

“...one of the most highly regarded and expertly designed C++ library projects in the world.”

— Herb Sutter and Andrei Alexandrescu, *C++ Coding Standards*

---

## Chapter 10. Boost.Intrusive

Olaf Krzikalla

Ion Gaztanaga

Copyright © 2005 Olaf Krzikalla, 2006-2010 Ion Gaztanaga

Distributed under the Boost Software License, Version 1.0. (See accompanying file LICENSE\_1\_0.txt or copy at [http://www.boost.org/LICENSE\\_1\\_0.txt](http://www.boost.org/LICENSE_1_0.txt))

### Table of Contents

#### Introduction

- Presenting Boost.Intrusive
- Building Boost.Intrusive

#### Intrusive and non-intrusive containers

- Differences between intrusive and non-intrusive containers
- Properties of Boost.Intrusive containers

#### How to use Boost.Intrusive

- Using base hooks
- Using member hooks
- Using both hooks
- Object lifetime

#### When to use?

#### Concept summary

#### Presenting Boost.Intrusive containers

#### Safe hooks

- Features of the safe mode
- Configuring safe-mode assertions

#### Auto-unlink hooks

- What's an auto-unlink hook?
- Auto-unlink hook example
- Auto-unlink hooks and containers with constant-time `size()`

#### Intrusive singly linked list: `slist`

- `slist` hooks
- `slist` container
- Example

#### Intrusive doubly linked list: `list`

- `list` hooks
- `list` container
- Example

#### Intrusive associative containers: `set`, `multiset`, `rbtree`

- `set`, `multiset` and `rbtree` hooks
- `set`, `multiset` and `rbtree` containers
- Example

#### Semi-Intrusive unordered associative containers: `unordered_set`, `unordered_multiset`

- `unordered_set` and `unordered_multiset` performance notes
- `unordered_set` and `unordered_multiset` hooks
- `unordered_set` and `unordered_multiset` containers
- Example
- Custom bucket traits

#### Intrusive splay tree based associative containers: `splay_set`, `splay_multiset` and `splay_tree`

- Advantages and disadvantages of splay tree based containers
- `splay_set`, `splay_multiset` and `splaytree` hooks
- `splay_set`, `splay_multiset` and `splaytree` containers
- Splay trees with BST hooks

### Example

Intrusive avl tree based associative containers: `avl_set`, `avl_multiset` and `avltree`

`avl_set`, `avl_multiset` and `avltree` hooks

`avl_set`, `avl_multiset` and `avltree` containers

### Example

Intrusive scapegoat tree based associative containers: `sg_set`, `sg_multiset` and `sgtree`

Using binary search tree hooks: `bs_set_base_hook` and `bs_set_member_hook`

`sg_set`, `sg_multiset` and `sgtree` containers

### Example

Intrusive treap based associative containers: `treap_set`, `treap_multiset` and `treap`

Using binary search tree hooks: `bs_set_base_hook` and `bs_set_member_hook`

`treap_set`, `treap_multiset` and `treap` containers

Exception safety of treap-based intrusive containers

### Example

Advanced lookup and insertion functions for associative containers

Advanced lookups

Advanced insertions

Positional insertions

Erasing and disposing values from Boost.Intrusive containers

Cloning Boost.Intrusive containers

Using function hooks

Recursive Boost.Intrusive containers

Using smart pointers with Boost.Intrusive containers

Requirements for smart pointers compatible with Boost.Intrusive

Obtaining iterators from values

Any Hooks: A single hook for any Intrusive container

Concepts explained

Node algorithms with custom NodeTraits

Intrusive singly linked list algorithms

Intrusive doubly linked list algorithms

Intrusive red-black tree algorithms

Intrusive splay tree algorithms

Intrusive avl tree algorithms

Intrusive treap algorithms

Containers with custom ValueTraits

ValueTraits interface

Custom ValueTraits example

Reusing node algorithms for different values

Simplifying value traits definition

Stateful value traits

Thread safety guarantees

Obtaining the same types and reducing symbol length

Design Notes

Boost.Intrusive in performance sensitive environments

Boost.Intrusive in space constrained environments

Boost.Intrusive as a basic building block

Extending Boost.Intrusive

Performance

Back insertion and destruction

Reversing

Sorting

Write access

Conclusions

Release Notes

Boost 1.45 Release

Boost 1.40 Release

Boost 1.39 Release

Boost 1.38 Release

Boost 1.37 Release

Boost 1.36 Release

Tested compilers

References

Acknowledgements

Reference

Header `<boost/intrusive/any_hook.hpp>`

Header `<boost/intrusive/avl_set.hpp>`

Header `<boost/intrusive/avl_set_hook.hpp>`

Header `<boost/intrusive/avltree.hpp>`

Header `<boost/intrusive/avltree_algorithms.hpp>`

```
Header <boost/intrusive/bs_set_hook.hpp>
Header <boost/intrusive/circular_list_algorithms.hpp>
Header <boost/intrusive/circular_slist_algorithms.hpp>
Header <boost/intrusive/derivation_value_traits.hpp>
Header <boost/intrusive/hashtable.hpp>
Header <boost/intrusive/intrusive_fwd.hpp>
Header <boost/intrusive/linear_slist_algorithms.hpp>
Header <boost/intrusive/link_mode.hpp>
Header <boost/intrusive/list.hpp>
Header <boost/intrusive/list_hook.hpp>
Header <boost/intrusive/member_value_traits.hpp>
Header <boost/intrusive/options.hpp>
Header <boost/intrusive/parent_from_member.hpp>
Header <boost/intrusive/pointer_plus_bits.hpp>
Header <boost/intrusive/priority_compare.hpp>
Header <boost/intrusive/rbtree.hpp>
Header <boost/intrusive/rbtree_algorithms.hpp>
Header <boost/intrusive/set.hpp>
Header <boost/intrusive/set_hook.hpp>
Header <boost/intrusive/sg_set.hpp>
Header <boost/intrusive/sgtree.hpp>
Header <boost/intrusive/sgtree_algorithms.hpp>
Header <boost/intrusive/slist.hpp>
Header <boost/intrusive/slist_hook.hpp>
Header <boost/intrusive/splay_set.hpp>
Header <boost/intrusive/splay_set_hook.hpp>
Header <boost/intrusive/splaytree.hpp>
Header <boost/intrusive/splaytree_algorithms.hpp>
Header <boost/intrusive/treap.hpp>
Header <boost/intrusive/treap_algorithms.hpp>
Header <boost/intrusive/treap_set.hpp>
Header <boost/intrusive/trivial_value_traits.hpp>
Header <boost/intrusive/unordered_set.hpp>
Header <boost/intrusive/unordered_set_hook.hpp>
```

## Introduction

Presenting Boost.Intrusive  
Building Boost.Intrusive

## Presenting Boost.Intrusive

**Boost.Intrusive** is a library presenting some intrusive containers to the world of C++. Intrusive containers are special containers that offer better performance and exception safety guarantees than non-intrusive containers (like STL containers).

The performance benefits of intrusive containers makes them ideal as a building block to efficiently construct complex containers like multi-index containers or to design high performance code like memory allocation algorithms.

While intrusive containers were and are widely used in C, they became more and more forgotten in C++ due to the presence of the standard containers which don't support intrusive techniques. **Boost.Intrusive** not only reintroduces this technique to C++, but also encapsulates the implementation in STL-like interfaces. Hence anyone familiar with standard containers can easily use **Boost.Intrusive**.

## Building Boost.Intrusive

There is no need to compile anything to use **Boost.Intrusive**, since it's a header only library. Just include your Boost header directory in your compiler include path.

Last revised: March 08, 2011 at 23:39:05 GMT