A Proposal to add two iostream manipulators to the C++ Standard Library

Document number: JTC 1/SC22/WG21/N1842=05-0102
Date: 2005-07-27, version 1
Project: Languages C++
References: C++ ISO/IEC IS 14882:1998(E),

Reply to: Paul A Bristow, pbristow@hetp.u-net.com, J16/04-0108

1 Background & motivation

Why is this important? What kinds of problems does it address, and what kinds of programmers is it intended to support? Is it based on existing practice?

May I suggest, for completeness, the following additional trivial iostream manipulators which return the stream back to the default state:

```
ios_base& automatic(ios_base& str)
{
      str.unsetf(ios_base::floatfield); // Effectively zero.
      return (str);
}

so cout << automatic ... // means not fixed nor scientific (default)

ios_base& noadjust(ios_base& str)
{
      str.unsetf(ios_base::adjustfield); // Effectively zero.
      return (str);
}

and cout << no adjust … // means no left, no right and no internal
```

My experience is that without these manipulators, for which I propose (tentatively) the names `automatic` and `noadjust`, quite often layout cannot be performed with manipulators. While this is possible with the single bit flags like showpos, showpoint that have simple negates in noshowpos, noshowpoint, for multi-bit fields like `floatfield` and `adjustfield`, there are no equivalents so it is necessary to use the `unsetf`() function directly, which looks unsightly, and interrupts the sequence of manipulators.

Novices expect, most reasonably, that everything simple to do with formatting is possible with manipulators.

Novices using cout << … should not need to know about `unsetf` nor the details of the float and adjust fields.

2 Impact on the C++ Standard

This proposal is for a pure addition to existing iostream manipulators. It does not require any C++ language change, nor any change to the existing values provided by ios.

Draft of Proposed Revised Text for

27.4.5.4 floatfield manipulators [lib.floatfield.manip]

`ios_base & fixed ( ios_base & str );`
1 Effects: Calls
`str .setf(ios_base::fixed, ios_base::floatfield).`
2 Returns: `str`

`ios_base & scientific ( ios_base & str );`
3 Effects: Calls `str .setf(ios_base::scientific, ios_base::floatfield).`
4 Returns: `str`

Proposed addition to 27.4.5.4

`ios_base & automatic ( ios_base & str );`
3 Effects: Calls `str .unsetf(ios_base::floatfield)`
4 Returns: `str`

and to 27.4.5.2 adjustfield manipulators [lib.adjustfield.manip]

`ios_base & internal ( ios_base & str );`
1 Effects: Calls `str .setf(ios_base::internal, ios_base::adjustfield).`
2 Returns: `str .`

`ios_base & left ( ios_base & str );`
3 Effects: Calls `str .setf(ios_base::left, ios_base::adjustfield).`
4 Returns: `str .`
`ios_base & right ( ios_base & str );`
5 Effects: `Calls str .setf(ios_base::right, ios_base::adjustfield).`
6 Returns: `str .`

add

`ios_base & noadjust ( ios_base & str );`
1 Effects: Calls `str .unsetf(ios_base::adjustfield).`
2 Returns: `str .`

In passing, I also note that a ***possible*** interpretation of Table 85: fmtflags constants

Constant	Allowable values
adjustfield	left | right | internal
basefield	dec | oct | hex
floatfield	scientific | fixed

is that adjustfield MUST be either left, right, or internal, implying that if none of these are set, the implementation is non-conforming, (and similarly for basefield and floatfield). However 27.4.4.1 basic_ios constructors declares:

void init (basic_streambuf <charT ,traits >* sb);

Postconditions: The postconditions of this function are indicated in Table 90.

Since the only flags set by init are flags() skipws | dec, so the adjustfield and floatfields must both be zero. So one can infer that zero must be an allowed setting.

Table 85 could be made clearer by adding “0 | ” for floatfield and adjustfield (but NOT basefield) to read:

adjustfield	0 | left | right | internal
basefield	dec | oct | hex
floatfield	0 | scientific | fixed

Assuming basefield = 0 is NOT a Standard value?

For float fields, the effect of the default zero, neither fixed nor scientific, is to chose the most appropriate format for the value.

It might be useful to clarify if other values of these fields are undefined behaviour or, much better in my view, implementation defined. (Since the iword/pword system is so awkward to use, allowing all additional bit combinations seems sensible).