My Position about the templatized Iostream for Austin Meeting

Norihiro Kumagai
Sharp Corporation,
IPSJ/ITSCJ/SC22/C++WG

In this paper, I discuss for the two documents about Iostream (X3J16/94-0203, "Iostream motion for Valley Forge by Jerry Schwarz", and X3J16/94-0197, "Comments for Jerry's proposals about the Templatized Iostream", by me) in order to make my position clear for Austin Meeting.

Conclusion

1) I agree with Jerry's proposal of X3J16/94-0203, under the following condition:

1a) introduce wstreampos, wstreamoff in the wchar_t version of ios_traits (relating to "Proposal 1").
1b) move fill member function from the ios_base to basic_ios (relating to "Proposal 8").

2) I agree with Jerry's proposal of deleting basic_convbuf from the Working Paper (see X3J16/94-0163).

3) I summarize my paper of X3J16/94-0197 as specifying locale-dependency of basic_filebuf default implementation. I propose the following modification for the Working Paper so as to achieve it:

3a) rewrite the description of the basic_filebuf so as to clarify that implements shall perform conversion between external source/sink byte stream in a file and charT stream in the put/get buffer with the codecnv facet in the locale object imbued in the basic_filebuf.
3b) introduce two member functions:

```cpp
locale imbue (locale new_loc);
locale getloc () const;
```

to the `basic_streambuf` (not to `basic_filebuf`) so as to achieve synchronization of changing locale value between a `basic_istream`/`basic_ostream` object and its related `basic_streambuf` object.

3c) introduce the following attribute(one of the private members) in the `basic_filebuf`:

```cpp
state_type state;
```

3d) [optional] introduce a new value to the return value and modify the behavior of `codecnv` facet of the `locale` class so as to avoid unnecessary copying to the `basic_filebuf` implementation.

**Discussion**

1: Jerry's document "Iostream motion for Valley Forge"

(X3J16/94-0203)

---

Proposal 1: Reorganize the "traits" classes as follows:

```
....
Allow (but do not require) an implementation to specialize `ios_traits<char>`.

`ios_traits<wchar_t>` shall have the default values except that `int_type` shall be `wint_t`.
```

I agree with it other than the default values of `ios_traits<wchar_t>`. Instead of the above description, I believe we should specify:

```
`ios_traits<wchar_t>` shall have the default values except that the following:
* `int_type` shall be `wint_t`,
* `pos_type` shall be `wstreampos`,
```

- 2 -
* off_type shall be wstreamoff.

We should leave freedom to choose a separate implementation of widechar version of Iostream from the skinny character version. In a shift encoding environment, pos_type and off_type should hold a shift state as well as file position offset information. I am afraid that if the Standard imposed streampos, streamoff for pos_type, off_type in widechar version as well as in skinny character version, we should have given up to enjoy the 32bit range of seek facility under some shift encoding environment.

Proposal 1a: Add traits to ios_traits.

Agree.

Proposal 2: Change the declaration of not_eof to ...

Agree.

Proposal 3: Clarify that POS_T and OFF_T are merely shorthand notations ...

Agree.

Proposal 4: Rewrite the requirement on pos_type and off_type ...
Proposal 5: Determining the offset associated with ...
Proposal 6: Explicitly allow streampos and streamoff to be the same type.
Proposal 7: The identity, streampos(streamoff (n)) == n; is required...

Agree.

Proposal 8: Introduce a class containing some of the types, constants and functions that are currently declared in basic_ios, but which do not depend on the template parameters of basic_ios.

I agree with the opinion. I have found that the ios_base definition appeared in Jerry's document had included the fill member function although it has int_type as its argument type, which closely relates to the template parameters of basic_ios. I believe the fill function should be moved to basic_ios.
Proposal 8: Change specification for various types to use typedefs rather than derived classes.

```cpp
typedef basic_ios<char> ios;
typedef basic_ios<wchar_t> wios;
...
```

Agree.

Proposal 9: A new header `iosfwd` that will contain an incomplete declarations of `basic_ios, ios, wios, basic_istream, istream, wistream, basic_ostream, ostream` and `wostream`.

Agree.

Proposal 12: Declare wide "standard stream" `win, wout, werr` and `wlog` with streambufs directed to the same sources/sinks of characters as the skinny standard streams.

Mixing operations on corresponding wide and skinny streams (or stdio `FILES`) will follow the same semantics as mixing wide and skinny operations as in amendment 1 of the ISO C standard.

Agree. I much appreciate the specification of the mixing operation.

Proposal 13: Eliminate `write_byte` and `read_byte` from the WP.

Agree.

Proposal 14: Describe `pbackfail` using a style consistent with the descriptions of the default (protocol) behavior of `underflow` and `overflow` as currently present in editorial box 154 of the WP.

Agree.

Proposal 15: Add a sentence ...
Proposal 16: Delete all functions from the working paper that mention `signed charT` or `unsigned charT` is the template class parameter. And add editorial boxes noting that the existing code which calls these functions when T is `char` may quietly change: actual arguments of type `signed char` or `unsigned char` will resolve to `int` instead.

Agree.

I think the resulting declaration of `ios_traits` are as follows:

```cpp
template <class charT, class traits=ios_traits<charT> >
class ios_traits { 

public:

typedef charT char_type;
typedef int int_type; // 94-0203
static char_type to_char_type (int_type c);
static int_type to_int_type (char_type c);
static bool eq_char_type (char_type, char_type);
static bool eq_int_type (int_type, int_type);
static int_type eof ();
static int_type not_eof (char_type c); // 94-0203
static bool is_eof (int_type);
static char_type newline ();
static bool is_whitespace (ctype<char_type>, char_type);
static char_type eos ();
static size_t length (); // streamsize?
static char_type* copy (char_type* dst,
    const char_type* src, size_t n); // size_t --> streamsize?

  typedef streampos pos_type; // 94-0203
  typedef streamoff off_type; // 94-0203
  typedef T1 state_type; // 94-0197(1-1) && 94-0203,
      // but WP shows int instead of T1.
```
2: Deleting basic_convbuf

I decided to accept Jerry's response about extensibility of conversion functionality (see c++std-lib-3515). I understand his idea to provide the extensibility by means of mbstreambuf class and agree with him that the Standard has enough extensibility without basic_convbuf. So I agree to delete it from the Working Paper.

3: basic_filebuf(X3J16/94-0203)

In general, we have to implement a conversion between charT sequence and byte sequence in a templatized streambuf class whose external source/sink is a byte stream. Note that this is not only the case in wide characters (wchar_t) but it is the common case for any charT specialization's.

In the Working Paper, the basic_filebuf class is the only templatized streambuf class whose external is byte-oriented. In order to implement the basic_filebuf, we have to generate a charT sequence for filling the get buffer from the byte sequence in a file and to generate a byte sequence from the charT sequence in the put buffer in the
protected virtuals (underflow, uflow, and overflow) of the basic_filebuf. Therefore, we need some templatized method to generate a charT sequences from a byte sequence and to generate a byte sequence from a charT sequence in order to implement basic_filebuf. Because all of the templatized method needed for implementation shall be specified in the Standard by some means(either in a template argument, in a traits, or through another templatized functions), we shall specify templatized conversion functions as well.

Deletion of basic_convbuf from the Working Paper will cause the basic_filebuf to lose the templatized conversion functions. So we have to do something to recover a templatized conversion functions to it.

There are at least three alternatives to provide templatized conversion functions as follows:

* use the convert member function of the codecvt facet in the locale class,
* use the widen/narrow member functions of the ctype facet in the locale class,
* introduce a new templatized class for conversion.

Among the above three, the widen/narrow approach is based on the concept of conversion between one byte and one character and provides no ways to handle shift states. So it is not appropriate as the solution.

The specification of convert member function of the codecvt facet is as follows(see 22.2.7):

```cpp
result convert (stateT& state,
    const fromT* from, const fromT* f_end, const fromT*& f_next,
    toT*   to,       toT* t_end,    toT*& t_next) const;
```

In the above interface, fromT, toT, and stateT is the template parameters of the codecvt facet. This function provides an in-core memory to memory conversion between arbitral two character classes, fromT and toT. It also has the stateT as one of the template parameter so as to customize shift state definition necessary to support arbitral shift encodings. In case the parameter fromT is fixed to char, the convert function is available to convert from a byte sequence to a charT sequence in a basic_filebuf implementation. So it is suitable for one of the solutions.

The 'mbstate' class, a new templatized class for conversion introduced in my
message, c++std-lib-3511, is functionally equivalent to the convert member function of the codecnv facet. It provides an in-core memory to memory templatized conversion and can handle shift state by deriving from it, too. Instead of introducing 'yet another' redundant functionality to the Standard, I will give up the mbstate and follow an approach based on the codecnv facet in order to specify templatized conversion functionality to the basic_filebuf.

3.1 Synchronization between templatized istream/ostream and templatized streambuf:

Changing the locale value in a basic_ifstream/basic_ofstream object causes to change the encoding scheme for the its basic_filebuf object(rdbuf) if the new locale object provides different codecnv facet object to the previous one.

So the basic_filebuf shall have a locale object (or at least a codecnv facet object) as one of the attributes and shall have a member function to alter its locale attribute value. So I might propose the following two members, imbue and setloc and one attribute, loc (for exposition only), as same declaration as in the ios_base:

template <class charT, class traits=ios_traits<charT> >
class basic_filebuf : public basic_streambuf {
public:
    ....
    locale imbue (locale loc);
    // alter the locale attribute value. Mainly for
    // implementing basic_ifstream::imbue.
    locale getloc () const;
    // get the locale attribute value.

private:
    locale loc;   // for exposition only
    ....
};

Exactly speaking, this specification is not enough to implement the ios_base::imbue, because the implementation of ios_base:imbue will refer the stream pointer (the return value of rdbuf()), as the basic_streambuf class pointer, not as the basic_filebuf pointer. Therefore, the class basic_streambuf shall have two member, imbue and getloc, and one attribute, loc.
Although introducing a locale value to the `basic_streambuf`, it does not affect the behavior of other member functions than `imbue`, `getloc` at all because it is not useful unless its external source/sink is byte-oriented. It affects only the behavior of the `basic_filebuf` in the Standard.

### 3.2 Behavior of the `basic_filebuf`

Once we decide the `basic_streambuf` has a locale value as an attribute, we can specify the behavior of the `basic_filebuf` as follows:

* In `underflow, uflow` member function, before acquiring `charT` values to fill the get buffer a conversion with the `codecnv` facet of the attribute `loc` as if by invoking the following function:

  ```
  loc.use<locale::codecnv<char, charT, state_type> >().convert
  (state_type& state,
   const char* from, const char* f_end, const char*& f_next,
   charT* to,    charT* t_end,     charT*& t_next);
  ```

  the input buffer beginning pointed to by `from` and the next byte pointed to by `f_end` filled with the next of the external source byte sequence in a file.

* In `overflow` member function, before writing the byte sequence to the external sink byte sequence in a file, a conversion with the `codecnv` facet of the attribute `loc` as if by invoking the following function:

  ```
  loc.use<locale::codecnv<charT, char, state_type> >().convert
  (state_type& state,
   const charT* from, const charT* f_end, const charT*& f_next,
   char* to,      char* t_end,    char*& t_next);
  ```

  with the `charT` character sequence from the put buffer.

* Another private member, state, is needed to implement the above behaviors.

  ```
  state_type state;
  ```
3.3 Performance improvement

Although it is inevitable some conversion in the default behavior in the basic_filebuf class, we would like to avoid the conversion function invocation and relating handling charge in some specialization's where no conversion needed (for example, filebuf in the ASCII environment). In the ASCII environment, at least we would like to do without unnecessary copying byte sequence from pointed to by from to to in the convert function of the codecnv facet.

Fundamentally, there is no right to judge the necessity of conversion in the basic_filebuf protected virtuals because it depends on the feature of the codecnv facet.

In order to allow a convert function implementation where no copying occurs to return immediately, we can introduce the means for the convert function to notify its decision of no-copying and availability of the source data to the basic_filebuf. When a convert function invocation results to notify the no-copying occurrence, the invoker, one of the basic_filebuf protected virtuals is pleased to use the data pointed to by from instead of those pointed to by to to fill its get buffer, or to perform a physical write operation to the file.

We have already had an enum, result, in the codecnv declaration, so as to utilize it as the good means to the notification. We can introduce the forth value, noconv in the enum result to do so. In case the convert returns the noconv value, the protected virtuals use the sequence between pointed to by from and by f_next instead of those between by to and by t_next.

* introduce a new value, noconv, in the enum, locale::codecnv::result.

```
enum result { ok, partial, error, noconv };
```