

fswatch

Generated by Doxygen 1.9.3

1 fswatch	1
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
2 C++ API	5
2.1 Usage	5
2.2 Example	6
3 C API	7
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
4 History	9
4.1 12:0:0	9
4.2 11:1:0	9
4.3 11:0:0	9
4.4 10:1:1	9
4.5 10:0:1	10
4.6 9:0:0	10
4.7 8:0:2	10
4.8 8:0:2	10
4.9 8:0:2	10
4.10 5:0:2	11
4.11 4:0:1	11
4.12 3:0:0	11
5 Path Filtering	13
6 Namespace Index	15
6.1 Namespace List	15
7 Hierarchical Index	17
7.1 Class Hierarchy	17
8 Class Index	19
8.1 Class List	19

9 File Index	21
9.1 File List	21
10 Namespace Documentation	23
10.1 fsw Namespace Reference	23
10.1.1 Detailed Description	25
10.1.2 Typedef Documentation	25
10.1.2.1 FSW_EVENT_CALLBACK	25
10.1.2.2 fsw_hash_map	26
10.1.2.3 fsw_hash_set	26
10.1.2.4 monitor_filter	26
10.1.3 Function Documentation	26
10.1.3.1 fsw_realpath()	26
10.1.3.2 get_directory_children()	27
10.1.3.3 lstat_path()	27
10.1.3.4 operator<<()	28
10.1.3.5 read_from_file()	28
10.1.3.6 read_link_path()	29
10.1.3.7 stat_path()	30
10.1.4 Variable Documentation	31
10.1.4.1 extended	31
10.2 fsw::string_utils Namespace Reference	31
10.2.1 Detailed Description	31
10.2.2 Function Documentation	31
10.2.2.1 string_from_format()	31
10.2.2.2 vstring_from_format()	32
10.3 fsw::win_paths Namespace Reference	32
10.3.1 Detailed Description	32
10.3.2 Function Documentation	33
10.3.2.1 posix_to_win_w()	33
10.3.2.2 win_w_to_posix()	33
10.4 fsw::win_strings Namespace Reference	34
10.4.1 Detailed Description	34
10.4.2 Function Documentation	34
10.4.2.1 wstring_to_string() [1/2]	34
10.4.2.2 wstring_to_string() [2/2]	35
11 Class Documentation	37
11.1 fsw::compiled_monitor_filter Struct Reference	37
11.2 fsw::directory_change_event Class Reference	37
11.2.1 Detailed Description	38
11.3 fsw::event Class Reference	38
11.3.1 Detailed Description	39

11.3.2 Constructor & Destructor Documentation	39
11.3.2.1 event()	39
11.3.2.2 ~event()	40
11.3.3 Member Function Documentation	40
11.3.3.1 get_event_flag_by_name()	40
11.3.3.2 get_event_flag_name()	40
11.3.3.3 get_flags()	41
11.3.3.4 get_path()	41
11.3.3.5 get_time()	41
11.4 fsw::fen_monitor Class Reference	42
11.4.1 Detailed Description	42
11.4.2 Member Function Documentation	42
11.4.2.1 run()	42
11.5 fsw::fsevents_monitor Class Reference	43
11.5.1 Detailed Description	43
11.5.2 Member Function Documentation	43
11.5.2.1 run()	44
11.5.3 Member Data Documentation	44
11.5.3.1 DARWIN_EVENTSTREAM_NO_DEFER	44
11.6 fsw_cevent Struct Reference	44
11.6.1 Detailed Description	45
11.7 fsw_cmonitor_filter Struct Reference	45
11.8 fsw_event_type_filter Struct Reference	45
11.8.1 Detailed Description	46
11.9 fsw::inotify_monitor Class Reference	46
11.9.1 Detailed Description	46
11.9.2 Member Function Documentation	46
11.9.2.1 run()	47
11.10 fsw::inotify_monitor_impl Struct Reference	47
11.11 fsw::kqueue_monitor Class Reference	47
11.11.1 Detailed Description	48
11.11.2 Member Function Documentation	48
11.11.2.1 run()	48
11.12 fsw::libfsd_exception Class Reference	49
11.12.1 Detailed Description	49
11.12.2 Constructor & Destructor Documentation	49
11.12.2.1 libfsd_exception()	49
11.12.3 Member Function Documentation	50
11.12.3.1 error_code()	50
11.12.3.2 what()	50
11.13 fsw::monitor Class Reference	50
11.13.1 Detailed Description	53

11.13.2 Constructor & Destructor Documentation	54
11.13.2.1 monitor()	54
11.13.2.2 ~monitor()	55
11.13.3 Member Function Documentation	55
11.13.3.1 accept_event_type()	55
11.13.3.2 accept_path()	56
11.13.3.3 add_event_type_filter()	56
11.13.3.4 add_filter()	57
11.13.3.5 filter_flags()	57
11.13.3.6 get_context()	57
11.13.3.7 get_property()	58
11.13.3.8 is_running()	58
11.13.3.9 notify_events()	58
11.13.3.10 notify_overflow()	59
11.13.3.11 on_stop()	59
11.13.3.12 run()	59
11.13.3.13 set_allow_overflow()	60
11.13.3.14 set_bubble_events()	60
11.13.3.15 set_context()	60
11.13.3.16 set_directory_only()	61
11.13.3.17 set_event_type_filters()	62
11.13.3.18 set_filters()	62
11.13.3.19 set_fire_idle_event()	62
11.13.3.20 set_follow_symlinks()	63
11.13.3.21 set_latency()	63
11.13.3.22 set_properties()	64
11.13.3.23 set_property()	64
11.13.3.24 set_recursive()	65
11.13.3.25 set_watch_access()	65
11.13.3.26 start()	66
11.13.3.27 stop()	66
11.13.4 Member Data Documentation	66
11.13.4.1 callback	67
11.13.4.2 fire_idle_event	67
11.13.4.3 paths	67
11.13.4.4 properties	67
11.14 fsw::monitor_factory Class Reference	68
11.14.1 Detailed Description	68
11.14.2 Member Function Documentation	68
11.14.2.1 create_monitor() [1/2]	68
11.14.2.2 create_monitor() [2/2]	69
11.14.2.3 exists_type()	70

11.14.2.4 <code>get_types()</code>	71
11.15 <code>fsw::poll_monitor</code> Class Reference	71
11.15.1 Detailed Description	72
11.15.2 Member Function Documentation	72
11.15.2.1 <code>run()</code>	72
11.16 <code>fsw::win_error_message</code> Class Reference	73
11.16.1 Detailed Description	73
11.16.2 Constructor & Destructor Documentation	73
11.16.2.1 <code>win_error_message()</code> [1/2]	73
11.16.2.2 <code>win_error_message()</code> [2/2]	74
11.16.3 Member Function Documentation	74
11.16.3.1 <code>current()</code>	74
11.16.3.2 <code>get_error_code()</code>	74
11.16.3.3 <code>get_message()</code>	75
11.16.3.4 <code>operator std::wstring()</code>	75
11.17 <code>fsw::win_flag_type</code> Struct Reference	75
11.18 <code>fsw::win_handle</code> Class Reference	75
11.18.1 Detailed Description	76
11.18.2 Constructor & Destructor Documentation	76
11.18.2.1 <code>~win_handle()</code>	76
11.18.2.2 <code>win_handle()</code>	77
11.18.3 Member Function Documentation	77
11.18.3.1 <code>is_valid()</code> [1/2]	77
11.18.3.2 <code>is_valid()</code> [2/2]	77
11.18.3.3 <code>operator=()</code> [1/2]	78
11.18.3.4 <code>operator=()</code> [2/2]	78
11.19 <code>fsw::windows_monitor</code> Class Reference	79
11.19.1 Detailed Description	79
11.19.2 Member Function Documentation	80
11.19.2.1 <code>run()</code>	80
12 File Documentation	81
12.1 <code>libfswatch/c++/event.hpp</code> File Reference	81
12.1.1 Detailed Description	82
12.2 <code>event.hpp</code>	82
12.3 <code>libfswatch/c++/fen_monitor.hpp</code> File Reference	83
12.3.1 Detailed Description	83
12.4 <code>fen_monitor.hpp</code>	84
12.5 <code>libfswatch/c++/filter.hpp</code> File Reference	84
12.5.1 Detailed Description	85
12.6 <code>filter.hpp</code>	86
12.7 <code>libfswatch/c++/fsevents_monitor.hpp</code> File Reference	86

12.7.1 Detailed Description	87
12.8 fsevents_monitor.hpp	87
12.9 libfswatch/c++/inotify_monitor.hpp File Reference	88
12.9.1 Detailed Description	88
12.10 inotify_monitor.hpp	89
12.11 libfswatch/c++/kqueue_monitor.hpp File Reference	89
12.11.1 Detailed Description	90
12.12 kqueue_monitor.hpp	90
12.13 libfswatch/c++/libfswatch_exception.hpp File Reference	91
12.13.1 Detailed Description	92
12.14 libfswatch_exception.hpp	92
12.15 libfswatch/c++/libfswatch_map.hpp File Reference	93
12.15.1 Detailed Description	93
12.16 libfswatch_map.hpp	94
12.17 libfswatch/c++/libfswatch_set.hpp File Reference	94
12.17.1 Detailed Description	95
12.18 libfswatch_set.hpp	95
12.19 libfswatch/c++/monitor.hpp File Reference	96
12.19.1 Detailed Description	96
12.20 monitor.hpp	97
12.21 libfswatch/c++/monitor_factory.hpp File Reference	98
12.21.1 Detailed Description	99
12.22 monitor_factory.hpp	100
12.23 libfswatch/c++/path_utils.hpp File Reference	100
12.23.1 Detailed Description	101
12.24 path_utils.hpp	101
12.25 libfswatch/c++/poll_monitor.hpp File Reference	102
12.25.1 Detailed Description	102
12.26 poll_monitor.hpp	103
12.27 libfswatch/c++/string/string_utils.hpp File Reference	104
12.27.1 Detailed Description	104
12.28 string_utils.hpp	105
12.29 libfswatch/c++/windows/win_directory_change_event.hpp File Reference	105
12.29.1 Detailed Description	106
12.30 win_directory_change_event.hpp	106
12.31 libfswatch/c++/windows/win_error_message.hpp File Reference	107
12.31.1 Detailed Description	107
12.32 win_error_message.hpp	108
12.33 libfswatch/c++/windows/win_handle.hpp File Reference	108
12.33.1 Detailed Description	109
12.34 win_handle.hpp	109
12.35 libfswatch/c++/windows/win_paths.hpp File Reference	110

12.35.1 Detailed Description	110
12.36 win_paths.hpp	111
12.37 libswatch/c++/windows/win_strings.hpp File Reference	111
12.37.1 Detailed Description	112
12.38 win_strings.hpp	112
12.39 libswatch/c++/windows_monitor.hpp File Reference	112
12.39.1 Detailed Description	113
12.40 windows_monitor.hpp	113
12.41 libswatch/c/cevent.h File Reference	114
12.41.1 Detailed Description	115
12.41.2 Typedef Documentation	115
12.41.2.1 fsw_cevent	115
12.41.2.2 FSW_CEVENT_CALLBACK	116
12.41.3 Enumeration Type Documentation	116
12.41.3.1 fsw_event_flag	116
12.41.4 Function Documentation	118
12.41.4.1 fsw_get_event_flag_by_name()	118
12.41.4.2 fsw_get_event_flag_name()	119
12.42 cevent.h	119
12.43 libswatch/c/cfilter.h File Reference	120
12.43.1 Detailed Description	121
12.44 cfilter.h	122
12.45 libswatch/c/cmonitor.h File Reference	122
12.45.1 Detailed Description	123
12.45.2 Enumeration Type Documentation	123
12.45.2.1 fsw_monitor_type	123
12.46 cmonitor.h	124
12.47 libswatch/c/error.h File Reference	124
12.47.1 Detailed Description	125
12.47.2 Macro Definition Documentation	125
12.47.2.1 FSW_ERR_CALLBACK_NOT_SET	125
12.47.2.2 FSW_ERR_INVALID_CALLBACK	126
12.47.2.3 FSW_ERR_INVALID_LATENCY	126
12.47.2.4 FSW_ERR_INVALID_PATH	126
12.47.2.5 FSW_ERR_INVALID_PROPERTY	126
12.47.2.6 FSW_ERR_INVALID_REGEX	126
12.47.2.7 FSW_ERR_MEMORY	126
12.47.2.8 FSW_ERR_MISSING_CONTEXT	126
12.47.2.9 FSW_ERR_MONITOR_ALREADY_EXISTS	126
12.47.2.10 FSW_ERR_MONITOR_ALREADY_RUNNING	127
12.47.2.11 FSW_ERR_PATHS_NOT_SET	127
12.47.2.12 FSW_ERR_SESSION_UNKNOWN	127

12.47.2.13 FSW_ERR_UNKNOWN_ERROR	127
12.47.2.14 FSW_ERR_UNKNOWN_MONITOR_TYPE	127
12.47.2.15 FSW_ERR_UNKNOWN_VALUE	127
12.47.2.16 FSW_OK	127
12.48 error.h	128
12.49 libfswatch/c/libfswatch.cpp File Reference	128
12.49.1 Detailed Description	130
12.49.2 Function Documentation	130
12.49.2.1 fsw_add_event_type_filter()	130
12.49.2.2 fsw_add_filter()	130
12.49.2.3 fsw_add_path()	131
12.49.2.4 fsw_add_property()	131
12.49.2.5 fsw_destroy_session()	131
12.49.2.6 fsw_init_library()	131
12.49.2.7 fsw_init_session()	132
12.49.2.8 fsw_is_running()	132
12.49.2.9 fsw_is_verbose()	132
12.49.2.10 fsw_last_error()	132
12.49.2.11 fsw_set_allow_overflow()	133
12.49.2.12 fsw_set_callback()	133
12.49.2.13 fsw_set_directory_only()	133
12.49.2.14 fsw_set_follow_symlinks()	133
12.49.2.15 fsw_set_latency()	133
12.49.2.16 fsw_set_recursive()	134
12.49.2.17 fsw_set_verbose()	134
12.49.2.18 fsw_start_monitor()	134
12.49.2.19 fsw_stop_monitor()	134
12.50 libfswatch/c/libfswatch.h File Reference	134
12.50.1 Detailed Description	135
12.50.2 Function Documentation	135
12.50.2.1 fsw_add_event_type_filter()	136
12.50.2.2 fsw_add_filter()	136
12.50.2.3 fsw_add_path()	136
12.50.2.4 fsw_add_property()	136
12.50.2.5 fsw_destroy_session()	136
12.50.2.6 fsw_init_library()	137
12.50.2.7 fsw_init_session()	137
12.50.2.8 fsw_is_running()	138
12.50.2.9 fsw_is_verbose()	138
12.50.2.10 fsw_last_error()	138
12.50.2.11 fsw_set_allow_overflow()	138
12.50.2.12 fsw_set_callback()	138

12.50.2.13 fsw_set_directory_only()	138
12.50.2.14 fsw_set_follow_symlinks()	139
12.50.2.15 fsw_set_latency()	139
12.50.2.16 fsw_set_recursive()	139
12.50.2.17 fsw_set_verbose()	139
12.50.2.18 fsw_start_monitor()	139
12.50.2.19 fsw_stop_monitor()	139
12.51 libfswatch.h	140
12.52 libfswatch/c/libfswatch_log.h File Reference	141
12.52.1 Detailed Description	141
12.52.2 Function Documentation	142
12.52.2.1 fsw_flog()	142
12.52.2.2 fsw_flogf()	142
12.52.2.3 fsw_log()	142
12.52.2.4 fsw_log_perror()	142
12.52.2.5 fsw_logf()	142
12.52.2.6 fsw_logf_perror()	142
12.53 libfswatch_log.h	143
12.54 libfswatch/c/libfswatch_types.h File Reference	143
12.54.1 Detailed Description	143
12.55 libfswatch_types.h	144
12.56 gettext.h	144
12.57 gettext_defs.h	148
Index	149

Chapter 1

fswatch

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 macOS.
- A monitor based on *kqueue*, an event notification interface introduced in FreeBSD 4.1 and supported on most *BSD systems (including macOS).
- 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 12:0:0

- Implement event bubbling (see the monitor class) by grouping events using the (time, path) tuple, to emit a single event with the union of all the flags.
- Refactor the `fsevents_monitor` class to replace usages of the deprecated function `FSEventStreamSchedule↔WithRunLoop` and substituting it with `FSEventStreamSetDispatchQueue` and an implementation based on dispatch queues.
- Improve the responsiveness of the `fsevents_monitor` on macOS by adding support for the `kFSEventStream↔CreateFlagNoDefer` flag.

4.2 11:1:0

- Fix `monitor_factory::create_monitor` ignoring the monitor type and always returning the default system monitor. (Issue 218: `fswatch v1.13` ignores the `-monitor` parameter and always uses the default monitor).

4.3 11:0:0

- Refactor the `monitor_factory` class so that available monitor types are determined at compile time. (Issue 142: Static library will not have any monitor type available).

4.4 10:1:1

- Migrate usages of POSIX regular expressions (`<regex.h>`) to the C++11 regex library (`<regex>`).
- Wrong error message is printed when inotify event queue overflows.

4.5 10:0:1

- Fix C99 compatibility in [cevent.h](#) by not implying enum.
- Free session memory.
- Fix segmentation fault when starting monitor.
- Add [fsw_is_running\(\)](#) function to the C API to check that a monitor is running.
- Fix stop sequence in [fsw::fsevents_monitor::run\(\)](#) and in [fsw::fsevents_monitor::on_stop\(\)](#).

4.6 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.

4.7 8:0:2

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

4.8 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.

4.9 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.

4.10 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()`.

4.11 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`

4.12 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:

fsw	Main namespace of <code>libfswatch</code>	23
fsw::string_utils	This namespace contains string manipulation functions	31
fsw::win_paths	Path conversion functions	32
fsw::win_strings	String conversion functions	34

Chapter 7

Hierarchical Index

7.1 Class Hierarchy

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

fsw::compiled_monitor_filter	37
fsw::directory_change_event	37
fsw::event	38
std::exception	
fsw::libfsw_exception	49
fsw_cevent	44
fsw_cmonitor_filter	45
fsw_event_type_filter	45
fsw::inotify_monitor_impl	47
fsw::monitor	50
fsw::fen_monitor	42
fsw::fsevents_monitor	43
fsw::inotify_monitor	46
fsw::kqueue_monitor	47
fsw::poll_monitor	71
fsw::windows_monitor	79
fsw::monitor_factory	68
fsw::win_error_message	73
fsw::win_flag_type	75
fsw::win_handle	75

Chapter 8

Class Index

8.1 Class List

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

fsw::compiled_monitor_filter	37
fsw::directory_change_event	
Header of the fsw::directory_change_event class, a helper class to wrap Microsoft Windows' <code>ReadDirectoryChangesW</code> function and a common workflow to detect file system changes	37
fsw::event	
Type representing a file change event	38
fsw::fen_monitor	
Solaris/Illumos monitor	42
fsw::fsevents_monitor	
MacOS FSEvents monitor	43
fsw_cevent	44
fsw_cmonitor_filter	45
fsw_event_type_filter	
Event type filter	45
fsw::inotify_monitor	
Solaris/Illumos monitor	46
fsw::inotify_monitor_impl	47
fsw::kqueue_monitor	
Solaris/Illumos monitor	47
fsw::libfsw_exception	
Base exception of the <code>libfswatch</code> library	49
fsw::monitor	
Base class of all monitors	50
fsw::monitor_factory	
Object factory class for fsw::monitor instances	68
fsw::poll_monitor	
<code>stat()</code> -based monitor	71
fsw::win_error_message	
Helper class to get the system-defined error message for a Microsoft Windows' error code	73
fsw::win_flag_type	75
fsw::win_handle	
A RAII wrapper around Microsoft Windows <code>HANDLE</code>	75
fsw::windows_monitor	
Windows monitor	79

Chapter 9

File Index

9.1 File List

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

libfswatch/ gettext.h	144
libfswatch/ gettext_defs.h	148
libfswatch/c++/ event.hpp Header of the fsw::event class	81
libfswatch/c++/ fen_monitor.hpp Solaris/Illumos monitor	83
libfswatch/c++/ filter.hpp Header of the fsw::monitor_filter class	84
libfswatch/c++/ fsevents_monitor.hpp MacOS FSEvents monitor	86
libfswatch/c++/ inotify_monitor.hpp Solaris/Illumos monitor	88
libfswatch/c++/ kqueue_monitor.hpp kqueue monitor	89
libfswatch/c++/ libfswatch_exception.hpp Base exception of the <code>libfswatch</code> library	91
libfswatch/c++/ libfswatch_map.hpp Header defining the associative container used by the library	93
libfswatch/c++/ libfswatch_set.hpp Header defining the default set type used by the library	94
libfswatch/c++/ monitor.hpp Header of the fsw::monitor class	96
libfswatch/c++/ monitor_factory.hpp Header of the fsw::monitor_factory class	98
libfswatch/c++/ path_utils.hpp Header defining utility functions to manipulate paths	100
libfswatch/c++/ poll_monitor.hpp stat() based monitor	102
libfswatch/c++/ windows_monitor.hpp Windows monitor	112
libfswatch/c++/string/ string_utils.hpp Header of the fsw::string_utils namespace	104
libfswatch/c++/windows/ win_directory_change_event.hpp Header of the fsw::directory_change_event class	105
libfswatch/c++/windows/ win_error_message.hpp Header of the fsw::win_error_message class	107

libfswatch/c++/windows/win_handle.hpp	
Header of the fsw::win_handle class	108
libfswatch/c++/windows/win_paths.hpp	
Header of the fsw::win_paths namespace	110
libfswatch/c++/windows/win_strings.hpp	
Header of the fsw::win_strings namespace	111
libfswatch/c/cevent.h	
Event type manipulation	114
libfswatch/c/cfilter.h	
Header of the <code>libfswatch</code> library functions for filter management	120
libfswatch/c/cmonitor.h	
Header of the <code>libfswatch</code> library defining the monitor types	122
libfswatch/c/error.h	
Error values	124
libfswatch/c/libfswatch.cpp	
Main <code>libfswatch</code> source file	128
libfswatch/c/libfswatch.h	
Header of the <code>libfswatch</code> library	134
libfswatch/c/libfswatch_log.h	
Header of the <code>libfswatch</code> library containing logging functions.	141
libfswatch/c/libfswatch_types.h	
Header of the <code>libfswatch</code> library containing common types	143

Chapter 10

Namespace Documentation

10.1 fsw Namespace Reference

Main namespace of `libfswatch`.

Namespaces

- namespace [string_utils](#)
This namespace contains string manipulation functions.
- namespace [win_paths](#)
Path conversion functions.
- namespace [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.
- class [fsevents_monitor](#)
macOS 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.
- 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

- using [monitor_filter](#) = [monitor_filter](#) { std::string text
Path filters used to accept or reject file change events.
- using [FSEventFlagType](#) = FSEventFlagType { FSEventStreamEventFlags flag
- 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.
- using [poll_monitor_data](#) = poll_monitor::poll_monitor_data { [fsw_hash_map](#)< string, poll_monitor↵
::watched_file_info > tracked_files

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)
- static bool [parse_filter](#) (string filter, [monitor_filter](#) &filter_object, void(*err_handler)(string))
- static std::vector< [monitor_filter](#) > [read_from_file](#) (const std::string &path, void(*err_handler)(std↵
::string)=nullptr)
Load filters from the specified file.
- static vector< FSEventFlagType > [create_flag_type_vector](#) ()
- static vector< [fsw_event_flag](#) > [decode_flags](#) (FSEventStreamEventFlags flag)
- static [monitor](#) * [create_default_monitor](#) (std::vector< std::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)
- std::string [fsw_realpath](#) (const char *path, char *resolved_path)
A thin wrapper about `realpath`.
- 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.*
- static vector< `win_flag_type` > `create_flag_type_vector` ()
- static vector< `fsw_event_flag` > `decode_flags` (DWORD flag)

Variables

- `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.*
- 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

```
using fsw::monitor_filter = typedef monitor_filter { std::string text
```

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 fsw_realpath()

```
std::string fsw::fsw_realpath (  
    const char * path,  
    char * resolved_path )
```

A thin wrapper about `realpath`.

Parameters

<i>path</i>	The path to resolve.
<i>resolved_path</i>	A pointer to a buffer where the resolved path is stored.

Returns

If there is no error, `realpath()` returns a string, otherwise it throws a `std::system_error`.

10.1.3.2 `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`.

10.1.3.3 `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.

10.1.3.4 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.

10.1.3.5 read_from_file()

```
static std::vector< monitor_filter > fsw::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: $^{\wedge}([+-])([ei]*) (.+)\$$
- 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.
-------------------------	---

10.1.3.6 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.

10.1.3.7 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.

10.1.4 Variable Documentation

10.1.4.1 extended

```
bool fsw::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

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 (
    const std::wstring & s )
```

Converts a wide character string into a string.

Parameters

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

Returns

The converted string.

10.4.2.2 wstring_to_string() [2/2]

```
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 con- vert.
---	---

Returns

The converted string.

Chapter 11

Class Documentation

11.1 `fsw::compiled_monitor_filter` Struct Reference

Public Attributes

- `std::regex` `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], [default]
```

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

<i>libfsw_exception</i>	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

<i>libfsw_exception</i>	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:

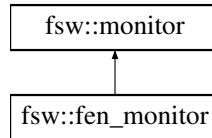
- [libfswatch/c++/event.hpp](#)
- [libfswatch/c++/event.cpp](#)

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:

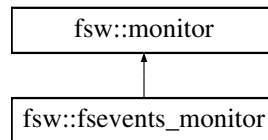
- [libfswatch/c++/fen_monitor.hpp](#)

11.5 fsw::fsevents_monitor Class Reference

macOS 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.
- **fsevents_monitor** (const fsevents_monitor &orig)=delete
- **fsevents_monitor & operator=** (const fsevents_monitor &that)=delete

Static Public Attributes

- static constexpr const char * **DARWIN_EVENTSTREAM_NO_DEFER** = "darwin.eventStream.noDefer"
Custom monitor property used to enable the kFSEventStreamCreateFlagNoDefer flag in the event stream.

Protected Member Functions

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

Additional Inherited Members

11.5.1 Detailed Description

macOS FSEvents monitor.

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

11.5.2 Member Function Documentation

11.5.2.1 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](#).

11.5.3 Member Data Documentation

11.5.3.1 DARWIN_EVENTSTREAM_NO_DEFER

```
constexpr const char* fsw::fsevents_monitor::DARWIN_EVENTSTREAM_NO_DEFER = "darwin.event↔  
Stream.noDefer" [static], [constexpr]
```

Custom monitor property used to enable the kFSEventStreamCreateFlagNoDefer flag in the event stream.

If you specify this flag and more than latency seconds have elapsed since the last event, your app will receive the event immediately. The delivery of the event resets the latency timer and any further events will be delivered after latency seconds have elapsed. This flag is useful for apps that are interactive and want to react immediately to changes but avoid getting swamped by notifications when changes are occurring in rapid succession. If you do not specify this flag, then when an event occurs after a period of no events, the latency timer is started. Any events that occur during the next latency seconds will be delivered as one group (including that first event). The delivery of the group of events resets the latency timer and any further events will be delivered after latency seconds. This is the default behavior and is more appropriate for background, daemon or batch processing apps.

See also

<https://developer.apple.com/documentation/coreservices/kfseventstreamcreateflagnode>

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

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

11.6 fsw_cevent Struct Reference

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


Public Attributes

- char * **path**
- time_t **evt_time**
- enum [fsw_event_flag](#) * **flags**
- unsigned int **flags_num**

11.6.1 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](#)

11.7 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](#)

11.8 fsw_event_type_filter Struct Reference

Event type filter.

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

Public Attributes

- enum [fsw_event_flag](#) **flag**

11.8.1 Detailed Description

Event type filter.

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

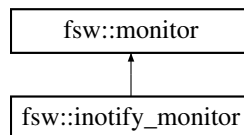
- [libfswatch/c/cfilter.h](#)

11.9 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

11.9.1 Detailed Description

Solaris/Illumos monitor.

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

11.9.2 Member Function Documentation

11.9.2.1 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](#)

11.10 fsw::inotify_monitor_impl Struct Reference

Public Attributes

- int **inotify_monitor_handle** = -1
- std::vector< [event](#) > **events**
- [fsw_hash_set](#)< int > **watched_descriptors**
- [fsw_hash_map](#)< std::string, int > **path_to_wd**
- [fsw_hash_map](#)< int, std::string > **wd_to_path**
- [fsw_hash_set](#)< int > **descriptors_to_remove**
- [fsw_hash_set](#)< int > **watches_to_remove**
- std::vector< std::string > **paths_to_rescan**
- time_t **curr_time**

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

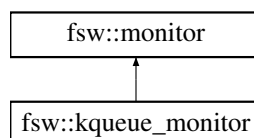
- [libfswatch/c++/inotify_monitor.cpp](#)

11.11 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.
- **~kqueue_monitor** () override
Destroys an instance of this class.

Protected Member Functions

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

Additional Inherited Members

11.11.1 Detailed Description

Solaris/Illumos monitor.

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

11.11.2 Member Function Documentation

11.11.2.1 run()

```
void fsw::kqueue_monitor::run ( ) [final], [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:

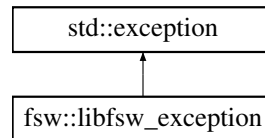
- [libfswatch/c++/kqueue_monitor.hpp](#)

11.12 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`
- `const char *` `what` () `const noexcept` *override*
Gets the error message.
- `virtual int` `error_code` () `const noexcept`
Gets the error code.
- `~libfsw_exception` () `noexcept` *override*
Destructs an instance of this class.
- `operator int` () `const noexcept`
Gets the error code.

11.12.1 Detailed Description

Base exception of the `libfswatch` library.

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

11.12.2 Constructor & Destructor Documentation

11.12.2.1 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.

11.12.3 Member Function Documentation

11.12.3.1 error_code()

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

Gets the error code.

Returns

The error code.

11.12.3.2 what()

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

Gets the error message.

Returns

The error message.

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

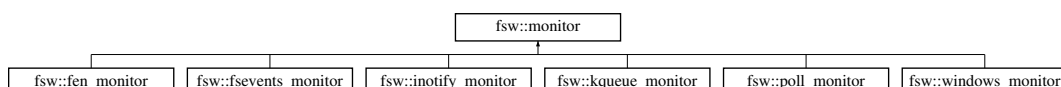
- [libfswatch/c++/libfswatch_exception.hpp](#)
- [libfswatch/c++/libfswatch_exception.cpp](#)

11.13 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 [set_bubble_events](#) (bool [bubble_events](#))
 - Set the bubble events flag.*
- 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.
- 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.
- bool `bubble_events` = false
Bubble events by joining flags received for the same (time, path) pair.
- 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.

11.13.1 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.

11.13.2 Constructor & Destructor Documentation

11.13.2.1 `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 call-back to which change events will be notified. The call-back cannot be null, otherwise a <code>libfsw_exception</code> will be thrown.

Parameters

<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> .
----------------	--

11.13.2.2 `~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\(\)](#)

11.13.3 Member Function Documentation**11.13.3.1** `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

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

Returns

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

11.13.3.2 accept_path()

```
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

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

Returns

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

11.13.3.3 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.
---------------	-------------------------------

11.13.3.4 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.
---------------	--------------------

11.13.3.5 filter_flags()

```
std::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.

11.13.3.6 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.

11.13.3.7 get_property()

```
std::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.

11.13.3.8 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.

11.13.3.9 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\(\)](#)

11.13.3.10 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\(\)](#)

11.13.3.11 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\(\)](#)

11.13.3.12 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::inotify_monitor](#), [fsw::poll_monitor](#), [fsw::windows_monitor](#), [fsw::kqueue_monitor](#), [fsw::fen_monitor](#), and [fsw::fsevents_monitor](#).

11.13.3.13 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.
-----------------	---

11.13.3.14 set_bubble_events()

```
void fsw::monitor::set_bubble_events (
    bool bubble_events )
```

Set the bubble events flag.

This function sets the bubble events flags, instructing the monitor to consolidate the event flags for all events with the same time and path received in the same batch.

Parameters

<i>bubble_events</i>	The bubble events flag.
----------------------	-------------------------

11.13.3.15 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.
----------------	----------------------------------

11.13.3.16 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.
-----------------------	--

11.13.3.17 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.
----------------	---------------------

11.13.3.18 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.
----------------	---------------------

11.13.3.19 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>	true if idle events should be fired, false other- wise.
------------------------	---

11.13.3.20 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>	true if sym- bolic links should be fol- lowed, false other- wise.
---------------	--

11.13.3.21 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.
----------------	--------------------

11.13.3.22 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.
----------------	---

11.13.3.23 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.
-------------	---------------------------

Parameters

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

11.13.3.24 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.
------------------	---

11.13.3.25 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.

11.13.3.26 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\(\)](#)

11.13.3.27 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\(\)](#)

11.13.4 Member Data Documentation

11.13.4.1 callback

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

Callback to which change events should be notified.

See also

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

11.13.4.2 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.

11.13.4.3 paths

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

List of paths to watch.

See also

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

11.13.4.4 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](#)

11.14 fsw::monitor_factory Class Reference

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

```
#include <monitor_factory.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.

11.14.1 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 are registered at compile time.

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.

11.14.2 Member Function Documentation

11.14.2.1 create_monitor() [1/2]

```
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

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

See also

fsw::monitor()

11.14.2.2 create_monitor() [2/2]

```
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

<i>libfsw_exception</i>	if a monitor of the specified <code>type</code> cannot be found.
---	--

See also

`fsw::monitor()`

11.14.2.3 exists_type()

```
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.

11.14.2.4 get_types()

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

Get the available monitor types.

Returns

A vector with the available monitor types.

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

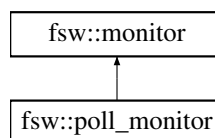
- [libfswatch/c++/monitor_factory.hpp](#)
- [libfswatch/c++/monitor_factory.cpp](#)

11.15 fsw::poll_monitor Class Reference

`stat()`-based monitor.

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

Inheritance diagram for `fsw::poll_monitor`:



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

11.15.1 Detailed Description

`stat()`-based monitor.

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

11.15.2 Member Function Documentation

11.15.2.1 `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](#)

11.16 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()`.

11.16.1 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.

11.16.2 Constructor & Destructor Documentation

11.16.2.1 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

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

11.16.2.2 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\(\)](#)

11.16.3 Member Function Documentation**11.16.3.1 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\(\)](#)

11.16.3.2 get_error_code()

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

Gets the error code.

Returns

The error code.

11.16.3.3 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.

11.16.3.4 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](#)

11.17 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](#)

11.18 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.

11.18.1 Detailed Description

A RAII wrapper around Microsoft Windows `HANDLE`.

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

11.18.2 Constructor & Destructor Documentation

11.18.2.1 ~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\(const HANDLE &\)](#)

11.18.2.2 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.
--------------	---------------------

11.18.3 Member Function Documentation

11.18.3.1 is_valid() [1/2]

```
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\(\)](#)

11.18.3.2 is_valid() [2/2]

```
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.

11.18.3.3 operator=() [1/2]

```
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

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

11.18.3.4 operator=() [2/2]

```
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.
--------------	---------------------

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

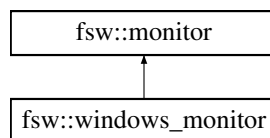
- [libfswatch/c++/windows/win_handle.hpp](#)

11.19 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

11.19.1 Detailed Description

Windows monitor.

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

11.19.2 Member Function Documentation

11.19.2.1 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

- namespace `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:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.2 event.hpp

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2014-2015 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16 #ifndef FSW_EVENT_H
17 # define FSW_EVENT_H
18
19 # include <string>
20 # include <ctime>
21 # include <vector>
22 # include <iostream>
23 # include "../c/cevent.h"
24
25 namespace fsw
26 {
27     class event
28     {
29     public:
30         event(std::string path, time_t evt_time, std::vector<fsw_event_flag> flags);
31
32         virtual ~event();
33
34         std::string get_path() const;
35
36         time_t get_time() const;
37
38         std::vector<fsw_event_flag> get_flags() const;
39
40         static fsw_event_flag get_event_flag_by_name(const std::string& name);
41
42         static std::string get_event_flag_name(const fsw_event_flag& flag);
43
44     private:
45         std::string path;
46         time_t evt_time;
47         std::vector<fsw_event_flag> evt_flags;
48     };
49
50     std::ostream& operator<<(std::ostream& out, const fsw_event_flag flag);
51 }
52 #endif /* FSW_EVENT_H */

```

12.3 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

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

12.3.1 Detailed Description

Solaris/Illumos monitor.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.4 fen_monitor.hpp

[Go to the documentation of this file.](#)

```

1 /*
2 * Copyright (c) 2015-2016 Enrico M. Crisostomo
3 *
4 * This program is free software; you can redistribute it and/or modify it under
5 * the terms of the GNU General Public License as published by the Free Software
6 * Foundation; either version 3, or (at your option) any later version.
7 *
8 * This program is distributed in the hope that it will be useful, but WITHOUT
9 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16
17 #ifndef FSW_FEN_MONITOR_H
18 # define FSW_FEN_MONITOR_H
19
20 # include "monitor.hpp"
21 # include <string>
22 # include <vector>
23
24 namespace fsw
25 {
26     struct fen_monitor_load;
27
28     struct fen_info;
29
30     class fen_monitor : public monitor
31     {
32     public:
33         fen_monitor(std::vector<std::string> paths,
34                     FSW_EVENT_CALLBACK *callback,
35                     void *context = nullptr);
36
37         virtual ~fen_monitor();
38
39     protected:
40         void run() override;
41
42     private:
43         fen_monitor(const fen_monitor& orig) = delete;
44         fen_monitor& operator=(const fen_monitor& that) = delete;
45
46         void scan_root_paths();
47         bool scan(const std::string& path, bool is_root_path = true);
48         bool is_path_watched(const std::string& path) const;
49         bool add_watch(const std::string& path, const struct stat& fd_stat);
50         bool associate_port(struct fen_info *finfo, const struct stat& fd_stat);
51         void process_events(struct fen_info *obj, int events);
52         void rescan_removed();
53         void rescan_pending();
54
55         // pimpl
56         fen_monitor_load *load;
57     };
58 }
59
60 #endif /* FSW_FEN_MONITOR_H */

```

12.5 libfswatch/c++/filter.hpp File Reference

Header of the `fsw::monitor_filter` class.

```

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

```

Namespaces

- namespace `fsw`

Main namespace of libfswatch.

Typedefs

- using `fsw::monitor_filter` = `monitor_filter` { `std::string` `text`
Path filters used to accept or reject file change events.

Functions

- static `std::vector< monitor_filter >` `fsw::read_from_file` (`const std::string &path`, `void(*err_handler)(std::string)=nullptr`)
Load filters from the specified file.

Variables

- `fsw_filter_type` `fsw::type`
Filter type.
- bool `fsw::case_sensitive`
Flag indicating whether `monitor_filter::text` is a case sensitive regular expression.
- bool `fsw::extended`
Flag indicating whether `monitor_filter::text` is an extended regular expression.

12.5.1 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:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.6 filter.hpp

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2014-2021 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16
17 #ifndef FSW__FILTER_H
18 #define FSW__FILTER_H
19
20 #include <string>
21 #include "libfswatch/c/cfilter.h"
22 #include <vector>
23
24 namespace fsw
25 {
26     using monitor_filter = struct monitor_filter
27     {
28         std::string text;
29         fsw_filter_type type;
30         bool case_sensitive;
31         bool extended;
32
33         static std::vector<monitor_filter> read_from_file(const std::string& path,
34                                                         void (*err_handler)(
35                                                             std::string) = nullptr);
36     };
37 }
38 #endif /* FSW__FILTER_H */

```

12.7 libfswatch/c++/fsevents_monitor.hpp File Reference

macOS FSEvents monitor.

```

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

```

Classes

- class [fsw::fsevents_monitor](#)
macOS FSEvents monitor.

Namespaces

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

12.7.1 Detailed Description

macOS FSEvents monitor.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.8 fsevents_monitor.hpp

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2014-2021 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16 #ifndef FSW_FSEVENT_MONITOR_H
17 # define FSW_FSEVENT_MONITOR_H
18
19 # include "monitor.hpp"
20 # include <CoreServices/CoreServices.h>
21
22 namespace fsw
23 {
24     class fsevents_monitor : public monitor
25     {
26     public:
27
28         static constexpr const char *DARWIN_EVENTSTREAM_NO_DEFER = "darwin.eventStream.noDefer";
29
30         fsevents_monitor(std::vector<std::string> paths,
31                         FSW_EVENT_CALLBACK *callback,
32                         void *context = nullptr);
33         fsevents_monitor(const fsevents_monitor& orig) = delete;
34         fsevents_monitor& operator=(const fsevents_monitor& that) = delete;
35
36     protected:
37         void run() override;
38
39     private:
40         static void fsevents_callback(ConstFSEventStreamRef streamRef,
41                                     void *clientCallbackInfo,
42                                     size_t numEvents,
43                                     void *eventPaths,
44                                     const FSEventStreamEventFlags eventFlags[],
45                                     const FSEventStreamEventId eventIds[]);
46
47         FSEventStreamRef stream = nullptr;
48         dispatch_queue_t fsevents_queue = nullptr;
49         bool no_defer();
50         void create_stream(CFArrayRef pathsToWatch);
51     };
52 }
53
54 #endif /* FSW_FSEVENT_MONITOR_H */

```

12.9 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

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

12.9.1 Detailed Description

Solaris/Illumos monitor.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.10 inotify_monitor.hpp

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2014-2015 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16 #ifndef FSW_INOTIFY_MONITOR_H
17 # define FSW_INOTIFY_MONITOR_H
18
19 # include "monitor.hpp"
20 # include <sys/inotify.h>
21 # include <string>
22 # include <vector>
23 # include <sys/stat.h>
24
25 namespace fsw
26 {
27     struct inotify_monitor_impl;
28
29     class inotify_monitor : public monitor
30     {
31     public:
32         inotify_monitor(std::vector<std::string> paths,
33                         FSW_EVENT_CALLBACK *callback,
34                         void *context = nullptr);
35
36         virtual ~inotify_monitor();
37
38     protected:
39         void run();
40
41     private:
42         inotify_monitor(const inotify_monitor& orig) = delete;
43         inotify_monitor& operator=(const inotify_monitor& that) = delete;
44
45         void scan_root_paths();
46         bool is_watched(const std::string& path) const;
47         void preprocess_dir_event(struct inotify_event *event);
48         void preprocess_event(struct inotify_event *event);
49         void preprocess_node_event(struct inotify_event *event);
50         void scan(const std::string& path, const bool accept_non_dirs = true);
51         bool add_watch(const std::string& path,
52                       const struct stat& fd_stat);
53         void process_pending_events();
54         void remove_watch(int fd);
55
56         inotify_monitor_impl *impl;
57     };
58 }
59 #endif /* FSW_INOTIFY_MONITOR_H */

```

12.11 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

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

12.11.1 Detailed Description

kqueue monitor.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.12 kqueue_monitor.hpp

[Go to the documentation of this file.](#)

```
1 /*
2  * Copyright (c) 2014-2021 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16 #ifndef FSW_KQUEUE_MONITOR_H
17 # define FSW_KQUEUE_MONITOR_H
18
19 # include "monitor.hpp"
20 # include <string>
21 # include <vector>
22 # include <sys/stat.h>
23 # include <sys/event.h>
24
25 namespace fsw
```

```

36 {
41     struct kqueue_monitor_load;
42
48     class kqueue_monitor : public monitor
49     {
50     public:
54         kqueue_monitor(std::vector<std::string> paths,
55                         FSW_EVENT_CALLBACK *callback,
56                         void *context = nullptr);
57
61         ~kqueue_monitor() override;
62
63     protected:
71         void run() final;
72
73     private:
74         kqueue_monitor(const kqueue_monitor& orig) = delete;
75         kqueue_monitor& operator=(const kqueue_monitor& that) = delete;
76
77         void initialize_kqueue();
78         void terminate_kqueue();
79         bool scan(const std::string& path, bool is_root_path = true);
80         bool add_watch(const std::string& path, const struct stat& fd_stat);
81         bool is_path_watched(const std::string& path) const;
82         void remove_deleted();
83         void rescan_pending();
84         void scan_root_paths();
85         int wait_for_events(const std::vector<struct kevent>& changes,
86                             std::vector<struct kevent>& event_list) const;
87         void process_events(const std::vector<struct kevent>& event_list,
88                             int event_num);
89
90         int kq = -1;
91         // initial load
92         kqueue_monitor_load *load;
93     };
94 }
95
96 #endif /* FSW_KQUEUE_MONITOR_H */

```

12.13 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

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

12.13.1 Detailed Description

Base exception of the `libfswatch` library.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.14 libfswatch_exception.hpp

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2014-2015 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16 #ifndef LIBFSW_EXCEPTION_H
17 # define LIBFSW_EXCEPTION_H
18
19 # include "../c/error.h"
20 # include <exception>
21 # include <string>
22
23 namespace fsw
24 {
25     class libfsw_exception : public std::exception
26     {
27     public:
28         libfsw_exception(std::string cause, int code = FSW_ERR_UNKNOWN_ERROR);
29         libfsw_exception( const libfsw_exception& other ) noexcept;
30         libfsw_exception& operator=(const libfsw_exception&) noexcept;
31         const char *what() const noexcept override;
32         virtual int error_code() const noexcept;
33         ~libfsw_exception() noexcept override;
34         explicit operator int() const noexcept;
35     private:
36         std::string cause;
37         int code;
38     };
39 }
40 #endif /* LIBFSW_EXCEPTION_H */

```


12.15 libfswatch/c++/libfswatch_map.hpp File Reference

Header defining the associative container used by the library.

```
#include "libfswatch/libfswatch_config.h"
#include <map>
```

Namespaces

- namespace [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.

12.15.1 Detailed Description

Header defining the associative container used by the library.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.16 libfswatch_map.hpp

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2014-2021 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16 #ifndef LIBFSW_MAP_H
17 # define LIBFSW_MAP_H
18
19 #include "libfswatch/libfswatch_config.h"
20
21 # ifdef HAVE_UNORDERED_MAP
22 #   include <unordered_map>
23
24 namespace fsw
25 {
26     template<typename K, typename V>
27     using fsw_hash_map = std::unordered_map<K, V>;
28 }
29 # else
30 #   include <map>
31
32 namespace fsw
33 {
34     template <typename K, typename V>
35     using fsw_hash_map = std::map<K, V>;
36 }
37 # endif
38 /* LIBFSW_MAP_H */

```

12.17 libfswatch/c++/libfswatch_set.hpp File Reference

Header defining the default set type used by the library.

```

#include "libfswatch/libfswatch_config.h"
#include <set>

```

Namespaces

- namespace `fsw`

Main namespace of libfswatch.

Typedefs

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

Default set type used by libfswatch.

12.17.1 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

12.18 libfswatch_set.hpp

[Go to the documentation of this file.](#)

```
1 /*
2 * Copyright (c) 2014-2021 Enrico M. Crisostomo
3 *
4 * This program is free software; you can redistribute it and/or modify it under
5 * the terms of the GNU General Public License as published by the Free Software
6 * Foundation; either version 3, or (at your option) any later version.
7 *
8 * This program is distributed in the hope that it will be useful, but WITHOUT
9 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16 #ifndef LIBFSWATCH_SET_H
17 # define LIBFSWATCH_SET_H
18
19 # include "libfswatch/libfswatch_config.h"
20
21 # if defined(HAVE_UNORDERED_SET)
22 #   include <unordered_set>
23
24 namespace fsw
25 {
26     template<typename K>
27     using fsw_hash_set = std::unordered_set<K>;
28 }
29
30 # else
31 #   include <set>
32
33 namespace fsw
34 {
35     template <typename K>
36     using fsw_hash_set = std::set<K>;
37 }
38
39 # endif
40 /* LIBFSWATCH_SET_H */
```

12.19 libfswatch/c++/monitor.hpp File Reference

Header of the `fsw::monitor` class.

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

Classes

- class `fsw::monitor`
Base class of all monitors.

Namespaces

- namespace `fsw`
Main namespace of libfswatch.

Typedefs

- typedef void `fsw::FSW_EVENT_CALLBACK`(const std::vector< event > &, void *)
Function definition of an event callback.

12.19.1 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

12.20 monitor.hpp

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2014-2021 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16
17 #ifndef FSW__MONITOR_H
18 # define FSW__MONITOR_H
19
20 # include "libfswatch/libfswatch_config.h"
21 # include "filter.hpp"
22 # include <vector>
23 # include <string>
24 # ifdef HAVE_CXX_MUTEX
25 #   include <mutex>
26 # endif
27 # include <atomic>
28 # include <chrono>
29 # include <map>
30 # include "event.hpp"
31 # include "libfswatch/c/cmonitor.h"
32
33 namespace fsw
34 {
35     typedef void FSW_EVENT_CALLBACK(const std::vector<event>&, void *);
36
37     struct compiled_monitor_filter;
38
39     class monitor
40     {
41     public:
42         monitor(std::vector<std::string> paths,
43                FSW_EVENT_CALLBACK *callback,
44                void *context = nullptr);
45
46         virtual ~monitor();
47
48         monitor(const monitor& orig) = delete;
49
50         monitor& operator=(const monitor& that) = delete;
51
52         void set_property(const std::string& name, const std::string& value);
53
54         void set_properties(const std::map<std::string, std::string> options);
55
56         std::string get_property(std::string name);
57
58         void set_latency(double latency);
59
60         void set_fire_idle_event(bool fire_idle_event);
61
62         void set_allow_overflow(bool overflow);
63
64         void set_recursive(bool recursive);
65
66         void set_directory_only(bool directory_only);
67
68         void add_filter(const monitor_filter& filter);
69
70         void set_filters(const std::vector<monitor_filter>& filters);
71
72         void set_follow_symlinks(bool follow);
73
74         void *get_context() const;
75
76         void set_context(void *context);
77
78         void set_bubble_events(bool bubble_events);
79
80         void start();
81
82         void stop();
83
84     };
85
86 }
87
88 #endif

```

```

415     bool is_running();
416
424     void add_event_type_filter(const fsw_event_type_filter& filter);
425
434     void set_event_type_filters(
435         const std::vector<fsw_event_type_filter>& filters);
436
443     void set_watch_access(bool access);
444
445 protected:
455     bool accept_event_type(fsw_event_flag event_type) const;
456
466     bool accept_path(const std::string& path) const;
467
475     void notify_events(const std::vector<event>& events) const;
476
487     void notify_overflow(const std::string& path) const;
488
498     std::vector<fsw_event_flag> filter_flags(const event& evt) const;
499
514     virtual void run() = 0;
515
526     virtual void on_stop();
527
528 protected:
534     std::vector<std::string> paths;
535
542     std::map<std::string, std::string> properties;
543
549     FSW_EVENT_CALLBACK *callback;
550
554     void *context = nullptr;
555
559     double latency = 1.0;
560
566     bool fire_idle_event = false;
567
572     bool allow_overflow = false;
573
577     bool recursive = false;
578
582     bool follow_symlinks = false;
583
587     bool directory_only = false;
588
592     bool watch_access = false;
593
597     bool running = false;
598
602     bool should_stop = false;
603
608     bool bubble_events = false;
609
610 # ifdef HAVE_CXX_MUTEX
615     mutable std::mutex run_mutex;
616
620     mutable std::mutex notify_mutex;
621 # endif
622
623 private:
624     std::chrono::milliseconds get_latency_ms() const;
625     std::vector<compiled_monitor_filter> filters;
626     std::vector<fsw_event_type_filter> event_type_filters;
627
628 #ifdef HAVE_CXX_MUTEX
629 # ifdef HAVE_CXX_ATOMIC
630 #   define HAVE_INACTIVITY_CALLBACK
631     static void inactivity_callback(monitor *mon);
632     mutable std::atomic<std::chrono::milliseconds> last_notification;
633 # endif
634 #endif
635 };
636 }
637
638 #endif /* FSW__MONITOR_H */

```

12.21 libfswatch/c++/monitor_factory.hpp File Reference

Header of the `fsw::monitor_factory` class.

```
#include "monitor.hpp"
#include "libfswatch_set.hpp"
```

Classes

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

Namespaces

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

12.21.1 Detailed Description

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

This header file defines the [fsw::monitor_factory](#) class, the base type of a `libfswatch` monitor factory.

Copyright

Copyright (c) 2014-2018 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.22 monitor_factory.hpp

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2014-2018 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16
17 #ifndef FSW__MONITOR_FACTORY_H
18 # define FSW__MONITOR_FACTORY_H
19
20 #include "monitor.hpp"
21 #include "libfswatch_set.hpp"
22
23 namespace fsw
24 {
25     class monitor_factory
26     {
27     public:
28         static monitor *create_monitor(fsw_monitor_type type,
29                                       std::vector<std::string> paths,
30                                       FSW_EVENT_CALLBACK *callback,
31                                       void *context = nullptr);
32
33         static monitor *create_monitor(const std::string& name,
34                                       std::vector<std::string> paths,
35                                       FSW_EVENT_CALLBACK *callback,
36                                       void *context = nullptr);
37
38         static std::vector<std::string> get_types();
39
40         static bool exists_type(const std::string& name);
41
42         monitor_factory() = delete;
43         monitor_factory(const monitor_factory& orig) = delete;
44         monitor_factory& operator=(const monitor_factory& that) = delete;
45     private:
46         static std::map<std::string, fsw_monitor_type>& creators_by_string();
47     };
48 }
49
50 #endif /* FSW__MONITOR_FACTORY_H */

```

12.23 libfswatch/c++/path_utils.hpp File Reference

Header defining utility functions to manipulate paths.

```

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

```

Namespaces

- namespace `fsw`

Main namespace of libfswatch.

Functions

- `std::string fsw::fsw_realpath` (const char *path, char *resolved_path)
A thin wrapper about realpath.
- `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.

12.23.1 Detailed Description

Header defining utility functions to manipulate paths.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.24 path_utils.hpp

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2014-2015 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16 #ifndef FSW_PATH_UTILS_H
17 # define FSW_PATH_UTILS_H
18
19 # include <string>
20 # include <vector>
21 # include <sys/stat.h>

```

```
32
33 namespace fsw
34 {
44     std::string fsw_realpath(const char *path, char *resolved_path);
45
52     std::vector<std::string> get_directory_children(const std::string& path);
53
66     bool read_link_path(const std::string& path, std::string& link_path);
67
76     bool lstat_path(const std::string& path, struct stat& fd_stat);
77
86     bool stat_path(const std::string& path, struct stat& fd_stat);
87 }
88 #endif /* FSW_PATH_UTILS_H */
```

12.25 libfswatch/c++/poll_monitor.hpp File Reference

stat() based monitor.

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

Classes

- class `fsw::poll_monitor`
stat() -based monitor.

Namespaces

- namespace `fsw`
Main namespace of libfswatch.

12.25.1 Detailed Description

stat() based monitor.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.26 poll_monitor.hpp

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2014-2021 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16
17 #ifndef FSW_POLL_MONITOR_H
18 # define FSW_POLL_MONITOR_H
19
20 # include "monitor.hpp"
21 # include <sys/stat.h>
22 # include <ctime>
23 # include <memory>
24
25 namespace fsw
26 {
27     class poll_monitor : public monitor
28     {
29     public:
30         poll_monitor(std::vector<std::string> paths,
31                     FSW_EVENT_CALLBACK *callback,
32                     void *context = nullptr);
33
34         virtual ~poll_monitor();
35
36     protected:
37         void run();
38
39     private:
40         static const unsigned int MIN_POLL_LATENCY = 1;
41
42         poll_monitor(const poll_monitor& orig) = delete;
43         poll_monitor& operator=(const poll_monitor& that) = delete;
44
45         typedef bool (poll_monitor::*poll_monitor_scan_callback)(
46             const std::string& path,
47             const struct stat& stat);
48
49         typedef struct watched_file_info
50         {
51             time_t mtime;
52             time_t ctime;
53         } watched_file_info;
54
55         struct poll_monitor_data;
56
57         void scan(const std::string& path, poll_monitor_scan_callback fn);
58         void collect_initial_data();
59         void collect_data();
60         bool add_path(const std::string& path,
61                     const struct stat& fd_stat,
62                     poll_monitor_scan_callback poll_callback);
63         bool initial_scan_callback(const std::string& path, const struct stat& stat);
64         bool intermediate_scan_callback(const std::string& path,
65                                       const struct stat& stat);
66
67         void find_removed_files();
68         void swap_data_containers();
69
70         std::unique_ptr<poll_monitor_data> previous_data;
71         std::unique_ptr<poll_monitor_data> new_data;
72
73         std::vector<event> events;
74         time_t curr_time;
75     };
76 }
77
78 #endif /* FSW_POLL_MONITOR_H */

```

12.27 libfswatch/c++/string/string_utils.hpp File Reference

Header of the `fsw::string_utils` namespace.

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

Namespaces

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

12.27.1 Detailed Description

Header of the `fsw::string_utils` namespace.

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.28 string_utils.hpp

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2015 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16 #ifndef FSW_STRING_UTILS_H
17 # define FSW_STRING_UTILS_H
18
19 #include <cstdlib>
20 #include <string>
21
22 namespace fsw
23 {
24     namespace string_utils
25     {
26         std::string string_from_format(const char *format, ...);
27
28         std::string vstring_from_format(const char *format, va_list args);
29     }
30 }
31 #endif /* FSW_STRING_UTILS_H */

```

12.29 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

- namespace `fsw`

Main namespace of `libfswatch`.

12.29.1 Detailed Description

Header of the `fsw::directory_change_event` class.

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.30 win_directory_change_event.hpp

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2015 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16
17 #ifndef FSW_WIN_DIRECTORY_CHANGE_EVENT_H
18 #define FSW_WIN_DIRECTORY_CHANGE_EVENT_H
19
20 #include <cstdlib>
21 #include <string>
22 #include <memory>
23 #include <vector>
24 #include <windows.h>
25 #include "win_handle.hpp"
26 #include "win_error_message.hpp"
27 #include "../event.hpp"
28
29 namespace fsw
30 {
31     class directory_change_event
32     {
33     public:
34         std::wstring path;
35         win_handle handle;
36         size_t buffer_size;
37         DWORD bytes_returned;
38         std::unique_ptr<void, decltype(free)*> buffer = {nullptr, free};
39         std::unique_ptr<OVERLAPPED, decltype(free)*> overlapped = {static_cast<OVERLAPPED *>(malloc(sizeof(OVERLAPPED))), free};
40         win_error_message read_error;
41
42         directory_change_event(size_t buffer_length = 16);
43         bool is_io_incomplete();
44         bool is_buffer_overflowed();
45         bool read_changes_async();
46         bool try_read();
47         void continue_read();
48         std::vector<event> get_events();
49     };
50 }
51
52 #endif /* WIN_DIRECTORY_CHANGE_EVENT_H */

```

12.31 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

- namespace [fsw](#)

Main namespace of libfswatch.

12.31.1 Detailed Description

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

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.32 win_error_message.hpp

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2015 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16
17 #ifndef FSW_WINDOWS_ERROR_MESSAGE_H
18 # define FSW_WINDOWS_ERROR_MESSAGE_H
19
20 # include <string>
21 # include <windows.h>
22
23 namespace fsw
24 {
25     class win_error_message
26     {
27     public:
28         static win_error_message current();
29         win_error_message(DWORD error_code);
30         win_error_message();
31         DWORD get_error_code() const;
32         std::wstring get_message() const;
33         operator std::wstring() const;
34     private:
35         mutable bool initialized = false;
36         mutable std::wstring msg;
37         DWORD err_code;
38     };
39 }
40
41 #endif /* FSW_WINDOWS_ERROR_MESSAGE_H */

```

12.33 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

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

12.33.1 Detailed Description

Header of the `fsw::win_handle` class.

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.34 win_handle.hpp

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2015 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16 #ifndef FSW_WINDOWS_HANDLE_H
17 #define FSW_WINDOWS_HANDLE_H
18
19 #include <windows.h>
20
21 namespace fsw
22 {
23     class win_handle
24     {
25     public:
26         static bool is_valid(const HANDLE & handle);
27
28         win_handle();
29
30         win_handle(HANDLE handle);
31
32         virtual ~win_handle();
33
34         operator HANDLE() const;
35
36         bool is_valid() const;
37
38         win_handle(const win_handle&) = delete;
39
40         win_handle& operator=(const win_handle&) = delete;
41
42         win_handle(win_handle&& other) noexcept;
43
44         win_handle& operator=(win_handle&& other) noexcept;
45
46         win_handle& operator=(const HANDLE& handle);
47     private:
48         HANDLE h;
49     };
50 }
51 #endif /* FSW_WINDOWS_HANDLE_H */

```

12.35 libfswatch/c++/windows/win_paths.hpp File Reference

Header of the `fsw::win_paths` namespace.

```
#include <string>
```

Namespaces

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

12.35.1 Detailed Description

Header of the `fsw::win_paths` namespace.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.36 win_paths.hpp

[Go to the documentation of this file.](#)

```
1 /*
2  * Copyright (c) 2016 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16 #ifndef FSW_WIN_PATHS_HPP
17 # define FSW_WIN_PATHS_HPP
18
19 # include <string>
20
21 namespace fsw
22 {
23     namespace win_paths
24     {
25         std::wstring posix_to_win_w(std::string path);
26
27         std::string win_w_to_posix(std::wstring path);
28     }
29 }
30 #endif /* FSW_WIN_PATHS_HPP */
```

12.37 libfswatch/c++/windows/win_strings.hpp File Reference

Header of the `fsw::win_strings` namespace.

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

Namespaces

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

12.37.1 Detailed Description

Header of the `fsw::win_strings` namespace.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.38 win_strings.hpp

[Go to the documentation of this file.](#)

```
1 /*
2  * Copyright (c) 2016 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16 #ifndef FSW_WIN_STRINGS_H
17 # define FSW_WIN_STRINGS_H
18
19 # include <string>
20 # include <wchar>
21
22 namespace fsw
23 {
24     namespace win_strings
25     {
26         std::string wstring_to_string(wchar_t *s);
27
28         std::string wstring_to_string(const std::wstring& s);
29     }
30 }
31
32 #endif /* FSW_WIN_STRINGS_H */
```

12.39 libfswatch/c++/windows_monitor.hpp File Reference

Windows monitor.

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

Classes

- class [fsw::windows_monitor](#)
Windows monitor.

Namespaces

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

12.39.1 Detailed Description

Windows monitor.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.40 windows_monitor.hpp

[Go to the documentation of this file.](#)

```
1 /*
2  * Copyright (c) 2014-2016 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16 #ifndef FSW_WINDOWS_MONITOR_H
17 # define FSW_WINDOWS_MONITOR_H
18
19 # include "monitor.hpp"
20 # include <string>
21 # include <vector>
22
23 namespace fsw
24 {
25     struct windows_monitor_load;
26 }
27
```

```

47 class windows_monitor : public monitor
48 {
49 public:
53     windows_monitor(std::vector<std::string> paths,
54                     FSW_EVENT_CALLBACK *callback,
55                     void *context = nullptr);
56
60     virtual ~windows_monitor();
61
62 protected:
70     void run();
71
72 private:
73     windows_monitor(const windows_monitor& orig) = delete;
74     windows_monitor& operator=(const windows_monitor& that) = delete;
75
76     void configure_monitor();
77     void initialize_windows_path_list();
78     void initialize_events();
79     bool init_search_for_path(const std::wstring path);
80     void stop_search_for_path(const std::wstring path);
81     void process_path(const std::wstring& path);
82     bool is_path_watched(std::wstring path);
83
84     // initial load
85     windows_monitor_load *load;
86 };
87 }
88
89 #endif /* FSW_WINDOWS_MONITOR_H */

```

12.41 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

- enum `fsw_event_flag` `FSW_ALL_EVENT_FLAGS` [15]

12.41.1 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:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.41.2 Typedef Documentation

12.41.2.1 `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.

12.41.2.2 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.

12.41.3 Enumeration Type Documentation

12.41.3.1 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 including 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.
------	------------------------

Enumerator

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.

Enumerator

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.

12.41.4 Function Documentation

12.41.4.1 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.

12.41.4.2 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.

12.42 cevent.h

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2014-2015 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16
17 #ifndef FSW__CEVENT_H
18 # define FSW__CEVENT_H
19
20 # include <time.h>
21 # include <limits.h>
22 # include "libfswatch_types.h"
23
24 # ifdef __cplusplus
25 extern "C"
26 {
27 # endif
28
29 enum fsw_event_flag
30 {
31     NoOp = 0,
32     PlatformSpecific = (1 << 0),
33     Created = (1 << 1),
34     Updated = (1 << 2),
35     Removed = (1 << 3),
36     Renamed = (1 << 4),
37     OwnerModified = (1 << 5),
38     AttributeModified = (1 << 6),
39     MovedFrom = (1 << 7),
40     MovedTo = (1 << 8),
41     IsFile = (1 << 9),
42     IsDir = (1 << 10),
43     IsSymLink = (1 << 11),
44     Link = (1 << 12),
45     Overflow = (1 << 13)
46 };
47
48 extern const enum fsw_event_flag FSW_ALL_EVENT_FLAGS[15];
49
50 FSW_STATUS fsw_get_event_flag_by_name(const char *name, enum fsw_event_flag *flag);
51
52 char *fsw_get_event_flag_name(const enum fsw_event_flag flag);
53
54 typedef struct fsw_cevent
55 {
56     char * path;
57     time_t evt_time;
58     enum fsw_event_flag * flags;
59     unsigned int flags_num;
60 } fsw_cevent;
61
62 typedef void (*FSW_CEVENT_CALLBACK)(fsw_cevent const *const events,
63                                     const unsigned int event_num,
64                                     void *data);
65
66 # ifdef __cplusplus
67 }
68 # endif
69 #endif /* FSW__CEVENT_H */

```

12.43 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.

12.43.1 Detailed Description

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

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.44 cfilter.h

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2014-2016 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16 #ifndef FSW_CFILTER_H
17 # define FSW_CFILTER_H
18 # include "cevent.h"
19
20 # ifdef __cplusplus
21 extern "C"
22 {
23 # endif
24
25     enum fsw_filter_type
26     {
27         filter_include,
28         filter_exclude
29     };
30
31     typedef struct fsw_cmonitor_filter
32     {
33         char * text;
34         enum fsw_filter_type type;
35         bool case_sensitive;
36         bool extended;
37     } fsw_cmonitor_filter;
38
39     typedef struct fsw_event_type_filter
40     {
41         enum fsw_event_flag flag;
42     } fsw_event_type_filter;
43
44 # ifdef __cplusplus
45 }
46 # endif
47 #endif /* FSW_CFILTER_H */

```

12.45 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.

12.45.1 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

12.45.2 Enumeration Type Documentation

12.45.2.1 `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>	macOS FSEvents monitor.
<code>kqueue_monitor_type</code>	BSD kqueue monitor.

Enumerator

inotify_monitor_type	Linux inotify monitor.
windows_monitor_type	Windows monitor.
poll_monitor_type	stat()- based poll monitor.
fen_monitor_type	Solaris/ illumos monitor.

12.46 cmonitor.h

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2014-2021 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16 #ifndef FSW__CMONITOR_H
17 # define FSW__CMONITOR_H
18
19 # include <time.h>
20
21 # ifdef __cplusplus
22 extern "C"
23 {
24 # endif
25
26 enum fsw_monitor_type
27 {
28     system_default_monitor_type = 0,
29     fsevents_monitor_type,
30     kqueue_monitor_type,
31     inotify_monitor_type,
32     windows_monitor_type,
33     poll_monitor_type,
34     fen_monitor_type
35 };
36
37 # ifdef __cplusplus
38 }
39 # endif
40 /* FSW__CMONITOR_H */

```

12.47 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)

12.47.1 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:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.47.2 Macro Definition Documentation

12.47.2.1 FSW_ERR_CALLBACK_NOT_SET

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

The callback has not been set.

12.47.2.2 FSW_ERR_INVALID_CALLBACK

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

The callback is invalid.

12.47.2.3 FSW_ERR_INVALID_LATENCY

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

The latency is invalid.

12.47.2.4 FSW_ERR_INVALID_PATH

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

The path is invalid.

12.47.2.5 FSW_ERR_INVALID_PROPERTY

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

The property is invalid.

12.47.2.6 FSW_ERR_INVALID_REGEX

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

The regular expression is invalid.

12.47.2.7 FSW_ERR_MEMORY

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

An error occurred while invoking a memory management routine.

12.47.2.8 FSW_ERR_MISSING_CONTEXT

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

The callback context has not been set.

12.47.2.9 FSW_ERR_MONITOR_ALREADY_EXISTS

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

The session already contains a monitor.

12.47.2.10 FSW_ERR_MONITOR_ALREADY_RUNNING

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

A monitor is already running in the specified session.

12.47.2.11 FSW_ERR_PATHS_NOT_SET

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

The paths to watch have not been set.

12.47.2.12 FSW_ERR_SESSION_UNKNOWN

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

The session specified by the handle is unknown.

12.47.2.13 FSW_ERR_UNKNOWN_ERROR

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

An unknown error has occurred.

12.47.2.14 FSW_ERR_UNKNOWN_MONITOR_TYPE

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

The specified monitor type does not exist.

12.47.2.15 FSW_ERR_UNKNOWN_VALUE

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

The value is unknown.

12.47.2.16 FSW_OK

```
#define FSW_OK 0
```

The call was successful.

12.48 error.h

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2014-2015 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16
17 #ifndef FSW__ERROR_H
18 # define FSW__ERROR_H
19
20 # ifdef __cplusplus
21 extern "C"
22 {
23 # endif
24
25 // Error codes
26 # define FSW_OK 0
27 # define FSW_ERR_UNKNOWN_ERROR (1 << 0)
28 # define FSW_ERR_SESSION_UNKNOWN (1 << 1)
29 # define FSW_ERR_MONITOR_ALREADY_EXISTS (1 << 2)
30 # define FSW_ERR_MEMORY (1 << 3)
31 # define FSW_ERR_UNKNOWN_MONITOR_TYPE (1 << 4)
32 # define FSW_ERR_CALLBACK_NOT_SET (1 << 5)
33 # define FSW_ERR_PATHS_NOT_SET (1 << 6)
34 # define FSW_ERR_MISSING_CONTEXT (1 << 7)
35 # define FSW_ERR_INVALID_PATH (1 << 8)
36 # define FSW_ERR_INVALID_CALLBACK (1 << 9)
37 # define FSW_ERR_INVALID_LATENCY (1 << 10)
38 # define FSW_ERR_INVALID_REGEX (1 << 11)
39 # define FSW_ERR_MONITOR_ALREADY_RUNNING (1 << 12)
40 # define FSW_ERR_UNKNOWN_VALUE (1 << 13)
41 # define FSW_ERR_INVALID_PROPERTY (1 << 14)
42 # ifdef __cplusplus
43 }
44 # endif
45 #endif /* FSW__ERROR_H */

```

12.49 libfswatch/c/libfswatch.cpp File Reference

Main libfswatch source file.

```

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

```

Typedefs

- using **FSW_SESSION** = FSW_SESSION { vector< string > paths
- using **fsw_callback_context** = fsw_callback_context { **FSW_HANDLE** handle

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)
- bool **fsw_is_running** (const **FSW_HANDLE** handle)
- **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

- **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**
- static bool **fsw_libfswatch_verbose** = false
- static FSW_THREAD_LOCAL **FSW_STATUS** **last_error**
- static FSW_EVENT_CALLBACK **libfsw_cpp_callback_proxy**

12.49.1 Detailed Description

Main `libfswatch` source file.

Copyright

Copyright (c) 2014-2022 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.17.0

12.49.2 Function Documentation

12.49.2.1 `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 [cfilter.h](#) for the definition of `fsw_event_type_filter`.

12.49.2.2 `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`.

12.49.2.3 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.

12.49.2.4 fsw_add_property()

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

Adds the specified monitor property.

12.49.2.5 fsw_destroy_session()

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

Destroys an existing session and invalidates its handle.

12.49.2.6 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.

12.49.2.7 `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.

12.49.2.8 `fsw_is_running()`

```
bool fsw_is_running (
    const FSW_HANDLE handle )
```

Checks if a monitor exists and is running.

12.49.2.9 `fsw_is_verbose()`

```
bool fsw_is_verbose ( )
```

Check whether the verbose mode is active.

12.49.2.10 `fsw_last_error()`

```
FSW_STATUS fsw_last_error ( )
```

Gets the last error code.

12.49.2.11 `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.

12.49.2.12 `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`.

12.49.2.13 `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.

12.49.2.14 `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.

12.49.2.15 `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.

12.49.2.16 `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.

12.49.2.17 `fsw_set_verbose()`

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

Set the verbose mode.

12.49.2.18 `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.

12.49.2.19 `fsw_stop_monitor()`

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

Stops a running monitor.

12.50 `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)
- [bool fsw_is_running](#) (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)

12.50.1 Detailed Description

Header of the `libfswatch` library.

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

Copyright

Copyright (c) 2014-2022 Enrico M. Crisostomo

License: GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.50.2 Function Documentation

12.50.2.1 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 [cfilter.h](#) for the definition of `fsw_event_type_filter`.

12.50.2.2 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`.

12.50.2.3 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.

12.50.2.4 fsw_add_property()

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

Adds the specified monitor property.

12.50.2.5 fsw_destroy_session()

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

Destroys an existing session and invalidates its handle.

12.50.2.6 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.

12.50.2.7 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.

12.50.2.8 fsw_is_running()

```
bool fsw_is_running (
    const FSW_HANDLE handle )
```

Checks if a monitor exists and is running.

12.50.2.9 fsw_is_verbose()

```
bool fsw_is_verbose ( )
```

Check whether the verbose mode is active.

12.50.2.10 fsw_last_error()

```
FSW_STATUS fsw_last_error ( )
```

Gets the last error code.

12.50.2.11 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.

12.50.2.12 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.

12.50.2.13 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.

12.50.2.14 `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.

12.50.2.15 `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.

12.50.2.16 `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.

12.50.2.17 `fsw_set_verbose()`

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

Set the verbose mode.

12.50.2.18 `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.

12.50.2.19 `fsw_stop_monitor()`

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

Stops a running monitor.

12.51 libfswatch.h

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2014-2022 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16
17 #ifndef LIBFSW_H
18 #define LIBFSW_H
19
20 #include <stdbool.h>
21 #include "libfswatch_types.h"
22 #include "cevent.h"
23 #include "cmonitor.h"
24 #include "cfilter.h"
25 #include "error.h"
26
27 # ifdef __cplusplus
28 extern "C"
29 {
30 # endif
31
32 FSW_STATUS fsw_init_library();
33
34 FSW_HANDLE fsw_init_session(const enum fsw_monitor_type type);
35
36 FSW_STATUS fsw_add_path(const FSW_HANDLE handle, const char * path);
37
38 FSW_STATUS fsw_add_property(const FSW_HANDLE handle, const char * name, const char * value);
39
40 FSW_STATUS fsw_set_allow_overflow(const FSW_HANDLE handle, const bool allow_overflow);
41
42 FSW_STATUS fsw_set_callback(const FSW_HANDLE handle,
43                             const FSW_CEVENT_CALLBACK callback,
44                             void * data);
45
46 FSW_STATUS fsw_set_latency(const FSW_HANDLE handle, const double latency);
47
48 FSW_STATUS fsw_set_recursive(const FSW_HANDLE handle, const bool recursive);
49
50 FSW_STATUS fsw_set_directory_only(const FSW_HANDLE handle, const bool directory_only);
51
52 FSW_STATUS fsw_set_follow_symlinks(const FSW_HANDLE handle,
53                                    const bool follow_symlinks);
54
55 FSW_STATUS fsw_add_event_type_filter(const FSW_HANDLE handle,
56                                     const fsw_event_type_filter event_type);
57
58 FSW_STATUS fsw_add_filter(const FSW_HANDLE handle,
59                           const fsw_cmonitor_filter filter);
60
61 FSW_STATUS fsw_start_monitor(const FSW_HANDLE handle);
62
63 FSW_STATUS fsw_stop_monitor(const FSW_HANDLE handle);
64
65 bool fsw_is_running(const FSW_HANDLE handle);
66
67 FSW_STATUS fsw_destroy_session(const FSW_HANDLE handle);
68
69 FSW_STATUS fsw_last_error();
70
71 bool fsw_is_verbose();
72
73 void fsw_set_verbose(bool verbose);
74
75 # ifdef __cplusplus
76 }
77 # endif
78 #endif /* LIBFSW_H */

```


12.52 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_logf_perror` (const char *format,...)

12.52.1 Detailed Description

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

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.52.2 Function Documentation

12.52.2.1 fsw_flog()

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

Prints the specified message to the specified file.

12.52.2.2 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.

12.52.2.3 fsw_log()

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

Prints the specified message to standard output.

12.52.2.4 fsw_log_perror()

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

Prints the specified message using perror.

12.52.2.5 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.

12.52.2.6 fsw_logf_perror()

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

Prints the specified message using perror. The message string format conforms with printf.

12.53 libfswatch_log.h

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2015 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16 #ifndef LIBFSWATCH_LOG_H
17 # define LIBFSWATCH_LOG_H
18
19 #include <stdio.h>
20
21 void fsw_log(const char * msg);
22
23 void fsw_flog(FILE * f, const char * msg);
24
25 void fsw_logf(const char * format, ...);
26
27 void fsw_flogf(FILE * f, const char * format, ...);
28
29 void fsw_log_perror(const char * msg);
30
31 void fsw_logf_perror(const char * format, ...);
32
33 # define FSW_LOG(msg)          fsw_logf("%s: ", __func__);          fsw_log(msg)
34
35 # define FSW_ELOG(msg)        fsw_flogf(stderr, "%s: ", __func__); fsw_flog(stderr, msg)
36
37 # define FSW_LOGF(msg, ...)   fsw_logf("%s: ", __func__);          fsw_logf(msg, __VA_ARGS__)
38
39 # define FSW_ELOGF(msg, ...)   fsw_flogf(stderr, "%s: ", __func__); fsw_flogf(stderr, msg,
40   __VA_ARGS__)
41
42 # define FSW_FLOGF(f, msg, ...) fsw_flogf(f, "%s: ", __func__);    fsw_flogf(f, msg, __VA_ARGS__)
43
44 #endif /* LIBFSWATCH_LOG_H */

```

12.54 libfswatch/c/libfswatch_types.h File Reference

Header of the `libfswatch` library containing common types.

Typedefs

- typedef struct FSW_SESSION * **FSW_HANDLE**
Handle to a monitoring session.
- typedef int **FSW_STATUS**
Status of a library call.

12.54.1 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:\n GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.55 libfswatch_types.h

[Go to the documentation of this file.](#)

```

1 /*
2  * Copyright (c) 2015-2022 Enrico M. Crisostomo
3  *
4  * This program is free software; you can redistribute it and/or modify it under
5  * the terms of the GNU General Public License as published by the Free Software
6  * Foundation; either version 3, or (at your option) any later version.
7  *
8  * This program is distributed in the hope that it will be useful, but WITHOUT
9  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
10 * FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
11 * details.
12 *
13 * You should have received a copy of the GNU General Public License along with
14 * this program. If not, see <http://www.gnu.org/licenses/>.
15 */
16
17 #ifndef LIBFSWATCH_TYPES_H
18 #define LIBFSWATCH_TYPES_H
19
20 #ifdef __cplusplus
21 extern "C"
22 {
23 #endif
24
25 struct FSW_SESSION;
26
27 typedef struct FSW_SESSION *FSW_HANDLE;
28
29 typedef int FSW_STATUS;
30
31 #ifdef __cplusplus
32 }
33 #endif
34 #endif /* LIBFSWATCH_TYPES_H */

```

12.56 gettext.h

```

1 /* Convenience header for conditional use of GNU <libintl.h>.
2  Copyright (C) 1995-1998, 2000-2002, 2004-2006, 2009-2016 Free Software
3  Foundation, Inc.
4
5  This program is free software: you can redistribute it and/or modify
6  it under the terms of the GNU General Public License as published by
7  the Free Software Foundation; either version 3 of the License, or
8  (at your option) any later version.
9
10 This program is distributed in the hope that it will be useful,
11 but WITHOUT ANY WARRANTY; without even the implied warranty of
12 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

```

```

13 GNU General Public License for more details.
14
15 You should have received a copy of the GNU General Public License
16 along with this program.  If not, see <http://www.gnu.org/licenses/>.  */
17
18 #ifndef _LIBGETTEXT_H
19 #define _LIBGETTEXT_H 1
20
21 /* NLS can be disabled through the configure --disable-nls option.  */
22 #if ENABLE_NLS
23
24 /* Get declarations of GNU message catalog functions.  */
25 # include <libintl.h>
26
27 /* You can set the DEFAULT_TEXT_DOMAIN macro to specify the domain used by
28 the gettext() and ngettext() macros.  This is an alternative to calling
29 textdomain(), and is useful for libraries.  */
30 # ifdef DEFAULT_TEXT_DOMAIN
31 #  undef gettext
32 #  define gettext(Msgid) \
33 dgettext (DEFAULT_TEXT_DOMAIN, Msgid)
34 #  undef ngettext
35 #  define ngettext(Msgid1, Msgid2, N) \
36 dngettext (DEFAULT_TEXT_DOMAIN, Msgid1, Msgid2, N)
37 # endif
38
39 #else
40
41 /* Solaris /usr/include/locale.h includes /usr/include/libintl.h, which
42 chokes if dgettext is defined as a macro.  So include it now, to make
43 later inclusions of <locale.h> a NOP.  We don't include <libintl.h>
44 as well because people using "gettext.h" will not include <libintl.h>,
45 and also including <libintl.h> would fail on SunOS 4, whereas <locale.h>
46 is OK.  */
47 #if defined(__sun)
48 # include <locale.h>
49 #endif
50
51 /* Many header files from the libstdc++ coming with g++ 3.3 or newer include
52 <libintl.h>, which chokes if dcgettext is defined as a macro.  So include
53 it now, to make later inclusions of <libintl.h> a NOP.  */
54 #if defined(__cplusplus) && defined(__GNUG__) && (__GNUC__ >= 3)
55 # include <cstdlib>
56 # if (__GLIBC__ >= 2 && !defined __UCLIBC__) || _GLIBCXX_HAVE_LIBINTL_H
57 # include <libintl.h>
58 # endif
59 #endif
60
61 /* Disabled NLS.
62 The casts to 'const char *' serve the purpose of producing warnings
63 for invalid uses of the value returned from these functions.
64 On pre-ANSI systems without 'const', the config.h file is supposed to
65 contain "#define const".  */
66 # undef gettext
67 # define gettext(Msgid) ((const char *) (Msgid))
68 # undef dgettext
69 # define dgettext(Domainname, Msgid) ((void) (Domainname), gettext (Msgid))
70 # undef dcgettext
71 # define dcgettext(Domainname, Msgid, Category) \
72 ((void) (Category), dgettext (Domainname, Msgid))
73 # undef ngettext
74 # define ngettext(Msgid1, Msgid2, N) \
75 ((N) == 1 \
76 ? ((void) (Msgid2), (const char *) (Msgid1)) \
77 : ((void) (Msgid1), (const char *) (Msgid2)))
78 # undef dngettext
79 # define dngettext(Domainname, Msgid1, Msgid2, N) \
80 ((void) (Domainname), ngettext (Msgid1, Msgid2, N))
81 # undef dcngettext
82 # define dcngettext(Domainname, Msgid1, Msgid2, N, Category) \
83 ((void) (Category), dngettext (Domainname, Msgid1, Msgid2, N))
84 # undef textdomain
85 # define textdomain(Domainname) ((const char *) (Domainname))
86 # undef bindtextdomain
87 # define bindtextdomain(Domainname, Dirname) \
88 ((void) (Domainname), (const char *) (Dirname))
89 # undef bind_textdomain_codeset
90 # define bind_textdomain_codeset(Domainname, Codeset) \
91 ((void) (Domainname), (const char *) (Codeset))
92
93 #endif
94
95 /* Prefer gnuilib's setlocale override over libintl's setlocale override.  */
96 #ifndef GNULIB_defined_setlocale
97 # undef setlocale
98 # define setlocale rpl_setlocale
99 #endif

```

```

100
101 /* A pseudo function call that serves as a marker for the automated
102 extraction of messages, but does not call gettext(). The run-time
103 translation is done at a different place in the code.
104 The argument, String, should be a literal string. Concatenated strings
105 and other string expressions won't work.
106 The macro's expansion is not parenthesized, so that it is suitable as
107 initializer for static 'char[]' or 'const char[]' variables. */
108 #define gettext_noop(String) String
109
110 /* The separator between msgctxt and msgid in a .mo file. */
111 #define GETTEXT_CONTEXT_GLUE "\\004"
112
113 /* Pseudo function calls, taking a MSGCTXT and a MSGID instead of just a
114 MSGID. MSGCTXT and MSGID must be string literals. MSGCTXT should be
115 short and rarely need to change.
116 The letter 'p' stands for 'particular' or 'special'. */
117 #ifndef DEFAULT_TEXT_DOMAIN
118 # define pgettext(Msgctxt, Msgid) \
119 pgettext_aux(DEFAULT_TEXT_DOMAIN, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, LC_MESSAGES)
120 #else
121 # define pgettext(Msgctxt, Msgid) \
122 pgettext_aux(NULL, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, LC_MESSAGES)
123 #endif
124 #define dpgettext(Domainname, Msgctxt, Msgid) \
125 pgettext_aux(Domainname, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, LC_MESSAGES)
126 #define dcpgettext(Domainname, Msgctxt, Msgid, Category) \
127 pgettext_aux(Domainname, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, Category)
128 #ifndef DEFAULT_TEXT_DOMAIN
129 # define npgettext(Msgctxt, Msgid, MsgidPlural, N) \
130 npgettext_aux(DEFAULT_TEXT_DOMAIN, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, MsgidPlural, N,
131 LC_MESSAGES)
132 #else
133 # define npgettext(Msgctxt, Msgid, MsgidPlural, N) \
134 npgettext_aux(NULL, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, MsgidPlural, N, LC_MESSAGES)
135 #endif
136 #define dnpgettext(Domainname, Msgctxt, Msgid, MsgidPlural, N) \
137 npgettext_aux(Domainname, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, MsgidPlural, N, LC_MESSAGES)
138 #define dcngettext(Domainname, Msgctxt, Msgid, MsgidPlural, N, Category) \
139 npgettext_aux(Domainname, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, MsgidPlural, N, Category)
140
141 #ifdef __GNUC__
142 #define __inline
143 #else
144 #define __inline __cplusplus
145 #endif
146 #endif
147 static const char *
148 pgettext_aux(const char *domain,
149             const char *msg_ctxt_id, const char *msgid,
150             int category)
151 {
152     const char *translation = dcgettext(domain, msg_ctxt_id, category);
153     if (translation == msg_ctxt_id)
154         return msgid;
155     else
156         return translation;
157 }
158
159 #ifdef __GNUC__
160 #define __inline
161 #else
162 #define __inline __cplusplus
163 #endif
164 #endif
165 #endif
166 static const char *
167 npgettext_aux(const char *domain,
168             const char *msg_ctxt_id, const char *msgid,
169             const char *msgid_plural, unsigned long int n,
170             int category)
171 {
172     const char *translation =
173     dcngettext(domain, msg_ctxt_id, msgid_plural, n, category);
174     if (translation == msg_ctxt_id || translation == msgid_plural)
175         return (n == 1 ? msgid : msgid_plural);
176     else
177         return translation;
178 }
179
180 /* The same thing extended for non-constant arguments. Here MSGCTXT and MSGID
181 can be arbitrary expressions. But for string literals these macros are
182 less efficient than those above. */
183
184 #include <string.h>
185

```

```

186 #if ((__GNUC__ >= 3 || __GNUC__ >= 2) && !defined __STRICT_ANSI_) \
187 /* || __STDC_VERSION__ >= 199901L */
188 # define _LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS 1
189 #else
190 # define _LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS 0
191 #endif
192
193 #if !_LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS
194 #include <stdlib.h>
195 #endif
196
197 #define pgettext_expr(Msgtxt, Msgid) \
198 dcpgettext_expr (NULL, Msgtxt, Msgid, LC_MESSAGES)
199 #define dpgettext_expr(Domainname, Msgtxt, Msgid) \
200 dcpgettext_expr (Domainname, Msgtxt, Msgid, LC_MESSAGES)
201
202 #ifdef __GNUC__
203 __inline
204 #else
205 #ifdef __cplusplus
206 inline
207 #endif
208 #endif
209 static const char *
210 dcpgettext_expr (const char *domain,
211                 const char *msgtxt, const char *msgid,
212                 int category)
213 {
214     size_t msgtxt_len = strlen (msgtxt) + 1;
215     size_t msgid_len = strlen (msgid) + 1;
216     const char *translation;
217     #if _LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS
218     char msg_ctxt_id[msgtxt_len + msgid_len];
219     #else
220     char buf[1024];
221     char *msg_ctxt_id =
222         (msgtxt_len + msgid_len <= sizeof (buf)
223          ? buf
224          : (char *) malloc (msgtxt_len + msgid_len));
225     if (msg_ctxt_id != NULL)
226     #endif
227     {
228         int found_translation;
229         memcpy (msg_ctxt_id, msgtxt, msgtxt_len - 1);
230         msg_ctxt_id[msgtxt_len - 1] = '\004';
231         memcpy (msg_ctxt_id + msgtxt_len, msgid, msgid_len);
232         translation = dcgettext (domain, msg_ctxt_id, category);
233         found_translation = (translation != msg_ctxt_id);
234     #if !_LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS
235         if (msg_ctxt_id != buf)
236             free (msg_ctxt_id);
237     #endif
238         if (found_translation)
239             return translation;
240     }
241     return msgid;
242 }
243
244 #define npgettext_expr(Msgtxt, Msgid, MsgidPlural, N) \
245 dcngettext_expr (NULL, Msgtxt, Msgid, MsgidPlural, N, LC_MESSAGES)
246 #define dnpgettext_expr(Domainname, Msgtxt, Msgid, MsgidPlural, N) \
247 dcngettext_expr (Domainname, Msgtxt, Msgid, MsgidPlural, N, LC_MESSAGES)
248
249 #ifdef __GNUC__
250 __inline
251 #else
252 #ifdef __cplusplus
253 inline
254 #endif
255 #endif
256 static const char *
257 dcngettext_expr (const char *domain,
258                 const char *msgtxt, const char *msgid,
259                 const char *msgid_plural, unsigned long int n,
260                 int category)
261 {
262     size_t msgtxt_len = strlen (msgtxt) + 1;
263     size_t msgid_len = strlen (msgid) + 1;
264     const char *translation;
265     #if _LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS
266     char msg_ctxt_id[msgtxt_len + msgid_len];
267     #else
268     char buf[1024];
269     char *msg_ctxt_id =
270         (msgtxt_len + msgid_len <= sizeof (buf)
271          ? buf
272          : (char *) malloc (msgtxt_len + msgid_len));

```

```

273     if (msg_ctxt_id != NULL)
274 #endif
275     {
276         int found_translation;
277         memcpy (msg_ctxt_id, msgctxt, msgctxt_len - 1);
278         msg_ctxt_id[msgctxt_len - 1] = '\004';
279         memcpy (msg_ctxt_id + msgctxt_len, msgid, msgid_len);
280         translation = dcgettext (domain, msg_ctxt_id, msgid_plural, n, category);
281         found_translation = !(translation == msg_ctxt_id || translation == msgid_plural);
282 #if !_LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS
283         if (msg_ctxt_id != buf)
284             free (msg_ctxt_id);
285 #endif
286         if (found_translation)
287             return translation;
288     }
289     return (n == 1 ? msgid : msgid_plural);
290 }
291
292 #endif /* _LIBGETTEXT_H */

```

12.57 gettext_defs.h

```

1 /*
2 * Copyright (C) 2014-2015 Enrico M. Crisostomo
3 *
4 * This program is free software; you can redistribute it and/or modify
5 * it under the terms of the GNU General Public License as published by
6 * the Free Software Foundation; either version 3, or (at your option)
7 * any later version.
8 *
9 * This program is distributed in the hope that it will be useful,
10 * but WITHOUT ANY WARRANTY; without even the implied warranty of
11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12 * GNU General Public License for more details.
13 *
14 * You should have received a copy of the GNU General Public License
15 * along with this program. If not, see <http://www.gnu.org/licenses/>.
16 */
17 #ifndef FSW_GETTEXT_DEFS_H
18 # define FSW_GETTEXT_DEFS_H
19
20 #include "gettext.h"
21
22 # ifdef __cplusplus
23 extern "C"
24 {
25 # endif
26
27 #define _(String) gettext(String)
28
29 # ifdef __cplusplus
30 }
31 # endif
32
33 #endif /* FSW_GETTEXT_DEFS_H */

```


Index

- ~event
 - fsw::event, [39](#)
- ~monitor
 - fsw::monitor, [55](#)
- ~win_handle
 - fsw::win_handle, [76](#)

- accept_event_type
 - fsw::monitor, [55](#)
- accept_path
 - fsw::monitor, [56](#)
- add_event_type_filter
 - fsw::monitor, [56](#)
- add_filter
 - fsw::monitor, [57](#)
- AttributeModified
 - cevent.h, [117](#)

- callback
 - fsw::monitor, [66](#)
- cevent.h
 - AttributeModified, [117](#)
 - Created, [117](#)
 - fsw_cevent, [115](#)
 - FSW_CEVENT_CALLBACK, [115](#)
 - fsw_event_flag, [116](#)
 - fsw_get_event_flag_by_name, [118](#)
 - fsw_get_event_flag_name, [119](#)
 - IsDir, [118](#)
 - IsFile, [118](#)
 - IsSymLink, [118](#)
 - Link, [118](#)
 - MovedFrom, [117](#)
 - MovedTo, [118](#)
 - NoOp, [116](#)
 - Overflow, [118](#)
 - OwnerModified, [117](#)
 - PlatformSpecific, [117](#)
 - Removed, [117](#)
 - Renamed, [117](#)
 - Updated, [117](#)
- cmonitor.h
 - fen_monitor_type, [124](#)
 - fsevents_monitor_type, [123](#)
 - fsw_monitor_type, [123](#)
 - inotify_monitor_type, [124](#)
 - kqueue_monitor_type, [123](#)
 - poll_monitor_type, [124](#)
 - system_default_monitor_type, [123](#)
 - windows_monitor_type, [124](#)

- create_monitor
 - fsw::monitor_factory, [68](#), [69](#)
- Created
 - cevent.h, [117](#)
- current
 - fsw::win_error_message, [74](#)

- DARWIN_EVENTSTREAM_NO_DEFER
 - fsw::fsevents_monitor, [44](#)

- error.h
 - FSW_ERR_CALLBACK_NOT_SET, [125](#)
 - FSW_ERR_INVALID_CALLBACK, [125](#)
 - FSW_ERR_INVALID_LATENCY, [126](#)
 - FSW_ERR_INVALID_PATH, [126](#)
 - FSW_ERR_INVALID_PROPERTY, [126](#)
 - FSW_ERR_INVALID_REGEX, [126](#)
 - FSW_ERR_MEMORY, [126](#)
 - FSW_ERR_MISSING_CONTEXT, [126](#)
 - FSW_ERR_MONITOR_ALREADY_EXISTS, [126](#)
 - FSW_ERR_MONITOR_ALREADY_RUNNING, [126](#)
 - FSW_ERR_PATHS_NOT_SET, [127](#)
 - FSW_ERR_SESSION_UNKNOWN, [127](#)
 - FSW_ERR_UNKNOWN_ERROR, [127](#)
 - FSW_ERR_UNKNOWN_MONITOR_TYPE, [127](#)
 - FSW_ERR_UNKNOWN_VALUE, [127](#)
 - FSW_OK, [127](#)
- error_code
 - fsw::libfsw_exception, [50](#)
- event
 - fsw::event, [39](#)
- exists_type
 - fsw::monitor_factory, [70](#)
- extended
 - fsw, [31](#)

- fen_monitor_type
 - cmonitor.h, [124](#)
- filter_flags
 - fsw::monitor, [57](#)
- fire_idle_event
 - fsw::monitor, [67](#)
- fsevents_monitor_type
 - cmonitor.h, [123](#)
- fsw, [23](#)
 - extended, [31](#)
 - FSW_EVENT_CALLBACK, [25](#)
 - fsw_hash_map, [25](#)
 - fsw_hash_set, [26](#)

- fsw_realpath, 26
- get_directory_children, 27
- lstat_path, 27
- monitor_filter, 26
- operator<<, 28
- read_from_file, 28
- read_link_path, 29
- stat_path, 30
- fsw::compiled_monitor_filter, 37
- fsw::directory_change_event, 37
- fsw::event, 38
 - ~event, 39
 - event, 39
 - get_event_flag_by_name, 40
 - get_event_flag_name, 40
 - get_flags, 41
 - get_path, 41
 - get_time, 41
- fsw::fen_monitor, 42
 - run, 42
- fsw::fsevents_monitor, 43
 - DARWIN_EVENTSTREAM_NO_DEFER, 44
 - run, 43
- fsw::inotify_monitor, 46
 - run, 46
- fsw::inotify_monitor_impl, 47
- fsw::kqueue_monitor, 47
 - run, 48
- fsw::libfsw_exception, 49
 - error_code, 50
 - libfsw_exception, 49
 - what, 50
- fsw::monitor, 50
 - ~monitor, 55
 - accept_event_type, 55
 - accept_path, 56
 - add_event_type_filter, 56
 - add_filter, 57
 - callback, 66
 - filter_flags, 57
 - fire_idle_event, 67
 - get_context, 57
 - get_property, 58
 - is_running, 58
 - monitor, 54
 - notify_events, 58
 - notify_overflow, 58
 - on_stop, 59
 - paths, 67
 - properties, 67
 - run, 59
 - set_allow_overflow, 59
 - set_bubble_events, 60
 - set_context, 60
 - set_directory_only, 61
 - set_event_type_filters, 61
 - set_filters, 62
 - set_fire_idle_event, 62
 - set_follow_symlinks, 63
 - set_latency, 63
 - set_properties, 64
 - set_property, 64
 - set_recursive, 65
 - set_watch_access, 65
 - start, 65
 - stop, 66
- fsw::monitor_factory, 68
 - create_monitor, 68, 69
 - exists_type, 70
 - get_types, 71
- fsw::poll_monitor, 71
 - run, 72
- fsw::string_utils, 31
 - string_from_format, 31
 - vstring_from_format, 32
- fsw::win_error_message, 73
 - current, 74
 - get_error_code, 74
 - get_message, 74
 - operator std::wstring, 75
 - win_error_message, 73, 74
- fsw::win_flag_type, 75
- fsw::win_handle, 75
 - ~win_handle, 76
 - is_valid, 77
 - operator=, 78
 - win_handle, 76
- fsw::win_paths, 32
 - posix_to_win_w, 33
 - win_w_to_posix, 33
- fsw::win_strings, 34
 - wstring_to_string, 34
- fsw::windows_monitor, 79
 - run, 80
- fsw_add_event_type_filter
 - libfswatch.cpp, 130
 - libfswatch.h, 135
- fsw_add_filter
 - libfswatch.cpp, 130
 - libfswatch.h, 136
- fsw_add_path
 - libfswatch.cpp, 130
 - libfswatch.h, 136
- fsw_add_property
 - libfswatch.cpp, 131
 - libfswatch.h, 136
- fsw_cevent, 44
 - cevent.h, 115
- FSW_CEVENT_CALLBACK
 - cevent.h, 115
- fsw_cmonitor_filter, 45
- fsw_destroy_session
 - libfswatch.cpp, 131
 - libfswatch.h, 136
- FSW_ERR_CALLBACK_NOT_SET
 - error.h, 125

- FSW_ERR_INVALID_CALLBACK
 - error.h, [125](#)
- FSW_ERR_INVALID_LATENCY
 - error.h, [126](#)
- FSW_ERR_INVALID_PATH
 - error.h, [126](#)
- FSW_ERR_INVALID_PROPERTY
 - error.h, [126](#)
- FSW_ERR_INVALID_REGEX
 - error.h, [126](#)
- FSW_ERR_MEMORY
 - error.h, [126](#)
- FSW_ERR_MISSING_CONTEXT
 - error.h, [126](#)
- FSW_ERR_MONITOR_ALREADY_EXISTS
 - error.h, [126](#)
- FSW_ERR_MONITOR_ALREADY_RUNNING
 - error.h, [126](#)
- FSW_ERR_PATHS_NOT_SET
 - error.h, [127](#)
- FSW_ERR_SESSION_UNKNOWN
 - error.h, [127](#)
- FSW_ERR_UNKNOWN_ERROR
 - error.h, [127](#)
- FSW_ERR_UNKNOWN_MONITOR_TYPE
 - error.h, [127](#)
- FSW_ERR_UNKNOWN_VALUE
 - error.h, [127](#)
- FSW_EVENT_CALLBACK
 - fsw, [25](#)
- fsw_event_flag
 - cevent.h, [116](#)
- fsw_event_type_filter, [45](#)
- fsw_flog
 - libfswatch_log.h, [142](#)
- fsw_flogf
 - libfswatch_log.h, [142](#)
- fsw_get_event_flag_by_name
 - cevent.h, [118](#)
- fsw_get_event_flag_name
 - cevent.h, [119](#)
- fsw_hash_map
 - fsw, [25](#)
- fsw_hash_set
 - fsw, [26](#)
- fsw_init_library
 - libfswatch.cpp, [131](#)
 - libfswatch.h, [136](#)
- fsw_init_session
 - libfswatch.cpp, [132](#)
 - libfswatch.h, [137](#)
- fsw_is_running
 - libfswatch.cpp, [132](#)
 - libfswatch.h, [137](#)
- fsw_is_verbose
 - libfswatch.cpp, [132](#)
 - libfswatch.h, [138](#)
- fsw_last_error
 - libfswatch.cpp, [132](#)
 - libfswatch.h, [138](#)
- fsw_log
 - libfswatch_log.h, [142](#)
- fsw_log_perror
 - libfswatch_log.h, [142](#)
- fsw_logf
 - libfswatch_log.h, [142](#)
- fsw_logf_perror
 - libfswatch_log.h, [142](#)
- fsw_monitor_type
 - cmonitor.h, [123](#)
- FSW_OK
 - error.h, [127](#)
- fsw_realpath
 - fsw, [26](#)
- fsw_set_allow_overflow
 - libfswatch.cpp, [132](#)
 - libfswatch.h, [138](#)
- fsw_set_callback
 - libfswatch.cpp, [133](#)
 - libfswatch.h, [138](#)
- fsw_set_directory_only
 - libfswatch.cpp, [133](#)
 - libfswatch.h, [138](#)
- fsw_set_follow_symlinks
 - libfswatch.cpp, [133](#)
 - libfswatch.h, [138](#)
- fsw_set_latency
 - libfswatch.cpp, [133](#)
 - libfswatch.h, [139](#)
- fsw_set_recursive
 - libfswatch.cpp, [133](#)
 - libfswatch.h, [139](#)
- fsw_set_verbose
 - libfswatch.cpp, [134](#)
 - libfswatch.h, [139](#)
- fsw_start_monitor
 - libfswatch.cpp, [134](#)
 - libfswatch.h, [139](#)
- fsw_stop_monitor
 - libfswatch.cpp, [134](#)
 - libfswatch.h, [139](#)
- get_context
 - fsw::monitor, [57](#)
- get_directory_children
 - fsw, [27](#)
- get_error_code
 - fsw::win_error_message, [74](#)
- get_event_flag_by_name
 - fsw::event, [40](#)
- get_event_flag_name
 - fsw::event, [40](#)
- get_flags
 - fsw::event, [41](#)
- get_message
 - fsw::win_error_message, [74](#)
- get_path

- fsw::event, 41
- get_property
 - fsw::monitor, 58
- get_time
 - fsw::event, 41
- get_types
 - fsw::monitor_factory, 71
- inotify_monitor_type
 - cmonitor.h, 124
- is_running
 - fsw::monitor, 58
- is_valid
 - fsw::win_handle, 77
- IsDir
 - cevent.h, 118
- IsFile
 - cevent.h, 118
- IsSymLink
 - cevent.h, 118
- kqueue_monitor_type
 - cmonitor.h, 123
- libfsw_exception
 - fsw::libfsw_exception, 49
- libfswatch.cpp
 - fsw_add_event_type_filter, 130
 - fsw_add_filter, 130
 - fsw_add_path, 130
 - fsw_add_property, 131
 - fsw_destroy_session, 131
 - fsw_init_library, 131
 - fsw_init_session, 132
 - fsw_is_running, 132
 - fsw_is_verbose, 132
 - fsw_last_error, 132
 - fsw_set_allow_overflow, 132
 - fsw_set_callback, 133
 - fsw_set_directory_only, 133
 - fsw_set_follow_symlinks, 133
 - fsw_set_latency, 133
 - fsw_set_recursive, 133
 - fsw_set_verbose, 134
 - fsw_start_monitor, 134
 - fsw_stop_monitor, 134
- libfswatch.h
 - fsw_add_event_type_filter, 135
 - fsw_add_filter, 136
 - fsw_add_path, 136
 - fsw_add_property, 136
 - fsw_destroy_session, 136
 - fsw_init_library, 136
 - fsw_init_session, 137
 - fsw_is_running, 137
 - fsw_is_verbose, 138
 - fsw_last_error, 138
 - fsw_set_allow_overflow, 138
 - fsw_set_callback, 138
 - fsw_set_directory_only, 138
 - fsw_set_follow_symlinks, 138
 - fsw_set_latency, 139
 - fsw_set_recursive, 139
 - fsw_set_verbose, 139
 - fsw_start_monitor, 139
 - fsw_stop_monitor, 139
- libfswatch/c++/event.hpp, 81, 82
- libfswatch/c++/fen_monitor.hpp, 83, 84
- libfswatch/c++/filter.hpp, 84, 86
- libfswatch/c++/fsevents_monitor.hpp, 86, 87
- libfswatch/c++/inotify_monitor.hpp, 88, 89
- libfswatch/c++/kqueue_monitor.hpp, 89, 90
- libfswatch/c++/libfswatch_exception.hpp, 91, 92
- libfswatch/c++/libfswatch_map.hpp, 93, 94
- libfswatch/c++/libfswatch_set.hpp, 94, 95
- libfswatch/c++/monitor.hpp, 96, 97
- libfswatch/c++/monitor_factory.hpp, 98, 100
- libfswatch/c++/path_utils.hpp, 100, 101
- libfswatch/c++/poll_monitor.hpp, 102, 103
- libfswatch/c++/string/string_utils.hpp, 104, 105
- libfswatch/c++/windows/win_directory_change_event.hpp, 105, 106
- libfswatch/c++/windows/win_error_message.hpp, 107, 108
- libfswatch/c++/windows/win_handle.hpp, 108, 109
- libfswatch/c++/windows/win_paths.hpp, 110, 111
- libfswatch/c++/windows/win_strings.hpp, 111, 112
- libfswatch/c++/windows_monitor.hpp, 112, 113
- libfswatch/c/cevent.h, 114, 119
- libfswatch/c/cfilter.h, 120, 122
- libfswatch/c/cmonitor.h, 122, 124
- libfswatch/c/error.h, 124, 128
- libfswatch/c/libfswatch.cpp, 128
- libfswatch/c/libfswatch.h, 134, 140
- libfswatch/c/libfswatch_log.h, 141, 143
- libfswatch/c/libfswatch_types.h, 143, 144
- libfswatch/gettext.h, 144
- libfswatch/gettext_defs.h, 148
- libfswatch_log.h
 - fsw_flog, 142
 - fsw_flogf, 142
 - fsw_log, 142
 - fsw_log_perror, 142
 - fsw_logf, 142
 - fsw_logf_perror, 142
- Link
 - cevent.h, 118
- Istat_path
 - fsw, 27
- monitor
 - fsw::monitor, 54
- monitor_filter
 - fsw, 26
- MovedFrom
 - cevent.h, 117
- MovedTo
 - cevent.h, 118

NoOp
 cevent.h, 116

notify_events
 fsw::monitor, 58

notify_overflow
 fsw::monitor, 58

on_stop
 fsw::monitor, 59

operator std::wstring
 fsw::win_error_message, 75

operator<<
 fsw, 28

operator=
 fsw::win_handle, 78

Overflow
 cevent.h, 118

OwnerModified
 cevent.h, 117

paths
 fsw::monitor, 67

PlatformSpecific
 cevent.h, 117

poll_monitor_type
 cmonitor.h, 124

posix_to_win_w
 fsw::win_paths, 33

properties
 fsw::monitor, 67

read_from_file
 fsw, 28

read_link_path
 fsw, 29

Removed
 cevent.h, 117

Renamed
 cevent.h, 117

run
 fsw::fen_monitor, 42
 fsw::fsevents_monitor, 43
 fsw::inotify_monitor, 46
 fsw::kqueue_monitor, 48
 fsw::monitor, 59
 fsw::poll_monitor, 72
 fsw::windows_monitor, 80

set_allow_overflow
 fsw::monitor, 59

set_bubble_events
 fsw::monitor, 60

set_context
 fsw::monitor, 60

set_directory_only
 fsw::monitor, 61

set_event_type_filters
 fsw::monitor, 61

set_filters
 fsw::monitor, 62

set_fire_idle_event
 fsw::monitor, 62

set_follow_symlinks
 fsw::monitor, 63

set_latency
 fsw::monitor, 63

set_properties
 fsw::monitor, 64

set_property
 fsw::monitor, 64

set_recursive
 fsw::monitor, 65

set_watch_access
 fsw::monitor, 65

start
 fsw::monitor, 65

stat_path
 fsw, 30

stop
 fsw::monitor, 66

string_from_format
 fsw::string_utils, 31

system_default_monitor_type
 cmonitor.h, 123

Updated
 cevent.h, 117

vstring_from_format
 fsw::string_utils, 32

what
 fsw::libfsw_exception, 50

win_error_message
 fsw::win_error_message, 73, 74

win_handle
 fsw::win_handle, 76

win_w_to_posix
 fsw::win_paths, 33

windows_monitor_type
 cmonitor.h, 124

wstring_to_string
 fsw::win_strings, 34