# libfswatch

# libfswatch

Cross-platform file change monitor C/C++ library with multiple backends
for `libfswatch` version 1.6.0, 30 September 2015

**Enrico M. Crisostomo**

This manual is for `libfswatch` (version 1.6.0, 30 September 2015), a cross-platform file change monitor C/C++ library with multiple backends, including Apple OS X File System Events API, *BSD kqueue, Linux inotify, Microsoft Windows and a `stat`-based backend.

# Short Contents

# Table of Contents

# 1 History

## 1.1 4:0:1

### 1.1.1 Feature Changes

The changes introducted in `4:0:1` are the following:

- A monitor for Microsoft Windows was added.
- A logging function has been added to the library to log verbose messages.

### 1.1.2 C++ Interface Changes

- The `windows_monitor` class has been added to provide a monitor that uses the Windows API.

### 1.1.3 C Interface Changes

- A family of functions and macros have been added to log diagnostic messages and conditionally print them only in verbose mode:
  - `fsw_log`
  - `fsw_flog`
  - `fsw_logf`
  - `fsw_flogf`
  - `fsw_log_perror`
  - `FSW_LOG`
  - `FSW_ELOG`
  - `FSW_LOGF`
  - `FSW_ELOGF`
  - `FSW_FLOGF`

## 1.2 3:0:0

### 1.2.1 Feature Changes

The changes introducted in `3:0:0` are the following:

- The possibility of filtering events by *event type* was added.
- All the monitors were refactored and the callback invocation logic was moved to the base `monitor` class.
- All the monitors were refactored and the filtering logic (both by path and event type) was moved to the base `monitor` class.

### 1.2.2 C++ Interface Changes

- Added the `monitor::add_event_type_filter` method to add an event type filter.
- Added the `monitor::set_event_type_filters` method to set the event type filters.
- Added the `monitor::notify_events` method to centralize event filtering and dispatching into the base monitor class.
- Added the `event::get_event_flag_by_name` method to lookup an event type by name.
- Added the `event::get_event_flag_name` method to get the name of an event type.
- Added the overloaded `ostream& operator<<(ostream& out, const fsw_event_flag flag)` operator to simplify printing the name of an event type on a stream.

### 1.2.3 C Interface Changes

- Added the `fsw_event_type_filter` structure to represent a event type filter.
- Added the `FSW_ERR_UNKNOWN_VALUE` exit code.
- Added the `fsw_add_event_type_filter` function to add an event type filter.
- Added the `fsw_get_event_flag_by_name` function to lookup an event type by name.
- Added the `fsw_get_event_flag_name` function to get the name of an event type.

# 2 Introduction

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

- A monitor based on the *FSEvents* API of Apple OS X.
- A monitor based on *kqueue*, an event notification interface introduced in FreeBSD 4.1 and supported on most *BSD systems (including OS X).
- A monitor based on *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.

The first releases of `fswatch` were monolithic, self-contained binaries whose output was typically piped to other applications for processing. Given the nature of the features provided by `fswatch`, however, we recognized the need to expose this functionality through a library and the `libfswatch` package was born. `fswatch` is now built upon it and all its functionality is provided by the `libfswatch` library, with the exception perhaps of some formatting routines printing results to the standard output.

The biggest issue we have faced while developing `fswatch` was abstracting the behaviour of different backends behind a common interface and `libfswatch` is the result of that effort. Instead of using different APIs, a programmer can use just one: `libfswatch`'s. The advantages of using `libfswatch` are many:

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

## 2.1 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.

## 2.2 Relation between `fswatch` and `libfswatch`

Although `fswatch` uses functionality provided by `libfswatch` and depends on it, `libfswatch` is currently a package nested into `fswatch`. If either component is updated, the whole package is, as well as their version numbers are. From the GNU Build System point of view, the package version of `libfswatch` is always kept in sync with `fswatch`'s.

The library API version, however, *is not*, and it is the only piece of information that should be kept into account when linking against `libfswatch`. Since we use `libtool` to build `libfswatch`, we adopt `libtool`'s versioning scheme for library interface versions.

## 2.3 `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.

## 2.4 The C and the C++ API

The C API is built on top of the C++ API but the two are very different and the main difference reflect the 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 'façade' 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.

## 2.5 Thread Safety

The C++ API does not deal with thread safety explicitely. 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.

The C API, a layer above the C++ API, has been designed in order to provide the same basic guarantee:

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

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 `configure`'s output 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.
```

## 2.6 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. If you with you may contact the authors at the addresses listed in the `AUTHORS` file.

When reporting a bug, please be sure to include as much detail as possible and sufficient information to reproduce it.

# 3 The C++ API

The C++ API provides users an easy to use, object-oriented, common and intuitive 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 typical API usage pattern is similar to the following:

- An instance of a `monitor` is either created directly or through the factory (see [Monitor Discovery], page 7).
- The monitor is configured according to the user needs (see [Monitors], page 8).
- The monitor is *run* and change events are waited for.

## 3.1 Monitor Discovery

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 `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 by *type* or *name*.

The `monitor_factory` class provides the following methods:

`static monitor * create_monitor`

Creates a monitor of the specified type with the specified constructor parameters (see [Monitors], page 8). A monitor of the platform default type can be created if `fsw_monitor_type::system_default_monitor_type`. If the named monitor is not available, then return `nullptr`.

`static monitor * create_monitor_by_name`

Creates a monitor of the specified type (by *name*) and constructor parameters (see [Monitors], page 8). If the named monitor is not available, then return `nullptr`.

`static std::vector<std::string> get_types()`
: Get the list of available monitor types. The type name can then be used to get a monitor instance by name using `create_monitor_by_name`.

`static bool exists_type(const std::string& name)`
: Query whether the specified monitor exists.

`static void register_type(const std::string& name, fsw_monitor_type type)`
: Register a monitor type in the list of available implementations.

## 3.2 Monitor Registration

In order for monitor types to be visible to the factory they have to be registered. Currently they can be registered using two helper macros, defined in `monitor.h`:

`REGISTER_MONITOR(classname, monitor_type)`
: This macro must be invoked into a class' header file and must be passed the class name and the monitor type.

`REGISTER_MONITOR_IMPL(classname, monitor_type)`
: This macro must be invoked into a class' source file and must be passed the class name and the monitor type.

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.

## 3.3 Monitors

The `monitor` class is the fundamental type of the C++ API: it defines the interface of every monitor and provides common functionality to inheritors of this class.

The public interface of a monitor is the following:

```
class monitor
{
public:
  monitor(std::vector<std::string> paths,
          FSW_EVENT_CALLBACK * callback,
          void * context = nullptr);
  virtual ~monitor();

  monitor(const monitor& orig) = delete;
  monitor& operator=(const monitor & that) = delete;

  void set_properties(
```

```
      std::map<std::string, std::string> options);
    std::string get_property(std::string name);
    void set_latency(double latency);
    void set_allow_overflow(bool overflow);
    void set_recursive(bool recursive);
    void add_filter(const monitor_filter &filter);
    void set_filters(
      const std::vector<monitor_filter> &filters);
    void set_follow_symlinks(bool follow);
    void * get_context() const;
    void set_context(void * context);
    void start();
    void add_event_type_filter(
      const fsw_event_type_filter &filter);
    void set_event_type_filters(
      const std::vector<fsw_event_type_filter> &filters);
  }
```

A monitor is thus a type with the following characteristics:

- It cannot be copied.
- It cannot be assigned.
- It is disegned for extension.

### 3.3.1 Business Interface

The business interface of the `monitor` class is the following:

`void set_properties(std::map<std::string, std::string> options);`

This function sets a map of monitor-specific properties.

`std::string get_property(std::string name);`

This function returns the property named `name`.

`void set_allow_overflow(bool allow_overflow);`

This function sets the allow overflow flag to the specified value. If this flag is set, then the monitor will report a monitor buffer overflow as a change event of type `fsw_event_flag::Overflow`.

`void set_latency(double latency);`

This function sets the latency of the monitor in seconds. This method only sets the latency value. The exact meaning of latency and how it is enforced depends on a monitor implementation.

`void set_recursive(bool recursive);`

This function sets the `recursive` flag of the monitor to indicate whether the monitor should recursively observe the contents of directories.

void add_filter(const monitor_filter &filter);
: This function adds a monitor_filter instance to the filter list of the current monitor.

void set_filters(const std::vector<monitor_filter> &filters);
: This function sets the filter list of the current monitor, substituting existing filter if any.

void set_follow_symlinks(bool follow);
: This function sets the follow_symlinks flag of the monitor to indicate whether the monitor should follow observed symbolic links or observe the links themselves.

void * get_context() const;
: This function gest the pointer to the context data that is passed to the callback by the monitor.

void set_context(void * context);
: This function sets the pointer to the context data that is passed to the callback by the monitor.

void start();
: This function starts the monitor so that it begins listening to file system change events.

void add_event_type_filter(const fsw_event_type_filter &filter);
: This function adds a fsw_event_type_filter instance to the event type filter list of the current monitor.

void set_event_type_filters(const std::vector<fsw_event_type_filter> &filters);
: This function sets the event type filter list of the current monitor, substituting existing filters if any.

### 3.3.2 Implementing Monitors

monitor is a class that declares the following protected functions:

bool accept_event_type(fsw_event_flag event_type) const
: This function checks whether the specified event_type can be accepted according to the list of event type filters of the monitor.

bool accept_path(const std::string &path) const
: This function checks whether the specified path can be accepted according to the list of filters of the monitor.

bool accept_path(const char *path) const
: This function checks whether the specified path can be accepted according to the list of filters of the monitor.

void notify_events(const std::vector<event> &events) const
: This function notifies that detection of the specified events.

`void notify_overflow(const std::string & path) const`
: This function notifies that the monitor has overflowed. *Overflowing* is a monitor-specific concept and not all monitors experience this behaviour.

`std::vector<fsw_event_flag> filter_flags(const event &evt) const`
: This function filters the list of flags of an event `evt` using the list of event type filters of the monitor. If no filters are set, all the flags are returned.

`virtual void run() = 0`
: This pure virtual function shall contain the logic of a monitor implementation. This function will be invoked by the monitor's `start` API function.

Since it contains a pure virtual function, `run()`, the `monitor` class is abstract. Inheritors are required to provide an implementation of the `run()` function containing the monitor logic and its 'event loop'.

### 3.3.3 The Anatomy of a Typical Monitor

The anatomy of monitors is typically very similar and it can be illustrated with the following algorithm (written in pseudo-code):

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

  while (true)
  {
    scan_paths();
    wait_for_events(latency);

    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);
    }
  }
```

```
}
```

Despite being a minimal implementation, this algorithm exemplifies the common tasks 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.
- It scans the paths that must be observed: this step might be necessary for example because some path may not have existed during the loop's previous iteration, 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 last approximately the *latency* configured into the monitor.
- Events are filtered to exclude those that refer to paths that do not satisfy the filters of the monitor.
- The `notify_events` method is called on the base class to filter the events and notify the caller.

## 3.4 Events

Events are modeled by the `fsw::event` class, defined in the `event.h` header:

```
class event
{
public:
  event(std::string path,
        time_t evt_time,
        std::vector<fsw_event_flag> flags);
  virtual ~event();

  std::string get_path() const;
  time_t get_time() const;
  std::vector<fsw_event_flag> get_flags() const;

  static fsw_event_flag
    get_event_flag_by_name(const std::string &name);
  static std::string
    get_event_flag_name(const fsw_event_flag &flag);

private:
  std::string path;
  time_t evt_time;
  std::vector<fsw_event_flag> evt_flags;
};
```

The `event` class provides a simple and uniform representation of an event to all the API. An event has got the following characteristics:

- The *path* it relates to.

- The timestamp *evt_time* of the moment the event was raised.
- The list event *flags* (see [Event Flags], page 20).

Currently the API provides no way for monitor implementors to provide additional, monitor-dependent fields to an event. Since the API stores events by value into collections (such as `vector`), an extended event would be *sliced* and additional fields would be lost.

### 3.4.1 Looking Up Event Types by Name

Event types can be looked up by *name* using the following function:

`fsw_event_flag get_event_flag_by_name(const std::string &name);`
: Returns the `fsw_event_flag` instance whose name is *name*. If no instance is found with the specified *name*, this function will throw a `libfsw_exception`.

### 3.4.2 Getting the Name of an Event Type

The name of an event type can be obtained using the following function:

`static std::string get_event_flag_name(const fsw_event_flag &flag);`
: This function returns the name of the specified event type.

Most of the times, the name of an event type is used when writing user output: to ease this task, the `event.h` header defines the following operator overload:

`ostream& operator<<(ostream& out, const fsw_event_flag flag);`
: This operator writes the name of the specified `fsw_event_flag` to the stream.

## 3.5 Path Filters

*Path filters* are regular expression used to accept or reject file change events based on the value of their path. A filter is represented by the `fsw::monitor_filter` type, defined in the `filter.h` header:

```
typedef struct monitor_filter
{
  std::string text;
  fsw_filter_type type;
  bool case_sensitive;
  bool extended;
} monitor_filter;
```

and has the following characteristics:

`text` The regular expression used to match paths.

`type` The filter type can either be an *inclusion* or *exclusion* filter.

`case_sensitive`
A flag indicating the filter case sensitivitiy.

`extended` A flag indicating whether `text` is an extended regular expression.

### 3.5.1 Filter Types

A filter type determine 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 definition of filters has no effect.

The `fswatch` Info documentation has a user-oriented discussion of how filters are used.

## 3.6 Event Type Filters

*Event type filters* let callers filter the events using a specified set of event types. An event type filter is represented by the `fsw_event_type_filter` type, defined in the `cfilter.h` header:

```
typedef struct fsw_event_type_filter
{
  fsw_event_flag flag;
} fsw_event_type_filter;
```

# 4 The C API

## 4.1 Overview

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

### 4.1.1 Translating the C++ API

The conventions used to translate C++ types into C types are rather common:

- `std::string` is represented as a 'NUL'-terminated `char *`.
- Lists are represented as arrays whose length is specified in a separate field: `flags_num` indicates how many elements are stored in the array pointed by `flags`.
- More complex types are usually translated as a `struct` containing data fields and a set of functions to operate on it.

### 4.1.2 Thread Safety

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 (see [Thread Safety], page 4).

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

## 4.2 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);
}
```

## 4.3 Status Codes and Errors

Most API functions return a status code of type `FSW_STATUS` (`error.h` header) 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. Currently, the following status codes are defined:

`FSW_OK`
`0` The operation completed successfully.

`FSW_ERR_UNKNOWN_ERROR`
`(1 << 0)` An error occurred.

`FSW_ERR_SESSION_UNKNOWN`
`(1 << 1)` The session identified by the specified handle does not exist.

`FSW_ERR_MONITOR_ALREADY_EXISTS`
`(1 << 2)` The session already contains a monitor.

`FSW_ERR_MEMORY`
`(1 << 3)` An error occurred while using a memory management routine.

`FSW_ERR_UNKNOWN_MONITOR_TYPE`
`(1 << 4)` The specified monitor type does not exist.

`FSW_ERR_CALLBACK_NOT_SET`
`(1 << 5)` The callback is not set.

`FSW_ERR_PATHS_NOT_SET`
`(1 << 6)` The paths are not set.

`FSW_ERR_UNKNOWN_MONITOR`
`(1 << 7)` Unused.

`FSW_ERR_MISSING_CONTEXT`
`(1 << 8)` The callback context is missing.

`FSW_ERR_INVALID_PATH`
`(1 << 9)` The path is invalid.

`FSW_ERR_INVALID_CALLBACK`
`(1 << 10)` The callback is invalid.

`FSW_ERR_INVALID_LATENCY`
`(1 << 11)` The latency is invalid.

`FSW_ERR_INVALID_REGEX`
`(1 << 12)` The regular expression is invalid.

`FSW_ERR_MONITOR_ALREADY_RUNNING`
`(1 << 13)` A monitor is already running in the specified session.

`FSW_ERR_STALE_MONITOR_THREAD`
`(1 << 14)` Unused.

`FSW_ERR_THREAD_FAULT`
`(1 << 15)` Unused.

`FSW_ERR_UNSUPPORTED_OPERATION`
`(1 << 16)` Unused.

`FSW_ERR_UNKNOWN_VALUE`
`(1 << 17)` Unused.

`FSW_ERR_INVALID_PROPERTY`
`(1 << 18)` The specified property is invalid.

## 4.4 Functions

The library `libfswatch.h` header file defines the functions listed in the following table. As seen in See [Status Codes and Errors], page 16, functions return `FSW_OK` if they succeed, otherwise they return an error code. Functions that modify an existing monitoring sessions accept the session handle of type `FSW_HANDLE`.

`FSW_STATUS`
`fsw_init_library()`
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 will not be usable.

`FSW_HANDLE`
`fsw_init_session(const fsw_monitor_type type = system_default_monitor_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.

`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.

`FSW_STATUS`
`fsw_add_property(const FSW_HANDLE handle, const char * name, const char * value)`
This function adds a new key-value pair `(name, value)` in the monitor's property map.

`FSW_STATUS`
`fsw_set_allow_overflow(const FSW_HANDLE handle, const bool allow_overflow)`
Sets the allow overflow flag to the specified value. If this flag is set, monitor buffer overflows will be reported as change events of type `fsw_event_flag::Overflow`.

`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 (see [Callbacks], page 19) and an optional pointer to context data (see [Context Data], page 19). The callback must be set in the current session in order for it to be valid.

`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 second.

`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.

`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, symbolic links are not followed.

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

Adds a filter to the current session. A filter (see [Filters], page 13) 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.

`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. A filter (see [Event Type Filters], page 14) contains the *name* of the event type to include into the output. A session may contain multiple event type filters.

`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.

`FSW_STATUS`
`fsw_destroy_session(const FSW_HANDLE handle)`
Destroys an existing session and invalidates its handle.

`FSW_STATUS`
`fsw_last_error()`
Gets the last error code.

`bool`
`fsw_is_verbose()`
Check whether the verbose mode is active.

## 4.5 Callbacks

When a monitor receives change events satisfying all the session criteria, a callback provided by the user is invoked and passed a copy of the events; a function pointer of type `FSW_CEVENT_CALLBACK` is used by the API as a callback:

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

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`, a pointer to an optional user-provided context.

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.

### 4.5.1 Context Data

A *context* may be passed to the callback when events are received. Context data may be useful to easily associate a 'state' to each monitoring session. A monitoring session does *not* acquire ownership of the context data pointer; therefore, the following are responsibilities of the caller:

- To keep the pointer valid throughout the life of a monitoring session that shares this pointer.
- To free the pointed memory *after* when the pointer is not shared with any monitoring session any longer.

## 4.6 Memory Management Functions

The C API published by the `libfswatch` library contains some memory management routines. These functions, defined in the `libfswatch_mem.h` header file, are the following:

`void * fsw_alloc(size_t size)`
: This function allocates a chunk of memory of the specified *size* and returns a pointer to it. If the memory could not be allocated, `nullptr` is returned instead.

`void fsw_free(void * ptr)`
: This function frees the memory pointed by *ptr*.

`fsw_free`

## 4.7 Events in the C API

The C API represents events as instances of the `fsw_cevent` structure (`cevent.h`) which is an exact translation of the `fsw:event` type (see [Events], page 12) where C++ types and collections are represented by C friendly equivalent types:

```
typedef struct fsw_cevent
{
  char * path;
  time_t evt_time;
  fsw_event_flag * flags;
  unsigned int flags_num;
} fsw_cevent;
```

## 4.8 Event Flags

Events flags are `enum` values shared by both the C++ and the C API and they are defined in the `cevent.h` header. The values of event flags are power of 2, that is numbers $f$ in the form $f = 2^n$ 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. The `fsw_event_flag` enumeration currently includes the following values:

`NoOp`
`0`
: This event flag is used as a marker.

`PlatformSpecific`
`1 << 0`
: This event flag represents a platform-specific flag that is not encoded as any other event flag by the API.

`Created`
`1 << 1`
: This event flag represents a file creation creation event.

`Updated`
`1 << 2`
: This event flag represents a file update update event.

`Removed`
`1 << 3`
: This event flag represents a file removal event.

`Renamed`
`1 << 4`
: This event flag represents a file rename event.

`OwnerModified`
`1 << 5` This event flag represents a file owner modification event.

`AttributeModified`
`1 << 6` This event flag represents a file attribute modification event.

`MovedFrom`
`1 << 7` This event flag represents a file rename event.

`MovedTo`
`1 << 8` This event flag represents a file rename event.

`IsFile`
`1 << 9` This event flag indicates that the modified object is a regular file.

`IsDir`
`1 << 10` This event flag indicates that the modified object is a directory.

`IsSymLink`
`1 << 11` This event flag indicates that the modified object is a symbolic link.

`Link`
`1 << 12` This event flag represents a file link event.

`Overflow`
`1 << 13` This event flag represents a monitor buffer overflow.

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 flags 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 the `PlatformSpecific` API flags. The API currently offers no way to retain a platform-specific event flag value in this case.

The `cevent.h` header also defines the following utility functions:

`fsw_event_flag fsw_get_event_flag_by_name(const char * name);`
: This function looks for a `fsw_event_flag` instace with the specified *name*. If a matching event type is not found, this function will return a negative number.

`char * fsw_get_event_flag_name(const fsw_event_flag flag);`
: This function returns a `char *` pointer to the name of the specified event type, *flag*, or `nullptr` if an error occurs. The memory pointed by the return value of this function should be freed with a call to `fsw_free` (see [Memory Management Functions], page 19).

## 4.9 Filters in the C API

The C API represents filters (see [Filters], page 13) as instances of the `fsw_cmonitor_filter` structure, defined in the `cfilter.h` header. This structure is a translation of the `monitor_filter` class using C equivalent types:

```
enum fsw_filter_type
{
  filter_include,
  filter_exclude
};

typedef struct fsw_cmonitor_filter
{
  char * text;
  fsw_filter_type type;
  bool case_sensitive;
  bool extended;
} fsw_cmonitor_filter;
```

## 4.10 Monitor Types

The `fsw_monitor_type` enumeration, defined in the `cmonitor.h` header, contains a list of monitor types built into the `libfswatch` library:

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

Members of this enumeration may be used with factory methods (see [Monitor Discovery], page 7 provided by the API to request a monitor of a specific type. The members of this enumeration must be known at compile time.

## 4.11 Logging

The `libfswatch` library never writes any output to the standard streams, unless the *verbose* mode is set: in this mode, diagnostic information is written to standard error. The library offers a set of logging functions to ease the task of conditionally writing both literal and formatted messages to the output, when the verbose flag is set.

### 4.11.1 Logging functions

`void fsw_log(const char * msg)`
: This function prints the specified message `msg` literally to the standard output.

`void fsw_flog(FILE * f, const char * msg)`
: This function prints the specified message `msg` literally to the file `f`.

`void fsw_logf(const char * format, ...);`
: This function prints and formats the specified message `format` to the standard output.

`void fsw_flogf(FILE * f, const char * format, ...)`
: This function prints and formats the specified message `format` to the file `f`.

`void fsw_log_perror(const char * msg)`
: This function prints and formats the specified message `msg` using `perror`.

### 4.11.2 Logging macros

The `libfswatch` library provides a set of macros that can be used to print diagnostic messages containing the name of the method where the macro is called from. The format of a message `msg` printed using one of these macros from inside a function called `func` will be similar to:

```
func: msg
```

`FSW_LOG(msg)`
: This macro prints to standard output the name of the function this macro is called from followed by the message `msg`.

`FSW_ELOG(msg)`
: This macro prints to standard error the name of the function this macro is called from followed by the message `msg`.

`FSW_LOGF(msg, ...)`
: This macro prints to standard output the name of the function this macro is called from followed by the message `msg`, formatted using the specified arguments.

`FSW_ELOGF(msg, ...)`
: This macro prints to standard error the name of the function this macro is called from followed by the message `msg`, formatted using the specified arguments.

`FSW_FLOGF(f, msg, ...)`
: This macro prints to the specified file the name of the function this macro is called from followed by the message `msg`, formatted using the specified arguments.

## 4.12 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);
```

# Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008



0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

   The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

## ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

# Appendix B Index of Functions

The fsw:: and std:: prefixes have been dropped by function names in the following index.

## E



## F

## M



## O



## R

# Appendix C Index of Data Types

The `fsw::` prefix has been dropped by function names in the following index.

## E



## F



## L



## M



## T



## W

# Appendix D Index of Programs

# Index

## S



## T



## V