

primesieve

7.5

Generated by Doxygen 1.8.16

| | |
|--|-----------|
| 1 Main Page | 1 |
| 1.1 About | 1 |
| 1.2 C++ API | 1 |
| 1.3 C API | 1 |
| 2 Namespace Index | 3 |
| 2.1 Namespace List | 3 |
| 3 Hierarchical Index | 5 |
| 3.1 Class Hierarchy | 5 |
| 4 Class Index | 7 |
| 4.1 Class List | 7 |
| 5 File Index | 9 |
| 5.1 File List | 9 |
| 6 Namespace Documentation | 11 |
| 6.1 primesieve Namespace Reference | 11 |
| 6.1.1 Detailed Description | 12 |
| 6.1.2 Function Documentation | 12 |
| 6.1.2.1 count_primes() | 12 |
| 6.1.2.2 count_quadruplets() | 13 |
| 6.1.2.3 count_quintuplets() | 13 |
| 6.1.2.4 count_sextuplets() | 13 |
| 6.1.2.5 count_triplets() | 13 |
| 6.1.2.6 count_twins() | 13 |
| 6.1.2.7 get_max_stop() | 14 |
| 6.1.2.8 nth_prime() | 14 |
| 6.1.2.9 set_num_threads() | 14 |
| 6.1.2.10 set_sieve_size() | 15 |
| 7 Class Documentation | 17 |
| 7.1 primesieve::iterator Class Reference | 17 |
| 7.1.1 Detailed Description | 17 |
| 7.1.2 Constructor & Destructor Documentation | 18 |
| 7.1.2.1 iterator() | 18 |
| 7.1.3 Member Function Documentation | 18 |
| 7.1.3.1 next_prime() | 18 |
| 7.1.3.2 prev_prime() | 18 |
| 7.1.3.3 skipto() | 19 |
| 7.2 primesieve::primesieve_error Class Reference | 20 |
| 7.2.1 Detailed Description | 21 |
| 7.3 primesieve_iterator Struct Reference | 21 |

| | |
|---|-----------|
| 7.3.1 Detailed Description | 21 |
| 8 File Documentation | 23 |
| 8.1 iterator.h File Reference | 23 |
| 8.1.1 Detailed Description | 24 |
| 8.1.2 Function Documentation | 24 |
| 8.1.2.1 primesieve_next_prime() | 24 |
| 8.1.2.2 primesieve_prev_prime() | 25 |
| 8.1.2.3 primesieve_skipto() | 25 |
| 8.2 iterator.hpp File Reference | 25 |
| 8.2.1 Detailed Description | 27 |
| 8.3 primesieve.h File Reference | 27 |
| 8.3.1 Detailed Description | 29 |
| 8.3.2 Enumeration Type Documentation | 29 |
| 8.3.2.1 anonymous enum | 29 |
| 8.3.3 Function Documentation | 29 |
| 8.3.3.1 primesieve_count_primes() | 29 |
| 8.3.3.2 primesieve_count_quadruplets() | 30 |
| 8.3.3.3 primesieve_count_quintuplets() | 30 |
| 8.3.3.4 primesieve_count_sextuplets() | 30 |
| 8.3.3.5 primesieve_count_triplets() | 30 |
| 8.3.3.6 primesieve_count_twins() | 31 |
| 8.3.3.7 primesieve_generate_n_primes() | 31 |
| 8.3.3.8 primesieve_generate_primes() | 31 |
| 8.3.3.9 primesieve_get_max_stop() | 32 |
| 8.3.3.10 primesieve_nth_prime() | 32 |
| 8.3.3.11 primesieve_set_num_threads() | 32 |
| 8.3.3.12 primesieve_set_sieve_size() | 33 |
| 8.4 primesieve.hpp File Reference | 33 |
| 8.4.1 Detailed Description | 35 |
| 8.5 primesieve_error.hpp File Reference | 35 |
| 8.5.1 Detailed Description | 36 |
| 9 Example Documentation | 37 |
| 9.1 count_primes.c | 37 |
| 9.2 count_primes.cpp | 37 |
| 9.3 nth_prime.c | 37 |
| 9.4 nth_prime.cpp | 38 |
| 9.5 prev_prime.c | 38 |
| 9.6 prev_prime.cpp | 38 |
| 9.7 primesieve_iterator.c | 38 |
| 9.8 primesieve_iterator.cpp | 39 |
| 9.9 store_primes_in_array.c | 39 |

| | |
|---|-----------|
| 9.10 store_primes_in_vector.cpp | 39 |
| Index | 41 |

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

For more information please visit <https://github.com/kimwalisch/primesieve>.

1.2 C++ API

- [primesieve.hpp](#) - primesieve C++ header.
- [primesieve_iterator.cpp](#) - Example that shows how to iterate over primes using `primesieve::iterator`.
- [store_primes_in_vector.cpp](#) - Example that shows how to store primes in a `std::vector`.
- [count_primes.cpp](#) - Example that shows how to count primes.

1.3 C API

- [primesieve.h](#) - primesieve C header.
- [primesieve_iterator.c](#) - Example that shows how to iterate over primes using `primesieve_iterator`.
- [store_primes_in_array.c](#) - Example that shows how to store primes in an array.
- [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 | |
| Contains primesieve's C++ functions and classes | 11 |

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

| | |
|--|----|
| primesieve::iterator | 17 |
| primesieve_iterator | 21 |
| runtime_error | |
| primesieve::primesieve_error | 20 |

Chapter 4

Class Index

4.1 Class List

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

| | | |
|--|---|----|
| primesieve::iterator | Primesieve::iterator allows to easily iterate over primes both forwards and backwards | 17 |
| primesieve::primesieve_error | Primesieve throws a primesieve_error exception if an error occurs e.g | 20 |
| primesieve_iterator | C prime iterator, please refer to iterator.h for more information | 21 |

Chapter 5

File Index

5.1 File List

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

| | | |
|--------------------------------------|---|----|
| iterator.h | 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 $\text{PrimePi}(n^{0.5}) * 8$ bytes | 23 |
| iterator.hpp | The iterator class allows to easily iterate (forwards and backwards) over prime numbers | 25 |
| primesieve.h | 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 | 27 |
| primesieve.hpp | Primesieve C++ API. primesieve is a library for fast prime number generation, in case an error occurs a primesieve::primesieve_error exception (derived from <code>std::runtime_error</code>) is thrown . . | 33 |
| primesieve_error.hpp | The primesieve_error class is used for all exceptions within primesieve | 35 |

Chapter 6

Namespace Documentation

6.1 primesieve Namespace Reference

Contains primesieve's C++ functions and classes.

Classes

- 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 e.g.

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 (uint64_t n, uint64_t start=0)`
Find the nth prime.
- `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].
 - `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.
 - `uint64_t get_max_stop ()`
Returns the largest valid stop number for primesieve.
 - `int get_sieve_size ()`
Get the current set sieve size in KiB.
 - `int get_num_threads ()`
Get the current set number of threads.
 - `void set_sieve_size (int sieve_size)`
Set the sieve size in KiB (kibibyte).
 - `void set_num_threads (int num_threads)`
Set the number of threads for use in primesieve::count_() and primesieve::nth_prime().*
 - `std::string primesieve_version ()`
Get the primesieve version number, in the form "i.j".

6.1.1 Detailed Description

Contains primesieve's C++ functions and classes.

6.1.2 Function Documentation

6.1.2.1 count_primes()

```
uint64_t primesieve::count_primes (
    uint64_t start,
    uint64_t stop )
```

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

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

Note that each call to `count_primes()` incurs an initialization overhead of $O(\sqrt{\text{stop}})$ even if the interval [start, stop] is tiny. Hence if you have written an algorithm that makes many calls to `count_primes()` it may be preferable to use a `primesieve::iterator` which needs to be initialized only once.

Examples

`count_primes.cpp`.

6.1.2.2 count_quadruplets()

```
uint64_t primesieve::count_quadruplets (
    uint64_t start,
    uint64_t stop )
```

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

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

6.1.2.3 count_quintuplets()

```
uint64_t primesieve::count_quintuplets (
    uint64_t start,
    uint64_t stop )
```

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

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

6.1.2.4 count_sextuplets()

```
uint64_t primesieve::count_sextuplets (
    uint64_t start,
    uint64_t stop )
```

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

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

6.1.2.5 count_triplets()

```
uint64_t primesieve::count_triplets (
    uint64_t start,
    uint64_t stop )
```

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

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

6.1.2.6 count_twins()

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

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

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

6.1.2.7 `get_max_stop()`

```
uint64_t primesieve::get_max_stop ( )
```

Returns the largest valid stop number for primesieve.

Returns

$2^{64}-1$ (UINT64_MAX).

6.1.2.8 `nth_prime()`

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

Find the nth prime.

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

Note that each call to `nth_prime(n, start)` incurs an initialization overhead of $O(\sqrt{\text{start}})$ even if `n` is tiny. Hence it is not a good idea to use [nth_prime\(\)](#) repeatedly in a loop to get the next (or previous) prime. For this use case it is better to use a [primesieve::iterator](#) which needs to be initialized only once.

Parameters

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

Examples

[nth_prime.cpp](#).

6.1.2.9 `set_num_threads()`

```
void primesieve::set_num_threads (
    int num_threads )
```

Set the number of threads for use in `primesieve::count_*`() and [primesieve::nth_prime\(\)](#).

By default all CPU cores are used.

6.1.2.10 set_sieve_size()

```
void primesieve::set_sieve_size (
    int sieve_size )
```

Set the sieve size in KiB (kibibyte).

The best sieving performance is achieved with a sieve size of your CPU's L1 or L2 cache size (per core).

Precondition

sieve_size >= 8 && <= 4096.

Chapter 7

Class Documentation

7.1 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.
- [iterator](#) (const [iterator](#) &)=delete
[primesieve::iterator](#) objects cannot be copied.
- [iterator](#) & **operator=** (const [iterator](#) &)=delete
- [iterator](#) ([iterator](#) &&) noexcept
[primesieve::iterator](#) objects support move semantics.
- [iterator](#) & **operator=** ([iterator](#) &&) noexcept
- void [skipto](#) (uint64_t start, uint64_t stop_hint=[get_max_stop](#)())
Reset the primesieve iterator to start.
- uint64_t [next_prime](#) ()
Get the next prime.
- uint64_t [prev_prime](#) ()
Get the previous prime.

7.1.1 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 $\text{PrimePi}(n^{0.5}) * 8$ bytes.

Examples

[prev_prime.cpp](#), and [primesieve_iterator.cpp](#).

7.1.2 Constructor & Destructor Documentation

7.1.2.1 iterator()

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

Create a new iterator object.

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. |

7.1.3 Member Function Documentation

7.1.3.1 next_prime()

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

Get the next prime.

Returns UINT64_MAX if next prime > 2⁶⁴.

Examples

[primesieve_iterator.cpp](#).

7.1.3.2 prev_prime()

```
uint64_t primesieve::iterator::prev_prime ( ) [inline]
```

Get the previous prime.

prev_prime(n) returns 0 for n ≤ 2. Note that [next_prime\(\)](#) runs up to 2x faster than [prev_prime\(\)](#). Hence if the same algorithm can be written using either [prev_prime\(\)](#) or [next_prime\(\)](#) it is preferable to use [next_prime\(\)](#).

Examples

[prev_prime.cpp](#).

7.1.3.3 skipto()

```
void primesieve::iterator::skipto (
    uint64_t start,
    uint64_t stop_hint = get\_max\_stop\(\) )
```

Reset the primesieve iterator 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. |

Examples

[prev_prime.cpp](#), and [primesieve_iterator.cpp](#).

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

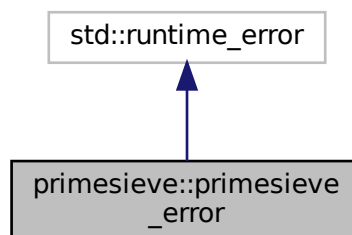
- [iterator.hpp](#)

7.2 primesieve::primesieve_error Class Reference

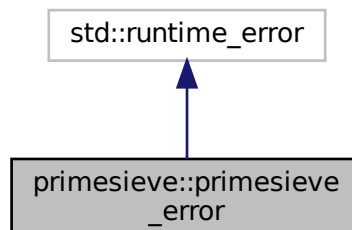
primesieve throws a [primesieve_error](#) exception if an error occurs e.g.

```
#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)

7.2.1 Detailed Description

primesieve throws a [primesieve_error](#) exception if an error occurs e.g.

prime > 2⁶⁴.

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

- [primesieve_error.hpp](#)

7.3 primesieve_iterator Struct Reference

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

```
#include <iterator.h>
```

Public Attributes

- size_t **i**
- size_t **last_idx**
- uint64_t **start**
- uint64_t **stop**
- uint64_t **stop_hint**
- uint64_t **dist**
- uint64_t * **primes**
- void * **vector**
- void * **primeGenerator**
- int **is_error**

7.3.1 Detailed Description

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

Examples

[prev_prime.c](#), and [primesieve_iterator.c](#).

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

- [iterator.h](#)

Chapter 8

File Documentation

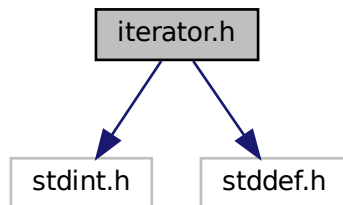
8.1 iterator.h File Reference

[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 $\text{PrimePi}(n^{0.5}) * 8$ bytes.

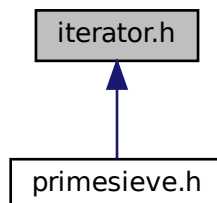
```
#include <stdint.h>
```

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

Include dependency graph for iterator.h:



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



Classes

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

Functions

- void [primesieve_init](#) ([primesieve_iterator](#) *it)
Initialize the primesieve iterator before first using it.
- void [primesieve_free_iterator](#) ([primesieve_iterator](#) *it)
Free all memory.
- void [primesieve_skipto](#) ([primesieve_iterator](#) *it, uint64_t start, uint64_t stop_hint)
Reset the primesieve iterator to start.
- static uint64_t [primesieve_next_prime](#) ([primesieve_iterator](#) *it)
Get the next prime.
- static uint64_t [primesieve_prev_prime](#) ([primesieve_iterator](#) *it)
Get the previous prime.

8.1.1 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 $\text{PrimePi}(n^{0.5}) * 8$ bytes.

The [primesieve_iterator.c](#) example shows how to use [primesieve_iterator](#). If any error occurs [primesieve_next_prime\(\)](#) and [primesieve_prev_prime\(\)](#) return PRIMESIEVE_ERROR. Furthermore [primesieve_iterator.is_error](#) is initialized to 0 and set to 1 if any error occurs.

Copyright (C) 2019 Kim Walisch, kim.walisch@gmail.com

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

8.1.2 Function Documentation

8.1.2.1 [primesieve_next_prime\(\)](#)

```
static uint64_t primesieve_next_prime (
    primesieve\_iterator * it ) [inline], [static]
```

Get the next prime.

Returns UINT64_MAX if next prime $> 2^{64}$.

Examples

[primesieve_iterator.c](#).

8.1.2.2 primesieve_prev_prime()

```
static uint64_t primesieve_prev_prime (
    primesieve_iterator * it ) [inline], [static]
```

Get the previous prime.

primesieve_prev_prime(n) returns 0 for $n \leq 2$. Note that [primesieve_next_prime\(\)](#) runs up to 2x faster than [primesieve_prev_prime\(\)](#). Hence if the same algorithm can be written using either [primesieve_prev_prime\(\)](#) or [primesieve_next_prime\(\)](#) it is preferable to use [primesieve_next_prime\(\)](#).

Examples

[prev_prime.c](#).

8.1.2.3 primesieve_skipto()

```
void primesieve_skipto (
    primesieve_iterator * it,
    uint64_t start,
    uint64_t stop_hint )
```

Reset 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 primesieve_get_max_stop() . |

Examples

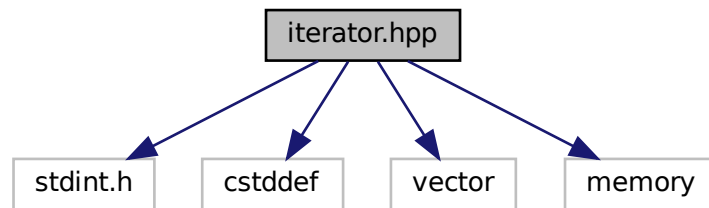
[prev_prime.c](#), and [primesieve_iterator.c](#).

8.2 iterator.hpp File Reference

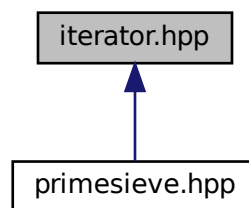
The iterator class allows to easily iterate (forwards and backwards) over prime numbers.

```
#include <stdint.h>
#include <cstdint>
#include <vector>
#include <memory>
```

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](#)
Contains primesieve's C++ functions and classes.

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 (forwards and backwards) over prime numbers.

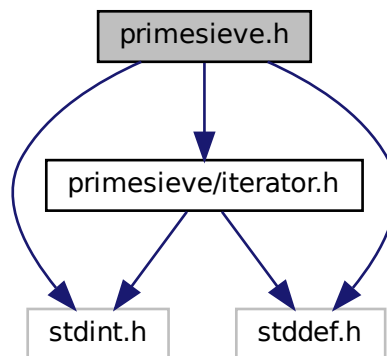
Copyright (C) 2019 Kim Walisch, kim.walisch@gmail.com

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

```
#include <primesieve/iterator.h>
#include <stdint.h>
#include <stddef.h>
Include dependency graph for primesieve.h:
```



Macros

- `#define PRIMESIEVE_VERSION "7.5"`
- `#define PRIMESIEVE_VERSION_MAJOR 7`
- `#define PRIMESIEVE_VERSION_MINOR 5`
- `#define PRIMESIEVE_ERROR ((uint64_t) ~((uint64_t) 0))`
primesieve functions return PRIMESIEVE_ERROR (UINT64_MAX) if any error occurs.

Enumerations

- `enum {`
`SHORT_PRIMES, USHORT_PRIMES, INT_PRIMES, UINT_PRIMES,`
`LONG_PRIMES, ULONG_PRIMES, LONGLONG_PRIMES, ULLONGLONG_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_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].
- 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.
- uint64_t [primesieve_get_max_stop](#) ()
Returns the largest valid stop number for primesieve.
- int [primesieve_get_sieve_size](#) ()
Get the current set sieve size in KiB.
- int [primesieve_get_num_threads](#) ()
Get the current set number of threads.
- void [primesieve_set_sieve_size](#) (int sieve_size)
Set the sieve size in KiB (kibibyte).
- void [primesieve_set_num_threads](#) (int num_threads)
Set the number of threads for use in [primesieve_count_\(\)](#) and [primesieve_nth_prime\(\)](#).*
- void [primesieve_free](#) (void *primes)
Deallocate a primes array created using the [primesieve_generate_primes\(\)](#) or [primesieve_generate_n_primes\(\)](#) functions.
- const char * [primesieve_version](#) ()
Get the primesieve version number, in the form "i.j"

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) 2019 Kim Walisch, kim.walisch@gmail.com

This file is distributed under the BSD License.

8.3.2 Enumeration Type Documentation

8.3.2.1 anonymous enum

anonymous enum

Enumerator

| | |
|------------------|---|
| 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 primesieve_count_primes()

```
uint64_t primesieve_count_primes (  
    uint64_t start,  
    uint64_t stop )
```

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

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

Note that each call to [primesieve_count_primes\(\)](#) incurs an initialization overhead of $O(\sqrt{\text{stop}})$ even if the interval $[\text{start}, \text{stop}]$ is tiny. Hence if you have written an algorithm that makes many calls to [primesieve_count_primes\(\)](#) it may be preferable to use a [primesieve::iterator](#) which needs to be initialized only once.

Examples

[count_primes.c](#).

8.3.3.2 primesieve_count_quadruplets()

```
uint64_t primesieve_count_quadruplets (
    uint64_t start,
    uint64_t stop )
```

Count the prime quadruplets within the interval $[\text{start}, \text{stop}]$.

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

8.3.3.3 primesieve_count_quintuplets()

```
uint64_t primesieve_count_quintuplets (
    uint64_t start,
    uint64_t stop )
```

Count the prime quintuplets within the interval $[\text{start}, \text{stop}]$.

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

8.3.3.4 primesieve_count_sextuplets()

```
uint64_t primesieve_count_sextuplets (
    uint64_t start,
    uint64_t stop )
```

Count the prime sextuplets within the interval $[\text{start}, \text{stop}]$.

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

8.3.3.5 primesieve_count_triplets()

```
uint64_t primesieve_count_triplets (
    uint64_t start,
    uint64_t stop )
```

Count the prime triplets within the interval $[\text{start}, \text{stop}]$.

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

8.3.3.6 primesieve_count_twins()

```
uint64_t primesieve_count_twins (
    uint64_t start,
    uint64_t stop )
```

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

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

8.3.3.7 primesieve_generate_n_primes()

```
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. |
|-------------|--|

Examples

[store_primes_in_array.c](#).

8.3.3.8 primesieve_generate_primes()

```
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. |

Examples

[store_primes_in_array.c](#).

8.3.3.9 primesieve_get_max_stop()

```
uint64_t primesieve_get_max_stop ( )
```

Returns the largest valid stop number for primesieve.

Returns

$2^{64}-1$ (UINT64_MAX).

8.3.3.10 primesieve_nth_prime()

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

Find the nth prime.

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

Note that each call to `primesieve_nth_prime(n, start)` incurs an initialization overhead of $O(\sqrt{\text{start}})$ even if `n` is tiny. Hence it is not a good idea to use [primesieve_nth_prime\(\)](#) repeatedly in a loop to get the next (or previous) prime. For this use case it is better to use a [primesieve::iterator](#) which needs to be initialized only once.

Parameters

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

Examples

[nth_prime.c](#).

8.3.3.11 primesieve_set_num_threads()

```
void primesieve_set_num_threads (
    int num_threads )
```

Set the number of threads for use in `primesieve_count_*`() and [primesieve_nth_prime\(\)](#).

By default all CPU cores are used.

8.3.3.12 primesieve_set_sieve_size()

```
void primesieve_set_sieve_size (
    int sieve_size )
```

Set the sieve size in KiB (kibibyte).

The best sieving performance is achieved with a sieve size of your CPU's L1 or L2 cache size (per core).

Precondition

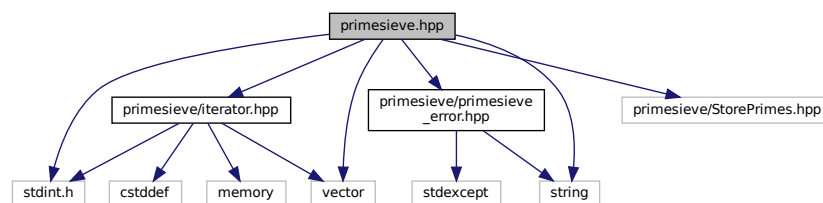
sieve_size >= 8 && <= 4096.

8.4 primesieve.hpp File Reference

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`) is thrown.

```
#include <primesieve/iterator.hpp>
#include <primesieve/primesieve_error.hpp>
#include <primesieve/StorePrimes.hpp>
#include <stdint.h>
#include <vector>
#include <string>
```

Include dependency graph for primesieve.hpp:



Namespaces

- [primesieve](#)

Contains primesieve's C++ functions and classes.

Macros

- `#define PRIMESIEVE_VERSION "7.5"`
- `#define PRIMESIEVE_VERSION_MAJOR 7`
- `#define PRIMESIEVE_VERSION_MINOR 5`

Functions

- `template<typename T >`
`void primesieve::generate_primes (uint64_t stop, std::vector< T > *primes)`
Store the primes \leq 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 \geq start in the primes vector.
- `uint64_t primesieve::nth_prime (uint64_t n, uint64_t start=0)`
Find the nth prime.
- `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].
- `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.
- `uint64_t primesieve::get_max_stop ()`
Returns the largest valid stop number for primesieve.
- `int primesieve::get_sieve_size ()`
Get the current set sieve size in KiB.
- `int primesieve::get_num_threads ()`
Get the current set number of threads.
- `void primesieve::set_sieve_size (int sieve_size)`
Set the sieve size in KiB (kibibyte).
- `void primesieve::set_num_threads (int num_threads)`
Set the number of threads for use in primesieve::count_() and primesieve::nth_prime().*
- `std::string primesieve::primesieve_version ()`
Get the primesieve version number, in the form "i.j".

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`) is thrown.

Copyright (C) 2019 Kim Walisch, kim.walisch@gmail.com

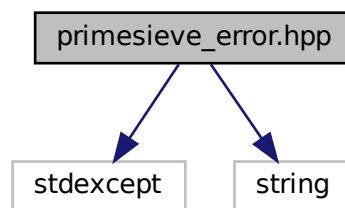
This file is distributed under the BSD License.

8.5 primesieve_error.hpp File Reference

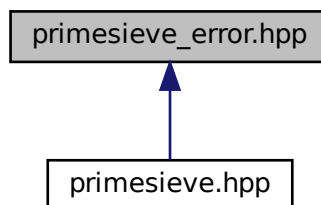
The `primesieve_error` class is used for all exceptions within primesieve.

```
#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 e.g.

Namespaces

- [primesieve](#)

Contains primesieve's C++ functions and classes.

8.5.1 Detailed Description

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

Copyright (C) 2017 Kim Walisch, kim.walisch@gmail.com

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

Chapter 9

Example Documentation

9.1 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);
    return 0;
}
```

9.2 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;
    return 0;
}
```

9.3 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 (argc > 1 && argv[1])
        n = atol(argv[1]);
    uint64_t prime = primesieve_nth_prime(n, 0);
    printf("%" PRIu64 "th prime = %" PRIu64 "\n", n, prime);
    return 0;
}
```

9.4 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;
}
```

9.5 prev_prime.c

Iterate backwards over primes using `primesieve_iterator`. Note that `primesieve_next_prime()` runs up to 2x faster and uses only half as much memory as `primesieve_prev_prime()`. Hence if it is possible to write the same algorithm using either `primesieve_prev_prime()` or `primesieve_next_prime()` then it is preferable to use `primesieve_next_prime()`.

```
#include <primesieve.h>
#include <inttypes.h>
#include <stdio.h>
int main()
{
    primesieve_iterator it;
    primesieve_init(&it);
    /* primesieve_skipto(&it, start_number, stop_hint) */
    primesieve_skipto(&it, 2000, 1000);
    uint64_t prime;
    /* iterate over primes from 2000 to 1000 */
    while ((prime = primesieve_prev_prime(&it)) >= 1000)
        printf("%" PRIu64 "\n", prime);
    primesieve_free_iterator(&it);
    return 0;
}
```

9.6 prev_prime.cpp

Iterate backwards over primes using `primesieve::iterator`.

```
#include <primesieve.hpp>
#include <iostream>
int main()
{
    primesieve::iterator it;
    it.skipto(2000);
    uint64_t prime = it.prev_prime();
    // iterate over primes from 2000 to 1000
    for (; prime >= 1000; prime = it.prev_prime())
        std::cout << prime << std::endl;
    return 0;
}
```

9.7 primesieve_iterator.c

Iterate over primes using C `primesieve_iterator`.

```
#include <primesieve.h>
#include <inttypes.h>
#include <stdio.h>
int main()
{
    primesieve_iterator it;
    primesieve_init(&it);
    uint64_t sum = 0;
```

```

uint64_t prime = 0;
/* iterate over the primes below 10^9 */
while ((prime = primesieve_next_prime(&it)) < 1000000000ull)
    sum += prime;
printf("Sum of the primes below 10^9 = %" PRIu64 "\n", sum);
/* generate primes > 1000 */
primesieve_skipto(&it, 1000, 1100);
while ((prime = primesieve_next_prime(&it)) < 1100)
    printf("%" PRIu64 "\n", prime);
primesieve_free_iterator(&it);
return 0;
}

```

9.8 primesieve_iterator.cpp

Iterate over primes using `primesieve::iterator`.

```

#include <primesieve.hpp>
#include <iostream>
int main()
{
    primesieve::iterator it;
    uint64_t prime = it.next_prime();
    uint64_t sum = 0;
    // iterate over the primes below 10^9
    for (; prime < 1000000000ull; prime = it.next_prime())
        sum += prime;
    std::cout << "Sum of the primes below 10^9 = " << sum << std::endl;
    // generate primes > 1000
    it.skipto(1000);
    prime = it.next_prime();
    for (; prime < 1100; prime = it.next_prime())
        std::cout << prime << std::endl;
    return 0;
}

```

9.9 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;
}

```

9.10 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 primes <= 1000

```

```
primesieve::generate_primes(1000, &primes);
primes.clear();
// Store primes inside [1000, 2000]
primesieve::generate_primes(1000, 2000, &primes);
primes.clear();
// Store first 1000 primes
primesieve::generate_n_primes(1000, &primes);
primes.clear();
// Store first 10 primes >= 1000
primesieve::generate_n_primes(10, 1000, &primes);
return 0;
}
```

Index

- count_primes
 - primesieve, [12](#)
- count_quadruplets
 - primesieve, [12](#)
- count_quintuplets
 - primesieve, [13](#)
- count_sextuplets
 - primesieve, [13](#)
- count_triplets
 - primesieve, [13](#)
- count_twins
 - primesieve, [13](#)
- get_max_stop
 - primesieve, [13](#)
- INT16_PRIMES
 - primesieve.h, [29](#)
- INT32_PRIMES
 - primesieve.h, [29](#)
- INT64_PRIMES
 - primesieve.h, [29](#)
- INT_PRIMES
 - primesieve.h, [29](#)
- iterator
 - primesieve::iterator, [18](#)
- iterator.h, [23](#)
 - primesieve_next_prime, [24](#)
 - primesieve_prev_prime, [24](#)
 - primesieve_skipto, [25](#)
- iterator.hpp, [25](#)
- LONG_PRIMES
 - primesieve.h, [29](#)
- LONGLONG_PRIMES
 - primesieve.h, [29](#)
- next_prime
 - primesieve::iterator, [18](#)
- nth_prime
 - primesieve, [14](#)
- prev_prime
 - primesieve::iterator, [18](#)
- primesieve, [11](#)
 - count_primes, [12](#)
 - count_quadruplets, [12](#)
 - count_quintuplets, [13](#)
 - count_sextuplets, [13](#)
 - count_triplets, [13](#)
 - count_twins, [13](#)
 - get_max_stop, [13](#)
 - nth_prime, [14](#)
 - set_num_threads, [14](#)
 - set_sieve_size, [14](#)
- primesieve.h, [27](#)
 - INT16_PRIMES, [29](#)
 - INT32_PRIMES, [29](#)
 - INT64_PRIMES, [29](#)
 - INT_PRIMES, [29](#)
 - LONG_PRIMES, [29](#)
 - LONGLONG_PRIMES, [29](#)
 - primesieve_count_primes, [29](#)
 - primesieve_count_quadruplets, [30](#)
 - primesieve_count_quintuplets, [30](#)
 - primesieve_count_sextuplets, [30](#)
 - primesieve_count_triplets, [30](#)
 - primesieve_count_twins, [30](#)
 - primesieve_generate_n_primes, [31](#)
 - primesieve_generate_primes, [31](#)
 - primesieve_get_max_stop, [31](#)
 - primesieve_nth_prime, [32](#)
 - primesieve_set_num_threads, [32](#)
 - primesieve_set_sieve_size, [32](#)
 - SHORT_PRIMES, [29](#)
 - UINT16_PRIMES, [29](#)
 - UINT32_PRIMES, [29](#)
 - UINT64_PRIMES, [29](#)
 - UINT_PRIMES, [29](#)
 - ULONG_PRIMES, [29](#)
 - ULONGLONG_PRIMES, [29](#)
 - USHORT_PRIMES, [29](#)
- primesieve.hpp, [33](#)
- primesieve::iterator, [17](#)
 - iterator, [18](#)
 - next_prime, [18](#)
 - prev_prime, [18](#)
 - skipto, [18](#)
- primesieve::primesieve_error, [20](#)
- primesieve_count_primes
 - primesieve.h, [29](#)
- primesieve_count_quadruplets
 - primesieve.h, [30](#)
- primesieve_count_quintuplets
 - primesieve.h, [30](#)
- primesieve_count_sextuplets
 - primesieve.h, [30](#)
- primesieve_count_triplets
 - primesieve.h, [30](#)
- primesieve_count_twins

- primesieve.h, [30](#)
- primesieve_error.hpp, [35](#)
- primesieve_generate_n_primes
 - primesieve.h, [31](#)
- primesieve_generate_primes
 - primesieve.h, [31](#)
- primesieve_get_max_stop
 - primesieve.h, [31](#)
- primesieve_iterator, [21](#)
- primesieve_next_prime
 - iterator.h, [24](#)
- primesieve_nth_prime
 - primesieve.h, [32](#)
- primesieve_prev_prime
 - iterator.h, [24](#)
- primesieve_set_num_threads
 - primesieve.h, [32](#)
- primesieve_set_sieve_size
 - primesieve.h, [32](#)
- primesieve_skipto
 - iterator.h, [25](#)
- set_num_threads
 - primesieve, [14](#)
- set_sieve_size
 - primesieve, [14](#)
- SHORT_PRIMES
 - primesieve.h, [29](#)
- skipto
 - primesieve::iterator, [18](#)
- UINT16_PRIMES
 - primesieve.h, [29](#)
- UINT32_PRIMES
 - primesieve.h, [29](#)
- UINT64_PRIMES
 - primesieve.h, [29](#)
- UINT_PRIMES
 - primesieve.h, [29](#)
- ULONG_PRIMES
 - primesieve.h, [29](#)
- ULONGLONG_PRIMES
 - primesieve.h, [29](#)
- USHORT_PRIMES
 - primesieve.h, [29](#)