

---

# **intervalset Documentation**

**Millian Poquet**

**May 25, 2023**



# CONTENTS

<b>1</b>	<b>Install</b>	<b>3</b>
1.1	From Nix . . . . .	3
1.2	Build It Yourself . . . . .	3
<b>2</b>	<b>Usage</b>	<b>5</b>
2.1	How to compile/link my program with intervalset? . . . . .	5
2.2	Quick example . . . . .	5
<b>3</b>	<b>API showcase</b>	<b>7</b>
3.1	Constructors . . . . .	7
3.2	Set operations . . . . .	7
3.3	Accessing elements . . . . .	8
3.4	Testing membership . . . . .	8
3.5	Iterating elements and intervals . . . . .	8
<b>4</b>	<b>API reference</b>	<b>11</b>
<b>5</b>	<b>Changelog</b>	<b>15</b>
5.1	Unreleased . . . . .	15
5.2	v1.2.0 . . . . .	15
5.3	v1.1.0 . . . . .	15
5.4	v1.0.0 . . . . .	16
	<b>Index</b>	<b>17</b>



**intervalset** is a C++ library to manage sets of closed intervals of integers. This is a simple wrapper around [Boost.Icl](#).



## INSTALL

### 1.1 From Nix

Nix is a package manager with amazing properties that is available on all Linux or Unix-like systems (*e.g.*, macOS).

**intervalset** is available on [NUR-Kapack](#) and directly from intervalset's git repository (flake or classical `default.nix`).

### 1.2 Build It Yourself

**intervalset** can be built with [Meson](#) and [Ninja](#).

```
# Get the code
git clone https://framagit.org/batsim/intervalset.git
cd intervalset

# Prepare build (call meson)
meson build # install prefix can be changed with --prefix
cd build

# Build the library (and the unit tests if google-test is found)
ninja

# Install
ninja install # may need privileges depending on prefix
```

---

**Note:**

You first need to install **intervalset**'s dependencies. Notably:

- decent C++ compiler
- decent Meson
- [Boost.Lcl](#)

Up-to-date dependencies are defined in the [intervalset](#) Nix package.

Give a look at the [CI build output](#) for versions.

---





## 2.1 How to compile/link my program with intervalset?

**intervalset** must be included and linked if one wants to use it.

- Ideally this is done via **pkg-config** after installing **intervalset** (cf. *Install*). `pkg-config --cflags --libs intervalset`

## 2.2 Quick example

```
#include <intervalset.hpp>

void usage_example()
{
    // Creation
    IntervalSet a; // empty
    IntervalSet b = IntervalSet::ClosedInterval(0,4); // [0,4] AKA {0,1,2,3,4}
    IntervalSet c = IntervalSet::from_string_hyphen("3-5,8"); // [3,5][8] AKA {3,4,5,8}

    // Format
    c.to_string_brackets(); // "[3,5][8]"
    c.to_string_hyphen(); // "3-5,8"
    c.to_string_elements(); // "3,4,5,8"

    // Set operations
    IntervalSet intersection_set = (a & c); // {}
    IntervalSet union_set = (b + c); // [0,5][8]
    IntervalSet difference_set = (b - c); // [0,2]

    // In-place set operations
    a += IntervalSet::empty_interval_set(); // a remains empty
    // -= and &= are also defined.

    // Common operations
    int number_of_elements = c.size(); // 4
    int first_element = c.first_element(); // 3
    IntervalSet first_two_elements = c.left(2); // [3-4]
    IntervalSet two_random_elements = c.random_pick(2); // Two different random values.
    ↪ from c
```

(continues on next page)

(continued from previous page)

```
c.contains(0); // false
a.is_subset_of(c); // true
}
```

## API SHOWCASE

### 3.1 Constructors

```
#include <intervalset.hpp>

void constructors_example()
{
    // Create an empty IntervalSet
    IntervalSet s1;

    // Copy an existing IntervalSet
    IntervalSet s2 = s1;

    // Create an IntervalSet from one interval
    IntervalSet s3 = IntervalSet::ClosedInterval(0,1);

    // Create an IntervalSet from one integer
    IntervalSet s4 = 2;
}
```

### 3.2 Set operations

```
#include <intervalset.hpp>

void set_operations_example()
{
    IntervalSet s1 = IntervalSet::from_string_hyphen("3,4-7,10-20,22,24-28");
    IntervalSet s2 = IntervalSet::from_string_hyphen("4,19-21,23");

    // Classical set operations
    IntervalSet s_union = (s1 + s2);
    IntervalSet s_intersection = (s1 & s2);
    IntervalSet s_difference = (s1 - s2);

    // In-place operations
    s1 += s2; // s1 = s1 s2
    s1 &= s2; // s1 = s1 s2
}
```

(continues on next page)

(continued from previous page)

```
s1 -= s2; // s1 = s1 \ s2
}
```

### 3.3 Accessing elements

```
#include <intervalset.hpp>

void access_example()
{
    IntervalSet s = IntervalSet::from_string_hyphen("3,10-16");

    s.first_element(); // 3
    s.left(1); // 3
    s.left(2); // {3,10}
    s.left(4); // {3,10,11,12}
    s.random_pick(2); // Two different random elements from s

    // Access can be done via operator[]
    // WARNING: This is very slow! Use iterators if performance matters.
    s[0]; // 3
    s[4]; // 13
}
```

### 3.4 Testing membership

```
#include <intervalset.hpp>

void membership_example()
{
    IntervalSet s1 = IntervalSet::from_string_hyphen("3,4-7,10-20,22,24-28");
    IntervalSet s2 = IntervalSet::from_string_hyphen("5-6,15,19,28");

    s1.contains(4); // Does s1 contains 4? Yup.
    s1.contains(IntervalSet::ClosedInterval(8,25)); // Nope.
    s2.is_subset_of(s1); // Yup, s2 s1
}
```

### 3.5 Iterating elements and intervals

```
#include <intervalset.hpp>

void traversal_example()
{
    IntervalSet s = IntervalSet::from_string_hyphen("4,19-21,23");
}
```

(continues on next page)

(continued from previous page)

```
// The intervals can be traversed
for (auto it = s.intervals_begin(); it != s.intervals_end(); ++it)
{
    // Use operator* to retrieve the interval
    const IntervalSet::ClosedInterval & itv = *it;

    // The two bounding elements can be retrieved this way
    int interval_minimum_element = lower(itv);
    int interval_maximum_element = upper(itv);
}

// The individual values can also be traversed
// Please DO note that this may be way slower than iterating over intervals
for (auto it = s.elements_begin(); it != s.elements_end(); ++it)
{
    // Use operator* to retrieve the element value
    int element = *it;
}
}
```



## API REFERENCE

struct **IntervalSet**

### Public Types

typedef boost::icl::closed\_interval<int, std::less> **ClosedInterval**

A closed interval of integers.

### Public Functions

**IntervalSet**()

Create an empty *IntervalSet*.

**IntervalSet**(const *ClosedInterval* &interval)

Create an *IntervalSet* made of a single interval.

**IntervalSet**(const *IntervalSet* &other)

Create an *IntervalSet* from another *IntervalSet* (copy constructor).

**IntervalSet**(int integer)

Create an *IntervalSet* from a single integer.

element\_const\_iterator **elements\_begin**() const

Iterator to beginning **element**.

**Note:** Iterating **intervals** is much more efficient (via *intervals\_begin()* and *intervals\_end()*).

element\_const\_iterator **elements\_end**() const

Iterator to ending **element**.

**Note:** Iterating **intervals** is much more efficient (via *intervals\_begin()* and *intervals\_end()*).

const\_iterator **intervals\_begin**() const

Iterator to beginning **interval**.

const\_iterator **intervals\_end**() const

Iterator to ending **interval**.

void **clear**()

Remove all the elements in the *IntervalSet*. In other words, make it empty.

void **insert**(const *IntervalSet* &interval\_set)

Insert an *IntervalSet* in another. This is similar to *operator+=(const IntervalSet &)*.

void **insert**(*ClosedInterval* interval)

Insert a *ClosedInterval* in an *IntervalSet*.

void **insert**(int integer)

Insert an integer in an *IntervalSet*.

void **remove**(const *IntervalSet* &interval\_set)

Remove an *IntervalSet* from another. This is similar to *operator-=(const Intervalset &)*.

void **remove**(*ClosedInterval* interval)

Remove a *ClosedInterval* from an *IntervalSet*.

void **remove**(int integer)

Remove an integer from an *IntervalSet*.

*IntervalSet* **left**(unsigned int nb\_integers) const

Create a sub-*IntervalSet* made of the nb\_integers leftmost elements of the source *IntervalSet*.

**Pre**

The source *IntervalSet* must contains nb\_integers or more elements.

*IntervalSet* **random\_pick**(unsigned int nb\_integers) const

Create a sub-*IntervalSet* made of nb\_integers randomly-picked elements from the source *IntervalSet*.

**Pre**

The source *IntervalSet* must contains nb\_integers or more elements.

const\_iterator **biggest\_interval**() const

Returns a const iterator to the biggest *ClosedInterval* in an *IntervalSet*.

int **first\_element**() const

Returns the value of the first element of an *IntervalSet*.

**Pre**

The *IntervalSet* must **NOT** be empty.

unsigned int **size**() const

Returns the number of **elements** of an *IntervalSet*.

bool **is\_empty**() const

Returns whether an *IntervalSet* is empty. An empty *IntervalSet* does not contain any element.

bool **contains**(int integer) const

Returns whether an *IntervalSet* contains an integer.

bool **contains**(const *ClosedInterval* &interval) const

Returns whether an *IntervalSet* fully contains a *ClosedInterval*.

bool **is\_subset\_of**(const *IntervalSet* &other) const

Returns whether an *IntervalSet* is a subset of another *IntervalSet*.

std::string **to\_string\_brackets**(const std::string &union\_str = "", const std::string &opening\_bracket = "[",  
const std::string &closing\_bracket = "]", const std::string &sep = ",") const

Returns a string representation of an *IntervalSet*.

This is the classical representation used in mathematics. For example, {1,2,3,7} is represented as [1,3][7].



std::string **to\_string\_hyphen**(const std::string &sep = ",", const std::string &joiner = "-") const

Returns a string representation of an *IntervalSet*.

This is a compact representation where {1,2,3,7} is represented as 1-3,7. Use sep=' ' to get a Batsim-compatible representation (see [Batsim documentation about Interval sets representation](#)).

std::string **to\_string\_elements**(const std::string &sep = ",") const

Returns a string representation of an *IntervalSet*.

This is the set representation of an *IntervalSet*. For example, {1,2,3,7} is represented as 1,2,3,7

*IntervalSet* &**operator**=(const *IntervalSet* &other)

Assignment operator. Reset an *IntervalSet* content to the one of another *IntervalSet*.

*IntervalSet* &**operator**=(const *IntervalSet::ClosedInterval* &interval)

Assignment operator. Reset an *IntervalSet* content to the one of a *ClosedInterval*.

bool **operator**==(const *IntervalSet* &other) const

Returns whether two *IntervalSet* exactly contain the same elements.

bool **operator**!=(const *IntervalSet* &other) const

Returns whether the content of two *IntervalSet* is different.

*IntervalSet* &**operator**--(const *IntervalSet* &other)

Difference + assignment operator. This is similar to *remove(const IntervalSet &)*.

a -= b; means "Set a's value to be a without the elements of b".

*IntervalSet* &**operator**+=(const *IntervalSet* &other)

Union + assignment operator. This is similar to *insert(const IntervalSet &)*.

a += b; means "Set a's value to be the union of a and b".

*IntervalSet* **operator**-(const *IntervalSet* &other) const

Difference operator. a - b returns an *IntervalSet* of the elements that are in a but are not in b.

*IntervalSet* **operator**+(const *IntervalSet* &other) const

Union operator. a + b returns an *IntervalSet* of the elements that are in a, in b, or both in a and b.

int **operator**[](int index) const

Subscript operator.

Returns the index-th element of the *IntervalSet*.

#### Pre

index must be positive and strictly smaller than *size()*

## Public Static Functions

static *IntervalSet* **from\_string\_hyphen**(const std::string &str, const std::string &sep = ",", const std::string &joiner = "-")

Parse an *IntervalSet* string representation and return the corresponding *IntervalSet*.

See *IntervalSet::to\_string\_hyphen* for representation details.

static *IntervalSet* **empty\_interval\_set**()

Returns an empty *IntervalSet*.



## CHANGELOG

All notable changes to this project will be documented in this file. The format is based on [Keep a Changelog](#) and intervalset adheres to [Semantic Versioning](#). The public API of intervalset is simply the public C++ functions and types defined by the library.

---

### 5.1 Unreleased

- [Commits since v1.2.0](#)
- 

### 5.2 v1.2.0

- [Commits since v1.1.0](#)
- Release date: 2019-02-22

#### 5.2.1 Added

- New `is_empty` method, that returns whether an intervalset is empty.
  - Full API is now documented on [readthedocs](#).
- 

### 5.3 v1.1.0

- [Commits since v1.0.0](#)
- Release date: 2018-11-09

### 5.3.1 Changed

- Build system changed from CMake to Meson.
- 

## 5.4 v1.0.0

- Release date: 2018-10-16

## INDEX

I

IntervalSet (C++ *struct*), 11  
IntervalSet::biggest\_interval (C++ *function*), 12  
IntervalSet::clear (C++ *function*), 11  
IntervalSet::ClosedInterval (C++ *type*), 11  
IntervalSet::contains (C++ *function*), 12  
IntervalSet::elements\_begin (C++ *function*), 11  
IntervalSet::elements\_end (C++ *function*), 11  
IntervalSet::empty\_interval\_set (C++ *function*),  
13  
IntervalSet::first\_element (C++ *function*), 12  
IntervalSet::from\_string\_hyphen (C++ *function*),  
13  
IntervalSet::insert (C++ *function*), 11, 12  
IntervalSet::intervals\_begin (C++ *function*), 11  
IntervalSet::intervals\_end (C++ *function*), 11  
IntervalSet::IntervalSet (C++ *function*), 11  
IntervalSet::is\_empty (C++ *function*), 12  
IntervalSet::is\_subset\_of (C++ *function*), 12  
IntervalSet::left (C++ *function*), 12  
IntervalSet::operator!= (C++ *function*), 13  
IntervalSet::operator+ (C++ *function*), 13  
IntervalSet::operator+= (C++ *function*), 13  
IntervalSet::operator= (C++ *function*), 13  
IntervalSet::operator== (C++ *function*), 13  
IntervalSet::operator- (C++ *function*), 13  
IntervalSet::operator-= (C++ *function*), 13  
IntervalSet::operator[] (C++ *function*), 13  
IntervalSet::random\_pick (C++ *function*), 12  
IntervalSet::remove (C++ *function*), 12  
IntervalSet::size (C++ *function*), 12  
IntervalSet::to\_string\_brackets (C++ *function*),  
12  
IntervalSet::to\_string\_elements (C++ *function*),  
13  
IntervalSet::to\_string\_hyphen (C++ *function*), 12