fswatch

1.11.3

Generated by Doxygen 1.8.13

Contents

Chapter 1

Main Page

1.1 Introduction

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

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

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

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

1.2 Changelog

See the History page.

1.3 Available Bindings

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

1.4 libtool's versioning scheme

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

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

1.5 The C and the C++ API

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

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

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

1.6 Thread Safety

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

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

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

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

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

1.7 C++11

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

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

1.8 Reporting Bugs and Suggestions

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

Chapter 2

C++ API

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

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

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

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

2.1 Usage

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

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

2.2 Example

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

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

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

Chapter 3

C API

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

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

3.1 Translating the C++ API to C

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

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

3.2 Thread Safety

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

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

3.3 Library Initialization

Before calling any library method, the library must be initialized by calling the fsw_init_library() function:

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

3.4 Status Codes and Errors

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

3.5 Example

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

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

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

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

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

Chapter 4

History

4.1 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().

4.2 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.

4.3 8:0:2

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

4.4 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.

4.5 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.

4.6 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()`.

4.7 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

4.8 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 lters, 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:

Chapter 7

Hierarchical Index

7.1 Class Hierarchy

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

Chapter 8

Class Index

8.1 Class List

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

Chapter 9

File Index

9.1 File List

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

Chapter 10

Namespace Documentation

10.1 fsw Namespace Reference

Main namespace of `libfswatch`.

Namespaces

- string_utils

  This namespace contains string manipulation functions.
- win_paths

  Path conversion functions.
- win_strings

  String conversion functions.

Classes

- struct compiled_monitor_filter
- class directory_change_event

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

  Type representing a file change event.
- class fen_monitor

  Solaris/Illumos monitor.
- struct FSEventFlagType
- class fsevents_monitor

  OS X FSEvents monitor.
- class inotify_monitor

  Solaris/Illumos monitor.
- struct inotify_monitor_impl
- class kqueue_monitor

  Solaris/Illumos monitor.
- class libfsw_exception

  Base exception of the `libfswatch` library.

- class monitor

  Base class of all monitors.
- class monitor_factory

  Object factory class for fsw::monitor instances.
- struct monitor_filter

  Path filters used to accept or reject file change events.
- class monitor_registrant

  Helper class to register monitor factories.
- class poll_monitor

  `stat()`-based monitor.
- class win_error_message

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

  A RAII wrapper around Microsoft Windows `HANDLE`.
- class windows_monitor

  Windows monitor.

Typedefs

- typedef struct fsw::monitor_filter monitor_filter

  Path filters used to accept or reject file change events.
- typedef struct fsw::FSEventFlagType **FSEventFlagType**
- template<typename K , typename V >
  using fsw_hash_map = std::map< K, V >

  Default associative container type used by `libfswatch`.
- template<typename K >
  using fsw_hash_set = std::set< K >

  Default set type used by `libfswatch`.
- typedef void FSW_EVENT_CALLBACK(const std::vector< event > &, void *)

  Function definition of an event callback.
- typedef monitor *(* **FSW_FN_MONITOR_CREATOR**) (std::vector< std::string > paths, FSW_EVENT_C↩ ALLBACK *callback, void *context)
- typedef struct fsw::poll_monitor::poll_monitor_data **poll_monitor_data**

Functions

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

  Overload of the << operator to print an event using `iostreams`.
- static bool **parse_filter** (std::string filter, monitor_filter &filter_object, void(*err_handler)(std::string))
- static bool **is_unescaped_space** (string &filter, long i)
- bool **parse_filter** (string filter, monitor_filter &filter_object, void(*err_handler)(string))
- static vector< FSEventFlagType > **create_flag_type_vector** ()
- **REGISTER_MONITOR_IMPL** (fsevents_monitor, fsevents_monitor_type)
- static vector< fsw_event_flag > **decode_flags** (FSEventStreamEventFlags flag)
- **REGISTER_MONITOR_IMPL** (inotify_monitor, inotify_monitor_type)
- static monitor * **create_default_monitor** (vector< string > paths, FSW_EVENT_CALLBACK *callback, void *context)
- vector< string > **get_directory_children** (const string &path)
- bool **read_link_path** (const string &path, string &link_path)

- bool **stat_path** (const string &path, struct stat &fd_stat)
- bool **lstat_path** (const string &path, struct stat &fd_stat)
- std::vector< std::string > get_directory_children (const std::string &path)

  Gets a vector of direct directory children.
- bool read_link_path (const std::string &path, std::string &link_path)

  Resolves a path name.
- bool lstat_path (const std::string &path, struct stat &fd_stat)

  Wraps a `lstat(path, fd_stat)` call that invokes `perror()` if it fails.
- bool stat_path (const std::string &path, struct stat &fd_stat)

  Wraps a `stat(path, fd_stat)` call that invokes `perror()` if it fails.
- **REGISTER_MONITOR_IMPL** (poll_monitor, poll_monitor_type)
- static vector< win_flag_type > **create_flag_type_vector** ()
- static vector< fsw_event_flag > **decode_flags** (DWORD flag)

Variables

- static const vector< FSEventFlagType > **event_flag_type** = create_flag_type_vector()
- static const unsigned int **BUFFER_SIZE** = (10 * ((sizeof(struct inotify_event)) + NAME_MAX + 1))
- static const vector< win_flag_type > **event_flag_type** = create_flag_type_vector()

10.1.1 Detailed Description

Main namespace of `libfswatch`.

10.1.2 Typedef Documentation

10.1.2.1 FSW_EVENT_CALLBACK

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

Function definition of an event callback.

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

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

10.1.2.2 fsw_hash_map

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

Default associative container type used by `libfswatch`.

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

10.1.2.3 fsw_hash_set

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

Default set type used by `libfswatch`.

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

10.1.2.4 monitor_filter

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

Path filters used to accept or reject file change events.

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

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

Further information about how filtering works in `libfswatch` can be found in Path Filtering.

10.1.3 Function Documentation

10.1.3.1 get_directory_children()

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

Gets a vector of direct directory children.

Parameters

path	The directory whose children must be returned.

Returns

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

10.1.3.2 lstat_path()

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

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

Parameters

path	The path to `lstat()`.
fd_stat	The `stat` structure where `lstat()` writes its results.

Returns

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

10.1.3.3 operator<<()

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

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

Parameters

out	A reference to the output stream.
flag	The flag to print.

Returns

A reference to the stream.

10.1.3.4 read_link_path()

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

Resolves a path name.

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

Parameters

path	The path to resolve.
link_path	A reference to a `std::string` where the resolved absolute path should be copied to.

Returns

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

10.1.3.5 stat_path()

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

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

Parameters

path	The path to `stat()`.
fd_stat	The `stat` structure where `stat()` writes its results.

Returns

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

10.2 fsw::string_utils Namespace Reference

This namespace contains string manipulation functions.

Functions

- string vstring_from_format (const char *format, va_list args)

  Create a `std::string` using a `printf()` format and a `va_list args`.
- string string_from_format (const char *format,...)

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

10.2.1 Detailed Description

This namespace contains string manipulation functions.

10.2.2 Function Documentation

10.2.2.1 string_from_format()

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

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

Parameters

format	The `printf()` 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

format	The `printf()` format.
args	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

path	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

path	The Windows path to convert to POSIX.

Returns

The converted POSIX path.

10.4 fsw::win_strings Namespace Reference

String conversion functions.

Functions

- string wstring_to_string (wchar_t *s)

  Converts a wide character string into a string.
- string **wstring_to_string** (const wstring &s)
- std::string wstring_to_string (const std::wstring &s)

  Converts a wide character string into a string.

10.4.1 Detailed Description

String conversion functions.

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

10.4.2 Function Documentation

10.4.2.1 wstring_to_string() [1/2]

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

Converts a wide character string into a string.

Parameters

s	The `wchar_t` array to convert.

Returns

The converted string.

10.4.2.2 wstring_to_string() [2/2]

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

Converts a wide character string into a string.

Parameters

s	The string to convert.

Returns

The converted string.

Chapter 11

Class Documentation

11.1 fsw::compiled_monitor_filter Struct Reference

Public Attributes

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

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

- libfswatch/c++/monitor.cpp

11.2 fsw::directory_change_event Class Reference

Header of the fsw::directory_change_event class, a helper class to wrap Microsoft Windows' `ReadDirectory↩ ChangesW` 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

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

11.3.2.2 ~event()

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

Destructs an event.

This is a virtual destructor that performs no operations.

11.3.3 Member Function Documentation

11.3.3.1 get_event_flag_by_name()

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

Get event flag by name.

Parameters

name	The name of the event flag to look for.

Returns

The event flag whose name is `name`, otherwise

Exceptions

libfsw_exception	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

flag	The event flag.

Returns

The name of `flag`.

Exceptions

libfsw_exception	if no event flag is found.

11.3.3.3 get_flags()

```
vector< fsw_event_flag > fsw::event::get_flags ( ) const
```

Returns the flags of the event.

Returns

The flags of the event.

11.3.3.4 get_path()

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

Returns the path of the event.

Returns

The path of the event.

11.3.3.5 get_time()

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

Returns the time of the event.

Returns

The time of the event.

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

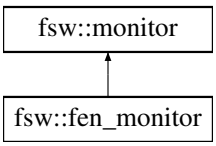
- libfswatch/c++/event.hpp
- libfswatch/c++/event.cpp

11.4 fsw::fen_monitor Class Reference

Solaris/Illumos monitor.

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

Inheritance diagram for fsw::fen_monitor:



Public Member Functions

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

Protected Member Functions

- void run () override

  Executes the monitor loop.

Additional Inherited Members

11.4.1 Detailed Description

Solaris/Illumos monitor.

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

11.4.2 Member Function Documentation

11.4.2.1 run()

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

Executes the monitor loop.

This call does not return until the monitor is stopped.

See also

stop()

Implements fsw::monitor.

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

- libfswatch/c++/fen_monitor.hpp

11.5 fsw::FSEventFlagType Struct Reference

Public Attributes

- FSEventStreamEventFlags **flag**
- fsw_event_flag **type**

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

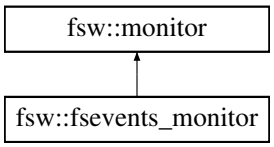
- libfswatch/c++/fsevents_monitor.cpp

11.6 fsw::fsevents_monitor Class Reference

OS X FSEvents monitor.

`#include <fsevents_monitor.hpp>`

Inheritance diagram for fsw::fsevents_monitor:



Public Member Functions

- fsevents_monitor (std::vector< std::string > paths, FSW_EVENT_CALLBACK *callback, void *context=nullptr)
  Constructs an instance of this class.
- **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.
- void on_stop () override
  Execute an implementation-specific stop handler.

Additional Inherited Members

11.6.1 Detailed Description

OS X FSEvents monitor.

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

11.6.2 Member Function Documentation

11.6.2.1 run()

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

Executes the monitor loop.

This call does not return until the monitor is stopped.

See also

stop()

Implements fsw::monitor.

11.6.3 Member Data Documentation

11.6.3.1 DARWIN_EVENTSTREAM_NO_DEFER

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

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/kfseventstreamcreateflagnodef

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

- libfswatch/c++/fsevents_monitor.hpp
- libfswatch/c++/fsevents_monitor.cpp

11.7 fsw_callback_context Struct Reference

Public Attributes

- FSW_HANDLE **handle**
- FSW_CEVENT_CALLBACK **callback**
- void * **data**

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

- libfswatch/c/libfswatch.cpp

11.8 fsw_cevent Struct Reference

`#include <cevent.h>`

Public Attributes

- char * **path**
- time_t **evt_time**
- enum fsw_event_flag * **flags**
- unsigned int **flags_num**

11.8.1 Detailed Description

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

- path is the path where the event was triggered.
- evt_time the time when the event was triggered.
- flags is an array of fsw_event_flag of size flags_num.
- flags_num is the size of the flags array.

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

- libfswatch/c/cevent.h

11.9 fsw_cmonitor_filter Struct Reference

Public Attributes

- char * **text**
- enum fsw_filter_type **type**
- bool **case_sensitive**
- bool **extended**

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

- libfswatch/c/cfilter.h

11.10 fsw_event_type_filter Struct Reference

Event type filter.

`#include <cfilter.h>`

Public Attributes

- enum fsw_event_flag **flag**

11.10.1 Detailed Description

Event type filter.

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

- libfswatch/c/cfilter.h

11.11 FSW_SESSION Struct Reference

Public Attributes

- vector< string > **paths**
- fsw_monitor_type **type**
- fsw::monitor * **monitor**
- FSW_CEVENT_CALLBACK **callback**
- double **latency**
- bool **allow_overflow**
- bool **recursive**
- bool **directory_only**
- bool **follow_symlinks**
- vector< monitor_filter > **filters**
- vector< fsw_event_type_filter > **event_type_filters**
- map< string, string > **properties**
- void * **data**

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

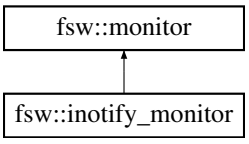
- libfswatch/c/libfswatch.cpp

11.12 fsw::inotify_monitor Class Reference

Solaris/Illumos monitor.

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

Inheritance diagram for fsw::inotify_monitor:



Public Member Functions

- inotify_monitor (std::vector< std::string > paths, FSW_EVENT_CALLBACK *callback, void *context=nullptr)

  Constructs an instance of this class.
- virtual ~inotify_monitor ()

  Destroys an instance of this class.

Protected Member Functions

- void run ()

  Executes the monitor loop.

Additional Inherited Members

11.12.1 Detailed Description

Solaris/Illumos monitor.

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

11.12.2 Member Function Documentation

11.12.2.1 run()

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

Executes the monitor loop.

This call does not return until the monitor is stopped.

See also

stop()

Implements fsw::monitor.

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

- libfswatch/c++/inotify_monitor.hpp
- libfswatch/c++/inotify_monitor.cpp

11.13 fsw::inotify_monitor_impl Struct Reference

Public Attributes

- int **inotify_monitor_handle** = -1
- vector< event > **events**
- fsw_hash_set< int > **watched_descriptors**
- fsw_hash_map< string, int > **path_to_wd**
- fsw_hash_map< int, string > **wd_to_path**
- fsw_hash_set< int > **descriptors_to_remove**
- fsw_hash_set< int > **watches_to_remove**
- vector< string > **paths_to_rescan**
- time_t **curr_time**

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

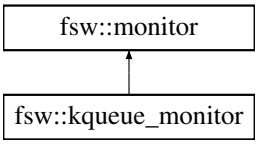
- libfswatch/c++/inotify_monitor.cpp

11.14 fsw::kqueue_monitor Class Reference

Solaris/Illumos monitor.

`#include <kqueue_monitor.hpp>`

Inheritance diagram for fsw::kqueue_monitor:



Public Member Functions

- kqueue_monitor (std::vector< std::string > paths, FSW_EVENT_CALLBACK *callback, void *context=nullptr)

  Constructs an instance of this class.
- virtual ~kqueue_monitor ()

  Destroys an instance of this class.

Protected Member Functions

- void run ()

  Executes the monitor loop.

Additional Inherited Members

11.14.1 Detailed Description

Solaris/Illumos monitor.

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

11.14.2 Member Function Documentation

11.14.2.1 run()

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

Executes the monitor loop.

This call does not return until the monitor is stopped.

See also

stop()

Implements fsw::monitor.

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

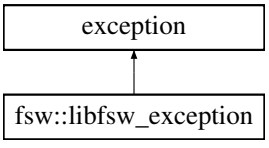
- libfswatch/c++/kqueue_monitor.hpp

11.15 fsw::libfsw_exception Class Reference

Base exception of the `libfswatch` library.

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

Inheritance diagram for fsw::libfsw_exception:



Public Member Functions

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

  Constructs an exception with the specified `cause` and error `code`.
- **libfsw_exception** (const libfsw_exception &other) noexcept
- libfsw_exception & **operator=** (const libfsw_exception &) noexcept
- virtual const char * what () const noexcept

  Gets the error message.
- virtual int error_code () const noexcept

  Gets the error code.
- virtual ~libfsw_exception () noexcept

  Destructs an instance of this class.
- operator int () const noexcept

  Gets the error code.

11.15.1 Detailed Description

Base exception of the `libfswatch` library.

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

11.15.2 Constructor & Destructor Documentation

11.15.2.1 libfsw_exception()

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

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

Parameters

cause	The error message.
code	The error code.

11.15.3 Member Function Documentation

11.15.3.1 error_code()

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

Gets the error code.

Returns

The error code.

11.15.3.2 what()

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

Gets the error message.

Returns

The error message.

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

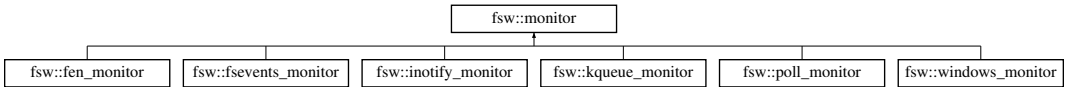
- libfswatch/c++/libfswatch_exception.hpp
- libfswatch/c++/libfswatch_exception.cpp

11.16 fsw::monitor Class Reference

Base class of all monitors.

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

Inheritance diagram for fsw::monitor:



Public Member Functions

- monitor (std::vector< std::string > paths, FSW_EVENT_CALLBACK *callback, void *context=nullptr)
  Constructs a monitor watching the specified `paths`.
- virtual ~monitor ()
  Destructs a monitor instance.
- monitor (const monitor &orig)=delete
  This class is not copy constructible.
- monitor & operator= (const monitor &that)=delete
  This class is not copy assignable.
- void set_property (const std::string &name, const std::string &value)
  Sets a custom property.
- void set_properties (const std::map< std::string, std::string > options)
  Sets the custom properties.
- std::string get_property (std::string name)

Gets the value of a property.

- void set_latency (double latency)

  Sets the latency.
- void set_fire_idle_event (bool fire_idle_event)

  Sets the fire idle event flag.
- void set_allow_overflow (bool overflow)

  Notify buffer overflows as change events.
- void set_recursive (bool recursive)

  Recursively scan subdirectories.
- void set_directory_only (bool directory_only)

  Watch directories only.
- void add_filter (const monitor_filter &filter)

  Add a path filter.
- void set_filters (const std::vector< monitor_filter > &filters)

  Set the path filters.
- void set_follow_symlinks (bool follow)

  Follow symlinks.
- void * get_context () const

  Get the pointer to the context data.
- void set_context (void *context)

  Set the context data.
- void start ()

  Start the monitor.
- void stop ()

  Stop the monitor.
- bool is_running ()

  Check whether the monitor is running.
- void add_event_type_filter (const fsw_event_type_filter &filter)

  Add an event type filter.
- void set_event_type_filters (const std::vector< fsw_event_type_filter > &filters)

  Set the event type filters.
- void set_watch_access (bool access)

  Monitor file access.

Protected Member Functions

- bool accept_event_type (fsw_event_flag event_type) const

  Check whether an event should be accepted.
- bool accept_path (const std::string &path) const

  Check whether a path should be accepted.
- bool accept_path (const char *path) const

  Check whether a path should be accepted.
- void notify_events (const std::vector< event > &events) const

  Notify change events.
- void notify_overflow (const std::string &path) const

  Notify an overflow event.
- std::vector< fsw_event_flag > filter_flags (const event &evt) const

  Filter event types.
- virtual void run ()=0

  Execute monitor loop.
- virtual void on_stop ()

  Execute an implementation-specific stop handler.

Protected Attributes

- std::vector< std::string > paths

  List of paths to watch.
- std::map< std::string, std::string > properties

  Map of custom properties.
- FSW_EVENT_CALLBACK * callback

  Callback to which change events should be notified.
- void * context = nullptr

  Pointer to context data that will be passed to the monitor::callback.
- double latency = 1.0

  Latency of the monitor.
- bool fire_idle_event = false

  If `true`, the monitor will notify an event when idle.
- bool allow_overflow = false

  If `true`, queue overflow events will be notified to the caller, otherwise the monitor will throw a libfsw_exception.
- bool recursive = false

  If `true`, directories will be scanned recursively.
- bool follow_symlinks = false

  If `true`, symbolic links are followed.
- bool directory_only = false

  Flag indicating whether only directories should be monitored.
- bool watch_access = false

  Flag indicating whether file access should be watched.
- bool running = false

  Flag indicating whether the monitor is in the running state.
- bool should_stop = false

  Flag indicating whether the monitor should preemptively stop.
- std::mutex run_mutex

  Mutex used to serialize access to the monitor state from multiple threads.
- std::mutex notify_mutex

  Mutex used to serialize access to the notify_events() method.

11.16.1 Detailed Description

Base class of all monitors.

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

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

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

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

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

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

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

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

    scan_paths();
    wait_for_events();

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

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

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

  terminate_api();
}
```

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

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

11.16.2 Constructor & Destructor Documentation

11.16.2.1 monitor()

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

Constructs a monitor watching the specified `paths`.

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

Parameters

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

11.16.2.2 ~monitor()

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

Destructs a monitor instance.

This destructor performs the following operations:

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

Warning

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

See also

stop()

11.16.3 Member Function Documentation

11.16.3.1 accept_event_type()

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

Check whether an event should be accepted.

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

Parameters

event_type	The event type to check.

Returns

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

11.16.3.2 accept_path() [1/2]

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

Check whether a path should be accepted.

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

Parameters

event_type	The path to check.

Returns

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

11.16.3.3 accept_path() [2/2]

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

Check whether a path should be accepted.

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

Parameters

event_type	The path to check.

Returns

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

11.16.3.4 add_event_type_filter()

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

Add an event type filter.

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

Parameters

filter	The event type filter to add.

11.16.3.5 add_filter()

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

Add a path filter.

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

Parameters

filter	The filter to add.

11.16.3.6 filter_flags()

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

Filter event types.

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

Parameters

evt	The event whose types must be filtered.

Returns

A vector containing the acceptable events.

11.16.3.7 get_context()

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

Get the pointer to the context data.

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

Returns

The pointer to the context data.

11.16.3.8 get_property()

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

Gets the value of a property.

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

Parameters

name	The name of the property.

Returns

The value of the property.

11.16.3.9 is_running()

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

Check whether the monitor is running.

State is checked thread-safely locking on monitor::run_mutex.

Returns

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

11.16.3.10 notify_events()

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

Notify change events.

This function notifies change events using the provided callback.

See also

monitor()

11.16.3.11 notify_overflow()

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

Notify an overflow event.

This function notifies an overflow event using the provided callback.

Warning

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

See also

monitor()

11.16.3.12 on_stop()

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

Execute an implementation-specific stop handler.

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

See also

stop()

Reimplemented in fsw::fsevents_monitor.

11.16.3.13 run()

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

Execute monitor loop.

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

This function should cooperatively check the monitor::should_stop field locking monitor::run_mutex and return if set to `true`.

See also

start()
stop()

Implemented in fsw::fsevents_monitor, fsw::fen_monitor, fsw::inotify_monitor, fsw::kqueue_monitor, fsw::windows_monitor, and fsw::poll_monitor.

11.16.3.14 set_allow_overflow()

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

Notify buffer overflows as change events.

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

Warning

The behaviour associated with this flag depends on the implementation.

Parameters

overflow	`true` if overflow should be notified, `false` otherwise.

11.16.3.15 set_context()

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

Set the context data.

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

Warning

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

Parameters

context	The pointer to the context data.

11.16.3.16 set_directory_only()

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

Watch directories only.

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

Warning

The behaviour associated with this flag depends on the implementation.

Parameters

directory_only	`true` if only directories should be watched, `flase` otherwise.

11.16.3.17 set_event_type_filters()

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

Set the event type filters.

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

Parameters

filters	The filters to set.

11.16.3.18 set_filters()

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

Set the path filters.

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

Parameters

filters	The filters to set.

11.16.3.19 set_fire_idle_event()

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

Sets the *fire idle event* flag.

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

Parameters

fire_idle_event	`true` if idle events should be fired, `false` otherwise.

11.16.3.20 set_follow_symlinks()

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

Follow symlinks.

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

Warning

The behaviour associated with this flag depends on the implementation.

Parameters

follow	`true` if symbolic links should be followed, `false` otherwise.

11.16.3.21 set_latency()

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

Sets the latency.

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

Warning

The behaviour associated with this flag depends on the implementation.

Parameters

latency	The latency value.

11.16.3.22 set_properties()

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

Sets the custom properties.

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

Parameters

options	The map containing the properties to set.

11.16.3.23 set_property()

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

Sets a custom property.

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

Parameters

name	The name of the property.
value	The value of the property.

11.16.3.24 set_recursive()

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

Recursively scan subdirectories.

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

Warning

The behaviour associated with this flag depends on the implementation.

Parameters

recursive	`true` if directories should be recursively, `false` otherwise.

11.16.3.25 set_watch_access()

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

Monitor file access.

Warning

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

11.16.3.26 start()

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

Start the monitor.

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

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

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

State changes are performed thread-safely locking on monitor::run_mutex.

See also

run()
stop()

11.16.3.27 stop()

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

Stop the monitor.

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

State changes are performed thread-safely locking on monitor::run_mutex.

See also

run()
start()

11.16.4 Member Data Documentation

11.16.4.1 callback

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

Callback to which change events should be notified.

See also

monitor::monitor()

11.16.4.2 fire_idle_event

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

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

An idle cycle is long as 110% of the monitor::latency value.

11.16.4.3 paths

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

List of paths to watch.

See also

monitor::monitor()

11.16.4.4 properties

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

Map of custom properties.

See also

monitor::set_property()
monitor::set_properties()

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

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

11.17 fsw::monitor_factory Class Reference

Object factory class for fsw::monitor instances.

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

Public Member Functions

- **monitor_factory** (const monitor_factory &orig)=delete
- monitor_factory & **operator=** (const monitor_factory &that)=delete

Static Public Member Functions

- static monitor * create_monitor (fsw_monitor_type type, std::vector< std::string > paths, FSW_EVENT_CALLBACK *callback, void *context=nullptr)

  Creates a monitor of the specified `type`.
- static monitor * create_monitor (const std::string &name, std::vector< std::string > paths, FSW_EVENT_CALLBACK *callback, void *context=nullptr)

  Creates a monitor whose type is the specified by `name`.
- static std::vector< std::string > get_types ()

  Get the available monitor types.
- static bool exists_type (const std::string &name)

  Checks whether a monitor of the type specified by `name` *exists.*
- static bool exists_type (const fsw_monitor_type &type)

  Checks whether a monitor of the type specified `type`.
- static void register_creator (const std::string &name, FSW_FN_MONITOR_CREATOR creator)

  Registers a `creator` *for the specified monitor type* `name`.
- static void register_creator_by_type (const fsw_monitor_type &type, FSW_FN_MONITOR_CREATOR creator)

  Registers a `creator` *for the specified monitor* `type`.

11.17.1 Detailed Description

Object factory class for fsw::monitor instances.

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

In order for monitor types to be visible to the factory they have to be *registered*. Currently, monitor implementations can be registered using the register_creator() and register_creator_by_type(), or using:

- The fsw::monitor_registrant helper class.
- The REGISTER_MONITOR macro.
- The REGISTER_MONITOR_IMPL macro.

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

11.17.2 Member Function Documentation

11.17.2.1 create_monitor() [1/2]

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

Creates a monitor of the specified `type`.

The other parameters are forwarded to the fsw::monitor() constructor.

Parameters

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

Returns

The newly created monitor.

Exceptions

libfsw_exception	if a monitor of the specified `type` cannot be found.

See also

fsw::monitor()

11.17.2.2 create_monitor() [2/2]

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

Creates a monitor whose type is the specified by `name`.

The other parameters are forwarded to the fsw::monitor() constructor.

Parameters

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

Returns

The newly created monitor.

Exceptions

libfsw_exception	if a monitor of the type specified by `name` cannot be found.

See also

fsw::monitor()

11.17.2.3 exists_type() [1/2]

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

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

Returns

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

Parameters

name	The name of the monitor type to look for.

Returns

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

11.17.2.4 exists_type() [2/2]

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

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

Parameters

type	The type of the monitor to look for.

Returns

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

11.17.2.5 get_types()

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

Get the available monitor types.

Returns

A vector with the available monitor types.

11.17.2.6 register_creator()

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

Registers a `creator` for the specified monitor type `name`.

Parameters

name	The name of the monitor type.
creator	The monitor creator function.

11.17.2.7 register_creator_by_type()

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

Registers a `creator` for the specified monitor `type`.

Parameters

type	The monitor type.
creator	The monitor creator function.

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

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

11.18 fsw::monitor_filter Struct Reference

Path filters used to accept or reject file change events.

`#include <filter.hpp>`

Static Public Member Functions

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

  Load filters from the specified file.

Public Attributes

- std::string text

  Regular expression used to match the paths.
- fsw_filter_type type

  Filter type.
- bool case_sensitive

  Flag indicating whether monitor_filter::text is a case sensitive regular expression.
- bool extended

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

11.18.1 Detailed Description

Path filters used to accept or reject file change events.

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

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

Further information about how filtering works in `libfswatch` can be found in Path Filtering.

11.18.2 Member Function Documentation

11.18.2.1 read_from_file()

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

Load filters from the specified file.

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

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

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

Parameters

path	The path of the file to read filters from.
err_handler	An optional error handler.

Returns

A vector containing the valid filters.

Exceptions

invalid_argument	If the specified path cannot be opened.

11.18.3 Member Data Documentation

11.18.3.1 extended

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

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

Further information about extended regular expressions can be found here:

http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html#tag↩_09_04

11.18.3.2 text

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

Regular expression used to match the paths.

Further information about regular expressions can be found here:

http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html

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

- libfswatch/c++/filter.hpp
- libfswatch/c++/filter.cpp

11.19 fsw::monitor_registrant< M > Class Template Reference

Helper class to register monitor factories.

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

Public Member Functions

- monitor_registrant (const std::string &name, const fsw_monitor_type &type)

  Constructs a monitor registrant for the specified `type`.

11.19.1 Detailed Description

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

Helper class to register monitor factories.

The constructor of this class perform the registration of the given (name, type) pair in the monitor_factory registry. This class is used by the REGISTER_MONITOR and REGISTER_MONITOR_IMPL macros.

See also

fsw::monitor_factory

11.19.2 Constructor & Destructor Documentation

11.19.2.1 monitor_registrant()

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

Constructs a monitor registrant for the specified `type`.

Parameters

name	The name of the type whose factory is being registered.
type	The type whose factory is being registered.

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

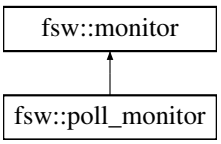
- libfswatch/c++/monitor.hpp

11.20 fsw::poll_monitor Class Reference

`stat()`-based monitor.

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

Inheritance diagram for fsw::poll_monitor:



Classes

- struct poll_monitor_data

Public Member Functions

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

Protected Member Functions

- void run ()
  Execute monitor loop.

Additional Inherited Members

11.20.1 Detailed Description

`stat()`-based monitor.

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

11.20.2 Member Function Documentation

11.20.2.1 run()

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

Execute monitor loop.

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

This function should cooperatively check the monitor::should_stop field locking monitor::run_mutex and return if set to `true`.

See also

start()
stop()

Implements fsw::monitor.

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

- libfswatch/c++/poll_monitor.hpp
- libfswatch/c++/poll_monitor.cpp

11.21 fsw::poll_monitor::poll_monitor_data Struct Reference

Public Attributes

- fsw_hash_map< string, poll_monitor::watched_file_info > **tracked_files**

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

- libfswatch/c++/poll_monitor.cpp

11.22 fsw::win_error_message Class Reference

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

`#include <win_error_message.hpp>`

Public Member Functions

- win_error_message (DWORD error_code)

  Constructs an error message using the specified `error_code`.
- win_error_message ()

  Constructs an error message using the last error code of the calling thread, retrieved with a call to `GetLast↩Error()`.
- 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 ths 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 `GetLast↩Error()`.

11.22.1 Detailed Description

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

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

11.22.2 Constructor & Destructor Documentation

11.22.2.1 win_error_message() [1/2]

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

Constructs an error message using the specified `error_code`.

Parameters

error_code	The error code.

11.22.2.2 win_error_message() [2/2]

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

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

See also

current()

11.22.3 Member Function Documentation

11.22.3.1 current()

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

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

See also

win_error_message()

11.22.3.2 get_error_code()

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

Gets the error code.

Returns

The error code.

11.22.3.3 get_message()

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

Gets the system-defined error message.

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

Returns

The error message.

11.22.3.4 operator std::wstring()

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

Gets ths system-defined error message.

See also

get_message()

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

- libfswatch/c++/windows/win_error_message.hpp

11.23 fsw::win_flag_type Struct Reference

Public Attributes

- DWORD **action**
- vector< fsw_event_flag > **types**

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

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

11.24 fsw::win_handle Class Reference

A RAII wrapper around Microsoft Windows `HANDLE`.

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

Public Member Functions

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

Static Public Member Functions

- static bool is_valid (const HANDLE &handle)
  Checks whether `handle` is valid.

11.24.1 Detailed Description

A RAII wrapper around Microsoft Windows `HANDLE`.

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

11.24.2 Constructor & Destructor Documentation

11.24.2.1 ~win_handle()

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

Destructs a handle.

If the handle is valid (is_valid()) it is closed invoking `CloseHandle()`.

See also

is_valid(const HANDLE &)

11.24.2.2 win_handle()

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

Move constructor.

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

Parameters

other	The handle to move.

11.24.3 Member Function Documentation

11.24.3.1 is_valid() [1/2]

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

Checks whether `handle` is valid.

A `handle` is valid is if its value is not `null` and if is not `INVALID_HANDLE_VALUE`.

Parameters

handle	The handle to check.

Returns

Returns `true` if `handle` is valid, `false` otherwise.

11.24.3.2 is_valid() [2/2]

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

Checks whether the handle is valid.

Returns

Returns `true` if the handle is valid, `false` otherwise.

See also

is_valid()

11.24.3.3 operator=() [1/2]

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

Move assignment operator.

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

Parameters

other	The handle to move.

11.24.3.4 operator=() [2/2]

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

Assigns a `handle` to the current instance.

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

Parameters

handle	The handle value to assign to the current instance.

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

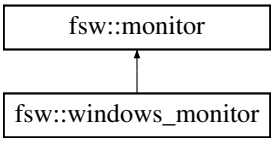
- libfswatch/c++/windows/win_handle.hpp

11.25 fsw::windows_monitor Class Reference

Windows monitor.

`#include <windows_monitor.hpp>`

Inheritance diagram for fsw::windows_monitor:



Public Member Functions

- windows_monitor (std::vector< std::string > paths, FSW_EVENT_CALLBACK *callback, void *context=nullptr)

  Constructs an instance of this class.
- virtual ~windows_monitor ()

  Destroys an instance of this class.

Protected Member Functions

- void run ()

  Executes the monitor loop.

Additional Inherited Members

11.25.1 Detailed Description

Windows monitor.

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

11.25.2 Member Function Documentation

11.25.2.1 run()

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

Executes the monitor loop.

This call does not return until the monitor is stopped.

See also

stop()

Implements fsw::monitor.

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

- libfswatch/c++/windows_monitor.hpp

Chapter 12

File Documentation

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

Header of the fsw::event class.

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

Classes

- class fsw::event

  Type representing a file change event.

Namespaces

- fsw

  Main namespace of `libfswatch`.

Functions

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

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

12.1.1 Detailed Description

Header of the fsw::event class.

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.2 libfswatch/c++/fen_monitor.hpp File Reference

Solaris/Illumos monitor.

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

Classes

- class fsw::fen_monitor

  Solaris/Illumos monitor.

Namespaces

- fsw

  Main namespace of `libfswatch`.

12.2.1 Detailed Description

Solaris/Illumos monitor.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

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

Header of the fsw::monitor_filter class.

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

Classes

- struct fsw::monitor_filter

  Path filters used to accept or reject file change events.

Namespaces

- fsw

  Main namespace of `libfswatch`.

Typedefs

- typedef struct fsw::monitor_filter fsw::monitor_filter

  Path filters used to accept or reject file change events.

12.3.1 Detailed Description

Header of the fsw::monitor_filter class.

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

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

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

OS X FSEvents monitor.

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

Classes

- class fsw::fsevents_monitor

  OS X FSEvents monitor.

Namespaces

- fsw

  Main namespace of `libfswatch`.

12.4.1 Detailed Description

OS X FSEvents monitor.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

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

Solaris/Illumos monitor.

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

Classes

- class fsw::inotify_monitor

  Solaris/Illumos monitor.

Namespaces

- fsw

  Main namespace of `libfswatch`.

12.5.1 Detailed Description

Solaris/Illumos monitor.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.6 libfswatch/c++/kqueue_monitor.hpp File Reference

`kqueue` monitor.

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

Classes

- class fsw::kqueue_monitor

  Solaris/Illumos monitor.

Namespaces

- fsw

  Main namespace of `libfswatch`.

12.6.1 Detailed Description

`kqueue` monitor.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

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

Base exception of the `libfswatch` library.

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

Classes

- class fsw::libfsw_exception

  Base exception of the `libfswatch` *library.*

Namespaces

- fsw

  Main namespace of `libfswatch`.

12.7.1 Detailed Description

Base exception of the `libfswatch` library.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

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

Header defining the associative container used by the library.

```
#include <map>
```

Namespaces

- fsw

  Main namespace of `libfswatch`.

Typedefs

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

  Default associative container type used by `libfswatch`.

12.8.1 Detailed Description

Header defining the associative container used by the library.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

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

Header defining the default set type used by the library.

```
#include <set>
```

Namespaces

- fsw

  Main namespace of `libfswatch`.

Typedefs

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

  Default set type used by `libfswatch`.

12.9.1 Detailed Description

Header defining the default set type used by the library.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

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

Header of the fsw::monitor class.

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

Classes

- class fsw::monitor

  Base class of all monitors.
- class fsw::monitor_factory

  Object factory class for fsw::monitor instances.
- class fsw::monitor_registrant< M >

  Helper class to register monitor factories.

Namespaces

- fsw

  Main namespace of `libfswatch`.

Macros

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

Typedefs

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

  Function definition of an event callback.
- typedef monitor *(* **fsw::FSW_FN_MONITOR_CREATOR**) (std::vector< std::string > paths, FSW_EVENT_CALLBACK *callback, void *context)

12.10.1 Detailed Description

Header of the fsw::monitor class.

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

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

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.10.2 Macro Definition Documentation

12.10.2.1 REGISTER_MONITOR

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

Value:

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

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

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

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

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

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

12.10.2.2 REGISTER_MONITOR_IMPL

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

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

A invocation of the REGISTER_MONITOR_IMPL macro must always be matched by an invocation of the REGIS↩TER_MONITOR macro in the class declaration.

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

[my_class.cpp]

REGISTER_MONITOR_IMPL(my_monitor, my_monitor_type);

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

Header defining utility functions to manipulate paths.

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

Namespaces

- fsw

  Main namespace of `libfswatch`.

Functions

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

  Gets a vector of direct directory children.
- bool fsw::read_link_path (const std::string &path, std::string &link_path)

  Resolves a path name.
- bool fsw::lstat_path (const std::string &path, struct stat &fd_stat)

  Wraps a `lstat(path, fd_stat)` call that invokes `perror()` if it fails.
- bool fsw::stat_path (const std::string &path, struct stat &fd_stat)

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

12.11.1 Detailed Description

Header defining utility functions to manipulate paths.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

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

`stat()` based monitor.

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

Classes

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

Namespaces

- fsw
  Main namespace of `libfswatch`.

12.12.1 Detailed Description

`stat()` based monitor.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

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

Header of the fsw::string_utils namespace.

```
#include <cstdarg>
#include <string>
```

Namespaces

- fsw

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

  This namespace contains string manipulation functions.

Functions

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

  Create a `std::string` using a `printf()` format and varargs.
- string fsw::string_utils::vstring_from_format (const char *format, va_list args)

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

12.13.1 Detailed Description

Header of the fsw::string_utils namespace.

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.14 libfswatch/c++/windows/win_directory_change_event.hpp File Reference

Header of the fsw::directory_change_event class.

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

Classes

- class fsw::directory_change_event

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

Namespaces

- fsw

  Main namespace of `libfswatch`.

12.14.1 Detailed Description

Header of the fsw::directory_change_event class.

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.15 libfswatch/c++/windows/win_error_message.hpp File Reference

Header of the fsw::win_error_message class.

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

Classes

- class fsw::win_error_message

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

Namespaces

- fsw

  Main namespace of `libfswatch`.

12.15.1 Detailed Description

Header of the fsw::win_error_message class.

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.16 libfswatch/c++/windows/win_handle.hpp File Reference

Header of the fsw::win_handle class.

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

Classes

- class fsw::win_handle

  A RAII wrapper around Microsoft Windows `HANDLE`.

Namespaces

- fsw

  Main namespace of `libfswatch`.

12.16.1 Detailed Description

Header of the fsw::win_handle class.

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

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

Header of the fsw::win_paths namespace.

```
#include <string>
```

Namespaces

- fsw

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

  Path conversion functions.

Functions

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

  Converts a POSIX path to Windows.
- std::string fsw::win_paths::win_w_to_posix (std::wstring path)

  Converts a Windows path to POSIX.

12.17.1 Detailed Description

Header of the fsw::win_paths namespace.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

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

Header of the fsw::win_strings namespace.

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

Namespaces

- fsw

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

  String conversion functions.

Functions

- string fsw::win_strings::wstring_to_string (wchar_t *s)

  Converts a wide character string into a string.
- std::string fsw::win_strings::wstring_to_string (const std::wstring &s)

  Converts a wide character string into a string.

12.18.1 Detailed Description

Header of the fsw::win_strings namespace.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

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

Windows monitor.

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

Classes

- class fsw::windows_monitor

  Windows monitor.

Namespaces

- fsw

  Main namespace of `libfswatch`.

12.19.1 Detailed Description

Windows monitor.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.20 libfswatch/c/cevent.h File Reference

Event type manipulation.

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

Classes

- struct fsw_cevent

Typedefs

- typedef struct fsw_cevent fsw_cevent
- typedef void(∗ FSW_CEVENT_CALLBACK) (fsw_cevent const ∗const events, const unsigned int event_num, void ∗data)

Enumerations

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

  Backend-agnostic change flags.

Functions

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

  Get event flag by name.
- char * fsw_get_event_flag_name (const enum fsw_event_flag flag)

  Get the name of an event flag.

Variables

- enum fsw_event_flag **FSW_ALL_EVENT_FLAGS** [15]

12.20.1 Detailed Description

Event type manipulation.

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

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.20.2 Typedef Documentation

12.20.2.1 fsw_cevent

```
typedef struct fsw_cevent fsw_cevent
```

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

- path is the path where the event was triggered.
- evt_time the time when the event was triggered.
- flags is an array of fsw_event_flag of size flags_num.
- flags_num is the size of the flags array.

12.20.2.2 FSW_CEVENT_CALLBACK

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

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

- events, a const pointer to an array of events of type const fsw_cevent.
- event_num, the size of the ∗events array.
- data, optional persisted data for a callback.

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

12.20.3 Enumeration Type Documentation

12.20.3.1 fsw_event_flag

```
enum fsw_event_flag
```

Backend-agnostic change flags.

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

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

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

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

Enumerator

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

12.20.4 Function Documentation

12.20.4.1 fsw_get_event_flag_by_name()

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

Get event flag by name.

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

Parameters

`in`	*name*	The name of the event flag to look for.
`out`	*flag*	The output variable where the event flag is returned.

Returns

FSW_OK if the functions succeeds, FSW_ERR_UNKNOWN_VALUE otherwise.

12.20.4.2 fsw_get_event_flag_name()

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

Get the name of an event flag.

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

Parameters

`in`	*flag*	The event flag to look for.

Returns

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

12.21 libfswatch/c/cfilter.h File Reference

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

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

Classes

- struct fsw_cmonitor_filter
- struct fsw_event_type_filter
  Event type filter.

Typedefs

- typedef struct fsw_cmonitor_filter **fsw_cmonitor_filter**
- typedef struct fsw_event_type_filter fsw_event_type_filter
  Event type filter.

Enumerations

- enum fsw_filter_type { **filter_include**, **filter_exclude** }
  Event filter type.

12.21.1 Detailed Description

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

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.22 libfswatch/c/cmonitor.h File Reference

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

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

Enumerations

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

  Available monitors.

12.22.1 Detailed Description

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

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.22.2 Enumeration Type Documentation

12.22.2.1 fsw_monitor_type

```
enum fsw_monitor_type
```

Available monitors.

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

Enumerator

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

12.23 libfswatch/c/error.h File Reference

Error values.

Macros

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

12.23.1 Detailed Description

Error values.

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

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.23.2 Macro Definition Documentation

12.23.2.1 FSW_ERR_CALLBACK_NOT_SET

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

The callback has not been set.

12.23.2.2 FSW_ERR_INVALID_CALLBACK

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

The callback is invalid.

12.23.2.3 FSW_ERR_INVALID_LATENCY

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

The latency is invalid.

12.23.2.4 FSW_ERR_INVALID_PATH

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

The path is invalid.

12.23.2.5 FSW_ERR_INVALID_PROPERTY

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

The property is invalid.

12.23.2.6 FSW_ERR_INVALID_REGEX

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

The regular expression is invalid.

12.23.2.7 FSW_ERR_MEMORY

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

An error occurred while invoking a memory management routine.

12.23.2.8 FSW_ERR_MISSING_CONTEXT

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

The callback context has not been set.

12.23.2.9 FSW_ERR_MONITOR_ALREADY_EXISTS

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

The session already contains a monitor.

12.23.2.10 FSW_ERR_MONITOR_ALREADY_RUNNING

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

A monitor is already running in the specified session.

12.23.2.11 FSW_ERR_PATHS_NOT_SET

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

The paths to watch have not been set.

12.23.2.12 FSW_ERR_SESSION_UNKNOWN

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

The session specified by the handle is unknown.

12.23.2.13 FSW_ERR_UNKNOWN_ERROR

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

An unknown error has occurred.

12.23.2.14 FSW_ERR_UNKNOWN_MONITOR_TYPE

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

The specified monitor type does not exist.

12.23.2.15 FSW_ERR_UNKNOWN_VALUE

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

The value is unknown.

12.23.2.16 FSW_OK

```
#define FSW_OK 0
```

The call was successful.

12.24 libfswatch/c/libfswatch.cpp File Reference

Main `libfswatch` source file.

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

Classes

- struct FSW_SESSION
- struct fsw_callback_context

Typedefs

- typedef struct FSW_SESSION **FSW_SESSION**
- typedef struct fsw_callback_context **fsw_callback_context**

Functions

- static FSW_SESSION * **get_session** (const FSW_HANDLE handle)
- static int **create_monitor** (FSW_HANDLE handle, const fsw_monitor_type type)
- static FSW_STATUS **fsw_set_last_error** (const int error)
- FSW_STATUS fsw_init_library ()
- void **libfsw_cpp_callback_proxy** (const std::vector< event > &events, void *context_ptr)
- FSW_HANDLE fsw_init_session (const fsw_monitor_type type)
- FSW_STATUS fsw_add_path (const FSW_HANDLE handle, const char *path)
- FSW_STATUS fsw_add_property (const FSW_HANDLE handle, const char *name, const char *value)
- FSW_STATUS fsw_set_callback (const FSW_HANDLE handle, const FSW_CEVENT_CALLBACK callback, void *data)
- FSW_STATUS fsw_set_allow_overflow (const FSW_HANDLE handle, const bool allow_overflow)
- FSW_STATUS fsw_set_latency (const FSW_HANDLE handle, const double latency)
- FSW_STATUS fsw_set_recursive (const FSW_HANDLE handle, const bool recursive)

- FSW_STATUS fsw_set_directory_only (const FSW_HANDLE handle, const bool directory_only)
- FSW_STATUS fsw_set_follow_symlinks (const FSW_HANDLE handle, const bool follow_symlinks)
- FSW_STATUS fsw_add_event_type_filter (const FSW_HANDLE handle, const fsw_event_type_filter event↩ _type)
- FSW_STATUS fsw_add_filter (const FSW_HANDLE handle, const fsw_cmonitor_filter filter)
- 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

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

12.24.1 Detailed Description

Main `libfswatch` source file.

Copyright

Copyright (c) 2014-2016 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.10.0

12.24.2 Function Documentation

12.24.2.1 fsw_add_event_type_filter()

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

Adds an event type filter to the current session.

See cfilter.h for the definition of fsw_event_type_filter.

12.24.2.2 fsw_add_filter()

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

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

See cfilter.h for the definition of fsw_cmonitor_filter.

12.24.2.3 fsw_add_path()

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

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

12.24.2.4 fsw_add_property()

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

Adds the specified monitor property.

12.24.2.5 fsw_destroy_session()

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

Destroys an existing session and invalidates its handle.

12.24.2.6 fsw_init_library()

```
FSW_STATUS fsw_init_library ( )
```

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

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

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

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

A basic session needs at least:

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

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

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

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

fsw_start_monitor(handle);
```

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

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

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

12.24.2.7 fsw_init_session()

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

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

See also

cmonitor.h for a list of all the available monitors.

12.24.2.8 fsw_is_running()

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

Checks if a monitor exists and is running.

12.24.2.9 fsw_is_verbose()

```
bool fsw_is_verbose ( )
```

Check whether the verbose mode is active.

12.24.2.10 fsw_last_error()

```
FSW_STATUS fsw_last_error ( )
```

Gets the last error code.

12.24.2.11 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.

12.24.2.12 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.

12.24.2.13 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.

12.24.2.14 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.

12.24.2.15 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.

12.24.2.16 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.

12.24.2.17 fsw_set_verbose()

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

Set the verbose mode.

12.24.2.18 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.

12.24.2.19 fsw_stop_monitor()

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

Stops a running monitor.

12.25 libfswatch/c/libfswatch.h File Reference

Header of the `libfswatch` library.

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

Functions

- FSW_STATUS fsw_init_library ()
- FSW_HANDLE fsw_init_session (const enum fsw_monitor_type type)
- FSW_STATUS fsw_add_path (const FSW_HANDLE handle, const char *path)
- FSW_STATUS fsw_add_property (const FSW_HANDLE handle, const char *name, const char *value)
- FSW_STATUS fsw_set_allow_overflow (const FSW_HANDLE handle, const bool allow_overflow)
- FSW_STATUS fsw_set_callback (const FSW_HANDLE handle, const FSW_CEVENT_CALLBACK callback, void *data)
- FSW_STATUS fsw_set_latency (const FSW_HANDLE handle, const double latency)
- FSW_STATUS fsw_set_recursive (const FSW_HANDLE handle, const bool recursive)
- FSW_STATUS fsw_set_directory_only (const FSW_HANDLE handle, const bool directory_only)
- FSW_STATUS fsw_set_follow_symlinks (const FSW_HANDLE handle, const bool follow_symlinks)
- FSW_STATUS fsw_add_event_type_filter (const FSW_HANDLE handle, const fsw_event_type_filter event_type)
- FSW_STATUS fsw_add_filter (const FSW_HANDLE handle, const fsw_cmonitor_filter filter)
- FSW_STATUS fsw_start_monitor (const FSW_HANDLE handle)
- FSW_STATUS fsw_stop_monitor (const FSW_HANDLE handle)
- 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)

12.25.1 Detailed Description

Header of the `libfswatch` library.

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

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.25.2 Function Documentation

12.25.2.1 fsw_add_event_type_filter()

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

Adds an event type filter to the current session.

See cfilter.h for the definition of fsw_event_type_filter.

12.25.2.2 fsw_add_filter()

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

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

See cfilter.h for the definition of fsw_cmonitor_filter.

12.25.2.3 fsw_add_path()

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

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

12.25.2.4 fsw_add_property()

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

Adds the specified monitor property.

12.25.2.5 fsw_destroy_session()

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

Destroys an existing session and invalidates its handle.

12.25.2.6 fsw_init_library()

```
FSW_STATUS fsw_init_library ( )
```

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

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

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

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

A basic session needs at least:

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

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

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

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

fsw_start_monitor(handle);
```

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

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

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

12.25.2.7 fsw_init_session()

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

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

See also

cmonitor.h for a list of all the available monitors.

12.25.2.8 fsw_is_running()

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

Checks if a monitor exists and is running.

12.25.2.9 fsw_is_verbose()

```
bool fsw_is_verbose ( )
```

Check whether the verbose mode is active.

12.25.2.10 fsw_last_error()

```
FSW_STATUS fsw_last_error ( )
```

Gets the last error code.

12.25.2.11 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.

12.25.2.12 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.

12.25.2.13 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.

12.25.2.14 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.

12.25.2.15 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.

12.25.2.16 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.

12.25.2.17 fsw_set_verbose()

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

Set the verbose mode.

12.25.2.18 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.

12.25.2.19 fsw_stop_monitor()

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

Stops a running monitor.

12.26 libfswatch/c/libfswatch_log.h File Reference

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

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

Macros

- #define FSW_LOG(msg) fsw_logf("%s: ", __func__); fsw_log(msg)

  Log the specified message to the standard output prepended by the source line number.
- #define FSW_ELOG(msg) fsw_flogf(stderr, "%s: ", __func__); fsw_flog(stderr, msg)

  Log the specified message to the standard error prepended by the source line number.
- #define FSW_LOGF(msg, ...) fsw_logf("%s: ", __func__); fsw_logf(msg, __VA_ARGS__)

  Log the specified `printf()`-like message to the standard output prepended by the source line number.
- #define FSW_ELOGF(msg, ...) fsw_flogf(stderr, "%s: ", __func__); fsw_flogf(stderr, msg, __VA_ARGS__)

  Log the specified `printf()`-like message to the standard error prepended by the source line number.
- #define FSW_FLOGF(f, msg, ...) fsw_flogf(f, "%s: ", __func__); fsw_flogf(f, msg, __VA_ARGS__)

  Log the specified `printf()`-like message to the specified file descriptor prepended by the source line number.

Functions

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

12.26.1 Detailed Description

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

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

12.26.2 Function Documentation

12.26.2.1 fsw_flog()

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

Prints the specified message to the specified file.

12.26.2.2 fsw_flogf()

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

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

12.26.2.3 fsw_log()

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

Prints the specified message to standard output.

12.26.2.4 fsw_log_perror()

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

Prints the specified message using perror.

12.26.2.5 fsw_logf()

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

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

12.26.2.6 fsw_logf_perror()

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

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

12.27 libfswatch/c/libfswatch_types.h File Reference

Header of the `libfswatch` library containing common types.

Macros

- #define **FSW_INVALID_HANDLE** -1
- #define **FSW_THREAD_LOCAL**

Typedefs

- typedef struct FSW_SESSION * FSW_HANDLE

  Handle to a monitoring session.
- typedef int FSW_STATUS

  Status of a library call.

12.27.1 Detailed Description

Header of the `libfswatch` library containing common types.

This header file defines the types used by the `libfswatch` library.

Copyright

Copyright (c) 2014-2015 Enrico M. Crisostomo

License:

GNU General Public License v. 3.0

Author

Enrico M. Crisostomo

Version

1.8.0

Index