I/O Stream Manipulator for Binary Integers

Document #:	P00000R0
Date	2025-02-25
Programming Language C++	
Audience	LEWG Incubator, SG16

Reply-To	Javier Estrada <javier.estrada@outlook.com>

1 Introduction

The **`std::dec`**, **`std::oct`** and **`std::hex`** standard manipulators for integrals numbers address decimal, octal and hexadecimal bases, input/output, respectively. However, there is no current solution for binary input/output, nor for binary literals.

2 Motivation and Scope

The goal and motivation of this paper is to fill a gap in the I/O streams representation without resorting to non-standard solutions. A solution that takes advantage of the existing I/O infrastructure is preferred.

Consider the following motivating example:

```
[Example 1:
#include <bitset>
#include <iostream>

int main() {
  // current solution for output, no solution for input
  std::cout << "The number 42 in binary: " << std::bitset<8>{42} << '\n';
  return 0;
}
— end example]
```

The previous example yields the following output:

```
The number 42 in binary: 00101010
```

The “bitset trick”, as it is known, only takes care of output, without consideration for width or formatting, showing or hiding the binary base. A standard solution would make use of a **`std::bin`** manipulator as follows:

```
[Example 2:
#include <iomanip>
#include <iostream>

using std::setw, std::setfill, std::showbase;

int main() {
  // take advantage of existing infrastructure
  std::cout << setw(8) << setfill('0') << showbase;
  std::cout << "The number 42 in binary: " << std::bin << 42 << '\n';
  return 0;
}
— end example]
```

The previous example would yield the following output, accounting for width, fill and base:

```
The number 42 in binary: 0b00101010
```

A similar example for binary input:

```
[Example 3:
#include <iomanip>
#include <iostream>

int main() {
  int n{};
  std::istringstream is("101010");

  is >> std::bin >> n;

  std::cout << "Parsing \"101010\" as binary: " << std::bin << n << '\n';
  return 0;
}
— end example]
```

The previous example would yield the following output:

```
Parsing "101010" as binary: 101010
```

3 Impact on the Standard

This is purely a library addition, requiring no changes to the language. It can be implemented using C++23 compilers with existing library features. The following sections and tables would be affected. However, these may not be the only places, but anywhere that the **`ios::basefield`** flags I/O for integral values are checked, most likely an additional check for **`ios_base::bin`** would need to be added.

This presumes that:

- the additional flag fits in the current bitmask used for the existing flags,
- already compiled operators wouldn't check the new flag.[1]

Additional changes may be needed for the **`num_get<>`** and **`num_put<>`** classes, not covered in this document, except what has been identified in section 3.6 of this document.

[1] Thanks to Jan Schultke for highlighting this.

3.1 Section 30.4.3.3.3

Table 110 adds two rows for binary output, **`showbase, noshowbase`** manipulators.

Table 110: Integer conversion [tab:facet.num.put.int]

State	`stdio` equivalent
`(basefield == ios_base::bin) && !uppercase`	`%b`
`(basefield == ios_base::bin)`	`%B`
`basefield == ios_base::oct`	`%o`
`(basefield == ios_base::hex) && !uppercase`	`%x`
`(basefield == ios_base::hex)`	`%X`
for a **`signed`** integral type	`%d`
for an **`unsigned`** integral type	`%u`

3.2 Section 31.5.1 Header <ios> synopsis

A new **`basefield`** would be added:

```
      // 31.5.5.3, basefield
+     ios_base& bin (ios_base& str);
      ios_base& dec (ios_base& str);
      ios_base& hex (ios_base& str);
      ios_base& oct (ios_base& str);
```

3.3 Section 31.5.2.1 General [ios.base.general]

A new format flag would be added in the class declaration for **`fmtflags`**:

```
      // 31.5.2.2.2, fmtflags
      using fmtflags = T1 ;
      static constexpr fmtflags boolalpha = unspecified ;
+     static constexpr fmtflags bin = unspecified ;
      static constexpr fmtflags dec = unspecified ;
      static constexpr fmtflags fixed = unspecified ;
      static constexpr fmtflags hex = unspecified ;
```

3.4 Table 118: `fmtflags` effects [tab:ios.fmtflags]

A new row is added before the **`boolalpha`** element to account for the **`ios_base::bin`** flag.

Table 118: Integer conversion [tab:facet.num .put.int]

Element	Effect(s) if set
`bin`	converts integer input or generates integer output in binary base
`boolalpha`	insert and extract the **`bool`** type in alphabetic format
Rest of the table stays as is.	

3.5 Section 31.5.5.3 `basefield` manipulators

Two paragraphs are added after paragraph 7 to describe the **`std::bin`** manipulator:

```
ios_base& bin(ios_base& str);
```

8. *Effects*: Calls **`str.setf(ios_base::bin, ios_base::basefield).`**
9. *Returns*: **`str.`**

3.6 Section 31.7.6.3.2 Arithmetic inserters

The example code in paragraph 1 dealing with **`num_get<>`** and **`num_put<>`** would need an additional flag check:

When **`val`** is of type **`short`** the formatting conversion occurs as if it performed the following code fragment:

```
ios_base::fmtflags baseflags = ios_base::flags() & ios_base::basefield;
   bool failed = use_facet<
      num_put<charT, ostreambuf_iterator<charT, traits>>
         >(getloc()).put(*this, *this, fill(),
         baseflags == ios_base::bin || ios_base::oct || baseflags == ios_base::hex
            ? static_cast<long>(static_cast<unsigned short>(val))
            : static_cast<long>(val)).failed();
```

When **`val`** is of type **`int`** the formatting conversion occurs as if it performed the following code fragment:

```
ios_base::fmtflags baseflags = ios_base::flags() & ios_base::basefield;
   bool failed = use_facet<
      num_put<charT, ostreambuf_iterator<charT, traits>>
         >(getloc()).put(*this, *this, fill(),
         baseflags == ios_base::bin || ios_base::oct || baseflags == ios_base::hex
            ? static_cast<long>(static_cast<unsigned int>(val))
            : static_cast<long>(val)).failed();
```

3.7 Section 31.7.7

Paragraph 4 would add an additional flag to check:

unspecified **setbase(int base);**

4. *Returns*: An object of unspecified type such that if **out** is an object of type **basic_ostream<charT, traits>** then the expression **out << setbase(base)** behaves as if it called **f(out, base)**, or if **in** is an object of type **basic_istream<charT, traits>** then the expression **in >> setbase(base)** behaves as if it called **f(in, base),** where the function **f** is defined as:

```
   void f(ios_base& str, int base) {
     // set basefield
-    str.setf(base == 8 ? ios_base::oct :
+    str.setf(base == 2 ? ios_base::bin :
+    base == 8 ? ios_base::oct :
     base == 10 ? ios_base::dec :
     base == 16 ? ios_base::hex :
     ios_base::fmtflags(0), ios_base::basefield);
   }
```

The expression **out << setbase(base)** has type **basic_ostream<charT, traits>&** and value **out**.
The expression **in >> setbase(base)** has type **basic_istream<charT, traits>&** and value **in**.

4 Design Decisions

Given that this proposal relies on an existing design and infrastructure, designs decisions are not applicable.

5 Technical Specifications

The sections, tables and paragraphs referenced in section 3 make use of the latest C++23 draft, namely N4950. Those elements are subject to change if they have shifted in the current draft (C++26).

6 Acknowledgements

Thanks to Chris Ryan, Victor Zverovich, Jonathan Wakely, and Jan Schultke for their input and corrections.

7 References

N4950, Working Draft, Standard for Programming Language C++

Langer, Angelika, Kreft, Klaus, *Standard C++ IOStreams and Locales: Advanced Programmer's Guide and Reference*, Addison Wesley Professional, 2000.