

primesieve

5.6.0

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
1.1	About . . . . .	1
1.2	C++ API . . . . .	1
1.3	C API . . . . .	1
<b>2</b>	<b>Namespace Index</b>	<b>3</b>
2.1	Namespace List . . . . .	3
<b>3</b>	<b>Hierarchical Index</b>	<b>5</b>
3.1	Class Hierarchy . . . . .	5
<b>4</b>	<b>Class Index</b>	<b>7</b>
4.1	Class List . . . . .	7
<b>5</b>	<b>File Index</b>	<b>9</b>
5.1	File List . . . . .	9
<b>6</b>	<b>Namespace Documentation</b>	<b>11</b>
6.1	primesieve Namespace Reference . . . . .	11
6.1.1	Detailed Description . . . . .	13
6.1.2	Enumeration Type Documentation . . . . .	13
6.1.2.1	anonymous enum . . . . .	13
6.1.3	Function Documentation . . . . .	13
6.1.3.1	callback_primes(uint64_t start, uint64_t stop, void(*callback)(uint64_t prime)) . .	13
6.1.3.2	callback_primes(uint64_t start, uint64_t stop, primesieve::Callback< uint64_t > *callback) . . . . .	13

6.1.3.3	<code>count_primes(uint64_t start, uint64_t stop)</code>	14
6.1.3.4	<code>count_quadruplets(uint64_t start, uint64_t stop)</code>	14
6.1.3.5	<code>count_quintuplets(uint64_t start, uint64_t stop)</code>	14
6.1.3.6	<code>count_sextuplets(uint64_t start, uint64_t stop)</code>	14
6.1.3.7	<code>count_triplets(uint64_t start, uint64_t stop)</code>	14
6.1.3.8	<code>count_twins(uint64_t start, uint64_t stop)</code>	15
6.1.3.9	<code>generate_n_primes(uint64_t n, uint64_t start, std::vector&lt; T &gt; *primes)</code>	15
6.1.3.10	<code>generate_primes(uint64_t stop, std::vector&lt; T &gt; *primes)</code>	15
6.1.3.11	<code>generate_primes(uint64_t start, uint64_t stop, std::vector&lt; T &gt; *primes)</code>	15
6.1.3.12	<code>get_max_stop()</code>	15
6.1.3.13	<code>get_num_threads()</code>	15
6.1.3.14	<code>nth_prime(int64_t n, uint64_t start=0)</code>	15
6.1.3.15	<code>parallel_count_primes(uint64_t start, uint64_t stop)</code>	16
6.1.3.16	<code>parallel_count_quadruplets(uint64_t start, uint64_t stop)</code>	16
6.1.3.17	<code>parallel_count_quintuplets(uint64_t start, uint64_t stop)</code>	16
6.1.3.18	<code>parallel_count_sextuplets(uint64_t start, uint64_t stop)</code>	17
6.1.3.19	<code>parallel_count_triplets(uint64_t start, uint64_t stop)</code>	17
6.1.3.20	<code>parallel_count_twins(uint64_t start, uint64_t stop)</code>	17
6.1.3.21	<code>parallel_nth_prime(int64_t n, uint64_t start=0)</code>	17
6.1.3.22	<code>primesieve_test()</code>	18
6.1.3.23	<code>print_primes(uint64_t start, uint64_t stop)</code>	18
6.1.3.24	<code>print_quadruplets(uint64_t start, uint64_t stop)</code>	18
6.1.3.25	<code>print_quintuplets(uint64_t start, uint64_t stop)</code>	18
6.1.3.26	<code>print_sextuplets(uint64_t start, uint64_t stop)</code>	18
6.1.3.27	<code>print_triplets(uint64_t start, uint64_t stop)</code>	19
6.1.3.28	<code>print_twins(uint64_t start, uint64_t stop)</code>	19
6.1.3.29	<code>set_num_threads(int num_threads)</code>	19
6.1.3.30	<code>set_sieve_size(int sieve_size)</code>	19

<b>7</b>	<b>Class Documentation</b>	<b>21</b>
7.1	<a href="#">primesieve::Callback&lt; T &gt; Class Template Reference</a>	21
7.1.1	<a href="#">Detailed Description</a>	21
7.2	<a href="#">primesieve::iterator Class Reference</a>	21
7.2.1	<a href="#">Detailed Description</a>	22
7.2.2	<a href="#">Constructor &amp; Destructor Documentation</a>	22
7.2.2.1	<a href="#">iterator(uint64_t start=0, uint64_t stop_hint=get_max_stop())</a>	22
7.2.3	<a href="#">Member Function Documentation</a>	22
7.2.3.1	<a href="#">next_prime()</a>	22
7.2.3.2	<a href="#">previous_prime()</a>	23
7.2.3.3	<a href="#">skipto(uint64_t start, uint64_t stop_hint=get_max_stop())</a>	23
7.3	<a href="#">primesieve::primesieve_error Class Reference</a>	23
7.3.1	<a href="#">Detailed Description</a>	24
7.4	<a href="#">primesieve_iterator Struct Reference</a>	25
7.4.1	<a href="#">Detailed Description</a>	25
<b>8</b>	<b>File Documentation</b>	<b>27</b>
8.1	<a href="#">Callback.hpp File Reference</a>	27
8.1.1	<a href="#">Detailed Description</a>	28
8.2	<a href="#">iterator.hpp File Reference</a>	28
8.2.1	<a href="#">Detailed Description</a>	29
8.3	<a href="#">primesieve.h File Reference</a>	29
8.3.1	<a href="#">Detailed Description</a>	31
8.3.2	<a href="#">Enumeration Type Documentation</a>	32
8.3.2.1	<a href="#">anonymous enum</a>	32
8.3.3	<a href="#">Function Documentation</a>	32
8.3.3.1	<a href="#">primesieve_callback_primes(uint64_t start, uint64_t stop, void(*callback)(uint64_t prime))</a>	32
8.3.3.2	<a href="#">primesieve_count_primes(uint64_t start, uint64_t stop)</a>	32
8.3.3.3	<a href="#">primesieve_count_quadruplets(uint64_t start, uint64_t stop)</a>	33
8.3.3.4	<a href="#">primesieve_count_quintuplets(uint64_t start, uint64_t stop)</a>	33

8.3.3.5	<code>primesieve_count_sextuplets(uint64_t start, uint64_t stop)</code>	33
8.3.3.6	<code>primesieve_count_triplets(uint64_t start, uint64_t stop)</code>	33
8.3.3.7	<code>primesieve_count_twins(uint64_t start, uint64_t stop)</code>	33
8.3.3.8	<code>primesieve_generate_n_primes(uint64_t n, uint64_t start, int type)</code>	33
8.3.3.9	<code>primesieve_generate_primes(uint64_t start, uint64_t stop, size_t *size, int type)</code>	34
8.3.3.10	<code>primesieve_get_max_stop()</code>	34
8.3.3.11	<code>primesieve_get_num_threads()</code>	34
8.3.3.12	<code>primesieve_get_sieve_size()</code>	34
8.3.3.13	<code>primesieve_nth_prime(int64_t n, uint64_t start)</code>	34
8.3.3.14	<code>primesieve_parallel_count_primes(uint64_t start, uint64_t stop)</code>	35
8.3.3.15	<code>primesieve_parallel_count_quadruplets(uint64_t start, uint64_t stop)</code>	35
8.3.3.16	<code>primesieve_parallel_count_quintuplets(uint64_t start, uint64_t stop)</code>	35
8.3.3.17	<code>primesieve_parallel_count_sextuplets(uint64_t start, uint64_t stop)</code>	36
8.3.3.18	<code>primesieve_parallel_count_triplets(uint64_t start, uint64_t stop)</code>	36
8.3.3.19	<code>primesieve_parallel_count_twins(uint64_t start, uint64_t stop)</code>	36
8.3.3.20	<code>primesieve_parallel_nth_prime(int64_t n, uint64_t start)</code>	36
8.3.3.21	<code>primesieve_print_primes(uint64_t start, uint64_t stop)</code>	37
8.3.3.22	<code>primesieve_print_quadruplets(uint64_t start, uint64_t stop)</code>	37
8.3.3.23	<code>primesieve_print_quintuplets(uint64_t start, uint64_t stop)</code>	37
8.3.3.24	<code>primesieve_print_sextuplets(uint64_t start, uint64_t stop)</code>	37
8.3.3.25	<code>primesieve_print_triplets(uint64_t start, uint64_t stop)</code>	37
8.3.3.26	<code>primesieve_print_twins(uint64_t start, uint64_t stop)</code>	37
8.3.3.27	<code>primesieve_set_num_threads(int num_threads)</code>	37
8.3.3.28	<code>primesieve_set_sieve_size(int sieve_size)</code>	38
8.3.3.29	<code>primesieve_test()</code>	38
8.3.3.30	<code>primesieve_version()</code>	38
8.4	<code>primesieve.hpp</code> File Reference	38
8.4.1	Detailed Description	41
8.5	<code>primesieve_error.hpp</code> File Reference	41
8.5.1	Detailed Description	42
8.6	<code>primesieve_iterator.h</code> File Reference	42
8.6.1	Detailed Description	43
8.6.2	Function Documentation	43
8.6.2.1	<code>primesieve_free_iterator(primesieve_iterator *pi)</code>	43
8.6.2.2	<code>primesieve_init(primesieve_iterator *pi)</code>	43
8.6.2.3	<code>primesieve_next_prime(primesieve_iterator *pi)</code>	44
8.6.2.4	<code>primesieve_previous_prime(primesieve_iterator *pi)</code>	44
8.6.2.5	<code>primesieve_skipto(primesieve_iterator *pi, uint64_t start, uint64_t stop_hint)</code>	44

<b>9</b>	<b>Example Documentation</b>	<b>45</b>
9.1	callback_primes.cpp . . . . .	45
9.2	count_primes.c . . . . .	45
9.3	count_primes.cpp . . . . .	46
9.4	nth_prime.c . . . . .	46
9.5	nth_prime.cpp . . . . .	46
9.6	previous_prime.c . . . . .	47
9.7	previous_prime.cpp . . . . .	47
9.8	primesieve_iterator.c . . . . .	47
9.9	primesieve_iterator.cpp . . . . .	48
9.10	store_primes_in_array.c . . . . .	48
9.11	store_primes_in_vector.cpp . . . . .	48
	<b>Index</b>	<b>51</b>



# Chapter 1

## Main Page

### 1.1 About

primesieve is a C/C++ library for fast prime number generation. It generates the primes below  $10^9$  in just 0.2 seconds on a single core of an Intel Core i7-6700 3.4GHz CPU. primesieve can generate primes and prime k-tuplets up to  $2^{64}$ . primesieve's memory requirement is about  $\pi(\sqrt{n}) * 8$  bytes per thread, its run-time complexity is  $O(n \log \log n)$  operations. For more information please visit <http://primesieve.org>.

The recommended way to get started is to first have a look at a few C/C++ example programs. The most common use cases are storing primes in a vector (or array) and iterating over primes using `next_prime()` or `previous_prime()`.

You can install libprimesieve either using your distribution's package manager (if it is available) or you can build and install it yourself, this is explained at <http://primesieve.org/build.html>.

### 1.2 C++ API

- [primesieve.hpp](#) - primesieve C++ header.
- [store\\_primes\\_in\\_vector.cpp](#) - Example that shows how to store primes in a `std::vector`.
- [primesieve\\_iterator.cpp](#) - Example that shows how to iterate over primes using `primesieve::iterator`.
- [count\\_primes.cpp](#) - Example that shows how to count primes.

### 1.3 C API

- [primesieve.h](#) - primesieve C header.
- [store\\_primes\\_in\\_array.c](#) - Example that shows how to store primes in an array.
- [primesieve\\_iterator.c](#) - Example that shows how to iterate over primes using `primesieve_iterator`.
- [count\\_primes.c](#) - Example that shows how to count primes.



## Chapter 2

# Namespace Index

### 2.1 Namespace List

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

[primesieve](#)

All of primesieve's C++ functions and classes are declared inside this namespace . . . . . [11](#)



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

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

primesieve::Callback< T > . . . . .	21
primesieve::iterator . . . . .	21
primesieve_iterator . . . . .	25
runtime_error	
primesieve::primesieve_error . . . . .	23



## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">primesieve::Callback&lt; T &gt;</a>	
<a href="#">Callback</a> interface class . . . . .	21
<a href="#">primesieve::iterator</a>	
Primesieve::iterator allows to easily iterate over primes both forwards and backwards . . . . .	21
<a href="#">primesieve::primesieve_error</a>	
Primesieve throws a <a href="#">primesieve_error</a> exception if an error occurs that cannot be handled e.g .	23
<a href="#">primesieve_iterator</a>	
C prime iterator, please refer to <a href="#">primesieve_iterator.h</a> for more information . . . . .	25



## Chapter 5

# File Index

### 5.1 File List

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

<a href="#">Callback.hpp</a>		
Callback interface classes	.....	27
<a href="#">iterator.hpp</a>		
The iterator class allows to easily iterate (forward and backward) over prime numbers	.....	28
<a href="#">primesieve.h</a>		
Primesieve C API	.....	29
<a href="#">primesieve.hpp</a>		
Primesieve C++ API	.....	38
<a href="#">primesieve_error.hpp</a>		
The primesieve_error class is used for all exceptions within primesieve	.....	41
<a href="#">primesieve_iterator.h</a>		
Primesieve_iterator allows to easily iterate over primes both forwards and backwards	.....	42



## Chapter 6

# Namespace Documentation

### 6.1 primesieve Namespace Reference

All of primesieve's C++ functions and classes are declared inside this namespace.

#### Classes

- class [Callback](#)  
*callback interface class.*
- class [iterator](#)  
*[primesieve::iterator](#) allows to easily iterate over primes both forwards and backwards.*
- class [primesieve\\_error](#)  
*primesieve throws a [primesieve\\_error](#) exception if an error occurs that cannot be handled e.g.*

#### Enumerations

- enum { [MAX\\_THREADS](#) = -1 }

#### Functions

- `template<typename T >`  
`void generate\_primes (uint64_t stop, std::vector< T > *primes)`  
*Store the primes  $\leq$  stop in the primes vector.*
- `template<typename T >`  
`void generate\_primes (uint64_t start, uint64_t stop, std::vector< T > *primes)`  
*Store the primes within the interval [start, stop] in the primes vector.*
- `template<typename T >`  
`void generate\_n\_primes (uint64_t n, std::vector< T > *primes)`  
*Store the first n primes in the primes vector.*
- `template<typename T >`  
`void generate\_n\_primes (uint64_t n, uint64_t start, std::vector< T > *primes)`  
*Store the first n primes  $\geq$  start in the primes vector.*
- `uint64_t nth\_prime (int64_t n, uint64_t start=0)`  
*Find the nth prime.*

- uint64\_t [parallel\\_nth\\_prime](#) (int64\_t n, uint64\_t start=0)  
*Find the nth prime in parallel.*
- uint64\_t [count\\_primes](#) (uint64\_t start, uint64\_t stop)  
*Count the primes within the interval [start, stop].*
- uint64\_t [count\\_twins](#) (uint64\_t start, uint64\_t stop)  
*Count the twin primes within the interval [start, stop].*
- uint64\_t [count\\_triplets](#) (uint64\_t start, uint64\_t stop)  
*Count the prime triplets within the interval [start, stop].*
- uint64\_t [count\\_quadruplets](#) (uint64\_t start, uint64\_t stop)  
*Count the prime quadruplets within the interval [start, stop].*
- uint64\_t [count\\_quintuplets](#) (uint64\_t start, uint64\_t stop)  
*Count the prime quintuplets within the interval [start, stop].*
- uint64\_t [count\\_sextuplets](#) (uint64\_t start, uint64\_t stop)  
*Count the prime sextuplets within the interval [start, stop].*
- uint64\_t [parallel\\_count\\_primes](#) (uint64\_t start, uint64\_t stop)  
*Count the primes within the interval [start, stop] in parallel.*
- uint64\_t [parallel\\_count\\_twins](#) (uint64\_t start, uint64\_t stop)  
*Count the twin primes within the interval [start, stop] in parallel.*
- uint64\_t [parallel\\_count\\_triplets](#) (uint64\_t start, uint64\_t stop)  
*Count the prime triplets within the interval [start, stop] in parallel.*
- uint64\_t [parallel\\_count\\_quadruplets](#) (uint64\_t start, uint64\_t stop)  
*Count the prime quadruplets within the interval [start, stop] in parallel.*
- uint64\_t [parallel\\_count\\_quintuplets](#) (uint64\_t start, uint64\_t stop)  
*Count the prime quintuplets within the interval [start, stop] in parallel.*
- uint64\_t [parallel\\_count\\_sextuplets](#) (uint64\_t start, uint64\_t stop)  
*Count the prime sextuplets within the interval [start, stop] in parallel.*
- void [print\\_primes](#) (uint64\_t start, uint64\_t stop)  
*Print the primes within the interval [start, stop] to the standard output.*
- void [print\\_twins](#) (uint64\_t start, uint64\_t stop)  
*Print the twin primes within the interval [start, stop] to the standard output.*
- void [print\\_triplets](#) (uint64\_t start, uint64\_t stop)  
*Print the prime triplets within the interval [start, stop] to the standard output.*
- void [print\\_quadruplets](#) (uint64\_t start, uint64\_t stop)  
*Print the prime quadruplets within the interval [start, stop] to the standard output.*
- void [print\\_quintuplets](#) (uint64\_t start, uint64\_t stop)  
*Print the prime quintuplets within the interval [start, stop] to the standard output.*
- void [print\\_sextuplets](#) (uint64\_t start, uint64\_t stop)  
*Print the prime sextuplets within the interval [start, stop] to the standard output.*
- void [callback\\_primes](#) (uint64\_t start, uint64\_t stop, void(\*callback)(uint64\_t prime))  
*Call back the primes within the interval [start, stop].*
- void [callback\\_primes](#) (uint64\_t start, uint64\_t stop, [primesieve::Callback](#)< uint64\_t > \*callback)  
*Call back the primes within the interval [start, stop].*
- int [get\\_sieve\\_size](#) ()  
*Get the current set sieve size in kilobytes.*
- int [get\\_num\\_threads](#) ()  
*Get the current set number of threads.*
- uint64\_t [get\\_max\\_stop](#) ()  
*Returns the largest valid stop number for primesieve.*
- void [set\\_sieve\\_size](#) (int sieve\_size)  
*Set the sieve size in kilobytes.*
- void [set\\_num\\_threads](#) (int num\_threads)

*Set the number of threads for use in subsequent primesieve::parallel\_\* function calls.*

- bool [primesieve\\_test](#) ()

*Run extensive correctness tests.*

- std::string [primesieve\\_version](#) ()

*Get the primesieve version number, in the form "i.j.k".*

### 6.1.1 Detailed Description

All of primesieve's C++ functions and classes are declared inside this namespace.

### 6.1.2 Enumeration Type Documentation

#### 6.1.2.1 anonymous enum

Enumerator

**MAX\_THREADS** Use all CPU cores for prime sieving.

### 6.1.3 Function Documentation

#### 6.1.3.1 void primesieve::callback\_primes ( uint64\_t start, uint64\_t stop, void (\*)(uint64\_t prime) callback )

Call back the primes within the interval [start, stop].

Parameters

<i>callback</i>	A callback function.
-----------------	----------------------

Precondition

$\text{stop} \leq 2^{64} - 2^{32} * 10.$

Examples:

[callback\\_primes.cpp](#).

#### 6.1.3.2 void primesieve::callback\_primes ( uint64\_t start, uint64\_t stop, primesieve::Callback< uint64\_t > \* callback )

Call back the primes within the interval [start, stop].

Parameters

<i>callback</i>	An object derived from <code>primesieve::Callback&lt;uint64_t&gt;</code> .
-----------------	--

**Precondition**
$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$
**6.1.3.3 uint64\_t primesieve::count\_primes ( uint64\_t start, uint64\_t stop )**

Count the primes within the interval [start, stop].

**Precondition**
$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$
**Examples:**

[count\\_primes.cpp](#).

**6.1.3.4 uint64\_t primesieve::count\_quadruplets ( uint64\_t start, uint64\_t stop )**

Count the prime quadruplets within the interval [start, stop].

**Precondition**
$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$
**6.1.3.5 uint64\_t primesieve::count\_quintuplets ( uint64\_t start, uint64\_t stop )**

Count the prime quintuplets within the interval [start, stop].

**Precondition**
$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$
**6.1.3.6 uint64\_t primesieve::count\_sextuplets ( uint64\_t start, uint64\_t stop )**

Count the prime sextuplets within the interval [start, stop].

**Precondition**
$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$
**6.1.3.7 uint64\_t primesieve::count\_triplets ( uint64\_t start, uint64\_t stop )**

Count the prime triplets within the interval [start, stop].

**Precondition**
$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$

6.1.3.8 `uint64_t primesieve::count_twins ( uint64_t start, uint64_t stop )`

Count the twin primes within the interval [start, stop].

**Precondition**

$\text{stop} \leq 2^{64} - 2^{32} * 10$ .

6.1.3.9 `template<typename T > void primesieve::generate_n_primes ( uint64_t n, uint64_t start, std::vector< T > * primes )`  
[inline]

Store the first n primes  $\geq$  start in the primes vector.

**Precondition**

$\text{start} \leq 2^{64} - 2^{32} * 10$ .

6.1.3.10 `template<typename T > void primesieve::generate_primes ( uint64_t stop, std::vector< T > * primes )`  
[inline]

Store the primes  $\leq$  stop in the primes vector.

**Precondition**

$\text{stop} \leq 2^{64} - 2^{32} * 10$ .

**Examples:**

[store\\_primes\\_in\\_vector.cpp](#).

6.1.3.11 `template<typename T > void primesieve::generate_primes ( uint64_t start, uint64_t stop, std::vector< T > * primes )`  
[inline]

Store the primes within the interval [start, stop] in the primes vector.

**Precondition**

$\text{stop} \leq 2^{64} - 2^{32} * 10$ .

6.1.3.12 `uint64_t primesieve::get_max_stop ( )`

Returns the largest valid stop number for primesieve.

**Returns**

$(2^{64}-1) - (2^{32}-1) * 10$ .

6.1.3.13 `int primesieve::get_num_threads ( )`

Get the current set number of threads.

**Note**

By default MAX\_THREADS (-1) is returned.

6.1.3.14 `uint64_t primesieve::nth_prime ( int64_t n, uint64_t start = 0 )`

Find the nth prime.

**Parameters**

<i>n</i>	if $n = 0$ finds the 1st prime $\geq$ start, if $n > 0$ finds the $n$ th prime $>$ start, if $n < 0$ finds the $n$ th prime $<$ start (backwards).
----------	--

**Precondition**

$\text{start} \leq 2^{64} - 2^{32} * 11.$

**Examples:**

[nth\\_prime.cpp](#).

**6.1.3.15 `uint64_t primesieve::parallel_count_primes ( uint64_t start, uint64_t stop )`**

Count the primes within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve::set\\_num\\_threads\(int\)](#) to change the number of threads.

**Precondition**

$\text{stop} \leq 2^{64} - 2^{32} * 10.$

**Examples:**

[count\\_primes.cpp](#).

**6.1.3.16 `uint64_t primesieve::parallel_count_quadruplets ( uint64_t start, uint64_t stop )`**

Count the prime quadruplets within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve::set\\_num\\_threads\(int\)](#) to change the number of threads.

**Precondition**

$\text{stop} \leq 2^{64} - 2^{32} * 10.$

**6.1.3.17 `uint64_t primesieve::parallel_count_quintuplets ( uint64_t start, uint64_t stop )`**

Count the prime quintuplets within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve::set\\_num\\_threads\(int\)](#) to change the number of threads.

**Precondition**

$\text{stop} \leq 2^{64} - 2^{32} * 10.$

**6.1.3.18** `uint64_t primesieve::parallel_count_sextuplets ( uint64_t start, uint64_t stop )`

Count the prime sextuplets within the interval [start, stop] in parallel.

By default all CPU cores are used, use `primesieve::set_num_threads(int)` to change the number of threads.

**Precondition**

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$
**6.1.3.19** `uint64_t primesieve::parallel_count_triplets ( uint64_t start, uint64_t stop )`

Count the prime triplets within the interval [start, stop] in parallel.

By default all CPU cores are used, use `primesieve::set_num_threads(int)` to change the number of threads.

**Precondition**

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$
**6.1.3.20** `uint64_t primesieve::parallel_count_twins ( uint64_t start, uint64_t stop )`

Count the twin primes within the interval [start, stop] in parallel.

By default all CPU cores are used, use `primesieve::set_num_threads(int)` to change the number of threads.

**Precondition**

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$
**6.1.3.21** `uint64_t primesieve::parallel_nth_prime ( int64_t n, uint64_t start = 0 )`

Find the nth prime in parallel.

By default all CPU cores are used, use `primesieve::set_num_threads(int)` to change the number of threads.

**Parameters**

<i>n</i>	if $n = 0$ finds the 1st prime $\geq$ start, if $n > 0$ finds the nth prime $>$ start, if $n < 0$ finds the nth prime $<$ start (backwards).
----------	--

**Precondition**

$$\text{start} \leq 2^{64} - 2^{32} * 11.$$

#### 6.1.3.22 `bool primesieve::primesieve_test ( )`

Run extensive correctness tests.

The tests last about one minute on a quad core CPU from 2013 and use up to 1 gigabyte of memory.

##### Returns

true if success else false.

#### 6.1.3.23 `void primesieve::print_primes ( uint64_t start, uint64_t stop )`

Print the primes within the interval [start, stop] to the standard output.

##### Precondition

$stop \leq 2^{64} - 2^{32} * 10$ .

#### 6.1.3.24 `void primesieve::print_quadruplets ( uint64_t start, uint64_t stop )`

Print the prime quadruplets within the interval [start, stop] to the standard output.

##### Precondition

$stop \leq 2^{64} - 2^{32} * 10$ .

#### 6.1.3.25 `void primesieve::print_quintuplets ( uint64_t start, uint64_t stop )`

Print the prime quintuplets within the interval [start, stop] to the standard output.

##### Precondition

$stop \leq 2^{64} - 2^{32} * 10$ .

#### 6.1.3.26 `void primesieve::print_sextuplets ( uint64_t start, uint64_t stop )`

Print the prime sextuplets within the interval [start, stop] to the standard output.

##### Precondition

$stop \leq 2^{64} - 2^{32} * 10$ .

**6.1.3.27 void primesieve::print\_triplets ( uint64\_t start, uint64\_t stop )**

Print the prime triplets within the interval [start, stop] to the standard output.

**Precondition**

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$
**6.1.3.28 void primesieve::print\_twins ( uint64\_t start, uint64\_t stop )**

Print the twin primes within the interval [start, stop] to the standard output.

**Precondition**

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$
**6.1.3.29 void primesieve::set\_num\_threads ( int num\_threads )**

Set the number of threads for use in subsequent primesieve::parallel\_\* function calls.

Note that this only changes the number of threads for the current process.

**Parameters**

<i>num_threads</i>	Number of threads for sieving or MAX_THREADS to use all CPU cores.
--------------------	--

**6.1.3.30 void primesieve::set\_sieve\_size ( int sieve\_size )**

Set the sieve size in kilobytes.

The best sieving performance is achieved with a sieve size of your CPU's L1 data cache size (per core). For sieving  $\geq 10^{17}$  a sieve size of your CPU's L2 cache size sometimes performs better.

**Parameters**

<i>sieve_size</i>	Sieve size in kilobytes.
-------------------	--------------------------

**Precondition**

$$\text{sieve\_size} \geq 1 \ \&\& \ \text{sieve\_size} \leq 2048.$$



# Chapter 7

## Class Documentation

### 7.1 primesieve::Callback< T > Class Template Reference

callback interface class.

```
#include <Callback.hpp>
```

#### Public Member Functions

- virtual void **callback** (T prime)=0

#### 7.1.1 Detailed Description

```
template<typename T>
class primesieve::Callback< T >
```

callback interface class.

Objects derived from this class can be passed to the [primesieve::generate\\_primes\(\)](#) functions.

#### Parameters

<i>T</i>	must be uint64_t.
----------	----------------------

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

- [Callback.hpp](#)

### 7.2 primesieve::iterator Class Reference

[primesieve::iterator](#) allows to easily iterate over primes both forwards and backwards.

```
#include <iterator.hpp>
```

## Public Member Functions

- `iterator` (`uint64_t start=0`, `uint64_t stop_hint=get_max_stop()`)  
*Create a new iterator object.*
- `void skipto` (`uint64_t start`, `uint64_t stop_hint=get_max_stop()`)  
*Reinitialize this iterator object to start.*
- `uint64_t next_prime` ()  
*Advance the iterator by one position.*
- `uint64_t previous_prime` ()  
*Get the previous prime, or 0 if input  $\leq 2$  e.g.*

### 7.2.1 Detailed Description

`primesieve::iterator` allows to easily iterate over primes both forwards and backwards.

Generating the first prime has a complexity of  $O(r \log \log r)$  operations with  $r = n^{0.5}$ , after that any additional prime is generated in amortized  $O(\log n \log \log n)$  operations. The memory usage is about  $\pi(n^{0.5}) * 16$  bytes. `primesieve::iterator` objects are very convenient to use at the cost of being slightly slower than the `callback_primes()` functions.

Examples:

`previous_prime.cpp`, and `primesieve_iterator.cpp`.

### 7.2.2 Constructor & Destructor Documentation

#### 7.2.2.1 `primesieve::iterator::iterator ( uint64_t start = 0, uint64_t stop_hint = get_max_stop () )`

Create a new iterator object.

Parameters

<code>start</code>	Generate primes $> start$ (or $< start$ ).
<code>stop_hint</code>	Stop number optimization hint, gives significant speed up if few primes are generated. E.g. if you want to generate the primes below 1000 use <code>stop_hint = 1000</code> .

Precondition

`start  $\leq 2^{64} - 2^{32} * 10$`

### 7.2.3 Member Function Documentation

#### 7.2.3.1 `uint64_t primesieve::iterator::next_prime ( ) [inline]`

Advance the iterator by one position.

**Returns**

The next prime.

**Examples:**

[primesieve\\_iterator.cpp](#).

**7.2.3.2 uint64\_t primesieve::iterator::previous\_prime ( ) [inline]**

Get the previous prime, or 0 if input  $\leq 2$  e.g.

previous\_prime(2) = 0.

**Examples:**

[previous\\_prime.cpp](#).

**7.2.3.3 void primesieve::iterator::skipto ( uint64\_t start, uint64\_t stop\_hint = get\_max\_stop ( ) )**

Reinitialize this iterator object to start.

**Parameters**

<i>start</i>	Generate primes $> start$ (or $< start$ ).
<i>stop_hint</i>	Stop number optimization hint, gives significant speed up if few primes are generated. E.g. if you want to generate the primes below 1000 use stop_hint = 1000.

**Precondition**

$start \leq 2^{64} - 2^{32} * 10$

**Examples:**

[previous\\_prime.cpp](#).

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

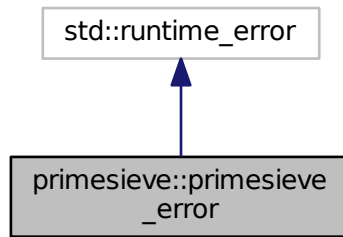
- [iterator.hpp](#)

**7.3 primesieve::primesieve\_error Class Reference**

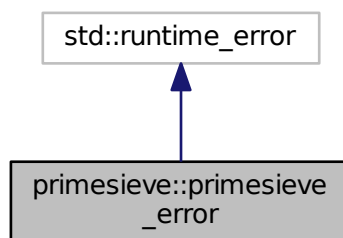
primesieve throws a [primesieve\\_error](#) exception if an error occurs that cannot be handled e.g.

```
#include <primesieve_error.hpp>
```

Inheritance diagram for `primesieve::primesieve_error`:



Collaboration diagram for `primesieve::primesieve_error`:



## Public Member Functions

- **`primesieve_error`** (`const std::string &msg`)

### 7.3.1 Detailed Description

`primesieve` throws a [primesieve\\_error](#) exception if an error occurs that cannot be handled e.g.

`stop > primesieve::max_stop()`.

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

- [primesieve\\_error.hpp](#)

## 7.4 primesieve\_iterator Struct Reference

C prime iterator, please refer to [primesieve\\_iterator.h](#) for more information.

```
#include <primesieve_iterator.h>
```

### Public Attributes

- `size_t i_`
- `size_t last_idx_`
- `uint64_t * primes_`
- `uint64_t * primes_pimpl_`
- `uint64_t start_`
- `uint64_t stop_`
- `uint64_t stop_hint_`
- `uint64_t tiny_cache_size_`
- `int is_error_`

### 7.4.1 Detailed Description

C prime iterator, please refer to [primesieve\\_iterator.h](#) for more information.

Examples:

[previous\\_prime.c](#), and [primesieve\\_iterator.c](#).

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

- [primesieve\\_iterator.h](#)



## Chapter 8

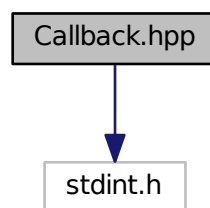
# File Documentation

### 8.1 Callback.hpp File Reference

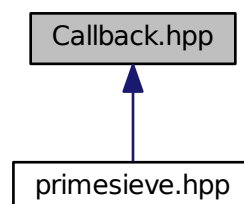
Callback interface classes.

```
#include <stdint.h>
```

Include dependency graph for Callback.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class `primesieve::Callback< T >`  
*callback interface class.*

## Namespaces

- `primesieve`  
*All of primesieve's C++ functions and classes are declared inside this namespace.*

### 8.1.1 Detailed Description

Callback interface classes.

Copyright (C) 2015 Kim Walisch, [kim.walisch@gmail.com](mailto:kim.walisch@gmail.com)

This file is distributed under the BSD License. See the COPYING file in the top level directory.

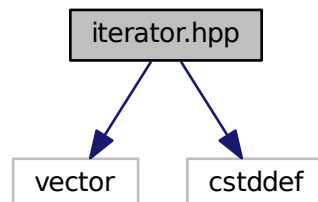
## 8.2 iterator.hpp File Reference

The iterator class allows to easily iterate (forward and backward) over prime numbers.

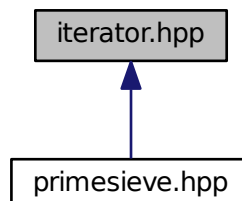
```
#include <vector>
```

```
#include <cstdint>
```

Include dependency graph for `iterator.hpp`:



This graph shows which files directly or indirectly include this file:



## Classes

- class `primesieve::iterator`  
*`primesieve::iterator` allows to easily iterate over primes both forwards and backwards.*

## Namespaces

- `primesieve`  
*All of `primesieve`'s C++ functions and classes are declared inside this namespace.*

## Functions

- `uint64_t primesieve::get_max_stop ()`  
*Returns the largest valid stop number for `primesieve`.*

### 8.2.1 Detailed Description

The iterator class allows to easily iterate (forward and backward) over prime numbers.

Copyright (C) 2015 Kim Walisch, [kim.walisch@gmail.com](mailto:kim.walisch@gmail.com)

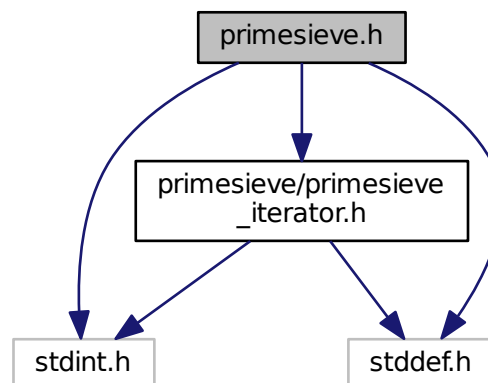
This file is distributed under the BSD License. See the COPYING file in the top level directory.

## 8.3 primesieve.h File Reference

primesieve C API.

```
#include <primesieve/primesieve_iterator.h>
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for `primesieve.h`:



## Macros

- `#define PRIMESIEVE_VERSION "5.6.0"`
- `#define PRIMESIEVE_VERSION_MAJOR 5`
- `#define PRIMESIEVE_VERSION_MINOR 6`
- `#define PRIMESIEVE_VERSION_PATCH 0`
- `#define PRIMESIEVE_ERROR ((uint64_t) ~((uint64_t) 0))`  
*primesieve functions return PRIMESIEVE\_ERROR (UINT64\_MAX) if any error occurs.*

## Enumerations

- `enum {`  
`MAX_THREADS = -1, SHORT_PRIMES, USHORT_PRIMES, INT_PRIMES,`  
`UINT_PRIMES, LONG_PRIMES, ULONG_PRIMES, LONGLONG_PRIMES,`  
`ULONGLONG_PRIMES, INT16_PRIMES, UINT16_PRIMES, INT32_PRIMES,`  
`UINT32_PRIMES, INT64_PRIMES, UINT64_PRIMES }`

## Functions

- `void * primesieve_generate_primes (uint64_t start, uint64_t stop, size_t *size, int type)`  
*Get an array with the primes inside the interval [start, stop].*
- `void * primesieve_generate_n_primes (uint64_t n, uint64_t start, int type)`  
*Get an array with the first n primes >= start.*
- `uint64_t primesieve_nth_prime (int64_t n, uint64_t start)`  
*Find the nth prime.*
- `uint64_t primesieve_parallel_nth_prime (int64_t n, uint64_t start)`  
*Find the nth prime in parallel.*
- `uint64_t primesieve_count_primes (uint64_t start, uint64_t stop)`  
*Count the primes within the interval [start, stop].*
- `uint64_t primesieve_count_twins (uint64_t start, uint64_t stop)`  
*Count the twin primes within the interval [start, stop].*
- `uint64_t primesieve_count_triplets (uint64_t start, uint64_t stop)`  
*Count the prime triplets within the interval [start, stop].*
- `uint64_t primesieve_count_quadruplets (uint64_t start, uint64_t stop)`  
*Count the prime quadruplets within the interval [start, stop].*
- `uint64_t primesieve_count_quintuplets (uint64_t start, uint64_t stop)`  
*Count the prime quintuplets within the interval [start, stop].*
- `uint64_t primesieve_count_sextuplets (uint64_t start, uint64_t stop)`  
*Count the prime sextuplets within the interval [start, stop].*
- `uint64_t primesieve_parallel_count_primes (uint64_t start, uint64_t stop)`  
*Count the primes within the interval [start, stop] in parallel.*
- `uint64_t primesieve_parallel_count_twins (uint64_t start, uint64_t stop)`  
*Count the twin primes within the interval [start, stop] in parallel.*
- `uint64_t primesieve_parallel_count_triplets (uint64_t start, uint64_t stop)`  
*Count the prime triplets within the interval [start, stop] in parallel.*
- `uint64_t primesieve_parallel_count_quadruplets (uint64_t start, uint64_t stop)`  
*Count the prime quadruplets within the interval [start, stop] in parallel.*
- `uint64_t primesieve_parallel_count_quintuplets (uint64_t start, uint64_t stop)`  
*Count the prime quintuplets within the interval [start, stop] in parallel.*
- `uint64_t primesieve_parallel_count_sextuplets (uint64_t start, uint64_t stop)`

- Count the prime sextuplets within the interval [start, stop] in parallel.*

  - void [primesieve\\_print\\_primes](#) (uint64\_t start, uint64\_t stop)

*Print the primes within the interval [start, stop] to the standard output.*
- void [primesieve\\_print\\_twins](#) (uint64\_t start, uint64\_t stop)

*Print the twin primes within the interval [start, stop] to the standard output.*
- void [primesieve\\_print\\_triplets](#) (uint64\_t start, uint64\_t stop)

*Print the prime triplets within the interval [start, stop] to the standard output.*
- void [primesieve\\_print\\_quadruplets](#) (uint64\_t start, uint64\_t stop)

*Print the prime quadruplets within the interval [start, stop] to the standard output.*
- void [primesieve\\_print\\_quintuplets](#) (uint64\_t start, uint64\_t stop)

*Print the prime quintuplets within the interval [start, stop] to the standard output.*
- void [primesieve\\_print\\_sextuplets](#) (uint64\_t start, uint64\_t stop)

*Print the prime sextuplets within the interval [start, stop] to the standard output.*
- void [primesieve\\_callback\\_primes](#) (uint64\_t start, uint64\_t stop, void(\*callback)(uint64\_t prime))

*Call back the primes within the interval [start, stop].*
- int [primesieve\\_get\\_sieve\\_size](#) ()

*Get the current set sieve size in kilobytes.*
- int [primesieve\\_get\\_num\\_threads](#) ()

*Get the current set number of threads.*
- uint64\_t [primesieve\\_get\\_max\\_stop](#) ()

*Returns the largest valid stop number for primesieve.*
- void [primesieve\\_set\\_sieve\\_size](#) (int sieve\_size)

*Set the sieve size in kilobytes.*
- void [primesieve\\_set\\_num\\_threads](#) (int num\_threads)

*Set the number of threads for use in subsequent `primesieve_parallel_*` function calls.*
- void [primesieve\\_free](#) (void \*primes)

*Deallocate a primes array created using the `primesieve_generate_primes()` or `primesieve_generate_n_primes()` functions.*
- int [primesieve\\_test](#) ()

*Run extensive correctness tests.*
- const char \* [primesieve\\_version](#) ()

*Get the primesieve version number, in the form "i.j.k".*

### 8.3.1 Detailed Description

primesieve C API.

primesieve is a library for fast prime number generation. In case an error occurs `errno` is set to `EDOM` and `PRIMESIEVE_ERROR` is returned.

Copyright (C) 2015 Kim Walisch, [kim.walisch@gmail.com](mailto:kim.walisch@gmail.com)

This file is distributed under the BSD License.

## 8.3.2 Enumeration Type Documentation

### 8.3.2.1 anonymous enum

#### Enumerator

**MAX\_THREADS** Use all CPU cores for prime sieving.

**SHORT\_PRIMES** Generate primes of short type.

**USHORT\_PRIMES** Generate primes of unsigned short type.

**INT\_PRIMES** Generate primes of int type.

**UINT\_PRIMES** Generate primes of unsigned int type.

**LONG\_PRIMES** Generate primes of long type.

**ULONG\_PRIMES** Generate primes of unsigned long type.

**LONGLONG\_PRIMES** Generate primes of long long type.

**ULONGLONG\_PRIMES** Generate primes of unsigned long long type.

**INT16\_PRIMES** Generate primes of int16\_t type.

**UINT16\_PRIMES** Generate primes of uint16\_t type.

**INT32\_PRIMES** Generate primes of int32\_t type.

**UINT32\_PRIMES** Generate primes of uint32\_t type.

**INT64\_PRIMES** Generate primes of int64\_t type.

**UINT64\_PRIMES** Generate primes of uint64\_t type.

## 8.3.3 Function Documentation

### 8.3.3.1 void primesieve\_callback\_primes ( uint64\_t start, uint64\_t stop, void (\*)(uint64\_t prime) callback )

Call back the primes within the interval [start, stop].

#### Parameters

<i>callback</i>	A callback function.
-----------------	----------------------

#### Precondition

$stop \leq 2^{64} - 2^{32} * 10.$

### 8.3.3.2 uint64\_t primesieve\_count\_primes ( uint64\_t start, uint64\_t stop )

Count the primes within the interval [start, stop].

#### Precondition

$stop \leq 2^{64} - 2^{32} * 10.$

#### Examples:

[count\\_primes.c](#).

#### 8.3.3.3 uint64\_t primesieve\_count\_quadruplets ( uint64\_t start, uint64\_t stop )

Count the prime quadruplets within the interval [start, stop].

##### Precondition

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$

#### 8.3.3.4 uint64\_t primesieve\_count\_quintuplets ( uint64\_t start, uint64\_t stop )

Count the prime quintuplets within the interval [start, stop].

##### Precondition

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$

#### 8.3.3.5 uint64\_t primesieve\_count\_sextuplets ( uint64\_t start, uint64\_t stop )

Count the prime sextuplets within the interval [start, stop].

##### Precondition

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$

#### 8.3.3.6 uint64\_t primesieve\_count\_triplets ( uint64\_t start, uint64\_t stop )

Count the prime triplets within the interval [start, stop].

##### Precondition

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$

#### 8.3.3.7 uint64\_t primesieve\_count\_twins ( uint64\_t start, uint64\_t stop )

Count the twin primes within the interval [start, stop].

##### Precondition

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$

#### 8.3.3.8 void\* primesieve\_generate\_n\_primes ( uint64\_t n, uint64\_t start, int type )

Get an array with the first n primes  $\geq$  start.

## Parameters

<i>type</i>	The type of the primes to generate, e.g. INT_PRIMES.
-------------	--

## Precondition

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$

## Examples:

[store\\_primes\\_in\\_array.c](#).

**8.3.3.9** `void* primesieve_generate_primes ( uint64_t start, uint64_t stop, size_t * size, int type )`

Get an array with the primes inside the interval [start, stop].

## Parameters

<i>size</i>	The size of the returned primes array.
<i>type</i>	The type of the primes to generate, e.g. INT_PRIMES.

## Precondition

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$

## Examples:

[store\\_primes\\_in\\_array.c](#).

**8.3.3.10** `uint64_t primesieve_get_max_stop ( )`

Returns the largest valid stop number for primesieve.

## Returns

$$(2^{64}-1) - (2^{32}-1) * 10.$$

**8.3.3.11** `int primesieve_get_num_threads ( )`

Get the current set number of threads.

## Note

By default MAX\_THREADS (-1) is returned.

**8.3.3.12** `int primesieve_get_sieve_size ( )`

Get the current set sieve size in kilobytes.

**8.3.3.13** `uint64_t primesieve_nth_prime ( int64_t n, uint64_t start )`

Find the nth prime.

## Parameters

$n$	if $n = 0$ finds the 1st prime $\geq$ start, if $n > 0$ finds the $n$ th prime $>$ start, if $n < 0$ finds the $n$ th prime $<$ start (backwards).
-----	--

## Precondition

$$\text{start} \leq 2^{64} - 2^{32} * 11.$$

## Examples:

[nth\\_prime.c](#).

## 8.3.3.14 uint64\_t primesieve\_parallel\_count\_primes ( uint64\_t start, uint64\_t stop )

Count the primes within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve\\_set\\_num\\_threads\(int\)](#) to change the number of threads.

## Precondition

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$

## Examples:

[count\\_primes.c](#).

## 8.3.3.15 uint64\_t primesieve\_parallel\_count\_quadruplets ( uint64\_t start, uint64\_t stop )

Count the prime quadruplets within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve\\_set\\_num\\_threads\(int\)](#) to change the number of threads.

## Precondition

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$

## 8.3.3.16 uint64\_t primesieve\_parallel\_count\_quintuplets ( uint64\_t start, uint64\_t stop )

Count the prime quintuplets within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve\\_set\\_num\\_threads\(int\)](#) to change the number of threads.

## Precondition

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$

### 8.3.3.17 `uint64_t primesieve_parallel_count_sextuplets ( uint64_t start, uint64_t stop )`

Count the prime sextuplets within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve\\_set\\_num\\_threads\(int\)](#) to change the number of threads.

#### Precondition

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$

### 8.3.3.18 `uint64_t primesieve_parallel_count_triplets ( uint64_t start, uint64_t stop )`

Count the prime triplets within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve\\_set\\_num\\_threads\(int\)](#) to change the number of threads.

#### Precondition

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$

### 8.3.3.19 `uint64_t primesieve_parallel_count_twins ( uint64_t start, uint64_t stop )`

Count the twin primes within the interval [start, stop] in parallel.

By default all CPU cores are used, use [primesieve\\_set\\_num\\_threads\(int\)](#) to change the number of threads.

#### Precondition

$$\text{stop} \leq 2^{64} - 2^{32} * 10.$$

### 8.3.3.20 `uint64_t primesieve_parallel_nth_prime ( int64_t n, uint64_t start )`

Find the nth prime in parallel.

By default all CPU cores are used, use [primesieve\\_set\\_num\\_threads\(int\)](#) to change the number of threads.

#### Parameters

<i>n</i>	if <i>n</i> = 0 finds the 1st prime $\geq$ start, if <i>n</i> > 0 finds the nth prime $>$ start, if <i>n</i> < 0 finds the nth prime $<$ start (backwards).
----------	---

#### Precondition

$$\text{start} \leq 2^{64} - 2^{32} * 11.$$

**8.3.3.21** void primesieve\_print\_primes ( uint64\_t *start*, uint64\_t *stop* )

Print the primes within the interval [start, stop] to the standard output.

**Precondition**

$\text{stop} \leq 2^{64} - 2^{32} * 10.$

**8.3.3.22** void primesieve\_print\_quadruplets ( uint64\_t *start*, uint64\_t *stop* )

Print the prime quadruplets within the interval [start, stop] to the standard output.

**Precondition**

$\text{stop} \leq 2^{64} - 2^{32} * 10.$

**8.3.3.23** void primesieve\_print\_quintuplets ( uint64\_t *start*, uint64\_t *stop* )

Print the prime quintuplets within the interval [start, stop] to the standard output.

**Precondition**

$\text{stop} \leq 2^{64} - 2^{32} * 10.$

**8.3.3.24** void primesieve\_print\_sextuplets ( uint64\_t *start*, uint64\_t *stop* )

Print the prime sextuplets within the interval [start, stop] to the standard output.

**Precondition**

$\text{stop} \leq 2^{64} - 2^{32} * 10.$

**8.3.3.25** void primesieve\_print\_triplets ( uint64\_t *start*, uint64\_t *stop* )

Print the prime triplets within the interval [start, stop] to the standard output.

**Precondition**

$\text{stop} \leq 2^{64} - 2^{32} * 10.$

**8.3.3.26** void primesieve\_print\_twins ( uint64\_t *start*, uint64\_t *stop* )

Print the twin primes within the interval [start, stop] to the standard output.

**Precondition**

$\text{stop} \leq 2^{64} - 2^{32} * 10.$

**8.3.3.27** void primesieve\_set\_num\_threads ( int *num\_threads* )

Set the number of threads for use in subsequent primesieve\_parallel\_\* function calls.

Note that this only changes the number of threads for the current process.

**Parameters**

<i>num_threads</i>	Number of threads for sieving or MAX_THREADS to use all CPU cores.
--------------------	--

**8.3.3.28 void primesieve\_set\_sieve\_size ( int *sieve\_size* )**

Set the sieve size in kilobytes.

The best sieving performance is achieved with a sieve size of your CPU's L1 data cache size (per core). For sieving  $\geq 10^{17}$  a sieve size of your CPU's L2 cache size sometimes performs better.

**Parameters**

<i>sieve_size</i>	Sieve size in kilobytes.
-------------------	--------------------------

**Precondition**

`sieve_size >= 1 && <= 2048.`

**8.3.3.29 int primesieve\_test ( )**

Run extensive correctness tests.

The tests last about one minute on a quad core CPU from 2013 and use up to 1 gigabyte of memory.

**Returns**

1 if success, 0 if error.

**8.3.3.30 const char\* primesieve\_version ( )**

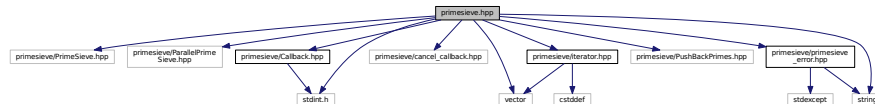
Get the primesieve version number, in the form "i.j.k".

## 8.4 primesieve.hpp File Reference

primesieve C++ API.

```
#include <primesieve/PrimeSieve.hpp>
#include <primesieve/ParallelPrimeSieve.hpp>
#include <primesieve/Callback.hpp>
#include <primesieve/cancel_callback.hpp>
#include <primesieve/iterator.hpp>
#include <primesieve/PushBackPrimes.hpp>
#include <primesieve/primesieve_error.hpp>
#include <stdint.h>
#include <vector>
#include <string>
```

Include dependency graph for primesieve.hpp:



## Namespaces

- [primesieve](#)

*All of primesieve's C++ functions and classes are declared inside this namespace.*

## Macros

- `#define PRimesieve_VERSION "5.6.0"`
- `#define PRimesieve_VERSION_MAJOR 5`
- `#define PRimesieve_VERSION_MINOR 6`
- `#define PRimesieve_VERSION_PATCH 0`

## Enumerations

- enum { [primesieve::MAX\\_THREADS](#) = -1 }

## Functions

- template<typename T >  
void [primesieve::generate\\_primes](#) (uint64\_t stop, std::vector< T > \*primes)  
*Store the primes <= stop in the primes vector.*
- template<typename T >  
void [primesieve::generate\\_primes](#) (uint64\_t start, uint64\_t stop, std::vector< T > \*primes)  
*Store the primes within the interval [start, stop] in the primes vector.*
- template<typename T >  
void [primesieve::generate\\_n\\_primes](#) (uint64\_t n, std::vector< T > \*primes)  
*Store the first n primes in the primes vector.*
- template<typename T >  
void [primesieve::generate\\_n\\_primes](#) (uint64\_t n, uint64\_t start, std::vector< T > \*primes)  
*Store the first n primes >= start in the primes vector.*
- uint64\_t [primesieve::nth\\_prime](#) (uint64\_t n, uint64\_t start=0)  
*Find the nth prime.*

- uint64\_t [primesieve::parallel\\_nth\\_prime](#) (int64\_t n, uint64\_t start=0)  
*Find the nth prime in parallel.*
- uint64\_t [primesieve::count\\_primes](#) (uint64\_t start, uint64\_t stop)  
*Count the primes within the interval [start, stop].*
- uint64\_t [primesieve::count\\_twins](#) (uint64\_t start, uint64\_t stop)  
*Count the twin primes within the interval [start, stop].*
- uint64\_t [primesieve::count\\_triplets](#) (uint64\_t start, uint64\_t stop)  
*Count the prime triplets within the interval [start, stop].*
- uint64\_t [primesieve::count\\_quadruplets](#) (uint64\_t start, uint64\_t stop)  
*Count the prime quadruplets within the interval [start, stop].*
- uint64\_t [primesieve::count\\_quintuplets](#) (uint64\_t start, uint64\_t stop)  
*Count the prime quintuplets within the interval [start, stop].*
- uint64\_t [primesieve::count\\_sextuplets](#) (uint64\_t start, uint64\_t stop)  
*Count the prime sextuplets within the interval [start, stop].*
- uint64\_t [primesieve::parallel\\_count\\_primes](#) (uint64\_t start, uint64\_t stop)  
*Count the primes within the interval [start, stop] in parallel.*
- uint64\_t [primesieve::parallel\\_count\\_twins](#) (uint64\_t start, uint64\_t stop)  
*Count the twin primes within the interval [start, stop] in parallel.*
- uint64\_t [primesieve::parallel\\_count\\_triplets](#) (uint64\_t start, uint64\_t stop)  
*Count the prime triplets within the interval [start, stop] in parallel.*
- uint64\_t [primesieve::parallel\\_count\\_quadruplets](#) (uint64\_t start, uint64\_t stop)  
*Count the prime quadruplets within the interval [start, stop] in parallel.*
- uint64\_t [primesieve::parallel\\_count\\_quintuplets](#) (uint64\_t start, uint64\_t stop)  
*Count the prime quintuplets within the interval [start, stop] in parallel.*
- uint64\_t [primesieve::parallel\\_count\\_sextuplets](#) (uint64\_t start, uint64\_t stop)  
*Count the prime sextuplets within the interval [start, stop] in parallel.*
- void [primesieve::print\\_primes](#) (uint64\_t start, uint64\_t stop)  
*Print the primes within the interval [start, stop] to the standard output.*
- void [primesieve::print\\_twins](#) (uint64\_t start, uint64\_t stop)  
*Print the twin primes within the interval [start, stop] to the standard output.*
- void [primesieve::print\\_triplets](#) (uint64\_t start, uint64\_t stop)  
*Print the prime triplets within the interval [start, stop] to the standard output.*
- void [primesieve::print\\_quadruplets](#) (uint64\_t start, uint64\_t stop)  
*Print the prime quadruplets within the interval [start, stop] to the standard output.*
- void [primesieve::print\\_quintuplets](#) (uint64\_t start, uint64\_t stop)  
*Print the prime quintuplets within the interval [start, stop] to the standard output.*
- void [primesieve::print\\_sextuplets](#) (uint64\_t start, uint64\_t stop)  
*Print the prime sextuplets within the interval [start, stop] to the standard output.*
- void [primesieve::callback\\_primes](#) (uint64\_t start, uint64\_t stop, void(\*callback)(uint64\_t prime))  
*Call back the primes within the interval [start, stop].*
- void [primesieve::callback\\_primes](#) (uint64\_t start, uint64\_t stop, [primesieve::Callback](#)< uint64\_t > \*callback)  
*Call back the primes within the interval [start, stop].*
- int [primesieve::get\\_sieve\\_size](#) ()  
*Get the current set sieve size in kilobytes.*
- int [primesieve::get\\_num\\_threads](#) ()  
*Get the current set number of threads.*
- uint64\_t [primesieve::get\\_max\\_stop](#) ()  
*Returns the largest valid stop number for primesieve.*
- void [primesieve::set\\_sieve\\_size](#) (int sieve\_size)  
*Set the sieve size in kilobytes.*
- void [primesieve::set\\_num\\_threads](#) (int num\_threads)

*Set the number of threads for use in subsequent `primesieve::parallel_*` function calls.*

- `bool primesieve::primesieve_test ()`

*Run extensive correctness tests.*

- `std::string primesieve::primesieve_version ()`

*Get the primesieve version number, in the form "i.j.k".*

### 8.4.1 Detailed Description

primesieve C++ API.

primesieve is a library for fast prime number generation, in case an error occurs a `primesieve::primesieve_error` exception (derived from `std::runtime_error`) will be thrown.

Copyright (C) 2015 Kim Walisch, [kim.walisch@gmail.com](mailto:kim.walisch@gmail.com)

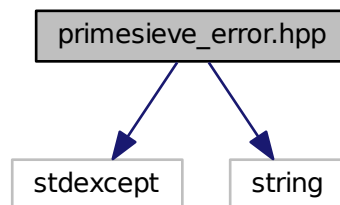
This file is distributed under the BSD License.

## 8.5 primesieve\_error.hpp File Reference

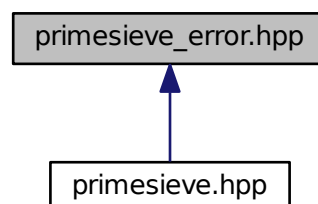
The `primesieve_error` class is used for all exceptions within primesieve.

```
#include <stdexcept>
#include <string>
```

Include dependency graph for `primesieve_error.hpp`:



This graph shows which files directly or indirectly include this file:



## Classes

- class `primesieve::primesieve_error`  
*primesieve throws a `primesieve_error` exception if an error occurs that cannot be handled e.g.*

## Namespaces

- `primesieve`  
*All of primesieve's C++ functions and classes are declared inside this namespace.*

### 8.5.1 Detailed Description

The `primesieve_error` class is used for all exceptions within `primesieve`.

Copyright (C) 2013 Kim Walisch, [kim.walisch@gmail.com](mailto:kim.walisch@gmail.com)

This file is distributed under the BSD License. See the COPYING file in the top level directory.

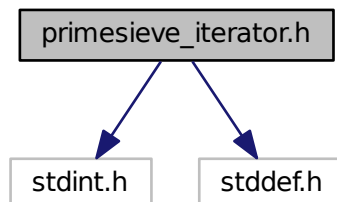
## 8.6 `primesieve_iterator.h` File Reference

`primesieve_iterator` allows to easily iterate over primes both forwards and backwards.

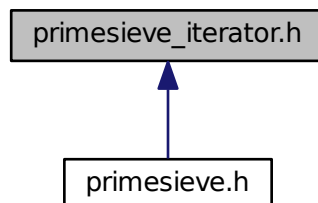
```
#include <stdint.h>
```

```
#include <stddef.h>
```

Include dependency graph for `primesieve_iterator.h`:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [primesieve\\_iterator](#)  
C prime iterator, please refer to [primesieve\\_iterator.h](#) for more information.

## Functions

- void [primesieve\\_init](#) ([primesieve\\_iterator](#) \*pi)  
*Initialize the primesieve iterator before first using it.*
- void [primesieve\\_free\\_iterator](#) ([primesieve\\_iterator](#) \*pi)  
*Free all memory.*
- void [primesieve\\_skipto](#) ([primesieve\\_iterator](#) \*pi, uint64\_t start, uint64\_t stop\_hint)  
*Set the primesieve iterator to start.*
- static uint64\_t [primesieve\\_next\\_prime](#) ([primesieve\\_iterator](#) \*pi)  
*Get the next prime.*
- static uint64\_t [primesieve\\_previous\\_prime](#) ([primesieve\\_iterator](#) \*pi)  
*Get the previous prime, or 0 if input  $\leq 2$  e.g.*

### 8.6.1 Detailed Description

[primesieve\\_iterator](#) allows to easily iterate over primes both forwards and backwards.

Generating the first prime has a complexity of  $O(r \log \log r)$  operations with  $r = n^{0.5}$ , after that any additional prime is generated in amortized  $O(\log n \log \log n)$  operations. The memory usage is about  $\pi(n^{0.5}) * 16$  bytes. [primesieve\\_iterator](#) objects are very convenient to use at the cost of being slightly slower than the [primesieve\\_↔callback\\_primes\(\)](#) functions.

The [primesieve\\_iterator.c](#) example shows how to use [primesieve\\_iterator](#). If any error occurs `errno` is set to `EDOM` and [primesieve\\_next\\_prime\(\)](#) and [primesieve\\_previous\\_prime\(\)](#) return `PRIMESIEVE_ERROR`.

Copyright (C) 2015 Kim Walisch, [kim.walisch@gmail.com](mailto:kim.walisch@gmail.com)

This file is distributed under the BSD License. See the COPYING file in the top level directory.

### 8.6.2 Function Documentation

#### 8.6.2.1 void primesieve\_free\_iterator ( primesieve\_iterator \* pi )

Free all memory.

Examples:

[previous\\_prime.c](#), and [primesieve\\_iterator.c](#).

#### 8.6.2.2 void primesieve\_init ( primesieve\_iterator \* pi )

Initialize the primesieve iterator before first using it.

Examples:

[previous\\_prime.c](#), and [primesieve\\_iterator.c](#).

**8.6.2.3** `static uint64_t primesieve_next_prime ( primesieve_iterator * pi )` `[inline],[static]`

Get the next prime.

Examples:

[primesieve\\_iterator.c](#).

**8.6.2.4** `static uint64_t primesieve_previous_prime ( primesieve_iterator * pi )` `[inline],[static]`

Get the previous prime, or 0 if input  $\leq 2$  e.g.

`previous_prime(2) = 0`.

Examples:

[previous\\_prime.c](#).

**8.6.2.5** `void primesieve_skipto ( primesieve_iterator * pi, uint64_t start, uint64_t stop_hint )`

Set the primesieve iterator to start.

Parameters

<i>start</i>	Generate primes $>$ start (or $<$ start).
<i>stop_hint</i>	Stop number optimization hint. E.g. if you want to generate the primes below 1000 use <code>stop_hint = 1000</code> , if you don't know use <a href="#">primesieve_get_max_stop()</a> .

Precondition

$start \leq 2^{64} - 2^{32} * 10$

Examples:

[previous\\_prime.c](#).

## Chapter 9

# Example Documentation

### 9.1 callback\_primes.cpp

This example shows how to use callback functions.

```
#include <primesieve.hpp>
#include <stdint.h>
#include <iostream>

void callback(uint64_t prime)
{
    std::cout << prime << std::endl;
}

int main()
{
    primesieve::callback_primes(2, 1000, callback);
    return 0;
}
```

### 9.2 count\_primes.c

C program that shows how to count primes.

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

int main()
{
    uint64_t count = primesieve_count_primes(0, 1000);
    printf("Primes below 1000 = %" PRIu64 "\n", count);

    /* use multi-threading for large intervals */
    count = primesieve_parallel_count_primes(0, 1000000000);
    printf("Primes below 10^9 = %" PRIu64 "\n", count);

    return 0;
}
```

## 9.3 count\_primes.cpp

This example shows how to count primes.

```
#include <primesieve.hpp>
#include <stdint.h>
#include <iostream>

int main()
{
    uint64_t count = primesieve::count_primes(0, 1000);
    std::cout << "Primes below 1000 = " << count << std::endl;

    uint64_t stop = 1000000000;

    // use multi-threading for large intervals
    count = primesieve::parallel_count_primes(0, stop);
    std::cout << "Primes below 10^9 = " << count << std::endl;

    return 0;
}
```

## 9.4 nth\_prime.c

C program that finds the nth prime.

```
#include <primesieve.h>
#include <stdlib.h>
#include <inttypes.h>
#include <stdio.h>

int main(int argc, char** argv)
{
    uint64_t n = 1000;
    if (argv[1])
        n = atol(argv[1]);

    uint64_t prime = primesieve_nth_prime(n, 0);
    printf("%" PRIu64 "th prime = %" PRIu64 "\n", n, prime);

    return 0;
}
```

## 9.5 nth\_prime.cpp

Find the nth prime.

```
#include <primesieve.hpp>
#include <stdint.h>
#include <iostream>
#include <cstdlib>

int main(int, char** argv)
{
    uint64_t n = 1000;
    if (argv[1])
        n = std::atol(argv[1]);

    uint64_t nth_prime = primesieve::nth_prime(n);
    std::cout << n << "th prime = " << nth_prime << std::endl;

    return 0;
}
```

## 9.6 previous\_prime.c

Iterate backwards over primes using [primesieve\\_iterator](#).

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

int main()
{
    primesieve_iterator pi;
    primesieve_init(&pi);

    /* primesieve_skipto(primesieve_iterator, start_number, stop_hint) */
    primesieve_skipto(&pi, 2000, 1000);
    uint64_t prime;

    /* iterate backwards over the primes between 2000 and 1000 */
    while ((prime = primesieve_previous_prime(&pi)) >= 1000)
        printf("%" PRIu64 "\n", prime);

    primesieve_free_iterator(&pi);
    return 0;
}
```

## 9.7 previous\_prime.cpp

This example shows how to iterate backwards over primes.

```
#include <primesieve.hpp>
#include <iostream>

int main()
{
    primesieve::iterator pi;
    pi.skipto(2000);

    uint64_t prime;

    // iterate backwards over the primes between 2000 and 1000
    while ((prime = pi.previous_prime()) >= 1000)
        std::cout << prime << std::endl;

    return 0;
}
```

## 9.8 primesieve\_iterator.c

Iterate over primes using C [primesieve\\_iterator](#).

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

int main()
{
    primesieve_iterator pi;
    primesieve_init(&pi);

    uint64_t sum = 0;
    uint64_t prime = 0;

    /* iterate over the primes below 10^10 */
    while ((prime = primesieve_next_prime(&pi)) < 10000000000ull)
        sum += prime;

    primesieve_free_iterator(&pi);
    printf("Sum of the primes below 10^10 = %" PRIu64 "\n", sum);
    return 0;
}
```

## 9.9 primesieve\_iterator.cpp

Iterate over primes using a `primesieve::iterator` object.

```
#include <primesieve.hpp>
#include <iostream>

int main()
{
    primesieve::iterator pi;
    uint64_t sum = 0;
    uint64_t prime;

    // iterate over primes below 10^10
    while ((prime = pi.next_prime()) < 10000000000ull)
        sum += prime;

    std::cout << "Sum of the primes below 10^10 = " << sum << std::endl;
    return 0;
}
```

## 9.10 store\_primes\_in\_array.c

Store primes in a C array.

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

int main()
{
    uint64_t start = 0;
    uint64_t stop = 1000;
    size_t i;
    size_t size;

    /* store the primes below 1000 */
    int* primes = (int*) primesieve_generate_primes(start, stop, &size,
        INT_PRIMES);

    for (i = 0; i < size; i++)
        printf("%i\n", primes[i]);

    primesieve_free(primes);
    uint64_t n = 1000;

    /* store the first 1000 primes */
    primes = (int*) primesieve_generate_n_primes(n, start,
        INT_PRIMES);

    for (i = 0; i < n; i++)
        printf("%i\n", primes[i]);

    primesieve_free(primes);
    return 0;
}
```

## 9.11 store\_primes\_in\_vector.cpp

Store primes in a `std::vector` using `primesieve`.

```
#include <primesieve.hpp>
#include <vector>

int main()
{
    std::vector<int> primes;

    // Store the primes <= 1000
    primesieve::generate_primes(1000, &primes);

    primes.clear();

    // Store the first 1000 primes
    primesieve::generate_n_primes(1000, &primes);

    return 0;
}
```



# Index

Callback.hpp, [27](#)  
callback\_primes  
    primesieve, [13](#)  
count\_primes  
    primesieve, [14](#)  
count\_quadruplets  
    primesieve, [14](#)  
count\_quintuplets  
    primesieve, [14](#)  
count\_sextuplets  
    primesieve, [14](#)  
count\_triplets  
    primesieve, [14](#)  
count\_twins  
    primesieve, [14](#)  
  
generate\_n\_primes  
    primesieve, [15](#)  
generate\_primes  
    primesieve, [15](#)  
get\_max\_stop  
    primesieve, [15](#)  
get\_num\_threads  
    primesieve, [15](#)  
  
INT16\_PRIMES  
    primesieve.h, [32](#)  
INT32\_PRIMES  
    primesieve.h, [32](#)  
INT64\_PRIMES  
    primesieve.h, [32](#)  
INT\_PRIMES  
    primesieve.h, [32](#)  
iterator  
    primesieve::iterator, [22](#)  
iterator.hpp, [28](#)  
  
LONG\_PRIMES  
    primesieve.h, [32](#)  
LONGLONG\_PRIMES  
    primesieve.h, [32](#)  
  
MAX\_THREADS  
    primesieve, [13](#)  
    primesieve.h, [32](#)  
  
next\_prime  
    primesieve::iterator, [22](#)  
nth\_prime  
    primesieve, [15](#)  
  
parallel\_count\_primes  
    primesieve, [16](#)  
parallel\_count\_quadruplets  
    primesieve, [16](#)  
parallel\_count\_quintuplets  
    primesieve, [16](#)  
parallel\_count\_sextuplets  
    primesieve, [16](#)  
parallel\_count\_triplets  
    primesieve, [17](#)  
parallel\_count\_twins  
    primesieve, [17](#)  
parallel\_nth\_prime  
    primesieve, [17](#)  
previous\_prime  
    primesieve::iterator, [23](#)  
primesieve, [11](#)  
    callback\_primes, [13](#)  
    count\_primes, [14](#)  
    count\_quadruplets, [14](#)  
    count\_quintuplets, [14](#)  
    count\_sextuplets, [14](#)  
    count\_triplets, [14](#)  
    count\_twins, [14](#)  
    generate\_n\_primes, [15](#)  
    generate\_primes, [15](#)  
    get\_max\_stop, [15](#)  
    get\_num\_threads, [15](#)  
    MAX\_THREADS, [13](#)  
    nth\_prime, [15](#)  
    parallel\_count\_primes, [16](#)  
    parallel\_count\_quadruplets, [16](#)  
    parallel\_count\_quintuplets, [16](#)  
    parallel\_count\_sextuplets, [16](#)  
    parallel\_count\_triplets, [17](#)  
    parallel\_count\_twins, [17](#)  
    parallel\_nth\_prime, [17](#)  
    primesieve\_test, [17](#)  
    print\_primes, [18](#)  
    print\_quadruplets, [18](#)  
    print\_quintuplets, [18](#)  
    print\_sextuplets, [18](#)  
    print\_triplets, [18](#)  
    print\_twins, [19](#)  
    set\_num\_threads, [19](#)  
    set\_sieve\_size, [19](#)  
primesieve.h, [29](#)  
    INT16\_PRIMES, [32](#)  
    INT32\_PRIMES, [32](#)

- INT64\_PRIMES, [32](#)
- INT\_PRIMES, [32](#)
- LONG\_PRIMES, [32](#)
- LONGLONG\_PRIMES, [32](#)
- MAX\_THREADS, [32](#)
- primesieve\_callback\_primes, [32](#)
- primesieve\_count\_primes, [32](#)
- primesieve\_count\_quadruplets, [32](#)
- primesieve\_count\_quintuplets, [33](#)
- primesieve\_count\_sextuplets, [33](#)
- primesieve\_count\_triplets, [33](#)
- primesieve\_count\_twins, [33](#)
- primesieve\_generate\_n\_primes, [33](#)
- primesieve\_generate\_primes, [34](#)
- primesieve\_get\_max\_stop, [34](#)
- primesieve\_get\_num\_threads, [34](#)
- primesieve\_get\_sieve\_size, [34](#)
- primesieve\_nth\_prime, [34](#)
- primesieve\_parallel\_count\_primes, [35](#)
- primesieve\_parallel\_count\_quadruplets, [35](#)
- primesieve\_parallel\_count\_quintuplets, [35](#)
- primesieve\_parallel\_count\_sextuplets, [35](#)
- primesieve\_parallel\_count\_triplets, [36](#)
- primesieve\_parallel\_count\_twins, [36](#)
- primesieve\_parallel\_nth\_prime, [36](#)
- primesieve\_print\_primes, [36](#)
- primesieve\_print\_quadruplets, [37](#)
- primesieve\_print\_quintuplets, [37](#)
- primesieve\_print\_sextuplets, [37](#)
- primesieve\_print\_triplets, [37](#)
- primesieve\_print\_twins, [37](#)
- primesieve\_set\_num\_threads, [37](#)
- primesieve\_set\_sieve\_size, [38](#)
- primesieve\_test, [38](#)
- primesieve\_version, [38](#)
- SHORT\_PRIMES, [32](#)
- UINT16\_PRIMES, [32](#)
- UINT32\_PRIMES, [32](#)
- UINT64\_PRIMES, [32](#)
- UINT\_PRIMES, [32](#)
- ULONG\_PRIMES, [32](#)
- ULONGLONG\_PRIMES, [32](#)
- USHORT\_PRIMES, [32](#)
- primesieve.hpp, [38](#)
- primesieve::Callback< T >, [21](#)
- primesieve::iterator, [21](#)
  - iterator, [22](#)
  - next\_prime, [22](#)
  - previous\_prime, [23](#)
  - skipto, [23](#)
- primesieve::primesieve\_error, [23](#)
- primesieve\_callback\_primes
  - primesieve.h, [32](#)
- primesieve\_count\_primes
  - primesieve.h, [32](#)
- primesieve\_count\_quadruplets
  - primesieve.h, [32](#)
- primesieve\_count\_sextuplets
  - primesieve.h, [32](#)
- primesieve\_count\_triplets
  - primesieve.h, [32](#)
- primesieve\_count\_twins
  - primesieve.h, [32](#)
- primesieve\_error.hpp, [41](#)
- primesieve\_free\_iterator
  - primesieve\_iterator.h, [43](#)
- primesieve\_generate\_n\_primes
  - primesieve.h, [33](#)
- primesieve\_generate\_primes
  - primesieve.h, [34](#)
- primesieve\_get\_max\_stop
  - primesieve.h, [34](#)
- primesieve\_get\_num\_threads
  - primesieve.h, [34](#)
- primesieve\_get\_sieve\_size
  - primesieve.h, [34](#)
- primesieve\_init
  - primesieve\_iterator.h, [43](#)
- primesieve\_iterator, [25](#)
- primesieve\_iterator.h, [42](#)
  - primesieve\_free\_iterator, [43](#)
  - primesieve\_init, [43](#)
  - primesieve\_next\_prime, [43](#)
  - primesieve\_previous\_prime, [44](#)
  - primesieve\_skipto, [44](#)
- primesieve\_next\_prime
  - primesieve\_iterator.h, [43](#)
- primesieve\_nth\_prime
  - primesieve.h, [34](#)
- primesieve\_parallel\_count\_primes
  - primesieve.h, [35](#)
- primesieve\_parallel\_count\_quadruplets
  - primesieve.h, [35](#)
- primesieve\_parallel\_count\_quintuplets
  - primesieve.h, [35](#)
- primesieve\_parallel\_count\_sextuplets
  - primesieve.h, [35](#)
- primesieve\_parallel\_count\_triplets
  - primesieve.h, [36](#)
- primesieve\_parallel\_count\_twins
  - primesieve.h, [36](#)
- primesieve\_parallel\_nth\_prime
  - primesieve.h, [36](#)
- primesieve\_previous\_prime
  - primesieve\_iterator.h, [44](#)
- primesieve\_print\_primes
  - primesieve.h, [36](#)
- primesieve\_print\_quadruplets
  - primesieve.h, [37](#)
- primesieve\_print\_quintuplets
  - primesieve.h, [37](#)
- primesieve\_print\_sextuplets
  - primesieve.h, [37](#)
- primesieve\_print\_triplets
  - primesieve.h, [37](#)

- primesieve.h, [37](#)
- primesieve\_print\_twins
  - primesieve.h, [37](#)
- primesieve\_set\_num\_threads
  - primesieve.h, [37](#)
- primesieve\_set\_sieve\_size
  - primesieve.h, [38](#)
- primesieve\_skipto
  - primesieve\_iterator.h, [44](#)
- primesieve\_test
  - primesieve, [17](#)
  - primesieve.h, [38](#)
- primesieve\_version
  - primesieve.h, [38](#)
- print\_primes
  - primesieve, [18](#)
- print\_quadruplets
  - primesieve, [18](#)
- print\_quintuplets
  - primesieve, [18](#)
- print\_sextuplets
  - primesieve, [18](#)
- print\_triplets
  - primesieve, [18](#)
- print\_twins
  - primesieve, [19](#)
- SHORT\_PRIMES
  - primesieve.h, [32](#)
- set\_num\_threads
  - primesieve, [19](#)
- set\_sieve\_size
  - primesieve, [19](#)
- skipto
  - primesieve::iterator, [23](#)
- UINT16\_PRIMES
  - primesieve.h, [32](#)
- UINT32\_PRIMES
  - primesieve.h, [32](#)
- UINT64\_PRIMES
  - primesieve.h, [32](#)
- UINT\_PRIMES
  - primesieve.h, [32](#)
- ULONG\_PRIMES
  - primesieve.h, [32](#)
- ULONGLONG\_PRIMES
  - primesieve.h, [32](#)
- USHORT\_PRIMES
  - primesieve.h, [32](#)