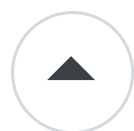


When should I concern myself with `std::iostream::sentry`?

Asked 14 years, 10 months ago Modified 14 years, 10 months ago

Viewed 5k times



31



Online references have rather brief and vague descriptions on the purpose of `std::iostream::sentry`.

When should I concern myself with this little critter? If it's only intended to be used internally, why make it public?

c++

iostream

Share

Improve this question

Follow

asked Feb 19, 2010 at 18:18



[Emile Cormier](#)

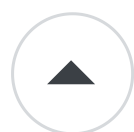
29.1k ● 15 ● 97 ● 130

Since when is an identifier a critter? And it is certainly not little! :D – [Hogan](#) Feb 19, 2010 at 18:21

3 Answers

Sorted by:

Highest score (default)



16

It's used whenever you need to extract or output data with a stream. That is, whenever you make an `operator>>`, the extraction operator, or `operator<<`, the insertion operator.



It's purpose is to simplify the logic: "Are any fail bits set? Synchronize the buffers. For input streams, optionally get any whitespace out of the way. Okay, ready?"



All extraction stream operators should begin with:

```
// second parameter to true to not skip whitespace, fo
const std::istream::sentry ok(stream, icareaboutwhites

if (ok)
{
    // ...
}
```

And all insertion stream operators should begin with:

```
const std::ostream::sentry ok(stream);

if (ok)
{
    // ...
}
```

It's just a cleaner way of doing (something similar to):

```
if (stream.good())
{
    if (stream.tie())
        stream.tie()->sync();

    // the second parameter
    if (!noskipwhitespace && stream.flags() & ios_base
    {
        stream >> std::ws;
    }
}
```

```
if (stream.good())  
{  
    // ...  
}
```

`ostream` just skips the whitespace part.

Share Improve this answer

edited Feb 19, 2010 at 18:48


Follow

answered Feb 19, 2010 at 18:31



GManNickG

503k ● 53 ● 502 ● 549

3 If I implement a custom `operator>>` in terms of other stream member functions, is it still necessary (or a good idea) to use the sentry? – [Emile Cormier](#) Feb 19, 2010 at 18:35


What about the insertion operator? ostream also defines sentry. – [Emile Cormier](#) Feb 19, 2010 at 18:36

@GMan: I hear ya. It seems one could write a whole textbook just on iostreams. :-) – [Emile Cormier](#) Feb 19, 2010 at 18:54

@GMan: at least twice -- there's an old (pre-standard) one by Teal, and a newer (much larger) one by Langer and Kreft that talks about the standardized version. – [Jerry Coffin](#) Feb 19, 2010 at 21:26

3 @EmileCormier: I'm revoking my previous comments. Contrary to what I previously suggested, you do not need to use sentries if your operators are defined in terms of existing operators, I'm not sure why I said otherwise. – [GManNickG](#) Aug 31, 2012 at 16:57



15



Most people will never write any code that needs to deal with creating sentry objects. A sentry object is needed when/if you extract data from (or insert it into) the stream buffer that underlies the stream object itself.

As long as your insertion/extraction operator uses other iostream members/operators to do its work, it does *not* have to deal with creating a sentry object (because those other iostream operators will create and destroy sentry objects as needed).

Share Improve this answer

Follow

answered Feb 19, 2010 at 20:13



[Jerry Coffin](#)

489k ● 83 ● 647 ● 1.1k

Ah, thanks. This makes the most sense to me (especially the underlying stream buffer thing). I was puzzled over why the textbook examples showing how to write your own custom stream operators never mentioned sentry objects.

– [Emile Cormier](#) Feb 19, 2010 at 20:35



1



Formatted input for anything but the basic types (int, double, etc.) doesn't make a lot of sense, and arguably only from them when taken from a non-interactive stream such as an `istream`. So you should probably not be implementing `op>>` in the first place, and thus not have to worry about sentry objects.



Share Improve this answer

[edited Feb 19, 2010 at 20:51](#)

Follow

answered Feb 19, 2010 at 20:23



anon

What if I want overload operator`>>` to conveniently do formatted input on a tuple of basic types? E.g.: `struct Point{double x, double y}; Point point; file >> point;` The overloaded operator`>>` is implemented in terms of operator`>>` for basic types. – [Emile Cormier](#) Feb 19, 2010 at 20:29

@Emile Assuming point is formatted as "x,y" then its suprisingly difficult to implement `op>>` so it works correctly for point AND works correctly when used in concert with all other formatted input operators. Basically, you are better off writing functions to parse the specific inputs you are expecting, rather than some generalised scheme which at the end of the day can't handle real world inputs. – anon Feb 19, 2010 at 20:34

@Neil: I see what you're saying. But perhaps a legitimate use of a custom operator `>>` would be to read back data from a file generated by symmetrical operator `<<`'s. This seems to be the premise behind the Boost.Serialization library.

– [Emile Cormier](#) Feb 19, 2010 at 20:43 

@Emile Well serialisation is a different thing. Typically it must output a token before the data - like `[POINT]120,70` that can be used to work out what the thing being read back in actually is. Such libraries often misuse (IMHO) operators like `op>>` when a named function like `Read()` might be better. I don't know anything about the Boost serialisation library though, and am a bit dubious about serialisation as a concept – anon Feb 19, 2010 at 20:50

@Neil: Hmm, no wait, I don't quite understand your argument in your first comment. :-) You're saying that `operator>>` `(istream& s, P& p){s>>p.x; s>>p.y;}` is bad, yet `getPoint(istream& s, P& p){s>>p.x; s>>p.y;}` is ok? – [Emile Cormier](#) Feb 19, 2010 at 20:53 