How do I allow move construction and disallow assignment and copy construction of a class

Asked 10 years, 2 months ago Modified 10 months ago Viewed 4k times

7

Is there a way to allow a move constructor and disallow copy construction and assignment. I can think of several classes with file pointers and buffer pointers (resource handles etc) that would benefit from being copy constructed and assigned.

I am using VC2010 & GCC 4.5.2. I know that I would have to declare empty private assignment and copy constructors in the VC2010 class headers and as far as I am aware GCC allows some sort of delete signature after the method to do the same thing.

If anyone has a good example of a skeleton class like this and the advantages I would be very grateful. Thanks in advance John

Here is an example of a class for which I would like to allow moves but I would also like to prevent direct asssignment. Is it simily a matter of making the copy constructor and operator=private?

```
class LoadLumScanner_v8002 : public ILoadLumScanner {
public:
// default constructor
LoadLumScanner_v8002();

// copy constructor
LoadLumScanner_v8002(const LoadLumScanner_v8002& rhs);

// move constructor
LoadLumScanner_v8002(LoadLumScanner_v8002&& rhs);

// non-throwing copy-and-swap idiom unified assignment
inline LoadLumScanner_v8002& operator=(LoadLumScanner_v8002 rhs) {
    rhs.swap(*this);
    return *this;
}

// non-throwing-swap idiom
inline void swap(LoadLumScanner_v8002& rhs) throw() {
    // enable ADL (not necessary in our case, but good practice)
    using std::swap;
    // swap base members
    // ...
    // swap members
    swap(mValidatedOk, rhs.mValidatedOk);
    swap(mFile, rhs.mFile);
    swap(mPartNo, rhs.mPartNo);
    swap(mMediaSequenceNo, rhs.mMediaSequenceNo);
    swap(mMaxMediaSequenceNo, rhs.mMaxMediaSequenceNo);
    swap(mLoadListOffset, rhs.mLoadListOffset);
    swap(mFirstLoadOffset, rhs.mFirstLoadOffset);
    swap(mLoadCount, rhs.mLoadCount);
    swap(mLoadIndex, rhs.mLoadIndex);
    swap(mLoadMediaSequenceNo, rhs.mLoadMediaSequenceNo);
    swap(mLoadPartNo, rhs.mLoadPartNo);
    swap(mLoadFilePath, rhs.mLoadFilePath);
```

```
}

// destructor
virtual ~LoadLumScanner_v8002();
}
```

c++ c++11 copy-constructor assignment-operator

Share Improve this question

Follow

edited May 6, 2012 at 18:30

ildjarn

61.1k 9 122 206

asked May 6, 2012 at 17:38

johnco3

2,319 4 31 60

1 Answer

Sorted by:

Highest score (default)

15

Both of the two solutions you mention work fine.

1.

```
class MoveOnly
{
   MoveOnly(const MoveOnly&);
   MoveOnly& operator=(const MoveOnly&);
public:
   MoveOnly(MoveOnly&&);
   MoveOnly& operator=(MoveOnly&&);
};
```

2.

```
class MoveOnly
{
public:
   MoveOnly(const MoveOnly&) = delete;
   MoveOnly& operator=(const MoveOnly&) = delete;
   MoveOnly(MoveOnly&&) = default;
   MoveOnly& operator=(MoveOnly&&) = default;
};
```

The "= delete" signature is new with C++11 (as is the rvalue reference), and means essentially the same thing as the C++03 technique (declare private and don't define). The advantage of the C++11 solution is that it will for sure catch mistakes at compile time, not delay until link time.

Your compiler may not yet support "= delete" and in that case, you'll have to fall back on the first solution.

A third solution is to default the copy members:

```
class MoveOnly
{
public:
   MoveOnly(MoveOnly&&);
   MoveOnly& operator=(MoveOnly&&);
};
```

When a move special member is declared, whether defaulted or not, then the compiler will implicitly add deleted copy members if you do not declare them otherwise. Your compiler may or may not implement this feature yet.

Share Improve this answer

Follow

edited Aug 25, 2021 at 17:07

plswork04

458 5 10

answered May 6, 2012 at 18:20

Howard Hinnant

194k 49 427 556

4 I like the `= delete` solution because it makes it explicitly clear that it's not just an accidental omission of a definition to anybody who reads it later. – Flexo ♦ May 7, 2012 at 9:22

I needed to add "= default" to the move constructor and move assignment in the second one. – Eyal Feb 10, 2021 at 22:22