Doc. No. : WG21 N0957R1
x3j16 96-0139R1
Date: July 27, 1996
Project: Programming Language C++
Reply to : Jerry Schwarz
`jerry@intrinsa.com`

pos_type and off_type

Jerry Schwarz

The descriptions of get_state, and get_pos traits in the current WP are unclear. The description of get_pos refers to fpos, which is not mentioned anywhere else in the WP. At the library wg mini-meeting in May there was some discussion of this problem and I made a proposal for a “cleanup” that included a name change and new traits. No consensus was reached and I agreed to prepare a formal analysis and proposal for Stockholm. As I reflected on it I came to realize that not only was my proposal seriously flawed, but the treatment of these traits in the WP needed to be significantly modified. This paper contains my analysis of the problem and a proposal that differs significantly from the one I made at the mini-meeting.

Why are pos_type and off_type traits?

If users are going to define their own arbitrary streambuf types (which was always a goal of iostreams) they should to be allowed to define their own positioning types. In classic iostreams that wasn’t possible. The positioning types were fixed to be `streampos` and `streamoff` and no variation was possible. A symptom of the problem was the universal assumption that a `streampos` would embed an `fpos_t` for use by `filebuf`. `filebuf`, by virtue of being part of the library, exercises a privilege that isn’t available to user defined streambuf’s. That special treatment is required is a symptom of a weakness in the design of classic iostreams, but one that was not easy to fix.

When the committee templatized the character type in iostreams it seemed a good opportunity to address this issue, which we did by including pos_type and off_type as traits. However the nature of these traits is different from that of other traits and this difference has, so far, not been reflected in the working paper. The difference has to do with of who is imposing constraints and who is using the generic interface. The traits are designed to allow generic coding of the classes (strings, streams, streambufs, ...) that manipulate characters (charT’s). The purpose of constraints on these traits is similar to the constraints on STL iterators or collections, they enable generic manipulations by template classes. The assumption is that all the operations that a stream or streambuf needs to perform on character types can be synthesized from the traits.

The assumptions with regards to `pos_type` and `off_type` go in the opposite direction. The users of a stream or streambuf should be able to write their code generically using only the constraints imposed by the WP, but the implementors of a specific streambuf class may need specific operations that are possible only with specific `pos_type`’s and `off_type`’s. The canonical example is, as usual, `filebuf`. The `pos_type` for a `filebuf` must embed specific information needed for `positioning` on a specific operating system. The implementation of `filebuf` must be allowed to assume that pos_type is (or is related to) `fpos_t`.

The conclusion from these considerations is that implementations must be allowed to specify which type `pos_type`'s and `off_type`'s are allowable for use in the standard streambuf classes (`filebuf`, `stringbuf.`)

Proposal A

Delete the traits get_state and get_pos from the character trait from table 47 in 21.1.2[lib.char.traits.require].

The description of these traits in the current WP doesn't make sense, and I see no way to make sense of it. The intended use of these traits is unclear and they are not otherwise referred to in the WP.

Proposal B

Add to 27.1.2[lib.iostreams.type.reqmts] the sentence

> The classes of this clause with template arguments `charT` and `traitsT` behave as described when `traitsT::pos_type` and `traitsT::off_type` are `streampos` and `streamoff` respectively. Their behavior when t`raits::pos_type` and traitsT::off_type are other types is implementation dependent.

state_type

Through facets of the locale class the WP supports arbitrary conversions between the character type of a `basic_filebuf` instance and the `char's` (or other type) actually stored in a file. These operations contemplate that a "position" will combine a "conversion state" and a "file position" in some way. There are sound reasons (given above) that the "file position" cannot be arbitrary . But If a program is going to supply conversion facets that rely on state information it will need more specific information about the type. The committee has gone to some trouble to ensure that the conversion is entirely encapsulated in the conversion facets.

In the present scheme the conversion state is a trait(`state_type)`. This trait is not included in proposal B because the clear intention is that `basic_filebuf` should be written without specific knowledge of `state_type` (which is encapsulated in a `locale` facet) and should work with an arbitrary `state_type`.

Proposal C

Add to (a new subsection of) 27.4 [lib.iostreams.base] a declaration of a class

```
template<class stateT> class fpos {
public:
      fpos(stateT);
      stateT state() const;
      void state(stateT);
private:
      stateT st; // exposition only
};
```

And definitions

`fpos(stateT s);`
Effects: Constructs an object and assigns s to st.
[Note: an fpos will also contain private information needed to position a file. This information will be initialized in an unspecified fashion]

`void state(stateT s);`
Effects: assign s to st.

`stateT state()`
Returns: st.

Add to 27.8.1.1[lib.filebuf] the sentence

An instance of basic_filebuf behaves as described in this clause provided `traits::pos_type` is `fpos<traitsT::state_type>`. Otherwise the behavior is undefined.

Assuming that N0954R1 is accepted, replace the declaration of streampos that paper adds to iosfwd in 27.2[lib.iostream.forward]

```
typedef fpos<char> streampos;
```

Move table 93(currently in editorial box 106) to (a new subsection of) 27.2 [lib.iostream.fwd] and add text

In the following table
-- P refers to an instance of template class `fpos`.
-- p and q refer to a value of type P
-- O refers to streamoff
-- o refers to a value of type streamoff
-- I refers to a value of type int

The operations specified in this table are permitted.

Stream operations that return a value of traitsT::pos_type return P(O(-1)) as an invalid `fpos` to signal an error. If this value is used as an argument to any stream or streambuf member that accepts a value of type traitsT::pos_type, then the behavior of that function is undefined.