

Re: ostream_iterator with namespaced types – BUG?

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.stl/2004-10/0064.html>

From: Igor Tandetnik (itandetnik_at_mvps.org)

Date: 10/15/04

Date: Fri, 15 Oct 2004 11:07:12 -0400

"Andy Coates" <n0_chance@n0_where.com> wrote in message
news:uyB%23iWssEHA.3580@TK2MSFTNGP10.phx.gbl
> So, changing the subject now that we are talking about streams...
>
> This doesn't work either:
>
> class Stream;
> Stream& bar(Stream& s);
>
> class Stream : public std::wostringstream
> {
> public:
> void foo()
> {
> ((std::wostringstream&)*this) << bar;
> }
>
> void works(){}
> };
>
> Stream& bar(Stream& s)
> {
> s.works();
> return s;
> }
>
> Stream& operator<<(Stream& s, Stream& (__cdecl *pFn)(Stream&))
> {
> (*pFn)(s);
> return s;
> }
>
> Because, again, it matches the void* inserter.

But of course. The first parameter of operator<< in your call is

microsoft.public.vc.stl: Re: ostream_iterator with namespaced types – BUG?

std::wostringstream&, not Stream, so your operator taking a Stream& does not have a chance. Remove the cast, forward-declare your operator<<() before the definition of Stream, and everything works as expected on my VC7.1

Note that deriving from standard stream classes and creating your own versions of operator<< is not a very good idea. Chaining breaks.

Consider:

```
Stream s;  
s << bar; // this works  
s << 1 << bar; // this does not
```

The first operator<<(ostream&, int) returns an ostream&, so then you effectively have

```
(ostream&)s << bar;
```

which we already know does not do what you want.

> *Is there a known, safe, solution to this kind of issue? Sorry if
> this is a bit of a muppet question – I'm a competent C++ dev, just
> never really tried deriving my own stream class before!*

Stream classes were not designed for being derived from. What are you trying to achieve?

--

With best wishes,

Igor Tandetnik

"On two occasions, I have been asked [by members of Parliament], 'Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?' I am not able to rightly apprehend the kind of confusion of ideas that could provoke such a question." -- Charles Babbage