatch`, therefore, is segregating access to each monitor instance from a different thread.

Similarly, the C API has been designed in order to provide the same guarantees offered by the C++ API:

- Concurrently manipulating different monitoring sessions is thread safe.
- Concurrently manipulating the same monitoring session is *not* thread safe.



## 1.7 C++11

There is an additional limitation which affects the C library only: the C binding implementation internally uses C++11 classes and keywords to provide the aforementioned guarantees. If compiler or library support is not found when building `libfswatch` the library will still build, but those guarantees will *not* be honoured. A warning such as the following will appear in the output of `configure` to inform the user:

```
configure: WARNING: libfswatch is not thread-safe because the current
combination of compiler and libraries do not support the thread_local
storage specifier.
```

## 1.8 Reporting Bugs and Suggestions

If you find problems or have suggestions about this program or this manual, please report them as new issues in the official GitHub repository of `fswatch` at <https://github.com/emcrisostomo/fswatch>. Please, read the `CONTRIBUTING.md` file for detailed instructions on how to contribute to `fswatch`.



## Chapter 2

# C++ API

The C++ API provides users an easy to use, object-oriented interface to a wide range of file monitoring APIs. This API provides a common facade to a set of heterogeneous APIs that not only greatly simplifies their usage, but provides an indirection layer that makes applications more portable: as far as there is an available monitor in another platform, an existing application will just work.

In reality, a monitor may have platform-specific behaviours that should be taken into account when writing portable applications using this library. This differences complicate the task of writing portable applications that are truly independent of the file monitoring API they may be using. However, monitors try to 'compensate' for any behavioural difference across implementations.

The `fsw::monitor` class is the basic type of the C++ API: it defines the interface of every monitor and provides common functionality to inheritors of this class, such as:

- Configuration and life cycle (`fsw::monitor`).
- Event filtering (`fsw::monitor`).
- Path filtering (`fsw::monitor`).
- Monitor registration (`fsw::monitor_factory`).
- Monitor discovery (`fsw::monitor_factory`).

### 2.1 Usage

The typical usage pattern of this API is similar to the following:

- An instance of a monitor is either created directly or through the factory (`fsw::monitor_factory`).
- The monitor is configured (`fsw::monitor`).
- The monitor is run and change events are waited for (`fsw::monitor::start()`).

## 2.2 Example

```
// Create the default platform monitor
monitor *active_monitor =
    monitor_factory::create_monitor(fsw_monitor_type::system_default_monitor_type,
                                    paths,
                                    process_events);

// Configure the monitor
active_monitor->set_properties(monitor_properties);
active_monitor->set_allow_overflow(allow_overflow);
active_monitor->set_latency(latency);
active_monitor->set_recursive(recursive);
active_monitor->set_directory_only(directory_only);
active_monitor->set_event_type_filters(event_filters);
active_monitor->set_filters(filters);
active_monitor->set_follow_symlinks(follow_symlinks);
active_monitor->set_watch_access(watch_access);

// Start the monitor
active_monitor->start();
```

## Chapter 3

# C API

The C API, whose main header file is [libfswatch.h](#), is a C-compatible lightweight wrapper around the C++ API that provides an easy to use binding to C clients. The central type in the C API is the *monitoring session*, an opaque type identified by a handle of type [FSW\\_HANDLE](#) that can be manipulated using the C functions of this library.

Session-modifying API calls (such as [fsw\\_add\\_path\(\)](#)) will take effect the next time a monitor is started with [fsw\\_↔start\\_monitor\(\)](#).

### 3.1 Translating the C++ API to C

The conventions used to translate C++ types into C types are simple:

- `std::string` is represented as a NUL-terminated `char *`.
- Lists are represented as arrays whose length is specified in a separate field.
- More complex types are usually translated as a `struct` containing data fields and a set of functions to operate on it.

### 3.2 Thread Safety

If the compiler and the C++ library used to build `libfswatch` support the `thread_local` storage specifier then this API is thread safe and a different state is maintained on a per-thread basis.

Even when `thread_local` is not available, manipulating *different* monitoring sessions concurrently from different threads is thread safe, since they share no data.

### 3.3 Library Initialization

Before calling any library method, the library must be initialized by calling the [fsw\\_init\\_library\(\)](#) function:

```
// Initialize the library
FSW_STATUS ret = fsw_init_library();
if (ret != FSW_OK)
{
    exit(1);
}
```

## 3.4 Status Codes and Errors

Most API functions return a status code of type `FSW_STATUS`, defined in the `error.h` header. A successful API call returns `FSW_OK` and the last error can be obtained calling the `fsw_last_error()` function.

## 3.5 Example

This is a basic example of how a monitor session can be constructed and run using the C API. To be valid, a session needs at least the following information:

- A path to watch.
- A *callback* to process the events sent by the monitor.

The next code fragment shows how to create and start a basic monitoring session (error checking code was omitted):

```
// Initialize the library
fsw_init_library();

// Use the default monitor.
const FSW_HANDLE handle = fsw_init_session();
fsw_add_path(handle, "my/path");
fsw_set_callback(handle, my_callback);
fsw_start_monitor(handle);
```

## Chapter 4

# History

### 4.1 9:0:0

- Add `fsw::monitor_filter::read_from_file()` to load filters from a file.
- Add `fsw_stop_monitor()` function to stop a running monitor.
- Change FSW\_HANDLE type.

### 4.2 8:0:2

- Add a mutex to protect the `fsw::monitor::notify_events()` method.
- Substitute C++ header names with C names in C headers.

### 4.3 8:0:2

- `fsw::monitor::~~monitor()`: update to invoke `fsw::monitor::stop()`.
- Close resources in `monitor::on_stop()` instead of doing it in destructors.
- Add inactivity callback.

### 4.4 8:0:2

- `fsw::monitor::stop()`: added.
- `fsw::monitor::monitor()`: update to move paths instead of copying them.
- `fsw::monitor_factory::exists_type(const std::string&)`: added.
- `fsw::monitor_factory::exists_type(const fsw_monitor_type&)`: added.
- `fsw::fsevents_monitor::set_numeric_event()`: removed.
- `fsw::string_utils::string_from_format`: added.
- `fsw::string_utils::vstring_from_format`: added.

## 4.5 5:0:2

- A monitor based on the Solaris/Illumos File Events Notification API has been added.
- The possibility of watching for directories only during a recursive scan. This feature helps reducing the number of open file descriptors if a generic change event for a directory is acceptable instead of events on directory children.
- `fsw::fen_monitor`: added to provide a monitor based on the Solaris/Illumos File Events Notification API.
- `fsw::monitor::set_directory_only()`: added to set a flag to only watch directories during a recursive scan.
- `fsw_set_directory_only()`: added to set a flag to only watch directories during a recursive scan.
- `fsw_logf_perror()`: added to log a `printf()`-style message using  `perror()`.

## 4.6 4:0:1

- `fsw::windows_monitor`: a monitor for Microsoft Windows was added.
- A logging function has been added to log verbose messages.
- A family of functions and macros have been added to log diagnostic messages:
  - `fsw_flog()`
  - `fsw_logf()`
  - `fsw_flogf()`
  - `fsw_log_perror()`
  - `FSW_LOG`
  - `FSW_ELOG`
  - `FSW_LOGF`
  - `FSW_ELOGF`
  - `FSW_FLOGF`

## 4.7 3:0:0

- Added ability to filter events *by type*:
  - `fsw::monitor::add_event_type_filter()`
  - `fsw::monitor::set_event_type_filters()`
- `fsw::monitor::notify_events()`: added to centralize event filtering and dispatching into the monitor base class.
- Added ability to get event types by name and stringify them:
  - `fsw::event::get_event_flag_by_name()`
  - `fsw::event::get_event_flag_name()`
  - `fsw_get_event_flag_by_name()`
  - `fsw_get_event_flag_name()`
- `fsw_event_type_filter`: added to represent an event type filter.
- `FSW_ERR_UNKNOWN_VALUE`: added error code.
- `fsw_add_event_type_filter()`: added to add an event type filter.



## Chapter 5

# Path Filtering

A *path filter* ([fsw::monitor\\_filter](#)) can be used to filter event paths. A filter type ([fsw\\_filter\\_type](#)) determines whether the filter regular expression is used to include and exclude paths from the list of the events processed by the library. `libfswatch` processes filters this way:

- If a path matches an including filter, the path is *accepted* no matter any other filter.
- If a path matches an excluding filter, the path is *rejected*.
- If a path matches no filters, the path is *accepted*.

Said another way:

- All paths are accepted *by default*, unless an exclusion filter says otherwise.
- Inclusion filters may override any other exclusion filter.
- The order in the filter definition has no effect.



## Chapter 6

# Namespace Index

### 6.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

<a href="#">fsw</a>	Main namespace of <code>libfswatch</code> . . . . .	21
<a href="#">fsw::string_utils</a>	This namespace contains string manipulation functions . . . . .	26
<a href="#">fsw::win_paths</a>	Path conversion functions . . . . .	27
<a href="#">fsw::win_strings</a>	String conversion functions . . . . .	29



## Chapter 7

# Hierarchical Index

### 7.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

fsw::compiled_monitor_filter . . . . .	31
fsw::directory_change_event . . . . .	31
fsw::event . . . . .	32
exception	
fsw::libfsw_exception . . . . .	43
fsw::FSEventFlagType . . . . .	36
fsw_callback_context . . . . .	38
fsw_cevent . . . . .	38
fsw_cmonitor_filter . . . . .	39
fsw_event_type_filter . . . . .	39
FSW_SESSION . . . . .	40
fsw::inotify_monitor_impl . . . . .	41
fsw::monitor . . . . .	44
fsw::fen_monitor . . . . .	35
fsw::fsevents_monitor . . . . .	37
fsw::inotify_monitor . . . . .	40
fsw::kqueue_monitor . . . . .	42
fsw::poll_monitor . . . . .	67
fsw::windows_monitor . . . . .	74
fsw::monitor_factory . . . . .	60
fsw::monitor_filter . . . . .	64
fsw::monitor_registrant< M > . . . . .	66
fsw::poll_monitor::poll_monitor_data . . . . .	68
fsw::win_error_message . . . . .	68
fsw::win_flag_type . . . . .	71
fsw::win_handle . . . . .	71



## Chapter 8

# Class Index

### 8.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">fsw::compiled_monitor_filter</a>	31
<a href="#">fsw::directory_change_event</a>	
Header of the <a href="#">fsw::directory_change_event</a> class, a helper class to wrap Microsoft Windows' <code>ReadDirectoryChangesW</code> function and a common workflow to detect file system changes	31
<a href="#">fsw::event</a>	
Type representing a file change event	32
<a href="#">fsw::fen_monitor</a>	
Solaris/Illumos monitor	35
<a href="#">fsw::FSEventFlagType</a>	36
<a href="#">fsw::fsevents_monitor</a>	
OS X FSEvents monitor	37
<a href="#">fsw_callback_context</a>	38
<a href="#">fsw_cevent</a>	38
<a href="#">fsw_cmonitor_filter</a>	39
<a href="#">fsw_event_type_filter</a>	
Event type filter	39
<a href="#">FSW_SESSION</a>	40
<a href="#">fsw::inotify_monitor</a>	
Solaris/Illumos monitor	40
<a href="#">fsw::inotify_monitor_impl</a>	41
<a href="#">fsw::kqueue_monitor</a>	
Solaris/Illumos monitor	42
<a href="#">fsw::libfsw_exception</a>	
Base exception of the <code>libfswatch</code> library	43
<a href="#">fsw::monitor</a>	
Base class of all monitors	44
<a href="#">fsw::monitor_factory</a>	
Object factory class for <a href="#">fsw::monitor</a> instances	60
<a href="#">fsw::monitor_filter</a>	
Path filters used to accept or reject file change events	64
<a href="#">fsw::monitor_registrant&lt; M &gt;</a>	
Helper class to register monitor factories	66
<a href="#">fsw::poll_monitor</a>	
<code>stat()</code> -based monitor	67
<a href="#">fsw::poll_monitor::poll_monitor_data</a>	68

<a href="#">fsw::win_error_message</a>	
Helper class to get the system-defined error message for a Microsoft Windows' error code . . .	68
<a href="#">fsw::win_flag_type</a> . . . . .	71
<a href="#">fsw::win_handle</a>	
A RAII wrapper around Microsoft Windows <code>HANDLE</code> . . . . .	71
<a href="#">fsw::windows_monitor</a>	
Windows monitor . . . . .	74



## Chapter 9

# File Index

### 9.1 File List

Here is a list of all documented files with brief descriptions:

libfswatch/ <b>gettext.h</b> . . . . .	??
libfswatch/ <b>gettext_defs.h</b> . . . . .	??
libfswatch/c++/ <b>event.hpp</b>	
Header of the <b>fsw::event</b> class . . . . .	77
libfswatch/c++/ <b>fen_monitor.hpp</b>	
Solaris/Illumos monitor . . . . .	78
libfswatch/c++/ <b>filter.hpp</b>	
Header of the <b>fsw::monitor_filter</b> class . . . . .	79
libfswatch/c++/ <b>fsevents_monitor.hpp</b>	
OS X FSEvents monitor . . . . .	80
libfswatch/c++/ <b>inotify_monitor.hpp</b>	
Solaris/Illumos monitor . . . . .	80
libfswatch/c++/ <b>kqueue_monitor.hpp</b>	
kqueue monitor . . . . .	81
libfswatch/c++/ <b>libfswatch_exception.hpp</b>	
Base exception of the <b>libfswatch</b> library . . . . .	82
libfswatch/c++/ <b>libfswatch_map.hpp</b>	
Header defining the associative container used by the library . . . . .	83
libfswatch/c++/ <b>libfswatch_set.hpp</b>	
Header defining the default set type used by the library . . . . .	84
libfswatch/c++/ <b>monitor.hpp</b>	
Header of the <b>fsw::monitor</b> class . . . . .	84
libfswatch/c++/ <b>path_utils.hpp</b>	
Header defining utility functions to manipulate paths . . . . .	87
libfswatch/c++/ <b>poll_monitor.hpp</b>	
<b>stat()</b> based monitor . . . . .	88
libfswatch/c++/ <b>windows_monitor.hpp</b>	
Windows monitor . . . . .	93
libfswatch/c++/string/ <b>string_utils.hpp</b>	
Header of the <b>fsw::string_utils</b> namespace . . . . .	88
libfswatch/c++/windows/ <b>win_directory_change_event.hpp</b>	
Header of the <b>fsw::directory_change_event</b> class . . . . .	89
libfswatch/c++/windows/ <b>win_error_message.hpp</b>	
Header of the <b>fsw::win_error_message</b> class . . . . .	90
libfswatch/c++/windows/ <b>win_handle.hpp</b>	
Header of the <b>fsw::win_handle</b> class . . . . .	91

<a href="#">libfswatch/c++/windows/win_paths.hpp</a>	
Header of the <a href="#">fsw::win_paths</a> namespace	92
<a href="#">libfswatch/c++/windows/win_strings.hpp</a>	
Header of the <a href="#">fsw::win_strings</a> namespace	92
<a href="#">libfswatch/c/cevent.h</a>	
Event type manipulation	94
<a href="#">libfswatch/c/cfilter.h</a>	
Header of the <code>libfswatch</code> library functions for filter management	98
<a href="#">libfswatch/c/cmonitor.h</a>	
Header of the <code>libfswatch</code> library defining the monitor types	99
<a href="#">libfswatch/c/error.h</a>	
Error values	100
<a href="#">libfswatch/c/libfswatch.cpp</a>	
Main <code>libfswatch</code> source file	103
<a href="#">libfswatch/c/libfswatch.h</a>	
Header of the <code>libfswatch</code> library	108
<a href="#">libfswatch/c/libfswatch_log.h</a>	
Header of the <code>libfswatch</code> library containing logging functions.	113
<a href="#">libfswatch/c/libfswatch_types.h</a>	
Header of the <code>libfswatch</code> library containing common types	116

## Chapter 10

# Namespace Documentation

### 10.1 fsw Namespace Reference

Main namespace of `libfswatch`.

#### Namespaces

- [string\\_utils](#)  
*This namespace contains string manipulation functions.*
- [win\\_paths](#)  
*Path conversion functions.*
- [win\\_strings](#)  
*String conversion functions.*

#### Classes

- struct [compiled\\_monitor\\_filter](#)
- class [directory\\_change\\_event](#)  
*Header of the `fsw::directory_change_event` class, a helper class to wrap Microsoft Windows' `ReadDirectoryChangesW` function and a common workflow to detect file system changes.*
- class [event](#)  
*Type representing a file change event.*
- class [fen\\_monitor](#)  
*Solaris/Illumos monitor.*
- struct [FSEventFlagType](#)
- class [fsevents\\_monitor](#)  
*OS X FSEvents monitor.*
- class [inotify\\_monitor](#)  
*Solaris/Illumos monitor.*
- struct [inotify\\_monitor\\_impl](#)
- class [kqueue\\_monitor](#)  
*Solaris/Illumos monitor.*
- class [libfsw\\_exception](#)  
*Base exception of the `libfswatch` library.*

- class [monitor](#)  
*Base class of all monitors.*
- class [monitor\\_factory](#)  
*Object factory class for [fsw::monitor](#) instances.*
- struct [monitor\\_filter](#)  
*Path filters used to accept or reject file change events.*
- class [monitor\\_registrant](#)  
*Helper class to register monitor factories.*
- class [poll\\_monitor](#)  
*`stat()`-based monitor.*
- class [win\\_error\\_message](#)  
*Helper class to get the system-defined error message for a Microsoft Windows' error code.*
- struct [win\\_flag\\_type](#)
- class [win\\_handle](#)  
*A RAII wrapper around Microsoft Windows `HANDLE`.*
- class [windows\\_monitor](#)  
*Windows monitor.*

## Typedefs

- typedef struct [fsw::monitor\\_filter](#) [monitor\\_filter](#)  
*Path filters used to accept or reject file change events.*
- typedef struct [fsw::FSEventFlagType](#) [FSEventFlagType](#)
- template<typename K, typename V >  
using [fsw\\_hash\\_map](#) = std::map< K, V >  
*Default associative container type used by `libfswatch`.*
- template<typename K >  
using [fsw\\_hash\\_set](#) = std::set< K >  
*Default set type used by `libfswatch`.*
- typedef void [FSW\\_EVENT\\_CALLBACK](#)(const std::vector< [event](#) > &, void \*)  
*Function definition of an event callback.*
- typedef [monitor](#) \*(\* [FSW\\_FN\\_MONITOR\\_CREATOR](#)) (std::vector< std::string > paths, [FSW\\_EVENT\\_CALLBACK](#) \*callback, void \*context)
- typedef struct [fsw::poll\\_monitor::poll\\_monitor\\_data](#) [poll\\_monitor\\_data](#)

## Functions

- ostream & [operator](#)<< (ostream &out, const [fsw\\_event\\_flag](#) flag)
- std::ostream & [operator](#)<< (std::ostream &out, const [fsw\\_event\\_flag](#) flag)  
*Overload of the << operator to print an event using `iostreams`.*
- static bool [parse\\_filter](#) (std::string filter, [monitor\\_filter](#) &filter\_object, void(\*err\_handler)(std::string))
- static bool [is\\_unescaped\\_space](#) (string &filter, long i)
- bool [parse\\_filter](#) (string filter, [monitor\\_filter](#) &filter\_object, void(\*err\_handler)(string))
- static vector< [FSEventFlagType](#) > [create\\_flag\\_type\\_vector](#) ()
- [REGISTER\\_MONITOR\\_IMPL](#) ([fsevents\\_monitor](#), [fsevents\\_monitor\\_type](#))
- static vector< [fsw\\_event\\_flag](#) > [decode\\_flags](#) (FSEventStreamEventFlags flag)
- [REGISTER\\_MONITOR\\_IMPL](#) ([inotify\\_monitor](#), [inotify\\_monitor\\_type](#))
- static [monitor](#) \* [create\\_default\\_monitor](#) (vector< string > paths, [FSW\\_EVENT\\_CALLBACK](#) \*callback, void \*context)
- vector< string > [get\\_directory\\_children](#) (const string &path)
- bool [read\\_link\\_path](#) (const string &path, string &link\_path)

- bool **stat\_path** (const string &path, struct stat &fd\_stat)
- bool **lstat\_path** (const string &path, struct stat &fd\_stat)
- std::vector< std::string > **get\_directory\_children** (const std::string &path)  
*Gets a vector of direct directory children.*
- bool **read\_link\_path** (const std::string &path, std::string &link\_path)  
*Resolves a path name.*
- bool **lstat\_path** (const std::string &path, struct stat &fd\_stat)  
*Wraps a `lstat(path, fd_stat)` call that invokes `perror()` if it fails.*
- bool **stat\_path** (const std::string &path, struct stat &fd\_stat)  
*Wraps a `stat(path, fd_stat)` call that invokes `perror()` if it fails.*
- **REGISTER\_MONITOR\_IMPL** (poll\_monitor, poll\_monitor\_type)
- static vector< win\_flag\_type > **create\_flag\_type\_vector** ()
- static vector< fsw\_event\_flag > **decode\_flags** (DWORD flag)

## Variables

- static const vector< FSEventFlagType > **event\_flag\_type** = create\_flag\_type\_vector()
- static const unsigned int **BUFFER\_SIZE** = (10 \* ((sizeof(struct inotify\_event)) + NAME\_MAX + 1))
- static const vector< win\_flag\_type > **event\_flag\_type** = create\_flag\_type\_vector()

### 10.1.1 Detailed Description

Main namespace of `libfswatch`.

### 10.1.2 Typedef Documentation

#### 10.1.2.1 FSW\_EVENT\_CALLBACK

```
typedef void fsw::FSW_EVENT_CALLBACK(const std::vector< event > &, void *)
```

Function definition of an event callback.

The event callback is a user-supplied function that is invoked by the monitor when an event is detected. The following parameters are passed to the callback:

- A reference to the vector of events.
- A pointer to the *context data* set by the caller.

### 10.1.2.2 fsw\_hash\_map

```
template<typename K , typename V >
using fsw::fsw_hash_map = typedef std::map<K, V>
```

Default associative container type used by libfswatch.

This type definition will be a synonym of `std::unordered_map` if the C++ library contains it, otherwise it will default to `std::map`.

### 10.1.2.3 fsw\_hash\_set

```
template<typename K >
using fsw::fsw_hash_set = typedef std::set<K>
```

Default set type used by libfswatch.

This type definition will be a synonym of `std::unordered_set` if the C++ library contains it, otherwise it will default to `std::set`.

### 10.1.2.4 monitor\_filter

```
typedef struct fsw::monitor_filter fsw::monitor_filter
```

Path filters used to accept or reject file change events.

A path filter is a regular expression used to accept or reject file change events based on the value of their path. A filter has the following characteristics:

- It has a *regular expression* (`monitor_filter::text`), used to match the paths.
- It can be an *inclusion* or an *exclusion* filter (`monitor_filter::type`).
- It can be case *sensitive* or *insensitive* (`monitor_filter::case_sensitive`).
- It can be an *extended* regular expression (`monitor_filter::extended`).

Further information about how filtering works in libfswatch can be found in [Path Filtering](#).

## 10.1.3 Function Documentation

### 10.1.3.1 get\_directory\_children()

```
std::vector<std::string> fsw::get_directory_children (
    const std::string & path )
```

Gets a vector of direct directory children.

## Parameters

<i>path</i>	The directory whose children must be returned.
-------------	--

## Returns

A vector containing the list of children of *path*.

10.1.3.2 `lstat_path()`

```
bool fsw::lstat_path (
    const std::string & path,
    struct stat & fd_stat )
```

Wraps a `lstat(path, fd_stat)` call that invokes `perror()` if it fails.

## Parameters

<i>path</i>	The path to <code>lstat()</code> .
<i>fd_stat</i>	The <code>stat</code> structure where <code>lstat()</code> writes its results.

## Returns

`true` if the function succeeds, `false` otherwise.

10.1.3.3 `operator<<()`

```
std::ostream& fsw::operator<< (
    std::ostream & out,
    const fsw_event_flag flag )
```

Overload of the `<<` operator to print an event using `iostreams`.

## Parameters

<i>out</i>	A reference to the output stream.
<i>flag</i>	The flag to print.

## Returns

A reference to the stream.

#### 10.1.3.4 read\_link\_path()

```
bool fsw::read_link_path (
    const std::string & path,
    std::string & link_path )
```

Resolves a path name.

This function resolves `path` using `realpath()` and stores the absolute pathname into `link_path`. The function returns `true` if it succeeds, `false` otherwise.

##### Parameters

<i>path</i>	The path to resolve.
<i>link_path</i>	A reference to a <code>std::string</code> where the resolved absolute path should be copied to.

##### Returns

`true` if the function succeeds, `false` otherwise.

#### 10.1.3.5 stat\_path()

```
bool fsw::stat_path (
    const std::string & path,
    struct stat & fd_stat )
```

Wraps a `stat(path, fd_stat)` call that invokes `perror()` if it fails.

##### Parameters

<i>path</i>	The path to <code>stat()</code> .
<i>fd_stat</i>	The <code>stat</code> structure where <code>stat()</code> writes its results.

##### Returns

`true` if the function succeeds, `false` otherwise.

## 10.2 fsw::string\_utils Namespace Reference

This namespace contains string manipulation functions.

### Functions

- string [vstring\\_from\\_format](#) (const char \*format, va\_list args)  
Create a `std::string` using a `printf()` format and a `va_list` args.
- string [string\\_from\\_format](#) (const char \*format,...)  
Create a `std::string` using a `printf()` format and `varargs`.



### 10.2.1 Detailed Description

This namespace contains string manipulation functions.

### 10.2.2 Function Documentation

#### 10.2.2.1 string\_from\_format()

```
std::string fsw::string_utils::string_from_format (
    const char * format,
    ... )
```

Create a `std::string` using a `printf()` format and `varargs`.

##### Parameters

<i>format</i>	The <code>printf()</code> format.
...	The arguments to format.

#### 10.2.2.2 vstring\_from\_format()

```
std::string fsw::string_utils::vstring_from_format (
    const char * format,
    va_list args )
```

Create a `std::string` using a `printf()` format and a `va_list` args.

##### Parameters

<i>format</i>	The <code>printf()</code> format.
<i>args</i>	The arguments to format.

## 10.3 fsw::win\_paths Namespace Reference

Path conversion functions.

### Functions

- wstring **posix\_to\_win\_w** (string path)
- string **win\_w\_to\_posix** (wstring path)

- `std::wstring posix_to_win_w (std::string path)`  
*Converts a POSIX path to Windows.*
- `std::string win_w_to_posix (std::wstring path)`  
*Converts a Windows path to POSIX.*

### 10.3.1 Detailed Description

Path conversion functions.

This namespace contains utility functions for POSIX to Windows and Windows to POSIX path conversion functions.

### 10.3.2 Function Documentation

#### 10.3.2.1 posix\_to\_win\_w()

```
std::wstring fsw::win_paths::posix_to_win_w (
    std::string path )
```

Converts a POSIX path to Windows.

##### Parameters

<i>path</i>	The POSIX path to convert to a Windows path.
-------------	--

##### Returns

The converted Windows path.

#### 10.3.2.2 win\_w\_to\_posix()

```
std::string fsw::win_paths::win_w_to_posix (
    std::wstring path )
```

Converts a Windows path to POSIX.

##### Parameters

<i>path</i>	The Windows path to convert to POSIX.
-------------	---------------------------------------

##### Returns

The converted POSIX path.

## 10.4 fsw::win\_strings Namespace Reference

String conversion functions.

### Functions

- string [wstring\\_to\\_string](#) (wchar\_t \*s)  
*Converts a wide character string into a string.*
- string **wstring\_to\_string** (const wstring &s)
- std::string [wstring\\_to\\_string](#) (const std::wstring &s)  
*Converts a wide character string into a string.*

### 10.4.1 Detailed Description

String conversion functions.

This namespace contains utility functions to convert wide character strings into strings.

### 10.4.2 Function Documentation

#### 10.4.2.1 wstring\_to\_string() [1/2]

```
std::string fsw::win_strings::wstring_to_string (  
    wchar_t * s )
```

Converts a wide character string into a string.

#### Parameters

s	The wchar_t array to convert.
---	-------------------------------

#### Returns

The converted string.

#### 10.4.2.2 wstring\_to\_string() [2/2]

```
std::string fsw::win_strings::wstring_to_string (  
    const std::wstring & s )
```

Converts a wide character string into a string.

**Parameters**

s	The string to convert.
---	------------------------

**Returns**

The converted string.

# Chapter 11

## Class Documentation

### 11.1 fsw::compiled\_monitor\_filter Struct Reference

#### Public Attributes

- `regex_t` **regex**
- `fsw_filter_type` **type**

The documentation for this struct was generated from the following file:

- `libfswatch/c++/monitor.cpp`

### 11.2 fsw::directory\_change\_event Class Reference

Header of the `fsw::directory_change_event` class, a helper class to wrap Microsoft Windows' `ReadDirectoryChangesW` function and a common workflow to detect file system changes.

```
#include <win_directory_change_event.hpp>
```

#### Public Member Functions

- **directory\_change\_event** (`size_t` `buffer_length`=16)
- `bool` **is\_io\_incomplete** ()
- `bool` **is\_buffer\_overflowed** ()
- `bool` **read\_changes\_async** ()
- `bool` **try\_read** ()
- `void` **continue\_read** ()
- `std::vector< event >` **get\_events** ()

## Public Attributes

- `std::wstring path`
- `win_handle handle`
- `size_t buffer_size`
- `DWORD bytes_returned`
- `std::unique_ptr< void, decltype(free) * > buffer = {nullptr, free}`
- `std::unique_ptr< OVERLAPPED, decltype(free) * > overlapped = {static_cast<OVERLAPPED *> (malloc(sizeof (OVERLAPPED))), free}`
- `win_error_message read_error`

### 11.2.1 Detailed Description

Header of the `fsw::directory_change_event` class, a helper class to wrap Microsoft Windows' `ReadDirectoryChangesW` function and a common workflow to detect file system changes.

The documentation for this class was generated from the following files:

- `libfswatch/c++/windows/win_directory_change_event.hpp`
- `libfswatch/c++/windows/win_directory_change_event.cpp`

## 11.3 fsw::event Class Reference

Type representing a file change event.

```
#include <event.hpp>
```

### Public Member Functions

- `event (std::string path, time_t evt_time, std::vector< fsw_event_flag > flags)`  
*Constructs an event.*
- `virtual ~event ()`  
*Destructs an event.*
- `std::string get_path () const`  
*Returns the path of the event.*
- `time_t get_time () const`  
*Returns the time of the event.*
- `std::vector< fsw_event_flag > get_flags () const`  
*Returns the flags of the event.*

### Static Public Member Functions

- `static fsw_event_flag get_event_flag_by_name (const std::string &name)`  
*Get event flag by name.*
- `static std::string get_event_flag_name (const fsw_event_flag &flag)`  
*Get the name of an event flag.*

### 11.3.1 Detailed Description

Type representing a file change event.

This class represents a file change event in the `libfswatch` API. An event contains:

- The path.
- The time the event was raised.
- A vector of flags specifying the type of the event.

### 11.3.2 Constructor & Destructor Documentation

#### 11.3.2.1 event()

```
fsw::event::event (
    std::string path,
    time_t evt_time,
    std::vector< fsw_event_flag > flags )
```

Constructs an event.

##### Parameters

<i>path</i>	The path the event refers to.
<i>evt_time</i>	The time the event was raised.
<i>flags</i>	The vector of flags specifying the type of the event.

#### 11.3.2.2 ~event()

```
fsw::event::~~event ( ) [virtual]
```

Destructs an event.

This is a virtual destructor that performs no operations.

### 11.3.3 Member Function Documentation

#### 11.3.3.1 get\_event\_flag\_by\_name()

```
fsw_event_flag fsw::event::get_event_flag_by_name (
    const std::string & name ) [static]
```

Get event flag by name.

## Parameters

<i>name</i>	The name of the event flag to look for.
-------------	---

## Returns

The event flag whose name is *name*, otherwise

## Exceptions

<a href="#"><i>libfsw_exception</i></a>	if no event flag is found.
---	----------------------------

11.3.3.2 `get_event_flag_name()`

```
string fsw::event::get_event_flag_name (
    const fsw\_event\_flag & flag ) [static]
```

Get the name of an event flag.

## Parameters

<i>flag</i>	The event flag.
-------------	-----------------

## Returns

The name of *flag*.

## Exceptions

<a href="#"><i>libfsw_exception</i></a>	if no event flag is found.
---	----------------------------

11.3.3.3 `get_flags()`

```
vector< fsw\_event\_flag > fsw::event::get_flags ( ) const
```

Returns the flags of the event.

## Returns

The flags of the event.



## 11.3.3.4 get\_path()

```
string fsw::event::get_path ( ) const
```

Returns the path of the event.

## Returns

The path of the event.

## 11.3.3.5 get\_time()

```
time_t fsw::event::get_time ( ) const
```

Returns the time of the event.

## Returns

The time of the event.

The documentation for this class was generated from the following files:

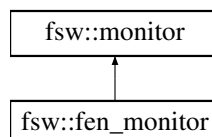
- libfswatch/c++/[event.hpp](#)
- libfswatch/c++/[event.cpp](#)

## 11.4 fsw::fen\_monitor Class Reference

Solaris/Illumos monitor.

```
#include <fen_monitor.hpp>
```

Inheritance diagram for fsw::fen\_monitor:



## Public Member Functions

- [fen\\_monitor](#) (std::vector< std::string > [paths](#), [FSW\\_EVENT\\_CALLBACK](#) \*[callback](#), void \*[context](#)=nullptr)  
Constructs an instance of this class.
- virtual [~fen\\_monitor](#) ()  
Destroys an instance of this class.

## Protected Member Functions

- void [run](#) () override  
*Executes the monitor loop.*

## Additional Inherited Members

### 11.4.1 Detailed Description

Solaris/Illumos monitor.

This monitor is built upon the *File Events Notification* API of the Solaris and Illumos kernels.

### 11.4.2 Member Function Documentation

#### 11.4.2.1 run()

```
void fsw::fen_monitor::run ( ) [override], [protected], [virtual]
```

Executes the monitor loop.

This call does not return until the monitor is stopped.

See also

[stop\(\)](#)

Implements [fsw::monitor](#).

The documentation for this class was generated from the following file:

- libfswatch/c++/[fen\\_monitor.hpp](#)

## 11.5 fsw::FSEventFlagType Struct Reference

### Public Attributes

- FSEventStreamEventFlags **flag**
- [fsw\\_event\\_flag](#) **type**

The documentation for this struct was generated from the following file:

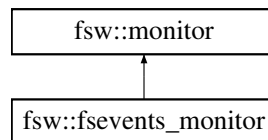
- libfswatch/c++/[fsevents\\_monitor.cpp](#)

## 11.6 fsw::fsevents\_monitor Class Reference

OS X FSEvents monitor.

```
#include <fsevents_monitor.hpp>
```

Inheritance diagram for fsw::fsevents\_monitor:



### Public Member Functions

- `fsevents_monitor` (std::vector< std::string > `paths`, `FSW_EVENT_CALLBACK` \*`callback`, void \*`context`=nullptr)  
*Constructs an instance of this class.*
- virtual `~fsevents_monitor` ()  
*Destroys an instance of this class.*

### Protected Member Functions

- void `run` () override  
*Executes the monitor loop.*
- void `on_stop` () override  
*Execute an implementation-specific stop handler.*

### Additional Inherited Members

#### 11.6.1 Detailed Description

OS X FSEvents monitor.

This monitor is built upon the *FSEvents* API of the Apple OS X kernel.

#### 11.6.2 Member Function Documentation

### 11.6.2.1 run()

```
void fsw::fsevents_monitor::run ( ) [override], [protected], [virtual]
```

Executes the monitor loop.

This call does not return until the monitor is stopped.

See also

[stop\(\)](#)

Implements [fsw::monitor](#).

The documentation for this class was generated from the following files:

- libfswatch/c++/fsevents\_monitor.hpp
- libfswatch/c++/fsevents\_monitor.cpp

## 11.7 fsw\_callback\_context Struct Reference

### Public Attributes

- [FSW\\_HANDLE](#) **handle**
- [FSW\\_CEVENT\\_CALLBACK](#) **callback**
- void \* **data**

The documentation for this struct was generated from the following file:

- libfswatch/c/libfswatch.cpp

## 11.8 fsw\_cevent Struct Reference

```
#include <cevent.h>
```

### Public Attributes

- char \* **path**
- time\_t **evt\_time**
- enum [fsw\\_event\\_flag](#) \* **flags**
- unsigned int **flags\_num**

### 11.8.1 Detailed Description

A file change event is represented as an instance of this struct where:

- path is the path where the event was triggered.
- evt\_time the time when the event was triggered.
- flags is an array of fsw\_event\_flag of size flags\_num.
- flags\_num is the size of the flags array.

The documentation for this struct was generated from the following file:

- libfswatch/c/[cevent.h](#)

## 11.9 fsw\_cmonitor\_filter Struct Reference

### Public Attributes

- char \* **text**
- enum [fsw\\_filter\\_type](#) **type**
- bool **case\_sensitive**
- bool **extended**

The documentation for this struct was generated from the following file:

- libfswatch/c/[cfilter.h](#)

## 11.10 fsw\_event\_type\_filter Struct Reference

Event type filter.

```
#include <cfilter.h>
```

### Public Attributes

- enum [fsw\\_event\\_flag](#) **flag**

### 11.10.1 Detailed Description

Event type filter.

The documentation for this struct was generated from the following file:

- libfswatch/c/[cfilter.h](#)

## 11.11 FSW\_SESSION Struct Reference

### Public Attributes

- vector< string > **paths**
- [fsw\\_monitor\\_type](#) **type**
- [fsw::monitor](#) \* **monitor**
- [FSW\\_CEVENT\\_CALLBACK](#) **callback**
- double **latency**
- bool **allow\_overflow**
- bool **recursive**
- bool **directory\_only**
- bool **follow\_symlinks**
- vector< [monitor\\_filter](#) > **filters**
- vector< [fsw\\_event\\_type\\_filter](#) > **event\_type\_filters**
- map< string, string > **properties**
- void \* **data**

The documentation for this struct was generated from the following file:

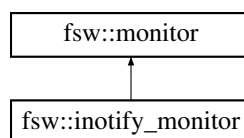
- [libfswatch/c/libfswatch.cpp](#)

## 11.12 fsw::inotify\_monitor Class Reference

Solaris/Illumos monitor.

```
#include <inotify_monitor.hpp>
```

Inheritance diagram for fsw::inotify\_monitor:



### Public Member Functions

- [inotify\\_monitor](#) (std::vector< std::string > [paths](#), [FSW\\_EVENT\\_CALLBACK](#) \*[callback](#), void \*[context](#)=nullptr)  
*Constructs an instance of this class.*
- virtual [~inotify\\_monitor](#) ()  
*Destroys an instance of this class.*

### Protected Member Functions

- void [run](#) ()  
*Executes the monitor loop.*

## Additional Inherited Members

### 11.12.1 Detailed Description

Solaris/Illumos monitor.

This monitor is built upon the *File Events Notification* API of the Solaris and Illumos kernels.

### 11.12.2 Member Function Documentation

#### 11.12.2.1 run()

```
void fsw::inotify_monitor::run ( ) [protected], [virtual]
```

Executes the monitor loop.

This call does not return until the monitor is stopped.

See also

[stop\(\)](#)

Implements [fsw::monitor](#).

The documentation for this class was generated from the following files:

- libfswatch/c++/inotify\_monitor.hpp
- libfswatch/c++/inotify\_monitor.cpp

## 11.13 fsw::inotify\_monitor\_impl Struct Reference

### Public Attributes

- int **inotify\_monitor\_handle** = -1
- vector< [event](#) > **events**
- [fsw\\_hash\\_set](#)< int > **watched\_descriptors**
- [fsw\\_hash\\_map](#)< string, int > **path\_to\_wd**
- [fsw\\_hash\\_map](#)< int, string > **wd\_to\_path**
- [fsw\\_hash\\_set](#)< int > **descriptors\_to\_remove**
- [fsw\\_hash\\_set](#)< int > **watches\_to\_remove**
- vector< string > **paths\_to\_rescan**
- time\_t **curr\_time**

The documentation for this struct was generated from the following file:

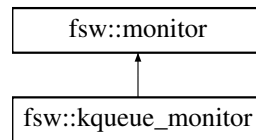
- libfswatch/c++/inotify\_monitor.cpp

## 11.14 fsw::kqueue\_monitor Class Reference

Solaris/Illumos monitor.

```
#include <kqueue_monitor.hpp>
```

Inheritance diagram for fsw::kqueue\_monitor:



### Public Member Functions

- [kqueue\\_monitor](#) (std::vector< std::string > [paths](#), [FSW\\_EVENT\\_CALLBACK](#) \*[callback](#), void \*[context](#)=nullptr)  
*Constructs an instance of this class.*
- virtual [~kqueue\\_monitor](#) ()  
*Destroys an instance of this class.*

### Protected Member Functions

- void [run](#) ()  
*Executes the monitor loop.*

### Additional Inherited Members

#### 11.14.1 Detailed Description

Solaris/Illumos monitor.

This monitor is built upon the `kqueue` API of the BSD kernels.

#### 11.14.2 Member Function Documentation

##### 11.14.2.1 run()

```
void fsw::kqueue_monitor::run ( ) [protected], [virtual]
```

Executes the monitor loop.

This call does not return until the monitor is stopped.

See also

[stop\(\)](#)

Implements [fsw::monitor](#).

The documentation for this class was generated from the following file:

- [libfswatch/c++/kqueue\\_monitor.hpp](#)

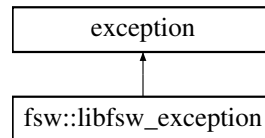


## 11.15 fsw::libfsw\_exception Class Reference

Base exception of the `libfswatch` library.

```
#include <libfswatch_exception.hpp>
```

Inheritance diagram for `fsw::libfsw_exception`:



### Public Member Functions

- `libfsw_exception` (`std::string cause`, `int code=FSW_ERR_UNKNOWN_ERROR`)  
*Constructs an exception with the specified `cause` and `error code`.*
- `libfsw_exception` (`const libfsw_exception &other`) `noexcept`
- `libfsw_exception & operator=` (`const libfsw_exception &`) `noexcept`
- `virtual const char * what` () `const noexcept`  
*Gets the error message.*
- `virtual int error_code` () `const noexcept`  
*Gets the error code.*
- `virtual ~libfsw_exception` () `noexcept`  
*Destructs an instance of this class.*
- `operator int` () `const noexcept`  
*Gets the error code.*

### 11.15.1 Detailed Description

Base exception of the `libfswatch` library.

An instance of this class stores an error message and an integer error code.

### 11.15.2 Constructor & Destructor Documentation

#### 11.15.2.1 libfsw\_exception()

```
fsw::libfsw_exception::libfsw_exception (
    std::string cause,
    int code = FSW_ERR_UNKNOWN_ERROR )
```

Constructs an exception with the specified `cause` and `error code`.

**Parameters**

<i>cause</i>	The error message.
<i>code</i>	The error code.

**11.15.3 Member Function Documentation****11.15.3.1 error\_code()**

```
int fsw::libfsw_exception::error_code ( ) const [virtual], [noexcept]
```

Gets the error code.

**Returns**

The error code.

**11.15.3.2 what()**

```
const char * fsw::libfsw_exception::what ( ) const [virtual], [noexcept]
```

Gets the error message.

**Returns**

The error message.

The documentation for this class was generated from the following files:

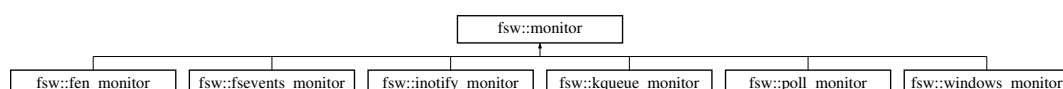
- [libfswatch/c++/libfswatch\\_exception.hpp](#)
- [libfswatch/c++/libfswatch\\_exception.cpp](#)

**11.16 fsw::monitor Class Reference**

Base class of all monitors.

```
#include <monitor.hpp>
```

Inheritance diagram for fsw::monitor:



## Public Member Functions

- [monitor](#) (std::vector< std::string > [paths](#), [FSW\\_EVENT\\_CALLBACK](#) \*[callback](#), void \*[context](#)=nullptr)  
*Constructs a monitor watching the specified [paths](#).*
- virtual [~monitor](#) ()  
*Destructs a monitor instance.*
- [monitor](#) (const [monitor](#) &[orig](#))=delete  
*This class is not copy constructible.*
- [monitor](#) & [operator=](#) (const [monitor](#) &[that](#))=delete  
*This class is not copy assignable.*
- void [set\\_property](#) (const std::string &[name](#), const std::string &[value](#))  
*Sets a custom property.*
- void [set\\_properties](#) (const std::map< std::string, std::string > [options](#))  
*Sets the custom properties.*
- std::string [get\\_property](#) (std::string [name](#))  
*Gets the value of a property.*
- void [set\\_latency](#) (double [latency](#))  
*Sets the latency.*
- void [set\\_fire\\_idle\\_event](#) (bool [fire\\_idle\\_event](#))  
*Sets the fire idle event flag.*
- void [set\\_allow\\_overflow](#) (bool [overflow](#))  
*Notify buffer overflows as change events.*
- void [set\\_recursive](#) (bool [recursive](#))  
*Recursively scan subdirectories.*
- void [set\\_directory\\_only](#) (bool [directory\\_only](#))  
*Watch directories only.*
- void [add\\_filter](#) (const [monitor\\_filter](#) &[filter](#))  
*Add a path filter.*
- void [set\\_filters](#) (const std::vector< [monitor\\_filter](#) > &[filters](#))  
*Set the path filters.*
- void [set\\_follow\\_symlinks](#) (bool [follow](#))  
*Follow symlinks.*
- void \* [get\\_context](#) () const  
*Get the pointer to the context data.*
- void [set\\_context](#) (void \*[context](#))  
*Set the context data.*
- void [start](#) ()  
*Start the monitor.*
- void [stop](#) ()  
*Stop the monitor.*
- bool [is\\_running](#) ()  
*Check whether the monitor is running.*
- void [add\\_event\\_type\\_filter](#) (const [fsw\\_event\\_type\\_filter](#) &[filter](#))  
*Add an event type filter.*
- void [set\\_event\\_type\\_filters](#) (const std::vector< [fsw\\_event\\_type\\_filter](#) > &[filters](#))  
*Set the event type filters.*
- void [set\\_watch\\_access](#) (bool [access](#))  
*Monitor file access.*

## Protected Member Functions

- bool `accept_event_type` (`fsw_event_flag` `event_type`) const  
*Check whether an event should be accepted.*
- bool `accept_path` (const std::string &path) const  
*Check whether a path should be accepted.*
- bool `accept_path` (const char \*path) const  
*Check whether a path should be accepted.*
- void `notify_events` (const std::vector< `event` > &events) const  
*Notify change events.*
- void `notify_overflow` (const std::string &path) const  
*Notify an overflow event.*
- std::vector< `fsw_event_flag` > `filter_flags` (const `event` &evt) const  
*Filter event types.*
- virtual void `run` ()=0  
*Execute monitor loop.*
- virtual void `on_stop` ()  
*Execute an implementation-specific stop handler.*

## Protected Attributes

- std::vector< std::string > `paths`  
*List of paths to watch.*
- std::map< std::string, std::string > `properties`  
*Map of custom properties.*
- `FSW_EVENT_CALLBACK` \* `callback`  
*Callback to which change events should be notified.*
- void \* `context` = nullptr  
*Pointer to context data that will be passed to the `monitor::callback`.*
- double `latency` = 1.0  
*Latency of the monitor.*
- bool `fire_idle_event` = false  
*If `true`, the monitor will notify an event when idle.*
- bool `allow_overflow` = false  
*If `true`, queue overflow events will be notified to the caller, otherwise the monitor will throw a `libfsw_exception`.*
- bool `recursive` = false  
*If `true`, directories will be scanned recursively.*
- bool `follow_symlinks` = false  
*If `true`, symbolic links are followed.*
- bool `directory_only` = false  
*Flag indicating whether only directories should be monitored.*
- bool `watch_access` = false  
*Flag indicating whether file access should be watched.*
- bool `running` = false  
*Flag indicating whether the monitor is in the running state.*
- bool `should_stop` = false  
*Flag indicating whether the monitor should preemptively stop.*
- std::mutex `run_mutex`  
*Mutex used to serialize access to the monitor state from multiple threads.*
- std::mutex `notify_mutex`  
*Mutex used to serialize access to the `notify_events()` method.*

### 11.16.1 Detailed Description

Base class of all monitors.

The `fsw::monitor` class is the base class of all monitors. This class encapsulates the common functionality of a monitor:

- Accessors to configuration parameters.
- `start()` and `stop()` lifecycle.
- Event filtering.
- Event notification to user-provided callback function.

Since some methods are designed to be called from different threads, this class provides an internal mutex (`monitor::run_mutex`) that implementors should lock on when accessing shared state. The mutex is available only when `HAVE_CXX_MUTEX` is defined.

At least the following tasks must be performed to implement a monitor:

- Providing an implementation of the `run()` method.
- Providing an implementation of the `on_stop()` method if the monitor cannot be stopped cooperatively from the `run()` method.

A basic monitor needs to implement the `run()` method, whose skeleton is often similar to the following:

```
void run()
{
    initialize_api();

    for (;;)
    {
#ifdef HAVE_CXX_MUTEX
        unique_lock<mutex> run_guard(run_mutex);
        if (should_stop) break;
        run_guard.unlock();
#endif

        scan_paths();
        wait_for_events();

        vector<change_events> evts = get_changes();
        vector<event> events;

        for (auto & evt : evts)
        {
            if (accept(evt.get_path()))
            {
                events.push_back({event from evt});
            }
        }

        if (events.size()) notify_events(events);
    }

    terminate_api();
}
```

Despite being a minimal implementation, it performs all the tasks commonly performed by a monitor:

- It initializes the API it uses to detect file system change events.
- It enters a loop, often infinite, where change events are waited for.
- If `HAVE_CXX_MUTEX` is defined, it locks on `monitor::run_mutex` to check whether `monitor::should_stop` is set to `true`. If it is, the monitor breaks the loop to return from `run()` as soon as possible.
- It scans the paths that must be observed: this step might be necessary for example because some path may not have existed during the previous iteration of the loop, or because some API may require the user to re-register a watch on a path after events are retrieved.
- Events are waited for and the wait should respect the specified *latency*.
- Events are *filtered* to exclude those referring to paths that do not satisfy the configured filters.
- The `notify_events()` method is called to filter the event types and notify the caller.

## 11.16.2 Constructor & Destructor Documentation

### 11.16.2.1 `monitor()`

```
fsw::monitor::monitor (
    std::vector< std::string > paths,
    FSW_EVENT_CALLBACK * callback,
    void * context = nullptr )
```

Constructs a monitor watching the specified `paths`.

The monitor will notify change events to the specified `callback`, passing it the pointer to the specified `context`.

#### Parameters

<i>paths</i>	The list of paths to watch.
<i>callback</i>	The callback to which change events will be notified. The callback cannot be null, otherwise a <code>libfsw_exception</code> will be thrown.
<i>context</i>	An optional pointer to context data. The monitor stores a copy of this pointer to pass it to the <code>callback</code> .

### 11.16.2.2 `~monitor()`

```
fsw::monitor::~~monitor ( ) [virtual]
```

Destructs a monitor instance.

This destructor performs the following operations:

- Stops the monitor.
- Frees the compiled regular expression of the path filters, if any.

### Warning

Destroying a monitor in the *running* state results in undefined behaviour.

### See also

[stop\(\)](#)

## 11.16.3 Member Function Documentation

### 11.16.3.1 accept\_event\_type()

```
bool fsw::monitor::accept_event_type (
    fsw_event_flag event_type ) const [protected]
```

Check whether an event should be accepted.

This function checks `event_type` against the event type filters of the monitor to determine whether it should be *accepted*.

#### Parameters

<code>event_type</code>	The event type to check.
-------------------------	--------------------------

#### Returns

`true` if the event is accepted, `false` otherwise.

### 11.16.3.2 accept\_path() [1/2]

```
bool fsw::monitor::accept_path (
    const std::string & path ) const [protected]
```

Check whether a path should be accepted.

This function checks `path` against the path filters of the monitor to determine whether it should be *accepted*.

#### Parameters

<code>event_type</code>	The path to check.
-------------------------	--------------------

#### Returns

`true` if the path is accepted, `false` otherwise.

### 11.16.3.3 accept\_path() [2/2]

```
bool fsw::monitor::accept_path (
    const char * path ) const [protected]
```

Check whether a path should be accepted.

This function checks `path` against the path filters of the monitor to determine whether it should be *accepted*.

#### Parameters

<i>event_type</i>	The path to check.
-------------------	--------------------

#### Returns

`true` if the path is accepted, `false` otherwise.

### 11.16.3.4 add\_event\_type\_filter()

```
void fsw::monitor::add_event_type_filter (
    const fsw_event_type_filter & filter )
```

Add an event type filter.

Adds a `fsw_event_type_filter` instance to filter events by *type*.

#### Parameters

<i>filter</i>	The event type filter to add.
---------------	-------------------------------

### 11.16.3.5 add\_filter()

```
void fsw::monitor::add_filter (
    const monitor_filter & filter )
```

Add a path filter.

This function adds a `monitor_filter` instance instance to the filter list.

#### Parameters

<i>filter</i>	The filter to add.
---------------	--------------------



### 11.16.3.6 filter\_flags()

```
vector< fsw_event_flag > fsw::monitor::filter_flags (
    const event & evt ) const [protected]
```

Filter event types.

This function filters the event types of an event leaving only the types allowed by the configured filters.

#### Parameters

<i>evt</i>	The event whose types must be filtered.
------------	---

#### Returns

A vector containing the acceptable events.

### 11.16.3.7 get\_context()

```
void * fsw::monitor::get_context ( ) const
```

Get the pointer to the context data.

This function gets the pointer to the context data that is passed to the callback by the monitor.

#### Returns

The pointer to the context data.

### 11.16.3.8 get\_property()

```
string fsw::monitor::get_property (
    std::string name )
```

Gets the value of a property.

This method gets the value of the property *name*. If the property *name* is not set, this method returns an empty string.

#### Parameters

<i>name</i>	The name of the property.
-------------	---------------------------

**Returns**

The value of the property.

**11.16.3.9 is\_running()**

```
bool fsw::monitor::is_running ( )
```

Check whether the monitor is running.

State is checked thread-safely locking on [monitor::run\\_mutex](#).

**Returns**

`true` if the monitor is running, `false` otherwise.

**11.16.3.10 notify\_events()**

```
void fsw::monitor::notify_events (
    const std::vector< event > & events ) const [protected]
```

Notify change events.

This function notifies change events using the provided callback.

**See also**

[monitor\(\)](#)

**11.16.3.11 notify\_overflow()**

```
void fsw::monitor::notify_overflow (
    const std::string & path ) const [protected]
```

Notify an overflow event.

This function notifies an overflow event using the provided callback.

**Warning**

Experiencing an overflow and the ability to notify it is an implementation-defined behaviour.

**See also**

[monitor\(\)](#)

### 11.16.3.12 on\_stop()

```
void fsw::monitor::on_stop ( ) [protected], [virtual]
```

Execute an implementation-specific stop handler.

This function is executed by the [stop\(\)](#) method, after requesting the monitor to stop. This handler is required if the thread running [run\(\)](#) is not able to preemptively stop its execution by checking the [monitor::should\\_stop](#) flag.

See also

[stop\(\)](#)

Reimplemented in [fsw::fsevents\\_monitor](#).

### 11.16.3.13 run()

```
virtual void fsw::monitor::run ( ) [protected], [pure virtual]
```

Execute monitor loop.

This function implements the monitor event watching logic. This function is called from [start\(\)](#) and it is executed on its thread. This function should *block* until the monitoring loop terminates: when it returns, the monitor is marked as stopped.

This function should cooperatively check the [monitor::should\\_stop](#) field locking [monitor::run\\_mutex](#) and return if set to `true`.

See also

[start\(\)](#)

[stop\(\)](#)

Implemented in [fsw::fen\\_monitor](#), [fsw::inotify\\_monitor](#), [fsw::kqueue\\_monitor](#), [fsw::windows\\_monitor](#), [fsw::fsevents\\_monitor](#), and [fsw::poll\\_monitor](#).

### 11.16.3.14 set\_allow\_overflow()

```
void fsw::monitor::set_allow_overflow (
    bool overflow )
```

Notify buffer overflows as change events.

If this flag is set, the monitor will report a monitor buffer overflow as a change event of type `fsw_event_flag::Overflow`.

Warning

The behaviour associated with this flag depends on the implementation.

## Parameters

<i>overflow</i>	true if overflow should be notified, false otherwise.
-----------------	---

**11.16.3.15 set\_context()**

```
void fsw::monitor::set_context (
    void * context )
```

Set the context data.

This function sets the pointer to the *context data*. The context data is opaque data that the monitor passes to the event callback.

## Warning

The monitor stores the pointer to the context data throughout its life. The caller must ensure it points to valid data until the monitor is running.

## Parameters

<i>context</i>	The pointer to the context data.
----------------	----------------------------------

**11.16.3.16 set\_directory\_only()**

```
void fsw::monitor::set_directory_only (
    bool directory_only )
```

Watch directories only.

This function sets the directory only flag to the specified value. If this flag is set, then the monitor will only watch directories during a recursive scan. This functionality is only supported by monitors whose backend fires change events on a directory when one its children is changed. If a monitor backend does not support this functionality, the flag is ignored.

## Warning

The behaviour associated with this flag depends on the implementation.

## Parameters

<i>directory_only</i>	true if only directories should be watched, false otherwise.
-----------------------	--

### 11.16.3.17 set\_event\_type\_filters()

```
void fsw::monitor::set_event_type_filters (
    const std::vector< fsw_event_type_filter > & filters )
```

Set the event type filters.

This function sets the list of event type filters, substituting existing filters if any.

#### Parameters

<i>filters</i>	The filters to set.
----------------	---------------------

### 11.16.3.18 set\_filters()

```
void fsw::monitor::set_filters (
    const std::vector< monitor_filter > & filters )
```

Set the path filters.

This function sets the list of path filters, substituting existing filters if any.

#### Parameters

<i>filters</i>	The filters to set.
----------------	---------------------

### 11.16.3.19 set\_fire\_idle\_event()

```
void fsw::monitor::set_fire_idle_event (
    bool fire_idle_event )
```

Sets the *fire idle event* flag.

When `true`, the *fire idle event* flag instructs the monitor to fire a fake event at the event of an *idle* cycle. An idle cycle is a period of time whose length is 110% of the `monitor::latency` where no change events were detected.

#### Parameters

<i>fire_idle_event</i>	<code>true</code> if idle events should be fired, <code>false</code> otherwise.
------------------------	---

### 11.16.3.20 set\_follow\_symlinks()

```
void fsw::monitor::set_follow_symlinks (
    bool follow )
```

Follow symlinks.

This function sets the `follow_symlinks` flag of the monitor to indicate whether the monitor should follow symbolic links or observe the links themselves.

#### Warning

The behaviour associated with this flag depends on the implementation.

#### Parameters

<i>follow</i>	<code>true</code> if symbolic links should be followed, <code>false</code> otherwise.
---------------	---

#### 11.16.3.21 `set_latency()`

```
void fsw::monitor::set_latency (
    double latency )
```

Sets the latency.

This method sets the *latency* of the monitor to `latency`. The latency is a positive number that indicates to a monitor implementation how often events must be retrieved or waited for: the shortest the latency, the quicker events are processed.

#### Warning

The behaviour associated with this flag depends on the implementation.

#### Parameters

<i>latency</i>	The latency value.
----------------	--------------------

#### 11.16.3.22 `set_properties()`

```
void fsw::monitor::set_properties (
    const std::map< std::string, std::string > options )
```

Sets the custom properties.

This method *replaces* all the existing properties using the pairs contained into `options`.

#### Parameters

<i>options</i>	The map containing the properties to set.
----------------	---

### 11.16.3.23 set\_property()

```
void fsw::monitor::set_property (
    const std::string & name,
    const std::string & value )
```

Sets a custom property.

This method sets the custom property *name* to *value*.

#### Parameters

<i>name</i>	The name of the property.
<i>value</i>	The value of the property.

### 11.16.3.24 set\_recursive()

```
void fsw::monitor::set_recursive (
    bool recursive )
```

Recursively scan subdirectories.

This function sets the recursive flag of the monitor to indicate whether the monitor should recursively observe the contents of directories. The behaviour associated with this flag is an implementation-specific detail. This class only stores the value of the flag.

#### Warning

The behaviour associated with this flag depends on the implementation.

#### Parameters

<i>recursive</i>	true if directories should be recursively, false otherwise.
------------------	---

### 11.16.3.25 set\_watch\_access()

```
void fsw::monitor::set_watch_access (
    bool access )
```

Monitor file access.

#### Warning

The ability of monitoring file access depends on a monitor implementation.

### 11.16.3.26 start()

```
void fsw::monitor::start ( )
```

Start the monitor.

The monitor status is marked as *running* and it starts watching for change events. This function performs the following tasks:

- Atomically marks the thread state as *running*, locking on [monitor::run\\_mutex](#).
- Calls the [run\(\)](#) function: the [monitor::run\\_mutex](#) is **not** locked during this call.
- When [run\(\)](#) returns, it atomically marks the thread state as *stopped*, locking on [monitor::run\\_mutex](#).

This call does *not* return until the monitor is stopped and events are notified from its thread.

State changes are performed thread-safely locking on [monitor::run\\_mutex](#).

See also

[run\(\)](#)  
[stop\(\)](#)

### 11.16.3.27 stop()

```
void fsw::monitor::stop ( )
```

Stop the monitor.

This function asks the monitor to stop. Since [start\(\)](#) is designed to execute the monitoring loop in its thread and to not return until the monitor is stopped, [stop\(\)](#) is designed to be called from another thread. [stop\(\)](#) is a cooperative signal that must be handled in an implementation-specific way in the [run\(\)](#) function.

State changes are performed thread-safely locking on [monitor::run\\_mutex](#).

See also

[run\(\)](#)  
[start\(\)](#)

## 11.16.4 Member Data Documentation



#### 11.16.4.1 callback

`FSW_EVENT_CALLBACK* fsw::monitor::callback` [protected]

Callback to which change events should be notified.

See also

[monitor::monitor\(\)](#)

#### 11.16.4.2 fire\_idle\_event

`bool fsw::monitor::fire_idle_event = false` [protected]

If `true`, the monitor will notify an event when idle.

An idle cycle is long as 110% of the [monitor::latency](#) value.

#### 11.16.4.3 paths

`std::vector<std::string> fsw::monitor::paths` [protected]

List of paths to watch.

See also

[monitor::monitor\(\)](#)

#### 11.16.4.4 properties

`std::map<std::string, std::string> fsw::monitor::properties` [protected]

Map of custom properties.

See also

[monitor::set\\_property\(\)](#)  
[monitor::set\\_properties\(\)](#)

The documentation for this class was generated from the following files:

- [libfswatch/c++/monitor.hpp](#)
- [libfswatch/c++/monitor.cpp](#)

## 11.17 fsw::monitor\_factory Class Reference

Object factory class for [fsw::monitor](#) instances.

```
#include <monitor.hpp>
```

### Public Member Functions

- [monitor\\_factory](#) (const [monitor\\_factory](#) &orig)=delete
- [monitor\\_factory](#) & [operator=](#) (const [monitor\\_factory](#) &that)=delete

### Static Public Member Functions

- static [monitor](#) \* [create\\_monitor](#) ([fsw\\_monitor\\_type](#) type, std::vector< std::string > paths, [FSW\\_EVENT\\_CALLBACK](#) \*callback, void \*context=nullptr)  
*Creates a monitor of the specified `type`.*
- static [monitor](#) \* [create\\_monitor](#) (const std::string &name, std::vector< std::string > paths, [FSW\\_EVENT\\_CALLBACK](#) \*callback, void \*context=nullptr)  
*Creates a monitor whose type is the specified by `name`.*
- static std::vector< std::string > [get\\_types](#) ()  
*Get the available monitor types.*
- static bool [exists\\_type](#) (const std::string &name)  
*Checks whether a monitor of the type specified by `name` exists.*
- static bool [exists\\_type](#) (const [fsw\\_monitor\\_type](#) &type)  
*Checks whether a monitor of the type specified `type`.*
- static void [register\\_creator](#) (const std::string &name, FSW\_FN\_MONITOR\_CREATOR creator)  
*Registers a `creator` for the specified monitor type `name`.*
- static void [register\\_creator\\_by\\_type](#) (const [fsw\\_monitor\\_type](#) &type, FSW\_FN\_MONITOR\_CREATOR creator)  
*Registers a `creator` for the specified monitor `type`.*

### 11.17.1 Detailed Description

Object factory class for [fsw::monitor](#) instances.

Since multiple monitor implementations exist and the caller potentially ignores which monitors will be available at run time, there must exist a way to query the API for the list of available monitor and request a particular instance. The [fsw::monitor\\_factory](#) is an object factory class that provides basic monitor *registration* and *discovery* functionality: API clients can query the monitor registry to get a list of available monitors and get an instance of a monitor either by *type* or by *name*.

In order for monitor types to be visible to the factory they have to be *registered*. Currently, monitor implementations can be registered using the [register\\_creator\(\)](#) and [register\\_creator\\_by\\_type\(\)](#), or using:

- The [fsw::monitor\\_registrant](#) helper class.
- The [REGISTER\\_MONITOR](#) macro.
- The [REGISTER\\_MONITOR\\_IMPL](#) macro.

The same monitor type cannot be used to register multiple monitor implementations. No checks are in place to detect this situation and the registration will succeed; however, the registration process of multiple monitor implementations for the same monitor type is *not* deterministic.

## 11.17.2 Member Function Documentation

### 11.17.2.1 create\_monitor() [1/2]

```
static monitor* fsw::monitor_factory::create_monitor (
    fsw_monitor_type type,
    std::vector< std::string > paths,
    FSW_EVENT_CALLBACK * callback,
    void * context = nullptr ) [static]
```

Creates a monitor of the specified type.

The other parameters are forwarded to the [fsw::monitor\(\)](#) constructor.

#### Parameters

<i>type</i>	The monitor type.
<i>paths</i>	The paths to watch.
<i>callback</i>	The callback to invoke during the notification of a change event.

#### Returns

The newly created monitor.

#### Exceptions

<a href="#"><i>libfsw_exception</i></a>	if a monitor of the specified type cannot be found.
---	---

#### See also

[fsw::monitor\(\)](#)

### 11.17.2.2 create\_monitor() [2/2]

```
static monitor* fsw::monitor_factory::create_monitor (
    const std::string & name,
    std::vector< std::string > paths,
    FSW_EVENT_CALLBACK * callback,
    void * context = nullptr ) [static]
```

Creates a monitor whose type is the specified by name.

The other parameters are forwarded to the [fsw::monitor\(\)](#) constructor.

**Parameters**

<i>name</i>	The monitor type.
<i>paths</i>	The paths to watch.
<i>callback</i>	The callback to invoke during the notification of a change event.

**Returns**

The newly created monitor.

**Exceptions**

<a href="#"><i>libfsw_exception</i></a>	if a monitor of the type specified by <i>name</i> cannot be found.
---	--

**See also**

[`fsw::monitor\(\)`](#)

**11.17.2.3 exists\_type()** [1/2]

```
static bool fsw::monitor_factory::exists_type (
    const std::string & name ) [static]
```

Checks whether a monitor of the type specified by *name* exists.

**Returns**

`true` if *name* specifies a valid monitor type, `false` otherwise.

**Parameters**

<i>name</i>	The name of the monitor type to look for.
-------------	---

**Returns**

`true` if the type *name* exists, `false` otherwise.

**11.17.2.4 exists\_type()** [2/2]

```
bool fsw::monitor_factory::exists_type (
    const fsw_monitor_type & type ) [static]
```

Checks whether a monitor of the type specified *type*.

**Parameters**

<i>type</i>	The type of the monitor to look for.
-------------	--------------------------------------

**Returns**

true if name specifies a valid monitor type, false otherwise.

**11.17.2.5 get\_types()**

```
vector< string > fsw::monitor_factory::get_types ( ) [static]
```

Get the available monitor types.

**Returns**

A vector with the available monitor types.

**11.17.2.6 register\_creator()**

```
void fsw::monitor_factory::register_creator (
    const std::string & name,
    FSW_FN_MONITOR_CREATOR creator ) [static]
```

Registers a creator for the specified monitor type name.

**Parameters**

<i>name</i>	The name of the monitor type.
<i>creator</i>	The monitor creator function.

**11.17.2.7 register\_creator\_by\_type()**

```
void fsw::monitor_factory::register_creator_by_type (
    const fsw_monitor_type & type,
    FSW_FN_MONITOR_CREATOR creator ) [static]
```

Registers a creator for the specified monitor type.

**Parameters**

<i>type</i>	The monitor type.
<i>creator</i>	The monitor creator function.

The documentation for this class was generated from the following files:

- libfswatch/c++/[monitor.hpp](#)
- libfswatch/c++/[monitor.cpp](#)

## 11.18 fsw::monitor\_filter Struct Reference

Path filters used to accept or reject file change events.

```
#include <filter.hpp>
```

### Static Public Member Functions

- static `std::vector< monitor\_filter > read\_from\_file` (`const std::string &path, void(*err_handler)(std::string)=nullptr`)  
*Load filters from the specified file.*

### Public Attributes

- `std::string text`  
*Regular expression used to match the paths.*
- `fsw\_filter\_type type`  
*Filter type.*
- `bool case\_sensitive`  
*Flag indicating whether [monitor\\_filter::text](#) is a case sensitive regular expression.*
- `bool extended`  
*Flag indicating whether [monitor\\_filter::text](#) is an extended regular expression.*

### 11.18.1 Detailed Description

Path filters used to accept or reject file change events.

A path filter is a regular expression used to accept or reject file change events based on the value of their path. A filter has the following characteristics:

- It has a *regular expression* ([monitor\\_filter::text](#)), used to match the paths.
- It can be an *inclusion* or an *exclusion* filter ([monitor\\_filter::type](#)).
- It can be case *sensitive* or *insensitive* ([monitor\\_filter::case\\_sensitive](#)).
- It can be an *extended* regular expression ([monitor\\_filter::extended](#)).

Further information about how filtering works in `libfswatch` can be found in [Path Filtering](#).

### 11.18.2 Member Function Documentation

## 11.18.2.1 read\_from\_file()

```
vector< monitor_filter > fsw::monitor_filter::read_from_file (
    const std::string & path,
    void(*) (std::string) err_handler = nullptr ) [static]
```

Load filters from the specified file.

Filters can be loaded from a text file containing one filter per line. A filter has the following structure:

- It is validated by the following regular expression: `^ ([+-]) ([ei]*) (.+)$`
- The first character is the filter type: + if it is an *inclusion* filter, – if it is an *exclusion* filter.
- An optional list of flags:
  - e if it is an *extended* regular expression.
  - i if it is a *case insensitive* regular expression.
- A space.
- The filter regular expression text.

Parsing errors are notified through an optional error handler. The valid filters are returned in a vector.

## Parameters

<i>path</i>	The path of the file to read filters from.
<i>err_handler</i>	An optional error handler.

## Returns

A vector containing the valid filters.

## Exceptions

<i>invalid_argument</i>	If the specified path cannot be opened.
-------------------------	---

## 11.18.3 Member Data Documentation

## 11.18.3.1 extended

```
bool fsw::monitor_filter::extended
```

Flag indicating whether `monitor_filter::text` is an extended regular expression.

Further information about extended regular expressions can be found here:

[http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1\\_chap09.html#tag\\_09\\_04](http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html#tag_09_04)

### 11.18.3.2 text

```
std::string fsw::monitor_filter::text
```

Regular expression used to match the paths.

Further information about regular expressions can be found here:

[http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1\\_chap09.html](http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html)

The documentation for this struct was generated from the following files:

- [libfswatch/c++/filter.hpp](#)
- [libfswatch/c++/filter.cpp](#)

## 11.19 fsw::monitor\_registrant< M > Class Template Reference

Helper class to register monitor factories.

```
#include <monitor.hpp>
```

### Public Member Functions

- [monitor\\_registrant](#) (const std::string &name, const [fsw\\_monitor\\_type](#) &type)  
*Constructs a monitor registrant for the specified type.*

### 11.19.1 Detailed Description

```
template<class M>
class fsw::monitor_registrant< M >
```

Helper class to register monitor factories.

The constructor of this class perform the registration of the given (name, type) pair in the [monitor\\_factory](#) registry. This class is used by the REGISTER\_MONITOR and REGISTER\_MONITOR\_IMPL macros.

See also

[fsw::monitor\\_factory](#)

### 11.19.2 Constructor & Destructor Documentation

#### 11.19.2.1 monitor\_registrant()

```
template<class M >
fsw::monitor_registrant< M >::monitor_registrant (
    const std::string & name,
    const fsw\_monitor\_type & type ) [inline]
```

Constructs a monitor registrant for the specified type.



## Parameters

<i>name</i>	The name of the type whose factory is being registered.
<i>type</i>	The type whose factory is being registered.

The documentation for this class was generated from the following file:

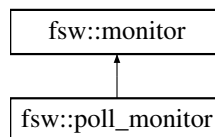
- libfswatch/c++/[monitor.hpp](#)

## 11.20 fsw::poll\_monitor Class Reference

`stat()`-based monitor.

```
#include <poll_monitor.hpp>
```

Inheritance diagram for `fsw::poll_monitor`:



### Classes

- struct [poll\\_monitor\\_data](#)

### Public Member Functions

- [poll\\_monitor](#) (`std::vector< std::string > paths`, `FSW_EVENT_CALLBACK *callback`, `void *context=nullptr`)  
*Constructs an instance of this class.*
- virtual [~poll\\_monitor](#) ()  
*Destroys an instance of this class.*

### Protected Member Functions

- void [run](#) ()  
*Execute monitor loop.*

### Additional Inherited Members

#### 11.20.1 Detailed Description

`stat()`-based monitor.

This monitor uses the `stat()` function to periodically check the observed paths and detect changes.

## 11.20.2 Member Function Documentation

### 11.20.2.1 run()

```
void fsw::poll_monitor::run ( ) [protected], [virtual]
```

Execute monitor loop.

This function implements the monitor event watching logic. This function is called from [start\(\)](#) and it is executed on its thread. This function should *block* until the monitoring loop terminates: when it returns, the monitor is marked as stopped.

This function should cooperatively check the [monitor::should\\_stop](#) field locking [monitor::run\\_mutex](#) and return if set to `true`.

See also

[start\(\)](#)  
[stop\(\)](#)

Implements [fsw::monitor](#).

The documentation for this class was generated from the following files:

- libfswatch/c++/poll\_monitor.hpp
- libfswatch/c++/poll\_monitor.cpp

## 11.21 fsw::poll\_monitor::poll\_monitor\_data Struct Reference

### Public Attributes

- [fsw\\_hash\\_map](#)< string, poll\_monitor::watched\_file\_info > **tracked\_files**

The documentation for this struct was generated from the following file:

- libfswatch/c++/poll\_monitor.cpp

## 11.22 fsw::win\_error\_message Class Reference

Helper class to get the system-defined error message for a Microsoft Windows' error code.

```
#include <win_error_message.hpp>
```

## Public Member Functions

- [win\\_error\\_message](#) (DWORD error\_code)  
*Constructs an error message using the specified error\_code.*
- [win\\_error\\_message](#) ()  
*Constructs an error message using the last error code of the calling thread, retrieved with a call to `GetLastError()`.*
- DWORD [get\\_error\\_code](#) () const  
*Gets the error code.*
- std::wstring [get\\_message](#) () const  
*Gets the system-defined error message.*
- [operator std::wstring](#) () const  
*Gets this system-defined error message.*

## Static Public Member Functions

- static [win\\_error\\_message current](#) ()  
*Constructs an instance of this class using the last error code of the calling thread, returned by a call to `GetLastError()`.*

### 11.22.1 Detailed Description

Helper class to get the system-defined error message for a Microsoft Windows' error code.

This class uses the `FormatMessage()` API to returns a `std::wstring` instance containing the system-defined error message for a Microsoft Windows' error code.

### 11.22.2 Constructor & Destructor Documentation

#### 11.22.2.1 win\_error\_message() [1/2]

```
fsw::win_error_message::win_error_message (
    DWORD error_code )
```

Constructs an error message using the specified `error_code`.

#### Parameters

<code>error_code</code>	The error code.
-------------------------	-----------------

#### 11.22.2.2 win\_error\_message() [2/2]

```
fsw::win_error_message::win_error_message ( )
```

Constructs an error message using the last error code of the calling thread, retrieved with a call to `GetLastError()`.

See also

[current\(\)](#)

### 11.22.3 Member Function Documentation

#### 11.22.3.1 `current()`

```
static win_error_message fsw::win_error_message::current ( ) [static]
```

Constructs an instance of this class using the last error code of the calling thread, returned by a call to `GetLastError()`.

See also

[win\\_error\\_message\(\)](#)

#### 11.22.3.2 `get_error_code()`

```
DWORD fsw::win_error_message::get_error_code ( ) const
```

Gets the error code.

Returns

The error code.

#### 11.22.3.3 `get_message()`

```
std::wstring fsw::win_error_message::get_message ( ) const
```

Gets the system-defined error message.

The system-defined error message is retrieved with a call to `FormatMessage` with the `FORMAT_MESSAGE_FROM_SYSTEM` formatting option.

Returns

The error message.

## 11.22.3.4 operator std::wstring()

```
fsw::win_error_message::operator std::wstring ( ) const
```

Gets the system-defined error message.

See also

[get\\_message\(\)](#)

The documentation for this class was generated from the following file:

- libfswatch/c++/windows/[win\\_error\\_message.hpp](#)

## 11.23 fsw::win\_flag\_type Struct Reference

## Public Attributes

- DWORD **action**
- vector< [fsw\\_event\\_flag](#) > **types**

The documentation for this struct was generated from the following file:

- libfswatch/c++/windows/win\_directory\_change\_event.cpp

## 11.24 fsw::win\_handle Class Reference

A RAII wrapper around Microsoft Windows `HANDLE`.

```
#include <win_handle.hpp>
```

## Public Member Functions

- [win\\_handle](#) ()  
*Constructs an instance wrapping `INVALID_HANDLE_VALUE`.*
- [win\\_handle](#) (HANDLE handle)  
*Constructs an instance wrapping `handle`.*
- virtual [~win\\_handle](#) ()  
*Destructs a handle.*
- [operator HANDLE](#) () const  
*Returns the handle value as `HANDLE` instance.*
- bool [is\\_valid](#) () const  
*Checks whether the handle is valid.*
- [win\\_handle](#) (const [win\\_handle](#) &)=delete  
*Deleted copy constructor.*
- [win\\_handle](#) & [operator=](#) (const [win\\_handle](#) &)=delete  
*Deleted copy assignment operator.*
- [win\\_handle](#) ([win\\_handle](#) &&other) noexcept  
*Move constructor.*
- [win\\_handle](#) & [operator=](#) ([win\\_handle](#) &&other) noexcept  
*Move assignment operator.*
- [win\\_handle](#) & [operator=](#) (const HANDLE &handle)  
*Assigns a `handle` to the current instance.*

## Static Public Member Functions

- static bool [is\\_valid](#) (const HANDLE &handle)  
*Checks whether `handle` is valid.*

### 11.24.1 Detailed Description

A RAI wrapper around Microsoft Windows `HANDLE`.

This class is a movable, non-copyable RAI wrapper on `HANDLE`.

### 11.24.2 Constructor & Destructor Documentation

#### 11.24.2.1 ~win\_handle()

```
virtual fsw::win_handle::~~win_handle ( ) [virtual]
```

Destructs a handle.

If the handle is valid ([is\\_valid\(\)](#)) it is closed invoking `CloseHandle()`.

See also

[is\\_valid\(const HANDLE &\)](#)

#### 11.24.2.2 win\_handle()

```
fsw::win_handle::win_handle (
    win_handle && other ) [noexcept]
```

Move constructor.

The move constructors moves the handle value wrapped by `other` to the target instance. The handle value in `other` is set to `INVALID_HANDLE_VALUE`. The previously wrapped instance is closed invoking `CloseHandle` if it is valid.

Parameters

<i>other</i>	The handle to move.
--------------	---------------------

### 11.24.3 Member Function Documentation

## 11.24.3.1 is\_valid() [1/2]

```
static bool fsw::win_handle::is_valid (
    const HANDLE & handle ) [static]
```

Checks whether handle is valid.

A handle is valid is if its value is not null and if is not INVALID\_HANDLE\_VALUE.

## Parameters

<i>handle</i>	The handle to check.
---------------	----------------------

## Returns

Returns true if handle is valid, false otherwise.

## 11.24.3.2 is\_valid() [2/2]

```
bool fsw::win_handle::is_valid ( ) const
```

Checks whether the handle is valid.

## Returns

Returns true if the handle is valid, false otherwise.

## See also

[is\\_valid\(\)](#)

## 11.24.3.3 operator=() [1/2]

```
win_handle& fsw::win_handle::operator= (
    win_handle && other ) [noexcept]
```

Move assignment operator.

The move assignment operator moves the handle value wrapped by `other` to the target instance. The handle value in `other` is set to INVALID\_HANDLE\_VALUE. The previously wrapped instance is closed invoking `CloseHandle` if it is valid.

## Parameters

<i>other</i>	The handle to move.
--------------	---------------------

#### 11.24.3.4 operator=() [2/2]

```
win_handle& fsw::win_handle::operator= (
    const HANDLE & handle )
```

Assigns a `handle` to the current instance.

The previously wrapped instance is closed invoking `CloseHandle` if it is valid.

##### Parameters

<code>handle</code>	The handle value to assign to the current instance.
---------------------	---

The documentation for this class was generated from the following file:

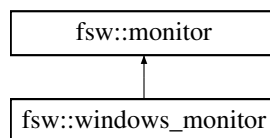
- `libfswatch/c++/windows/win_handle.hpp`

## 11.25 fsw::windows\_monitor Class Reference

Windows monitor.

```
#include <windows_monitor.hpp>
```

Inheritance diagram for `fsw::windows_monitor`:



### Public Member Functions

- `windows_monitor` (`std::vector< std::string > paths`, `FSW_EVENT_CALLBACK *callback`, `void *context=nullptr`)  
Constructs an instance of this class.
- `virtual ~windows_monitor ()`  
Destroys an instance of this class.

### Protected Member Functions

- `void run ()`  
Executes the monitor loop.



## Additional Inherited Members

### 11.25.1 Detailed Description

Windows monitor.

This monitor is built upon the `ReadDirectoryChanges` API of the Windows operating systems.

### 11.25.2 Member Function Documentation

#### 11.25.2.1 `run()`

```
void fsw::windows_monitor::run ( ) [protected], [virtual]
```

Executes the monitor loop.

This call does not return until the monitor is stopped.

See also

[stop\(\)](#)

Implements [fsw::monitor](#).

The documentation for this class was generated from the following file:

- [libfswatch/c++/windows\\_monitor.hpp](#)



## Chapter 12

# File Documentation

### 12.1 libfswatch/c++/event.hpp File Reference

Header of the `fsw::event` class.

```
#include <string>
#include <ctime>
#include <vector>
#include <iostream>
#include "../c/cevent.h"
```

#### Classes

- class `fsw::event`

*Type representing a file change event.*

#### Namespaces

- `fsw`

*Main namespace of `libfswatch`.*

#### Functions

- `std::ostream & fsw::operator<< (std::ostream &out, const fsw_event_flag flag)`

*Overload of the << operator to print an event using `iostreams`.*

### 12.1.1 Detailed Description

Header of the `fsw::event` class.

#### Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.2 libfswatch/c++/fen\_monitor.hpp File Reference

Solaris/Illumos monitor.

```
#include "monitor.hpp"
#include <string>
#include <vector>
```

### Classes

- class `fsw::fen_monitor`  
*Solaris/Illumos monitor.*

### Namespaces

- `fsw`  
*Main namespace of `libfswatch`.*

### 12.2.1 Detailed Description

Solaris/Illumos monitor.

#### Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.3 libfswatch/c++/filter.hpp File Reference

Header of the `fsw::monitor_filter` class.

```
#include <string>
#include "../c/cfilter.h"
#include <vector>
```

### Classes

- struct `fsw::monitor_filter`  
*Path filters used to accept or reject file change events.*

### Namespaces

- `fsw`  
*Main namespace of `libfswatch`.*

### Typedefs

- typedef struct `fsw::monitor_filter` `fsw::monitor_filter`  
*Path filters used to accept or reject file change events.*

#### 12.3.1 Detailed Description

Header of the `fsw::monitor_filter` class.

This header file defines the `fsw::monitor_filter` class, a type that represents a path filter.

#### Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.4 libfswatch/c++/fsevents\_monitor.hpp File Reference

OS X FSEvents monitor.

```
#include "monitor.hpp"
#include <CoreServices/CoreServices.h>
```

### Classes

- class [fsw::fsevents\\_monitor](#)  
*OS X FSEvents monitor.*

### Namespaces

- [fsw](#)  
*Main namespace of libfswatch.*

### 12.4.1 Detailed Description

OS X FSEvents monitor.

#### Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.5 libfswatch/c++/inotify\_monitor.hpp File Reference

Solaris/Illumos monitor.

```
#include "monitor.hpp"
#include <sys/inotify.h>
#include <string>
#include <vector>
#include <sys/stat.h>
```

## Classes

- class [fsw::inotify\\_monitor](#)  
*Solaris/Illumos monitor.*

## Namespaces

- [fsw](#)  
*Main namespace of `libfswatch`.*

### 12.5.1 Detailed Description

Solaris/Illumos monitor.

#### Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.6 libfswatch/c++/kqueue\_monitor.hpp File Reference

kqueue monitor.

```
#include "monitor.hpp"
#include <string>
#include <vector>
#include <sys/stat.h>
#include <sys/event.h>
```

## Classes

- class [fsw::kqueue\\_monitor](#)  
*Solaris/Illumos monitor.*

## Namespaces

- [fsw](#)

*Main namespace of `libfswatch`.*

### 12.6.1 Detailed Description

`kqueue` monitor.

#### Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.7 `libfswatch/c++/libfswatch_exception.hpp` File Reference

Base exception of the `libfswatch` library.

```
#include "../c/error.h"
#include <exception>
#include <string>
```

## Classes

- class [fsw::libfsw\\_exception](#)

*Base exception of the `libfswatch` library.*

## Namespaces

- [fsw](#)

*Main namespace of `libfswatch`.*



### 12.7.1 Detailed Description

Base exception of the `libfswatch` library.

#### Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.8 libfswatch/c++/libfswatch\_map.hpp File Reference

Header defining the associative container used by the library.

```
#include <map>
```

### Namespaces

- [fsw](#)

*Main namespace of `libfswatch`.*

### Typedefs

- `template<typename K, typename V >`  
`using fsw::fsw\_hash\_map = std::map< K, V >`  
*Default associative container type used by `libfswatch`.*

### 12.8.1 Detailed Description

Header defining the associative container used by the library.

#### Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.9 libfswatch/c++/libfswatch\_set.hpp File Reference

Header defining the default set type used by the library.

```
#include <set>
```

### Namespaces

- [fsw](#)  
*Main namespace of libfswatch.*

### Typedefs

- `template<typename K >`  
`using fsw::fsw\_hash\_set = std::set< K >`  
*Default set type used by libfswatch.*

### 12.9.1 Detailed Description

Header defining the default set type used by the library.

#### Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.10 libfswatch/c++/monitor.hpp File Reference

Header of the [fsw::monitor](#) class.

```
#include "filter.hpp"  
#include <vector>  
#include <string>  
#include <mutex>  
#include <atomic>  
#include <chrono>  
#include <map>  
#include "event.hpp"  
#include "../c/cmonitor.h"
```

## Classes

- class [fsw::monitor](#)  
*Base class of all monitors.*
- class [fsw::monitor\\_factory](#)  
*Object factory class for [fsw::monitor](#) instances.*
- class [fsw::monitor\\_registrant< M >](#)  
*Helper class to register monitor factories.*

## Namespaces

- [fsw](#)  
*Main namespace of `libfswatch`.*

## Macros

- `#define REGISTER_MONITOR(classname, monitor_type)`
- `#define REGISTER_MONITOR_IMPL(classname, monitor_type) const monitor_registrant<classname> classname::monitor_factory_registrant(#classname, monitor_type);`

## Typedefs

- `typedef void fsw::FSW\_EVENT\_CALLBACK(const std::vector< event > &, void *)`  
*Function definition of an event callback.*
- `typedef monitor *(* fsw::FSW\_FN\_MONITOR\_CREATOR) (std::vector< std::string > paths, FSW_EVENT_CALLBACK *callback, void *context)`

### 12.10.1 Detailed Description

Header of the [fsw::monitor](#) class.

This header file defines the [fsw::monitor](#) class, the base type of a `libfswatch` monitor and fundamental type of the C++ API.

If `HAVE_CXX_MUTEX` is defined, this header includes `<mutex>`.

#### Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.10.2 Macro Definition Documentation

### 12.10.2.1 REGISTER\_MONITOR

```
#define REGISTER_MONITOR(  
    classname,  
    monitor_type )
```

#### Value:

```
private: \  
static const monitor_registrant<classname> monitor_factory_registrant;
```

This macro is used to simplify the registration process of a monitor type. Since registration of a monitor type is usually performed once, a static private instance `monitor_factory_registrant` of the `monitor_registrant` class is declared by this macro in the enclosing class.

Beware that since this macro adds a private qualifier, every field declared after it must be correctly qualified.

The use of the `REGISTER_MONITOR` macro in a class must always be matched by a corresponding use of the `REGISTER_MONITOR_IMPL` macro in the class definition.

To register class `my_monitor` with type `my_type`, use the `REGISTER_MONITOR` macro as in the following example:

```
[my_class.h] class my_monitor { REGISTER_MONITOR(my_monitor, my_monitor_type); ... };
```

### 12.10.2.2 REGISTER\_MONITOR\_IMPL

```
#define REGISTER_MONITOR_IMPL(  
    classname,  
    monitor_type ) const monitor_registrant<classname> classname::monitor_factory_↵  
registrant(#classname, monitor_type);
```

This macro is used to simplify the registration process of a monitor type. Since registration of a monitor type is usually performed once, a static private instance `monitor_factory_registrant` of the `monitor_registrant` class is defined in the monitor class specified by `classname`.

A invocation of the `REGISTER_MONITOR_IMPL` macro must always be matched by an invocation of the `REGISTER_MONITOR` macro in the class declaration.

To register class `my_monitor` with type `my_type`, use the `REGISTER_MONITOR` macro as in the following example:

```
[my_class.cpp]
```

```
REGISTER_MONITOR_IMPL(my_monitor, my_monitor_type);
```

## 12.11 libfswatch/c++/path\_utils.hpp File Reference

Header defining utility functions to manipulate paths.

```
#include <string>
#include <vector>
#include <sys/stat.h>
```

### Namespaces

- [fsw](#)

*Main namespace of libfswatch.*

### Functions

- `std::vector< std::string > fsw::get_directory_children (const std::string &path)`  
*Gets a vector of direct directory children.*
- `bool fsw::read_link_path (const std::string &path, std::string &link_path)`  
*Resolves a path name.*
- `bool fsw::lstat_path (const std::string &path, struct stat &fd_stat)`  
*Wraps a `lstat (path, fd_stat)` call that invokes `perror()` if it fails.*
- `bool fsw::stat_path (const std::string &path, struct stat &fd_stat)`  
*Wraps a `stat (path, fd_stat)` call that invokes `perror()` if it fails.*

#### 12.11.1 Detailed Description

Header defining utility functions to manipulate paths.

#### Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.12 libfswatch/c++/poll\_monitor.hpp File Reference

`stat()` based monitor.

```
#include "monitor.hpp"
#include <sys/stat.h>
#include <ctime>
```

### Classes

- class [fsw::poll\\_monitor](#)  
*`stat()` -based monitor.*

### Namespaces

- [fsw](#)  
*Main namespace of `libfswatch`.*

### 12.12.1 Detailed Description

`stat()` based monitor.

#### Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.13 libfswatch/c++/string/string\_utils.hpp File Reference

Header of the [fsw::string\\_utils](#) namespace.

```
#include <cstdint>
#include <string>
```

## Namespaces

- [fsw](#)  
*Main namespace of `libfswatch`.*
- [fsw::string\\_utils](#)  
*This namespace contains string manipulation functions.*

## Functions

- string [fsw::string\\_utils::string\\_from\\_format](#) (const char \*format,...)  
*Create a `std::string` using a `printf()` format and varargs.*
- string [fsw::string\\_utils::vstring\\_from\\_format](#) (const char \*format, va\_list args)  
*Create a `std::string` using a `printf()` format and a `va_list` args.*

### 12.13.1 Detailed Description

Header of the [fsw::string\\_utils](#) namespace.

#### Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.14 libfswatch/c++/windows/win\_directory\_change\_event.hpp File Reference

Header of the [fsw::directory\\_change\\_event](#) class.

```
#include <cstdlib>
#include <string>
#include <memory>
#include <vector>
#include <windows.h>
#include "win_handle.hpp"
#include "win_error_message.hpp"
#include "../event.hpp"
```

## Classes

- class [fsw::directory\\_change\\_event](#)

Header of the [fsw::directory\\_change\\_event](#) class, a helper class to wrap Microsoft Windows' `ReadDirectoryChangesW` function and a common workflow to detect file system changes.

## Namespaces

- [fsw](#)

Main namespace of `libfswatch`.

### 12.14.1 Detailed Description

Header of the [fsw::directory\\_change\\_event](#) class.

#### Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.15 libfswatch/c++/windows/win\_error\_message.hpp File Reference

Header of the [fsw::win\\_error\\_message](#) class.

```
#include <string>
#include <windows.h>
```

## Classes

- class [fsw::win\\_error\\_message](#)

Helper class to get the system-defined error message for a Microsoft Windows' error code.

## Namespaces

- [fsw](#)

Main namespace of `libfswatch`.



### 12.15.1 Detailed Description

Header of the `fsw::win_error_message` class.

#### Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.16 libfswatch/c++/windows/win\_handle.hpp File Reference

Header of the `fsw::win_handle` class.

```
#include <windows.h>
```

### Classes

- class `fsw::win_handle`  
*A RAII wrapper around Microsoft Windows `HANDLE`.*

### Namespaces

- `fsw`  
*Main namespace of `libfswatch`.*

### 12.16.1 Detailed Description

Header of the `fsw::win_handle` class.

#### Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.17 libfswatch/c++/windows/win\_paths.hpp File Reference

Header of the [fsw::win\\_paths](#) namespace.

```
#include <string>
```

### Namespaces

- [fsw](#)  
*Main namespace of libfswatch.*
- [fsw::win\\_paths](#)  
*Path conversion functions.*

### Functions

- `std::wstring fsw::win\_paths::posix\_to\_win\_w (std::string path)`  
*Converts a POSIX path to Windows.*
- `std::string fsw::win\_paths::win\_w\_to\_posix (std::wstring path)`  
*Converts a Windows path to POSIX.*

### 12.17.1 Detailed Description

Header of the [fsw::win\\_paths](#) namespace.

#### Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.18 libfswatch/c++/windows/win\_strings.hpp File Reference

Header of the [fsw::win\\_strings](#) namespace.

```
#include <string>  
#include <cwchar>
```

## Namespaces

- [fsw](#)  
*Main namespace of libfswatch.*
- [fsw::win\\_strings](#)  
*String conversion functions.*

## Functions

- string [fsw::win\\_strings::wstring\\_to\\_string](#) (wchar\_t \*s)  
*Converts a wide character string into a string.*
- std::string [fsw::win\\_strings::wstring\\_to\\_string](#) (const std::wstring &s)  
*Converts a wide character string into a string.*

### 12.18.1 Detailed Description

Header of the [fsw::win\\_strings](#) namespace.

#### Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.19 libfswatch/c++/windows\_monitor.hpp File Reference

Windows monitor.

```
#include "monitor.hpp"  
#include <string>  
#include <vector>
```

## Classes

- class [fsw::windows\\_monitor](#)  
*Windows monitor.*

## Namespaces

- [fsw](#)

*Main namespace of `libfswatch`.*

### 12.19.1 Detailed Description

Windows monitor.

#### Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.20 libfswatch/c/cevent.h File Reference

Event type manipulation.

```
#include <time.h>
#include <limits.h>
#include "libfswatch_types.h"
```

## Classes

- struct [fsw\\_cevent](#)

## Typedefs

- typedef struct [fsw\\_cevent](#) [fsw\\_cevent](#)
- typedef void(\* [FSW\\_CEVENT\\_CALLBACK](#)) ([fsw\\_cevent](#) const \*const events, const unsigned int event\_num, void \*data)

## Enumerations

- enum `fsw_event_flag` {  
    `NoOp` = 0, `PlatformSpecific` = (1 << 0), `Created` = (1 << 1), `Updated` = (1 << 2),  
    `Removed` = (1 << 3), `Renamed` = (1 << 4), `OwnerModified` = (1 << 5), `AttributeModified` = (1 << 6),  
    `MovedFrom` = (1 << 7), `MovedTo` = (1 << 8), `IsFile` = (1 << 9), `IsDir` = (1 << 10),  
    `IsSymLink` = (1 << 11), `Link` = (1 << 12), `Overflow` = (1 << 13) }

*Backend-agnostic change flags.*

## Functions

- `FSW_STATUS fsw_get_event_flag_by_name` (const char \*name, enum `fsw_event_flag` \*flag)  
*Get event flag by name.*
- char \* `fsw_get_event_flag_name` (const enum `fsw_event_flag` flag)  
*Get the name of an event flag.*

## Variables

- `fsw_event_flag` `FSW_ALL_EVENT_FLAGS` [15]

### 12.20.1 Detailed Description

Event type manipulation.

This header file defines the event types of the `libfswatch` API.

#### Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

### 12.20.2 Typedef Documentation

### 12.20.2.1 fsw\_cevent

```
typedef struct fsw_cevent fsw_cevent
```

A file change event is represented as an instance of this struct where:

- path is the path where the event was triggered.
- evt\_time the time when the event was triggered.
- flags is an array of fsw\_event\_flag of size flags\_num.
- flags\_num is the size of the flags array.

### 12.20.2.2 FSW\_CEVENT\_CALLBACK

```
typedef void(* FSW_CEVENT_CALLBACK) (fsw_cevent const *const events, const unsigned int event↔_num, void *data)
```

A function pointer of type FSW\_CEVENT\_CALLBACK is used by the API as a callback to provide information about received events. The callback is passed the following arguments:

- events, a const pointer to an array of events of type const fsw\_cevent.
- event\_num, the size of the \*events array.
- data, optional persisted data for a callback.

The memory used by the fsw\_cevent objects will be freed at the end of the callback invocation. A callback should copy such data instead of storing a pointer to it.

## 12.20.3 Enumeration Type Documentation

### 12.20.3.1 fsw\_event\_flag

```
enum fsw_event_flag
```

Backend-agnostic change flags.

Each element of this enum represents a backend-agnostic change flag. No direct mapping to backend-specific change types is guaranteed to exist: a change type may be mapped to multiple fsw\_event\_flag instances included the PlatformSpecific flag.

The values of event flags are all powers of 2, that is numbers  $f = 2^n$  where  $n$  is an integer. This representation makes it easy to combine flags into a bit mask and encode multiple events flags into a single integer.

A monitor implementation is required to map implementation-specific flags into API flags. Sometimes, though, a perfect match is not possible and the following situation may arise:

- One platform-specific flag must be mapped into multiple API flags.
- Multiple platform-specific flags must be mapped into a single API flag.
- A mapping is not possible for some flags, in which case they should be mapped to fsw\_event\_flag::Platform↔Specific. The API currently offers no way to retain a platform-specific event flag value in this case.

## Enumerator

NoOp	No event has occurred.
PlatformSpecific	Platform-specific placeholder for event type that cannot currently be mapped.
Created	An object was created.
Updated	An object was updated.
Removed	An object was removed.
Renamed	An object was renamed.
OwnerModified	The owner of an object was modified.
AttributeModified	The attributes of an object were modified.
MovedFrom	An object was moved from this location.
MovedTo	An object was moved to this location.
IsFile	The object is a file.
IsDir	The object is a directory.
IsSymLink	The object is a symbolic link.
Link	The link count of an object has changed.
Overflow	The event queue has overflowed.

## 12.20.4 Function Documentation

## 12.20.4.1 fsw\_get\_event\_flag\_by\_name()

```
FSW_STATUS fsw_get_event_flag_by_name (
    const char * name,
    enum fsw_event_flag * flag )
```

Get event flag by name.

This function looks for an event flag called *name* and, if it exists, it writes its value onto *flag* and `FSW_OK`, otherwise *flag* is not modified and `FSW_ERR_UNKNOWN_VALUE` is returned.

## Parameters

in	<i>name</i>	The name of the event flag to look for.
out	<i>flag</i>	The output variable where the event flag is returned.

## Returns

`FSW_OK` if the functions succeeds, `FSW_ERR_UNKNOWN_VALUE` otherwise.

## 12.20.4.2 fsw\_get\_event\_flag\_name()

```
char* fsw_get_event_flag_name (
    const enum fsw_event_flag flag )
```

Get the name of an event flag.

This function looks for the name of the specified event `flag`. If it exists, it returns its name, otherwise `nullptr` is returned.

#### Parameters

in	<i>flag</i>	The event flag to look for.
----	-------------	-----------------------------

#### Returns

The name of `flag`, or `nullptr` if it does not exist.

## 12.21 libfswatch/c/cfilter.h File Reference

Header of the `libfswatch` library functions for filter management.

```
#include "cevent.h"
```

### Classes

- struct [fsw\\_cmonitor\\_filter](#)
- struct [fsw\\_event\\_type\\_filter](#)

*Event type filter.*

### Typedefs

- typedef struct [fsw\\_cmonitor\\_filter](#) **fsw\_cmonitor\_filter**
- typedef struct [fsw\\_event\\_type\\_filter](#) **fsw\_event\_type\_filter**

*Event type filter.*

### Enumerations

- enum [fsw\\_filter\\_type](#) { **filter\_include**, **filter\_exclude** }

*Event filter type.*

#### 12.21.1 Detailed Description

Header of the `libfswatch` library functions for filter management.

#### Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0



## 12.22 libfswatch/c/cmonitor.h File Reference

Header of the `libfswatch` library defining the monitor types.

```
#include <time.h>
```

### Enumerations

- enum `fsw_monitor_type` {  
    `system_default_monitor_type` = 0, `fsevents_monitor_type`, `kqueue_monitor_type`, `inotify_monitor_type`,  
    `windows_monitor_type`, `poll_monitor_type`, `fen_monitor_type` }

*Available monitors.*

### 12.22.1 Detailed Description

Header of the `libfswatch` library defining the monitor types.

#### Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

### 12.22.2 Enumeration Type Documentation

#### 12.22.2.1 `fsw_monitor_type`

```
enum fsw_monitor_type
```

Available monitors.

This enumeration lists all the available monitors, where the special `system_default_monitor_type` element refers to the platform-specific default monitor.

## Enumerator

<code>system_default_monitor_type</code>	System default monitor.
<code>fsevents_monitor_type</code>	OS X FSEvents monitor.
<code>kqueue_monitor_type</code>	BSD <code>kqueue</code> monitor.
<code>inotify_monitor_type</code>	Linux <code>inotify</code> monitor.
<code>windows_monitor_type</code>	Windows monitor.
<code>poll_monitor_type</code>	<code>stat()</code> -based poll monitor.
<code>fen_monitor_type</code>	Solaris/Illumos monitor.

## 12.23 libfswatch/c/error.h File Reference

Error values.

### Macros

- `#define FSW_OK 0`
- `#define FSW_ERR_UNKNOWN_ERROR (1 << 0)`
- `#define FSW_ERR_SESSION_UNKNOWN (1 << 1)`
- `#define FSW_ERR_MONITOR_ALREADY_EXISTS (1 << 2)`
- `#define FSW_ERR_MEMORY (1 << 3)`
- `#define FSW_ERR_UNKNOWN_MONITOR_TYPE (1 << 4)`
- `#define FSW_ERR_CALLBACK_NOT_SET (1 << 5)`
- `#define FSW_ERR_PATHS_NOT_SET (1 << 6)`
- `#define FSW_ERR_MISSING_CONTEXT (1 << 7)`
- `#define FSW_ERR_INVALID_PATH (1 << 8)`
- `#define FSW_ERR_INVALID_CALLBACK (1 << 9)`
- `#define FSW_ERR_INVALID_LATENCY (1 << 10)`
- `#define FSW_ERR_INVALID_REGEX (1 << 11)`
- `#define FSW_ERR_MONITOR_ALREADY_RUNNING (1 << 12)`
- `#define FSW_ERR_UNKNOWN_VALUE (1 << 13)`
- `#define FSW_ERR_INVALID_PROPERTY (1 << 14)`

### 12.23.1 Detailed Description

Error values.

This header file defines the error values used by the `libfswatch` API.

#### Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

## 12.23.2 Macro Definition Documentation

### 12.23.2.1 FSW\_ERR\_CALLBACK\_NOT\_SET

```
#define FSW_ERR_CALLBACK_NOT_SET (1 << 5)
```

The callback has not been set.

### 12.23.2.2 FSW\_ERR\_INVALID\_CALLBACK

```
#define FSW_ERR_INVALID_CALLBACK (1 << 9)
```

The callback is invalid.

### 12.23.2.3 FSW\_ERR\_INVALID\_LATENCY

```
#define FSW_ERR_INVALID_LATENCY (1 << 10)
```

The latency is invalid.

### 12.23.2.4 FSW\_ERR\_INVALID\_PATH

```
#define FSW_ERR_INVALID_PATH (1 << 8)
```

The path is invalid.

### 12.23.2.5 FSW\_ERR\_INVALID\_PROPERTY

```
#define FSW_ERR_INVALID_PROPERTY (1 << 14)
```

The property is invalid.

### 12.23.2.6 FSW\_ERR\_INVALID\_REGEX

```
#define FSW_ERR_INVALID_REGEX (1 << 11)
```

The regular expression is invalid.

### 12.23.2.7 FSW\_ERR\_MEMORY

```
#define FSW_ERR_MEMORY (1 << 3)
```

An error occurred while invoking a memory management routine.

#### 12.23.2.8 FSW\_ERR\_MISSING\_CONTEXT

```
#define FSW_ERR_MISSING_CONTEXT (1 << 7)
```

The callback context has not been set.

#### 12.23.2.9 FSW\_ERR\_MONITOR\_ALREADY\_EXISTS

```
#define FSW_ERR_MONITOR_ALREADY_EXISTS (1 << 2)
```

The session already contains a monitor.

#### 12.23.2.10 FSW\_ERR\_MONITOR\_ALREADY\_RUNNING

```
#define FSW_ERR_MONITOR_ALREADY_RUNNING (1 << 12)
```

A monitor is already running in the specified session.

#### 12.23.2.11 FSW\_ERR\_PATHS\_NOT\_SET

```
#define FSW_ERR_PATHS_NOT_SET (1 << 6)
```

The paths to watch have not been set.

#### 12.23.2.12 FSW\_ERR\_SESSION\_UNKNOWN

```
#define FSW_ERR_SESSION_UNKNOWN (1 << 1)
```

The session specified by the handle is unknown.

#### 12.23.2.13 FSW\_ERR\_UNKNOWN\_ERROR

```
#define FSW_ERR_UNKNOWN_ERROR (1 << 0)
```

An unknown error has occurred.

#### 12.23.2.14 FSW\_ERR\_UNKNOWN\_MONITOR\_TYPE

```
#define FSW_ERR_UNKNOWN_MONITOR_TYPE (1 << 4)
```

The specified monitor type does not exist.

#### 12.23.2.15 FSW\_ERR\_UNKNOWN\_VALUE

```
#define FSW_ERR_UNKNOWN_VALUE (1 << 13)
```

The value is unknown.

## 12.23.2.16 FSW\_OK

```
#define FSW_OK 0
```

The call was successful.

## 12.24 libfswatch/c/libfswatch.cpp File Reference

Main `libfswatch` source file.

```
#include "gettext_defs.h"
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <cstring>
#include <memory>
#include <vector>
#include <map>
#include "libfswatch.h"
#include "../c++/libfswatch_map.hpp"
#include "../c++/filter.hpp"
#include "../c++/monitor.hpp"
#include "../c++/libfswatch_exception.hpp"
```

## Classes

- struct [FSW\\_SESSION](#)
- struct [fsw\\_callback\\_context](#)

## Typedefs

- typedef struct [FSW\\_SESSION](#) [FSW\\_SESSION](#)
- typedef struct [fsw\\_callback\\_context](#) [fsw\\_callback\\_context](#)

## Functions

- static [FSW\\_SESSION](#) \* [get\\_session](#) (const [FSW\\_HANDLE](#) handle)
- static int [create\\_monitor](#) ([FSW\\_HANDLE](#) handle, const [fsw\\_monitor\\_type](#) type)
- static [FSW\\_STATUS](#) [fsw\\_set\\_last\\_error](#) (const int error)
- [FSW\\_STATUS](#) [fsw\\_init\\_library](#) ()
- void [libfsw\\_cpp\\_callback\\_proxy](#) (const std::vector< [event](#) > &events, void \*context\_ptr)
- [FSW\\_HANDLE](#) [fsw\\_init\\_session](#) (const [fsw\\_monitor\\_type](#) type)
- [FSW\\_STATUS](#) [fsw\\_add\\_path](#) (const [FSW\\_HANDLE](#) handle, const char \*path)
- [FSW\\_STATUS](#) [fsw\\_add\\_property](#) (const [FSW\\_HANDLE](#) handle, const char \*name, const char \*value)
- [FSW\\_STATUS](#) [fsw\\_set\\_callback](#) (const [FSW\\_HANDLE](#) handle, const [FSW\\_CEVENT\\_CALLBACK](#) callback, void \*data)
- [FSW\\_STATUS](#) [fsw\\_set\\_allow\\_overflow](#) (const [FSW\\_HANDLE](#) handle, const bool allow\_overflow)
- [FSW\\_STATUS](#) [fsw\\_set\\_latency](#) (const [FSW\\_HANDLE](#) handle, const double latency)
- [FSW\\_STATUS](#) [fsw\\_set\\_recursive](#) (const [FSW\\_HANDLE](#) handle, const bool recursive)

- [FSW\\_STATUS fsw\\_set\\_directory\\_only](#) (const [FSW\\_HANDLE](#) handle, const bool directory\_only)
- [FSW\\_STATUS fsw\\_set\\_follow\\_symlinks](#) (const [FSW\\_HANDLE](#) handle, const bool follow\_symlinks)
- [FSW\\_STATUS fsw\\_add\\_event\\_type\\_filter](#) (const [FSW\\_HANDLE](#) handle, const [fsw\\_event\\_type\\_filter](#) event\_type)
- [FSW\\_STATUS fsw\\_add\\_filter](#) (const [FSW\\_HANDLE](#) handle, const [fsw\\_cmonitor\\_filter](#) filter)
- [FSW\\_STATUS fsw\\_start\\_monitor](#) (const [FSW\\_HANDLE](#) handle)
- [FSW\\_STATUS fsw\\_stop\\_monitor](#) (const [FSW\\_HANDLE](#) handle)
- [FSW\\_STATUS fsw\\_destroy\\_session](#) (const [FSW\\_HANDLE](#) handle)
- [FSW\\_STATUS fsw\\_last\\_error](#) ()
- bool [fsw\\_is\\_verbose](#) ()
- void [fsw\\_set\\_verbose](#) (bool verbose)

## Variables

- static bool **fsw\_libfswatch\_verbose** = false
- static FSW\_THREAD\_LOCAL [FSW\\_STATUS last\\_error](#)
- static FSW\_EVENT\_CALLBACK **libfsw\_cpp\_callback\_proxy**

### 12.24.1 Detailed Description

Main `libfswatch` source file.

#### Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.10.0

### 12.24.2 Function Documentation

#### 12.24.2.1 `fsw_add_event_type_filter()`

```
FSW\_STATUS fsw_add_event_type_filter (
    const FSW\_HANDLE handle,
    const fsw\_event\_type\_filter event_type )
```

Adds an event type filter to the current session.

See [filter.h](#) for the definition of [fsw\\_event\\_type\\_filter](#).

### 12.24.2.2 fsw\_add\_filter()

```
FSW_STATUS fsw_add_filter (
    const FSW_HANDLE handle,
    const fsw_cmonitor_filter filter )
```

Adds a filter to the current session. A filter is a regular expression that, depending on whether the filter type is exclusion or not, must or must not be matched for an event path for the event to be accepted.

See [cfilter.h](#) for the definition of [fsw\\_cmonitor\\_filter](#).

### 12.24.2.3 fsw\_add\_path()

```
FSW_STATUS fsw_add_path (
    const FSW_HANDLE handle,
    const char * path )
```

Adds a path to watch to the specified session. At least one path must be added to the current session in order for it to be valid.

### 12.24.2.4 fsw\_add\_property()

```
FSW_STATUS fsw_add_property (
    const FSW_HANDLE handle,
    const char * name,
    const char * value )
```

Adds the specified monitor property.

### 12.24.2.5 fsw\_destroy\_session()

```
FSW_STATUS fsw_destroy_session (
    const FSW_HANDLE handle )
```

Destroys an existing session and invalidates its handle.

### 12.24.2.6 fsw\_init\_library()

```
FSW_STATUS fsw_init_library ( )
```

The `libfswatch` C API let users create monitor sessions and receive file system events matching the specified criteria. Most API functions return a status code of type `FSW_STATUS` which can take any value specified in the [error.h](#) header. A successful API call returns `FSW_OK` and the last error can be obtained calling the [fsw\\_last\\_error\(\)](#) function.

If the compiler and the C++ library used to build `libfswatch` support the `thread_local` storage specified then this API is thread safe and a different state is maintained on a per-thread basis.

Session-modifying API calls (such as `fsw_add_path`) will take effect the next time a monitor is started with `fsw_↔start_monitor`.

Currently not all monitors supports being stopped, in which case `fsw_start_monitor` is a non-returning API call.

A basic session needs at least:

- A path to watch.
- A callback to process the events sent by the monitor.

as shown in the next example (error checking code was omitted).

```
// Use the default monitor.
const FSW_HANDLE handle = fsw_init_session(system_default_monitor_type);

fsw_add_path(handle, "my/path");
fsw_set_callback(handle, my_callback);

fsw_start_monitor(handle);
```

A suitable callback function is a function pointer of type `FSW_CEVENT_CALLBACK`, that is it is a function conforming with the following signature:

```
void c_process_events(fsw_cevent const * const events,
                     const unsigned int event_num,
                     void * data);
```

When a monitor receives change events satisfying all the session criteria, the callback is invoked and passed a copy of the events. This function initializes the `libfswatch` library and must be invoked before any other calls to the C or C++ API. If the function succeeds, it returns `FSW_OK`, otherwise the initialization routine failed and the library should not be usable.

#### 12.24.2.7 fsw\_init\_session()

```
FSW_HANDLE fsw_init_session (
    const enum fsw_monitor_type type )
```

This function creates a new monitor session using the specified monitor and returns an handle to it. This function is the `libfswatch` API entry point.

#### See also

[cmonitor.h](#) for a list of all the available monitors.

#### 12.24.2.8 fsw\_is\_verbose()

```
bool fsw_is_verbose ( )
```

Check whether the verbose mode is active.

#### 12.24.2.9 fsw\_last\_error()

```
FSW_STATUS fsw_last_error ( )
```

Gets the last error code.



#### 12.24.2.10 fsw\_set\_allow\_overflow()

```
FSW_STATUS fsw_set_allow_overflow (
    const FSW_HANDLE handle,
    const bool allow_overflow )
```

Sets the allow overflow flag of the monitor. When this flag is set, a monitor is allowed to overflow and report it as a change event.

#### 12.24.2.11 fsw\_set\_callback()

```
FSW_STATUS fsw_set_callback (
    const FSW_HANDLE handle,
    const FSW_CEVENT_CALLBACK callback,
    void * data )
```

Sets the callback the monitor invokes when some events are received. The callback must be set in the current session in order for it to be valid.

See [cevent.h](#) for the definition of FSW\_CEVENT\_CALLBACK.

#### 12.24.2.12 fsw\_set\_directory\_only()

```
FSW_STATUS fsw_set_directory_only (
    const FSW_HANDLE handle,
    const bool directory_only )
```

Determines whether the monitor only watches a directory when performing a recursive scan. By default, a monitor accepts all kinds of files.

#### 12.24.2.13 fsw\_set\_follow\_symlinks()

```
FSW_STATUS fsw_set_follow_symlinks (
    const FSW_HANDLE handle,
    const bool follow_symlinks )
```

Determines whether a symbolic link is followed or not. By default, a symbolic link are not followed.

#### 12.24.2.14 fsw\_set\_latency()

```
FSW_STATUS fsw_set_latency (
    const FSW_HANDLE handle,
    const double latency )
```

Sets the latency of the monitor. By default, the latency is set to 1 s.

#### 12.24.2.15 fsw\_set\_recursive()

```
FSW_STATUS fsw_set_recursive (
    const FSW_HANDLE handle,
    const bool recursive )
```

Determines whether the monitor recursively scans each watched path or not. Recursive scanning is an optional feature which could not be implemented by all the monitors. By default, recursive scanning is disabled.

#### 12.24.2.16 fsw\_set\_verbose()

```
void fsw_set_verbose (
    bool verbose )
```

Set the verbose mode.

#### 12.24.2.17 fsw\_start\_monitor()

```
FSW_STATUS fsw_start_monitor (
    const FSW_HANDLE handle )
```

Starts the monitor if it is properly configured. Depending on the type of monitor this call might return when a monitor is stopped or not.

#### 12.24.2.18 fsw\_stop\_monitor()

```
FSW_STATUS fsw_stop_monitor (
    const FSW_HANDLE handle )
```

Stops a running monitor.

## 12.25 libfswatch/c/libfswatch.h File Reference

Header of the libfswatch library.

```
#include <stdbool.h>
#include "libfswatch_types.h"
#include "cevent.h"
#include "cmonitor.h"
#include "cfilter.h"
#include "error.h"
```

## Functions

- [FSW\\_STATUS fsw\\_init\\_library](#) ()
- [FSW\\_HANDLE fsw\\_init\\_session](#) (const enum [fsw\\_monitor\\_type](#) type)
- [FSW\\_STATUS fsw\\_add\\_path](#) (const [FSW\\_HANDLE](#) handle, const char \*path)
- [FSW\\_STATUS fsw\\_add\\_property](#) (const [FSW\\_HANDLE](#) handle, const char \*name, const char \*value)
- [FSW\\_STATUS fsw\\_set\\_allow\\_overflow](#) (const [FSW\\_HANDLE](#) handle, const bool allow\_overflow)
- [FSW\\_STATUS fsw\\_set\\_callback](#) (const [FSW\\_HANDLE](#) handle, const [FSW\\_CEVENT\\_CALLBACK](#) callback, void \*data)
- [FSW\\_STATUS fsw\\_set\\_latency](#) (const [FSW\\_HANDLE](#) handle, const double latency)
- [FSW\\_STATUS fsw\\_set\\_recursive](#) (const [FSW\\_HANDLE](#) handle, const bool recursive)
- [FSW\\_STATUS fsw\\_set\\_directory\\_only](#) (const [FSW\\_HANDLE](#) handle, const bool directory\_only)
- [FSW\\_STATUS fsw\\_set\\_follow\\_symlinks](#) (const [FSW\\_HANDLE](#) handle, const bool follow\_symlinks)
- [FSW\\_STATUS fsw\\_add\\_event\\_type\\_filter](#) (const [FSW\\_HANDLE](#) handle, const [fsw\\_event\\_type\\_filter](#) event\_type)
- [FSW\\_STATUS fsw\\_add\\_filter](#) (const [FSW\\_HANDLE](#) handle, const [fsw\\_cmonitor\\_filter](#) filter)
- [FSW\\_STATUS fsw\\_start\\_monitor](#) (const [FSW\\_HANDLE](#) handle)
- [FSW\\_STATUS fsw\\_stop\\_monitor](#) (const [FSW\\_HANDLE](#) handle)
- [FSW\\_STATUS fsw\\_destroy\\_session](#) (const [FSW\\_HANDLE](#) handle)
- [FSW\\_STATUS fsw\\_last\\_error](#) ()
- bool [fsw\\_is\\_verbose](#) ()
- void [fsw\\_set\\_verbose](#) (bool verbose)

### 12.25.1 Detailed Description

Header of the `libfswatch` library.

This header file defines the API of the `libfswatch` library.

#### Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

### 12.25.2 Function Documentation

#### 12.25.2.1 fsw\_add\_event\_type\_filter()

```
FSW_STATUS fsw_add_event_type_filter (
    const FSW_HANDLE handle,
    const fsw_event_type_filter event_type )
```

Adds an event type filter to the current session.

See [filter.h](#) for the definition of [fsw\\_event\\_type\\_filter](#).

#### 12.25.2.2 fsw\_add\_filter()

```
FSW_STATUS fsw_add_filter (
    const FSW_HANDLE handle,
    const fsw_cmonitor_filter filter )
```

Adds a filter to the current session. A filter is a regular expression that, depending on whether the filter type is exclusion or not, must or must not be matched for an event path for the event to be accepted.

See [filter.h](#) for the definition of [fsw\\_cmonitor\\_filter](#).

#### 12.25.2.3 fsw\_add\_path()

```
FSW_STATUS fsw_add_path (
    const FSW_HANDLE handle,
    const char * path )
```

Adds a path to watch to the specified session. At least one path must be added to the current session in order for it to be valid.

#### 12.25.2.4 fsw\_add\_property()

```
FSW_STATUS fsw_add_property (
    const FSW_HANDLE handle,
    const char * name,
    const char * value )
```

Adds the specified monitor property.

#### 12.25.2.5 fsw\_destroy\_session()

```
FSW_STATUS fsw_destroy_session (
    const FSW_HANDLE handle )
```

Destroys an existing session and invalidates its handle.

## 12.25.2.6 fsw\_init\_library()

```
FSW_STATUS fsw_init_library ( )
```

The `libfswatch` C API let users create monitor sessions and receive file system events matching the specified criteria. Most API functions return a status code of type `FSW_STATUS` which can take any value specified in the [error.h](#) header. A successful API call returns `FSW_OK` and the last error can be obtained calling the [fsw\\_last\\_error\(\)](#) function.

If the compiler and the C++ library used to build `libfswatch` support the `thread_local` storage specified then this API is thread safe and a different state is maintained on a per-thread basis.

Session-modifying API calls (such as `fsw_add_path`) will take effect the next time a monitor is started with `fsw_start_monitor`.

Currently not all monitors supports being stopped, in which case `fsw_start_monitor` is a non-returning API call.

A basic session needs at least:

- A path to watch.
- A callback to process the events sent by the monitor.

as shown in the next example (error checking code was omitted).

```
// Use the default monitor.
const FSW_HANDLE handle = fsw_init_session(system_default_monitor_type);

fsw_add_path(handle, "my/path");
fsw_set_callback(handle, my_callback);

fsw_start_monitor(handle);
```

A suitable callback function is a function pointer of type `FSW_CEVENT_CALLBACK`, that is it is a function conforming with the following signature:

```
void c_process_events(fsw_cevent const * const events,
                     const unsigned int event_num,
                     void * data);
```

When a monitor receives change events satisfying all the session criteria, the callback is invoked and passed a copy of the events. This function initializes the `libfswatch` library and must be invoked before any other calls to the C or C++ API. If the function succeeds, it returns `FSW_OK`, otherwise the initialization routine failed and the library should not be usable.

## 12.25.2.7 fsw\_init\_session()

```
FSW_HANDLE fsw_init_session (
    const enum fsw_monitor_type type )
```

This function creates a new monitor session using the specified monitor and returns an handle to it. This function is the `libfswatch` API entry point.

See also

[cmonitor.h](#) for a list of all the available monitors.

#### 12.25.2.8 fsw\_is\_verbose()

```
bool fsw_is_verbose ( )
```

Check whether the verbose mode is active.

#### 12.25.2.9 fsw\_last\_error()

```
FSW_STATUS fsw_last_error ( )
```

Gets the last error code.

#### 12.25.2.10 fsw\_set\_allow\_overflow()

```
FSW_STATUS fsw_set_allow_overflow (
    const FSW_HANDLE handle,
    const bool allow_overflow )
```

Sets the allow overflow flag of the monitor. When this flag is set, a monitor is allowed to overflow and report it as a change event.

#### 12.25.2.11 fsw\_set\_callback()

```
FSW_STATUS fsw_set_callback (
    const FSW_HANDLE handle,
    const FSW_CEVENT_CALLBACK callback,
    void * data )
```

Sets the callback the monitor invokes when some events are received. The callback must be set in the current session in order for it to be valid.

See [cevent.h](#) for the definition of FSW\_CEVENT\_CALLBACK.

#### 12.25.2.12 fsw\_set\_directory\_only()

```
FSW_STATUS fsw_set_directory_only (
    const FSW_HANDLE handle,
    const bool directory_only )
```

Determines whether the monitor only watches a directory when performing a recursive scan. By default, a monitor accepts all kinds of files.

#### 12.25.2.13 fsw\_set\_follow\_symlinks()

```
FSW_STATUS fsw_set_follow_symlinks (
    const FSW_HANDLE handle,
    const bool follow_symlinks )
```

Determines whether a symbolic link is followed or not. By default, a symbolic link are not followed.

#### 12.25.2.14 fsw\_set\_latency()

```
FSW_STATUS fsw_set_latency (
    const FSW_HANDLE handle,
    const double latency )
```

Sets the latency of the monitor. By default, the latency is set to 1 s.

#### 12.25.2.15 fsw\_set\_recursive()

```
FSW_STATUS fsw_set_recursive (
    const FSW_HANDLE handle,
    const bool recursive )
```

Determines whether the monitor recursively scans each watched path or not. Recursive scanning is an optional feature which could not be implemented by all the monitors. By default, recursive scanning is disabled.

#### 12.25.2.16 fsw\_set\_verbose()

```
void fsw_set_verbose (
    bool verbose )
```

Set the verbose mode.

#### 12.25.2.17 fsw\_start\_monitor()

```
FSW_STATUS fsw_start_monitor (
    const FSW_HANDLE handle )
```

Starts the monitor if it is properly configured. Depending on the type of monitor this call might return when a monitor is stopped or not.

#### 12.25.2.18 fsw\_stop\_monitor()

```
FSW_STATUS fsw_stop_monitor (
    const FSW_HANDLE handle )
```

Stops a running monitor.

## 12.26 libfswatch/c/libfswatch\_log.h File Reference

Header of the `libfswatch` library containing logging functions..

```
#include <stdio.h>
```

## Macros

- #define `FSW_LOG(msg) fsw_logf("%s: ", __func__); fsw_log(msg)`  
*Log the specified message to the standard output prepended by the source line number.*
- #define `FSW_ELOG(msg) fsw_flogf(stderr, "%s: ", __func__); fsw_flog(stderr, msg)`  
*Log the specified message to the standard error prepended by the source line number.*
- #define `FSW_LOGF(msg, ...) fsw_logf("%s: ", __func__); fsw_logf(msg, __VA_ARGS__)`  
*Log the specified `printf()`-like message to the standard output prepended by the source line number.*
- #define `FSW_ELOGF(msg, ...) fsw_flogf(stderr, "%s: ", __func__); fsw_flogf(stderr, msg, __VA_ARGS__)`  
*Log the specified `printf()`-like message to the standard error prepended by the source line number.*
- #define `FSW_FLOGF(f, msg, ...) fsw_flogf(f, "%s: ", __func__); fsw_flogf(f, msg, __VA_ARGS__)`  
*Log the specified `printf()`-like message to the specified file descriptor prepended by the source line number.*

## Functions

- void `fsw_log` (const char \*msg)
- void `fsw_flog` (FILE \*f, const char \*msg)
- void `fsw_logf` (const char \*format,...)
- void `fsw_flogf` (FILE \*f, const char \*format,...)
- void `fsw_log_perror` (const char \*msg)
- void `fsw_flog_perror` (const char \*format,...)

### 12.26.1 Detailed Description

Header of the `libfswatch` library containing logging functions..

#### Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

#### License:

GNU General Public License v. 3.0

#### Author

Enrico M. Crisostomo

#### Version

1.8.0

### 12.26.2 Function Documentation



### 12.26.2.1 fsw\_flog()

```
void fsw_flog (
    FILE * f,
    const char * msg )
```

Prints the specified message to the specified file.

### 12.26.2.2 fsw\_flogf()

```
void fsw_flogf (
    FILE * f,
    const char * format,
    ... )
```

Formats the specified message and prints it to the specified file. The message string format conforms with printf.

### 12.26.2.3 fsw\_log()

```
void fsw_log (
    const char * msg )
```

Prints the specified message to standard output.

### 12.26.2.4 fsw\_log\_perror()

```
void fsw_log_perror (
    const char * msg )
```

Prints the specified message using perror.

### 12.26.2.5 fsw\_logf()

```
void fsw_logf (
    const char * format,
    ... )
```

Formats the specified message and prints it to standard output. The message string format conforms with printf.

### 12.26.2.6 fsw\_logf\_perror()

```
void fsw_logf_perror (
    const char * format,
    ... )
```

Prints the specified message using perror. The message string format conforms with printf.

## 12.27 libfswatch/c/libfswatch\_types.h File Reference

Header of the `libfswatch` library containing common types.

### Macros

- `#define FSW_INVALID_HANDLE -1`
- `#define FSW_THREAD_LOCAL`

### Typedefs

- `typedef struct FSW_SESSION * FSW_HANDLE`  
*Handle to a monitoring session.*
- `typedef int FSW_STATUS`  
*Status of a library call.*

### 12.27.1 Detailed Description

Header of the `libfswatch` library containing common types.

This header file defines the types used by the `libfswatch` library.

### Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

### License:

GNU General Public License v. 3.0

### Author

Enrico M. Crisostomo

### Version

1.8.0

# Index

- ~event
  - fsw::event, [33](#)
- ~monitor
  - fsw::monitor, [48](#)
- ~win\_handle
  - fsw::win\_handle, [72](#)
- accept\_event\_type
  - fsw::monitor, [49](#)
- accept\_path
  - fsw::monitor, [49](#), [50](#)
- add\_event\_type\_filter
  - fsw::monitor, [50](#)
- add\_filter
  - fsw::monitor, [50](#)
- callback
  - fsw::monitor, [58](#)
- cevent.h
  - FSW\_CEVENT\_CALLBACK, [96](#)
  - fsw\_cevent, [95](#)
  - fsw\_event\_flag, [96](#)
  - fsw\_get\_event\_flag\_by\_name, [97](#)
  - fsw\_get\_event\_flag\_name, [97](#)
- cmonitor.h
  - fsw\_monitor\_type, [99](#)
- create\_monitor
  - fsw::monitor\_factory, [61](#)
- current
  - fsw::win\_error\_message, [70](#)
- error.h
  - FSW\_ERR\_CALLBACK\_NOT\_SET, [101](#)
  - FSW\_ERR\_INVALID\_CALLBACK, [101](#)
  - FSW\_ERR\_INVALID\_LATENCY, [101](#)
  - FSW\_ERR\_INVALID\_PATH, [101](#)
  - FSW\_ERR\_INVALID\_PROPERTY, [101](#)
  - FSW\_ERR\_INVALID\_REGEX, [101](#)
  - FSW\_ERR\_MEMORY, [101](#)
  - FSW\_ERR\_MISSING\_CONTEXT, [101](#)
  - FSW\_ERR\_MONITOR\_ALREADY\_EXISTS, [102](#)
  - FSW\_ERR\_MONITOR\_ALREADY\_RUNNING, [102](#)
  - FSW\_ERR\_PATHS\_NOT\_SET, [102](#)
  - FSW\_ERR\_SESSION\_UNKNOWN, [102](#)
  - FSW\_ERR\_UNKNOWN\_ERROR, [102](#)
  - FSW\_ERR\_UNKNOWN\_MONITOR\_TYPE, [102](#)
  - FSW\_ERR\_UNKNOWN\_VALUE, [102](#)
  - FSW\_OK, [102](#)
- error\_code
  - fsw::libfsw\_exception, [44](#)
- event
  - fsw::event, [33](#)
- exists\_type
  - fsw::monitor\_factory, [62](#)
- extended
  - fsw::monitor\_filter, [65](#)
- FSW\_CEVENT\_CALLBACK
  - cevent.h, [96](#)
- FSW\_ERR\_CALLBACK\_NOT\_SET
  - error.h, [101](#)
- FSW\_ERR\_INVALID\_CALLBACK
  - error.h, [101](#)
- FSW\_ERR\_INVALID\_LATENCY
  - error.h, [101](#)
- FSW\_ERR\_INVALID\_PATH
  - error.h, [101](#)
- FSW\_ERR\_INVALID\_PROPERTY
  - error.h, [101](#)
- FSW\_ERR\_INVALID\_REGEX
  - error.h, [101](#)
- FSW\_ERR\_MEMORY
  - error.h, [101](#)
- FSW\_ERR\_MISSING\_CONTEXT
  - error.h, [101](#)
- FSW\_ERR\_MONITOR\_ALREADY\_EXISTS
  - error.h, [102](#)
- FSW\_ERR\_MONITOR\_ALREADY\_RUNNING
  - error.h, [102](#)
- FSW\_ERR\_PATHS\_NOT\_SET
  - error.h, [102](#)
- FSW\_ERR\_SESSION\_UNKNOWN
  - error.h, [102](#)
- FSW\_ERR\_UNKNOWN\_ERROR
  - error.h, [102](#)
- FSW\_ERR\_UNKNOWN\_MONITOR\_TYPE
  - error.h, [102](#)
- FSW\_ERR\_UNKNOWN\_VALUE
  - error.h, [102](#)
- FSW\_EVENT\_CALLBACK
  - fsw, [23](#)
- FSW\_OK
  - error.h, [102](#)
- FSW\_SESSION, [40](#)
- filter\_flags
  - fsw::monitor, [51](#)
- fire\_idle\_event
  - fsw::monitor, [59](#)
- fsw, [21](#)

- FSW\_EVENT\_CALLBACK, 23
- fsw\_hash\_map, 23
- fsw\_hash\_set, 24
- get\_directory\_children, 24
- lstat\_path, 25
- monitor\_filter, 24
- operator<<, 25
- read\_link\_path, 25
- stat\_path, 26
- fsw::FSEventFlagType, 36
- fsw::compiled\_monitor\_filter, 31
- fsw::directory\_change\_event, 31
- fsw::event, 32
  - ~event, 33
  - event, 33
  - get\_event\_flag\_by\_name, 33
  - get\_event\_flag\_name, 34
  - get\_flags, 34
  - get\_path, 34
  - get\_time, 35
- fsw::fen\_monitor, 35
  - run, 36
- fsw::fsevents\_monitor, 37
  - run, 37
- fsw::inotify\_monitor, 40
  - run, 41
- fsw::inotify\_monitor\_impl, 41
- fsw::kqueue\_monitor, 42
  - run, 42
- fsw::libfsw\_exception, 43
  - error\_code, 44
  - libfsw\_exception, 43
  - what, 44
- fsw::monitor, 44
  - ~monitor, 48
  - accept\_event\_type, 49
  - accept\_path, 49, 50
  - add\_event\_type\_filter, 50
  - add\_filter, 50
  - callback, 58
  - filter\_flags, 51
  - fire\_idle\_event, 59
  - get\_context, 51
  - get\_property, 51
  - is\_running, 52
  - monitor, 48
  - notify\_events, 52
  - notify\_overflow, 52
  - on\_stop, 52
  - paths, 59
  - properties, 59
  - run, 53
  - set\_allow\_overflow, 53
  - set\_context, 54
  - set\_directory\_only, 54
  - set\_event\_type\_filters, 54
  - set\_filters, 55
  - set\_fire\_idle\_event, 55
  - set\_follow\_symlinks, 55
  - set\_latency, 56
  - set\_properties, 56
  - set\_property, 57
  - set\_recursive, 57
  - set\_watch\_access, 57
  - start, 57
  - stop, 58
- fsw::monitor\_factory, 60
  - create\_monitor, 61
  - exists\_type, 62
  - get\_types, 63
  - register\_creator, 63
  - register\_creator\_by\_type, 63
- fsw::monitor\_filter, 64
  - extended, 65
  - read\_from\_file, 64
  - text, 65
- fsw::monitor\_registrant
  - monitor\_registrant, 66
- fsw::monitor\_registrant< M >, 66
- fsw::poll\_monitor, 67
  - run, 68
- fsw::poll\_monitor::poll\_monitor\_data, 68
- fsw::string\_utils, 26
  - string\_from\_format, 27
  - vstring\_from\_format, 27
- fsw::win\_error\_message, 68
  - current, 70
  - get\_error\_code, 70
  - get\_message, 70
  - operator std::wstring, 70
  - win\_error\_message, 69
- fsw::win\_flag\_type, 71
- fsw::win\_handle, 71
  - ~win\_handle, 72
  - is\_valid, 72, 73
  - operator=, 73, 74
  - win\_handle, 72
- fsw::win\_paths, 27
  - posix\_to\_win\_w, 28
  - win\_w\_to\_posix, 28
- fsw::win\_strings, 29
  - wstring\_to\_string, 29
- fsw::windows\_monitor, 74
  - run, 75
- fsw\_add\_event\_type\_filter
  - libfswatch.cpp, 104
  - libfswatch.h, 109
- fsw\_add\_filter
  - libfswatch.cpp, 104
  - libfswatch.h, 110
- fsw\_add\_path
  - libfswatch.cpp, 105
  - libfswatch.h, 110
- fsw\_add\_property
  - libfswatch.cpp, 105
  - libfswatch.h, 110

- fsw\_callback\_context, 38
- fsw\_cevent, 38
  - cevent.h, 95
- fsw\_cmonitor\_filter, 39
- fsw\_destroy\_session
  - libfswatch.cpp, 105
  - libfswatch.h, 110
- fsw\_event\_flag
  - cevent.h, 96
- fsw\_event\_type\_filter, 39
- fsw\_flog
  - libfswatch\_log.h, 114
- fsw\_flogf
  - libfswatch\_log.h, 115
- fsw\_get\_event\_flag\_by\_name
  - cevent.h, 97
- fsw\_get\_event\_flag\_name
  - cevent.h, 97
- fsw\_hash\_map
  - fsw, 23
- fsw\_hash\_set
  - fsw, 24
- fsw\_init\_library
  - libfswatch.cpp, 105
  - libfswatch.h, 110
- fsw\_init\_session
  - libfswatch.cpp, 106
  - libfswatch.h, 111
- fsw\_is\_verbose
  - libfswatch.cpp, 106
  - libfswatch.h, 111
- fsw\_last\_error
  - libfswatch.cpp, 106
  - libfswatch.h, 112
- fsw\_log
  - libfswatch\_log.h, 115
- fsw\_log\_perror
  - libfswatch\_log.h, 115
- fsw\_logf
  - libfswatch\_log.h, 115
- fsw\_logf\_perror
  - libfswatch\_log.h, 115
- fsw\_monitor\_type
  - cmonitor.h, 99
- fsw\_set\_allow\_overflow
  - libfswatch.cpp, 106
  - libfswatch.h, 112
- fsw\_set\_callback
  - libfswatch.cpp, 107
  - libfswatch.h, 112
- fsw\_set\_directory\_only
  - libfswatch.cpp, 107
  - libfswatch.h, 112
- fsw\_set\_follow\_symlinks
  - libfswatch.cpp, 107
  - libfswatch.h, 112
- fsw\_set\_latency
  - libfswatch.cpp, 107
  - libfswatch.h, 112
- fsw\_set\_recursive
  - libfswatch.cpp, 107
  - libfswatch.h, 113
- fsw\_set\_verbose
  - libfswatch.cpp, 108
  - libfswatch.h, 113
- fsw\_start\_monitor
  - libfswatch.cpp, 108
  - libfswatch.h, 113
- fsw\_stop\_monitor
  - libfswatch.cpp, 108
  - libfswatch.h, 113
- get\_context
  - fsw::monitor, 51
- get\_directory\_children
  - fsw, 24
- get\_error\_code
  - fsw::win\_error\_message, 70
- get\_event\_flag\_by\_name
  - fsw::event, 33
- get\_event\_flag\_name
  - fsw::event, 34
- get\_flags
  - fsw::event, 34
- get\_message
  - fsw::win\_error\_message, 70
- get\_path
  - fsw::event, 34
- get\_property
  - fsw::monitor, 51
- get\_time
  - fsw::event, 35
- get\_types
  - fsw::monitor\_factory, 63
- is\_running
  - fsw::monitor, 52
- is\_valid
  - fsw::win\_handle, 72, 73
- libfsw\_exception
  - fsw::libfsw\_exception, 43
- libfswatch.cpp
  - fsw\_add\_event\_type\_filter, 104
  - fsw\_add\_filter, 104
  - fsw\_add\_path, 105
  - fsw\_add\_property, 105
  - fsw\_destroy\_session, 105
  - fsw\_init\_library, 105
  - fsw\_init\_session, 106
  - fsw\_is\_verbose, 106
  - fsw\_last\_error, 106
  - fsw\_set\_allow\_overflow, 106
  - fsw\_set\_callback, 107
  - fsw\_set\_directory\_only, 107
  - fsw\_set\_follow\_symlinks, 107
  - fsw\_set\_latency, 107

- fsw\_set\_recursive, 107
  - fsw\_set\_verbose, 108
  - fsw\_start\_monitor, 108
  - fsw\_stop\_monitor, 108
- libfswatch.h
  - fsw\_add\_event\_type\_filter, 109
  - fsw\_add\_filter, 110
  - fsw\_add\_path, 110
  - fsw\_add\_property, 110
  - fsw\_destroy\_session, 110
  - fsw\_init\_library, 110
  - fsw\_init\_session, 111
  - fsw\_is\_verbose, 111
  - fsw\_last\_error, 112
  - fsw\_set\_allow\_overflow, 112
  - fsw\_set\_callback, 112
  - fsw\_set\_directory\_only, 112
  - fsw\_set\_follow\_symlinks, 112
  - fsw\_set\_latency, 112
  - fsw\_set\_recursive, 113
  - fsw\_set\_verbose, 113
  - fsw\_start\_monitor, 113
  - fsw\_stop\_monitor, 113
- libfswatch/c++/event.hpp, 77
- libfswatch/c++/fen\_monitor.hpp, 78
- libfswatch/c++/filter.hpp, 79
- libfswatch/c++/fsevents\_monitor.hpp, 80
- libfswatch/c++/inotify\_monitor.hpp, 80
- libfswatch/c++/kqueue\_monitor.hpp, 81
- libfswatch/c++/libfswatch\_exception.hpp, 82
- libfswatch/c++/libfswatch\_map.hpp, 83
- libfswatch/c++/libfswatch\_set.hpp, 84
- libfswatch/c++/monitor.hpp, 84
- libfswatch/c++/path\_utils.hpp, 87
- libfswatch/c++/poll\_monitor.hpp, 88
- libfswatch/c++/string/string\_utils.hpp, 88
- libfswatch/c++/windows/win\_directory\_change\_event.h, 89
- libfswatch/c++/windows/win\_error\_message.hpp, 90
- libfswatch/c++/windows/win\_handle.hpp, 91
- libfswatch/c++/windows/win\_paths.hpp, 92
- libfswatch/c++/windows/win\_strings.hpp, 92
- libfswatch/c++/windows\_monitor.hpp, 93
- libfswatch/c/cevent.h, 94
- libfswatch/c/cfilter.h, 98
- libfswatch/c/cmonitor.h, 99
- libfswatch/c/error.h, 100
- libfswatch/c/libfswatch.cpp, 103
- libfswatch/c/libfswatch.h, 108
- libfswatch/c/libfswatch\_log.h, 113
- libfswatch/c/libfswatch\_types.h, 116
- libfswatch\_log.h
  - fsw\_flog, 114
  - fsw\_flogf, 115
  - fsw\_log, 115
  - fsw\_log\_perror, 115
  - fsw\_logf, 115
  - fsw\_logf\_perror, 115
- lstat\_path
  - fsw, 25
- monitor
  - fsw::monitor, 48
- monitor.hpp
  - REGISTER\_MONITOR\_IMPL, 86
  - REGISTER\_MONITOR, 86
- monitor\_filter
  - fsw, 24
- monitor\_registrant
  - fsw::monitor\_registrant, 66
- notify\_events
  - fsw::monitor, 52
- notify\_overflow
  - fsw::monitor, 52
- on\_stop
  - fsw::monitor, 52
- operator std::wstring
  - fsw::win\_error\_message, 70
- operator<<
  - fsw, 25
- operator=
  - fsw::win\_handle, 73, 74
- paths
  - fsw::monitor, 59
- posix\_to\_win\_w
  - fsw::win\_paths, 28
- properties
  - fsw::monitor, 59
- REGISTER\_MONITOR\_IMPL
  - monitor.hpp, 86
- REGISTER\_MONITOR
  - monitor.hpp, 86
- read\_from\_file
  - fsw::monitor\_filter, 64
- read\_link\_path
  - fsw, 25
- register\_creator
  - fsw::monitor\_factory, 63
- register\_creator\_by\_type
  - fsw::monitor\_factory, 63
- run
  - fsw::fen\_monitor, 36
  - fsw::fsevents\_monitor, 37
  - fsw::inotify\_monitor, 41
  - fsw::kqueue\_monitor, 42
  - fsw::monitor, 53
  - fsw::poll\_monitor, 68
  - fsw::windows\_monitor, 75
- set\_allow\_overflow
  - fsw::monitor, 53
- set\_context
  - fsw::monitor, 54
- set\_directory\_only

- fsw::monitor, [54](#)
- set\_event\_type\_filters
  - fsw::monitor, [54](#)
- set\_filters
  - fsw::monitor, [55](#)
- set\_fire\_idle\_event
  - fsw::monitor, [55](#)
- set\_follow\_symlinks
  - fsw::monitor, [55](#)
- set\_latency
  - fsw::monitor, [56](#)
- set\_properties
  - fsw::monitor, [56](#)
- set\_property
  - fsw::monitor, [57](#)
- set\_recursive
  - fsw::monitor, [57](#)
- set\_watch\_access
  - fsw::monitor, [57](#)
- start
  - fsw::monitor, [57](#)
- stat\_path
  - fsw, [26](#)
- stop
  - fsw::monitor, [58](#)
- string\_from\_format
  - fsw::string\_utils, [27](#)
- text
  - fsw::monitor\_filter, [65](#)
- vstring\_from\_format
  - fsw::string\_utils, [27](#)
- what
  - fsw::libfsw\_exception, [44](#)
- win\_error\_message
  - fsw::win\_error\_message, [69](#)
- win\_handle
  - fsw::win\_handle, [72](#)
- win\_w\_to\_posix
  - fsw::win\_paths, [28](#)
- wstring\_to\_string
  - fsw::win\_strings, [29](#)