SFML

* [Main Page](http://docs.google.com/index.htm)
* [Modules](http://docs.google.com/modules.htm)
* [Classes](http://docs.google.com/annotated.htm)
* [Files](http://docs.google.com/files.htm)
* [Class List](http://docs.google.com/annotated.htm)
* [Class Index](http://docs.google.com/classes.htm)
* [Class Hierarchy](http://docs.google.com/hierarchy.htm)
* [Class Members](http://docs.google.com/functions.htm)
* **sf**
* [Thread](http://docs.google.com/classsf_1_1Thread.htm)

[Public Member Functions](#_gjdgxs) | [List of all members](http://docs.google.com/classsf_1_1Thread-members.htm)

sf::Thread Class Reference

[System module](http://docs.google.com/group__system.htm)

Utility class to manipulate threads. [More...](http://docs.google.com/classsf_1_1Thread.htm#details)

#include <[Thread.hpp](http://docs.google.com/Thread_8hpp_source.htm)>

Inheritance diagram for sf::Thread:



| Public Member Functions | |
| --- | --- |
| template<typename F > | |
|  | [Thread](http://docs.google.com/classsf_1_1Thread.htm#a4cc65399bbb111cf8132537783b8e96c) (F function) |
|  | Construct the thread from a functor with no argument. |
|  | |
| template<typename F , typename A > | |
|  | [Thread](http://docs.google.com/classsf_1_1Thread.htm#a719b2cc067d92d52c35064a49d850a53) (F function, A argument) |
|  | Construct the thread from a functor with an argument. |
|  | |
| template<typename C > | |
|  | [Thread](http://docs.google.com/classsf_1_1Thread.htm#aa9f473c8cbb078900c62b1fd14a83a34) (void(C::\*function)(), C \*object) |
|  | Construct the thread from a member function and an object. |
|  | |
|  | [~Thread](http://docs.google.com/classsf_1_1Thread.htm#af77942fc1730af7c31bc4c3a913a9c1d) () |
|  | Destructor. |
|  | |
| void | [launch](http://docs.google.com/classsf_1_1Thread.htm#a74f75a9e86e1eb47479496314048b5f6) () |
|  | Run the thread. |
|  | |
| void | [wait](http://docs.google.com/classsf_1_1Thread.htm#a724b1f94c2d54f84280f2f78bde95fa0) () |
|  | Wait until the thread finishes. |
|  | |
| void | [terminate](http://docs.google.com/classsf_1_1Thread.htm#ad6b205d4f1ce38b8d44bba0f5501477c) () |
|  | Terminate the thread. |
|  | |
| template<typename F > | |
|  | **Thread** (F functor) |
|  | |
| template<typename F , typename A > | |
|  | **Thread** (F function, A argument) |
|  | |
| template<typename C > | |
|  | **Thread** (void(C::\*function)(), C \*object) |
|  | |

## Detailed Description

Utility class to manipulate threads.

Threads provide a way to run multiple parts of the code in parallel.

When you launch a new thread, the execution is split and both the new thread and the caller run in parallel.

To use a [sf::Thread](http://docs.google.com/classsf_1_1Thread.htm), you construct it directly with the function to execute as the entry point of the thread. [sf::Thread](http://docs.google.com/classsf_1_1Thread.htm) has multiple template constructors, which means that you can use several types of entry points:

* non-member functions with no argument
* non-member functions with one argument of any type
* functors with no argument (this one is particularly useful for compatibility with boost/std::bind)
* functors with one argument of any type
* member functions from any class with no argument

The function argument, if any, is copied in the [sf::Thread](http://docs.google.com/classsf_1_1Thread.htm) instance, as well as the functor (if the corresponding constructor is used). Class instances, however, are passed by pointer so you must make sure that the object won't be destroyed while the thread is still using it.

The thread ends when its function is terminated. If the owner [sf::Thread](http://docs.google.com/classsf_1_1Thread.htm) instance is destroyed before the thread is finished, the destructor will wait (see [wait()](http://docs.google.com/classsf_1_1Thread.htm#a724b1f94c2d54f84280f2f78bde95fa0))

Usage examples:

// example 1: non member function with one argument

void threadFunc(int argument)

{

...

}

[sf::Thread](http://docs.google.com/classsf_1_1Thread.htm) thread(&threadFunc, 5);

thread.launch(); // start the thread (internally calls threadFunc(5))

// example 2: member function

class Task

{

public :

void run()

{

...

}

};

Task task;

[sf::Thread](http://docs.google.com/classsf_1_1Thread.htm) thread(&Task::run, &task);

thread.launch(); // start the thread (internally calls task.run())

// example 3: functor

struct Task

{

void operator()()

{

...

}

};

[sf::Thread](http://docs.google.com/classsf_1_1Thread.htm) thread(Task());

thread.launch(); // start the thread (internally calls operator() on the Task instance)

Creating parallel threads of execution can be dangerous: all threads inside the same process share the same memory space, which means that you may end up accessing the same variable from multiple threads at the same time. To prevent this kind of situations, you can use mutexes (see [sf::Mutex](http://docs.google.com/classsf_1_1Mutex.htm)).

See Also[sf::Mutex](http://docs.google.com/classsf_1_1Mutex.htm)

Definition at line [48](http://docs.google.com/Thread_8hpp_source.htm#l00048) of file [Thread.hpp](http://docs.google.com/Thread_8hpp_source.htm).

## Constructor & Destructor Documentation

template<typename F >

| sf::Thread::Thread | ( | F | *function* | ) |  |
| --- | --- | --- | --- | --- | --- |

Construct the thread from a functor with no argument.

This constructor works for function objects, as well as free function.

Use this constructor for this kind of function:

void function();

// --- or ----

struct Functor

{

void operator()();

};

Note: this does *not* run the thread, use Launch().

Parameters

| function | Functor or free function to use as the entry point of the thread |
| --- | --- |

template<typename F , typename A >

| sf::Thread::Thread | ( | F | *function*, |
| --- | --- | --- | --- |
|  |  | A | *argument* |
|  | ) |  |  |

Construct the thread from a functor with an argument.

This constructor works for function objects, as well as free function. It is a template, which means that the argument can have any type (int, std::string, void\*, Toto, ...).

Use this constructor for this kind of function:

void function(int arg);

// --- or ----

struct Functor

{

void operator()(std::string arg);

};

Note: this does *not* run the thread, use Launch().

Parameters

| function | Functor or free function to use as the entry point of the thread |
| --- | --- |
| argument | argument to forward to the function |

template<typename C >

| sf::Thread::Thread | ( | void(C::\*)() | *function*, |
| --- | --- | --- | --- |
|  |  | C \* | *object* |
|  | ) |  |  |

Construct the thread from a member function and an object.

This constructor is template, which means that you can use it with any class. Use this constructor for this kind of function:

class MyClass

{

public :

void function();

};

Note: this does *not* run the thread, use Launch().

Parameters

| function | Entry point of the thread |
| --- | --- |
| object | Pointer to the object to use |

| sf::Thread::~Thread | ( |  | ) |  |
| --- | --- | --- | --- | --- |

Destructor.

This destructor calls Wait(), so that the internal thread cannot survive after its [sf::Thread](http://docs.google.com/classsf_1_1Thread.htm) instance is destroyed.

## Member Function Documentation

| void sf::Thread::launch | ( |  | ) |  |
| --- | --- | --- | --- | --- |

Run the thread.

This function starts the entry point passed to the thread's constructor, and returns immediately. After this function returns, the thread's function is running in parallel to the calling code.

| void sf::Thread::terminate | ( |  | ) |  |
| --- | --- | --- | --- | --- |

Terminate the thread.

This function immediately stops the thread, without waiting for its function to finish. Terminating a thread with this function is not safe, and can lead to local variables not being destroyed on some operating systems. You should rather try to make the thread function terminate by itself.

| void sf::Thread::wait | ( |  | ) |  |
| --- | --- | --- | --- | --- |

Wait until the thread finishes.

This function will block the execution until the thread's function ends. Warning: if the thread function never ends, the calling thread will block forever. If this function is called from its owner thread, it returns without doing anything.

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

* [Thread.hpp](http://docs.google.com/Thread_8hpp_source.htm)
* [Thread.inl](http://docs.google.com/Thread_8inl_source.htm)

Copyright � Laurent Gomila  ::  Documentation generated by [doxygen](http://www.doxygen.org/)  ::