By: Philippe Le Mouël
Rogue Wave Software Inc.
philippe@roguewave.com

Doc. No.: X3J16/96-0010
WG21/N0828
Date: January 30 1996

Bi-directional Iostreams Proposal

Introduction

This proposal describes the template classes needed to add the bi-directional streams object to chapter 27 « Input/output library ». The interface of each class is given as well as a short description of each of their member functions. The purpose of adding these new classes to the standard is twofold. First, maintains a backward compatibility with the old iostream, which provides bi-directional objects of type *fstream* and *strstream*. Second, it makes the « Input/output library » more user friendly (especially for beginners).

Synopsis of the bi-directional classes

Template class basic_iostream:

```
template<class charT, class traits = ios_traist<charT>>
class basic_iostream
: public basic_istream<charT,traits>, public basic_ostream<charT,traits>

typedef basic_iostream<char>    iostream;
typedef basic_iostream<wchar_t>  wiostream;
```

Template class basic_fstream

```
template<class charT, class traits = ios_traist<charT>>
class basic_fstream
:public basic_iostream<charT,traits>

typedef basic_fstream<char>    fstream;
typedef basic_fstream<wchar_t>  wfstream;
```

Template class basic_stringstream

```
template<class charT, class traits = ios_traist<charT>>
class basic_stringstream
:public basic_iostream<charT,traits>

typedef basic_stringstream<char>    stringstream;
typedef basic_stringstream<wchar_t>  wstringstream;
```

Class strstream

```
class strstream
:public basic_iostream<char>
```

Advantages of these new classes

The new classes add user friendliness to the « Input/Output library » while keeping its powerful template interface and they maintain backward compatibility with the old iostream. For instance, if you want to open a file on tiny characters in read/write mode with the current set of classes, you will have to write:

```
ofstream out("filename",ios_base::in | ios_base::out);
istream   in(out.rdbuf());
```

or

```
ifstream  in("filename", ios_base::in | ios_base::out);
ostream  out(in.rdbuf());
```

With the bi-directional classes you just have to write:

```
fstream inout("filename");
```

Here is the same example for a *stringstream* object on a user-defined character type called *char_user*.

Current set of classes:

```
basic_ostringstream<char_user,ios_traits<char_user>>   out(ios_base::in | ios_base::out);
basic_istream<char_user,ios_traits<char_user>>          in(out.rdbuf());
```

or

```
basic_istringstream<char_user,ios_traits<char_user>>    in(ios_base::in | ios_base::out);
basic_ostream<char_user,ios_traits<char_user>>           out(in.rdbuf());
```

With the bi-directional classes you just have to write:

basic_stringstream<char_user,ios_traits<char_user>> inout;

If you want to read and write into your objects, you write the following code:

Current set of classes:

```
out << x << y;
in >> z;
```

Bi-directional classes:

```
inout << x << y;
inout >> z;
```

Therefore old iostream code such as follow will work with the new « Input/Output library » if we add the bi-directional streams.

```
char c;
int x,y,z;

fstream inout("myfile");

inout << x << y;
inout >> z;
inout.get(c);
```

Template class basic_iostream

```
namespace std {
  template<class charT, class traits = ios_traits<charT> >
  class basic_iostream
  : public basic_istream<charT, traits>, public basic_ostream<charT, traits>  {

  public:
 // constructor/destructor:
   explicit basic_iostream(basic_streambuf<charT,traits> *sb);
   virtual ~basic_iostream();
 };
}
```

The class *basic_iostream* inherits a number of functions that assist in reading input and writing output to sequences controlled by a stream buffer.

basic_iostream constructors

explicit basic_iostream(basic_streambuf<charT,traits> *sb);

Effects: Constructs an object of class *basic_iostream*, assigning initial values to the base classes by calling *basic_istream<charT,traits>(sb)* (27.6.1.1.1) and *basic_ostream<charT,traits>(sb)* (27.6.2.2)
Postcondition: *rdbuf( ) == sb and gcount( ) == 0*

virtual ~basic_iostream();

Effects: Destroys an object of class *basic_iostream*.
Notes: Does not perform any operations on *rdbuf()*

Template class basic_fstream

```
namespace std {
  template<class charT, class traits = ios_traits<charT> >
  class basic_fstream
  : public basic_iostream<charT, traits>     {

  public:
  // Types
    typedef charT                          char_type;
    typedef typename traits::int_type      int_type;
    typedef typename traits::pos_type      pos_type;
    typedef typename traits::off_type      off_type;

 // constructors/destructor
    basic_fstream( );
    explicit basic_fstream(const char* s,
                           ios_base::openmode mode = ios_base::in | ios_base::out );

 // Members:
    basic_filebuf<charT,traits>* rdbuf( ) const;

  bool is_open( );
  void open(const char* s, ios_base::openmode mode = ios_base::in | ios_base::out );
  void close ( );

 private:
```

```
 //  basic_filebuf<charT,traits> sb;       exposition only
  };
}
```

The template class *basic_fstream<charT, traits>* supports reading and writing from named files. It uses a *basic_filebuf<charT,traits>* object to control the associated sequence. For the sake of exposition, the maintained data is presented here as:

- *sb*, the *filebuf* object.

basic_fstream constructors

basic_fstream( );

Effects: Constructs an object of class *basic_fstream<charT, traits>*, initializing the base class with *basic_iostream(&sb)* and initializing *sb* with *basic_filebuf<charT, traits>( ).*

explicit basic_fstream(const char* s,
ios_base::openmode mode = ios_base::in | ios_base::out);

Effects: Constructs an object of class *basic_fstream<charT, traits>*, initializing the base class with *basic_iostream(&sb)* and initializing *sb* with *basic_filebuf<charT, traits>( )*, then calls *rdbuf()->open(s,mode).*

Member functions

basic_filebuf<charT, traits>* rdbuf( ) const;

Returns: *(basic_filebuf<charT,traits>*) &sb.*

bool is_open( );

Returns: *rdbuf( )->is_open( ).*

void open(const char* s, ios_base::openmode mode = ios_base::in | ios_base::out);

Effects: Calls *rdbuf( )->open(s,mode)*, then if *is_open( )* returns false, calls *setstate(failbit)* (which may throw *ios_base::failure*(27.4.4.3)).

void close( );

Effects: Calls *rdbuf( )->close( )* and, if that function returns false, calls *setstate(failbit)* (which may throw *ios_base::failure*(27.4.4.3)).

Template class basic_stringstream

```
namespace std {
  template<class charT, class traits = ios_traits<charT> >
  class basic_stringstream
  : public basic_iostream<charT, traits>     {

  public:
  // Types
    typedef charT                          char_type;
    typedef typename traits::int_type      int_type;
    typedef typename traits::pos_type      pos_type;
    typedef typename traits::off_type      off_type;

 // constructors/destructor
   explicit basic_stringstream(ios_base::openmode which = ios_base::out | ios_base:: in );
   explicit basic_stringstream(const basic_string<charT>& str,
                               ios_base::openmode mode = ios_base::in | ios_base::out );

 // Members:
   basic_stringbuf<charT,traits>* rdbuf( ) const;

   basic_string<charT> str( ) const;
   void str(const basic_string<charT>& str);

 private:
 //  basic_stringbuf<charT,traits> sb;      exposition only
  };
}
```

The template class *basic_stringstream*<*charT, traits*> supports reading and writing from objects of class *basic_string*<*charT,traits*>. It uses a *basic_stringbuf*<*charT,traits*> object to control the associated sequence. For the sake of exposition, the maintained data is presented here as:

- *sb*, the *stringbuf* object.

basic_stringstream constructors

explicit basic_stringstream(ios_base::openmode which = ios_base::out | ios_base:: in);

Effects: Constructs an object of class *basic_stringstream<charT, traits>*, initializing the base class with *basic_iostream(&sb)* and initializing *sb* with *basic_stringbuf<charT, traits>(which)*.

explicit basic_stringstream(const basic_string<charT>& str,
ios_base::openmode mode = ios_base::in | ios_base::out);

Effects: Constructs an object of class *basic_stringstream<charT, traits>*, initializing the base class with *basic_iostream(&sb)* and initializing *sb* with *basic_stringbuf<charT, traits>(str, which)*.

Member functions

basic_stringbuf<charT, traits>* rdbuf( ) const;

Returns: *(basic_stringbuf<charT,traits>*) &sb.*

Basic_string<charT> str( ) const;

Returns: *rdbuf( )->str( ).*

void str(const basic_string<charT>& str);

Effects: Calls *rdbuf( )->str(str).*

Class strstream

```
namespace std {
  class strstream
  : public basic_iostream<char>     {

  public:
  // Types
    typedef char                                    char_type;
    typedef typename ios_traits<char>::int_type     int_type;
    typedef typename ios_traits<char>::pos_type     pos_type;
    typedef typename ios_traits<char>::off_type     off_type;

 // constructors/destructor
    strstream( );
    strstream(char* s, int n, ios_base::openmode mode = ios_base::in | ios_base::out);
    virtual ~strstream( );
```

```
  // Members:
    strstreambuf* rdbuf( ) const;

    void freeze(int freezefl = 1);
    int pcount( ) const;
    char * str( );

 private:
 //  strstreambuf sb;       exposition only
  };
}
```

The class *strstream* supports reading and writing from objects of class *strstreambuf*. It supplies a *strstreambuf* object to control the associated array object. For the sake of exposition, the maintained data is presented here as:

- *sb*, the *strstreambuf* object.

strstream constructors

strstream( );

Effects: Constructs an object of class *strstream*, initializing the base class with *iostream(&sb)* and initializing *sb* with *strstreambuf( )*.

strstream(char* s, int n, ios_base::openmode mode = ios_base::in | ios_base::out);

Effects: Constructs an object of class *strstream*, initializing the base class with *iostream(&sb)* and initializing *sb* with one of the two constructors:

- If *mode & app == 0*, then s shall designate the first element of an array of n elements. The constructor is *strstreambuf(s, n, s)*.

- If *mode & app != 0*, then s shall designate the first element of an array of n elements that contains an NTBS whose first element is designated by s. The constructor is *strstreambuf(s,n,s+ ::strlen(s))*.

virtual ~strstream();

Effects: Destroys an object of class *strstream*.

Member functions

strstreambuf* rdbuf( ) const;

Returns: *(strsteambuf*) &sb*.

void freeze(int freezefl = 1);

Effects: Calls *rdbuf( )->freeze(freezefl);*

char * str( );

Effects: Calls *rdbuf( )->str( ).*

int pcount( ) const;

Returns: *rdbuf( )->pcount( ).*