Subject: Re: How to mark std::array<T, N> moveable if only T is moveable
Posted by mirek on Thu, 25 Jul 2024 00:27:32 GMT
View Forum Message <> Reply to Message

busiek wrote on Tue, 23 July 2024 18:25mirek wrote on Thu, 18 July 2024 08:41Lance wrote on Wed, 03 July 2024 03:39Reminds me of some exploration I did with Moveable.

I vaguely recalled that when I tested a few years back, the way Moveable was designed were causing a compile time error with Windows/VSBuild. I would also welcome a redesign of Moveable using more recent language facilities.

Suggestions? (But it needs to be backward compatible).

I was thinking about something like that:

```
#include <Core/Core.h>
#include <type_traits>

namespace Upp {
// First way of marking class moveable is to be an ancestor of NtlMoveableBase
template <class T> class NtlMoveableBase {};

// Second way is to specialize NtlMoveableClass
template <class T>
struct NtlMoveableClass
: std::integral_constant<bool,
    std::is_trivial_v<T>
 || std::is_base_of_v<Upp::NtlMoveableBase<T>, T>
 // below is not needed if we make them derived from NtlMoveableBase
 || std::is_base_of_v<Upp::Moveable_<T>, T>
 || std::is_base_of_v<Upp::Moveable<T>, T>
 || std::is_base_of_v<Upp::MoveableAndDeepCopyOption<T>, T>> {};

// For instance mark std::array<T, N> moveable if only T is moveable
template <class T, size_t N>
struct NtlMoveableClass<std::array<T, N>>
: std::integral_constant<bool, NtlMoveableClass<T>::value> {};

// Helper for checking whether class is moveable
template <class T>
inline constexpr bool IsNtlMoveable = NtlMoveableClass<T>::value;

// Optional concept for c++20
template <class T>
concept NtlMoveable = IsNtlMoveable<T>;

}

using namespace Upp;
```

```
// use of concept
template <NtlMoveable T>
struct MyVector
{
 T *ptr;
 // Or use static_assert
 ~MyVector() { static_assert(IsNtlMoveable<T>); }
};

CONSOLE_APP_MAIN
{
 // Checking if given type is moveable
 static_assert(IsNtlMoveable<int>);
 static_assert(IsNtlMoveable<const void *>);
 static_assert(IsNtlMoveable<Vector<int>>);
 static_assert(IsNtlMoveable<std::array<int, 5>>);
 static_assert(IsNtlMoveable<std::array<Vector<int>, 5>>);
 static_assert(!IsNtlMoveable<Thread>);
}
```

You could redefine NTL_MOVEABLE macro as a specialization of NtlMoveableClass.
But you need to change an assertion AssertMoveable((T*)0) to static_assert(IsNtlMoveable<T>).
Are those ideas usable?

Yep. After tinkering and simplifying I think this should cover it all:

```
template <class T> struct Moveable {};

template <class T>
inline constexpr bool is_Moveable = std::is_trivially_copyable<T>::value ||
                                    std::is_base_of<moveable<T>, T>::value;
```

We will not need NTL_MOVEABLE macro anyway... Trivial types are covered and allowing non-trivial types to be explicitly marked Moveable is outright dangerous.

BTW, std::array passes is_Moveable out of box.

See any problem? (Apart for requiring C++17, but I guess we can go there for the next release)

Mirek