Compressed_Pair

Steve Cleary
Beman Dawes
Howard Hinnant
John Maddock

Copyright © 2000 Steve Cleary, Beman Dawes, Howard Hinnant & John Maddock

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)

Table of Contents

Overview 2
Synopsis 3
Acknowledgments 4

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Overview

All of the contents of `<boost/compressed_pair.hpp>` are defined inside `namespace boost`.

The class `compressed_pair` is very similar to `std::pair`, but if either of the template arguments are empty classes, then the *empty base-class optimisation* is applied to compress the size of the pair.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
template <class T1, class T2>
class compressed_pair
{
public:
    typedef T1                                                 first_type;
    typedef T2                                                 second_type;
    typedef typename call_traits<first_type>::param_type       first_param_type;
    typedef typename call_traits<second_type>::param_type      second_param_type;
    typedef typename call_traits<first_type>::reference        first_reference;
    typedef typename call_traits<second_type>::reference       second_reference;
    typedef typename call_traits<first_type>::const_reference  first_const_reference;
    typedef typename call_traits<second_type>::const_reference second_const_reference;

             compressed_pair() : base() {}
             compressed_pair(first_param_type x, second_param_type y);
    explicit compressed_pair(first_param_type x);
    explicit compressed_pair(second_param_type y);

    compressed_pair& operator=(const compressed_pair&);

    first_reference       first();
    first_const_reference first() const;

    second_reference       second();
    second_const_reference second() const;

    void swap(compressed_pair& y);
};
```

The two members of the pair can be accessed using the member functions `first()` and `second()`. Note that not all member functions can be instantiated for all template parameter types. In particular `compressed_pair` can be instantiated for reference and array types, however in these cases the range of constructors that can be used are limited. If types `T1` and `T2` are the same type, then there is only one version of the single-argument constructor, and this constructor initialises both values in the pair to the passed value.

Note that if either member is a POD type, then that member is not zero-initialized by the `compressed_pair` default constructor: it's up to you to supply an initial value for these types if you want them to have a default value.

Note that `compressed_pair` can not be instantiated if either of the template arguments is a union type, unless there is compiler support for `boost::is_union`, or if `boost::is_union` is specialised for the union type.

Finally, a word of caution for Visual C++ 6 users: if either argument is an empty type, then assigning to that member will produce memory corruption, unless the empty type has a "do nothing" assignment operator defined. This is due to a bug in the way VC6 generates implicit assignment operators.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Acknowledgments

Based on contributions by Steve Cleary, Beman Dawes, Howard Hinnant and John Maddock.

Maintained by John Maddock.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/