Doc No: N3179=10-0169
Date: 2010-10-18
Authors: Pablo Halpern
Intel Corp..
phalpern@halpernwightsoftware.com

Move and swap for I/O streams (US138)

Contents

National Body comments and issues

This paper proposes a resolution for comment US 138 to the July, 2010 FCD.

Document Conventions

All section names and numbers are relative to the August 2010 WP, **N3126**.

Existing working paper text is indented and shown in dark blue. Edits to the working paper are shown with ~~red strikeouts for deleted text~~ and green underlining for inserted text within the indented blue original text.

Comments and rationale mixed in with the proposed wording appears as shaded text.

Requests for LWG opinions and guidance appear with light (yellow) shading. It is expected that changes resulting from such guidance will be minor and will not delay acceptance of this proposal in the same meeting at which it is presented.

Discussion

For basic_istream, basic_ostream, and basic_iostream, the move constructor does not do a move construction, tthe move-assignment operation does not do move-assignment and swap does not perform a swap. Moreover, these functions are protected, precluding their use in reasonable code.

The resolution to issue 900 (and related issue 911) assumes that these functions would never be called from client code. However, these classes are not abstract. They can be instanced and there are use-cases for such instances. For example, one can create a filebuf outside of an fstream, then associate it with an ostream:

```
filebuf fb("name");
ostream fstr(&fb);
```

The above `ostream` is a full-fledged object that should be movable, and `copyable`. In that case, the move and copy operations should move and copy the *whole* object, including the `rdbuf()` member.

However, moving the `rdbuf()` member poses a problem for derived classes like `fstream`, that contain embedded `streambuf` objects and which want to ensure that the base class portion of the copy container a pointer to a copy of the streambuf, not a pointer to the original streambuf. However, such a problem can be overcome easily simply by calling `basic_ios::set_rdbuf` to set the stream buffers to their correct values. In order for that to work, the `iostream` functions to move and swap must not modify the stream buffers in any way. This requirement is already met by every implementation I have seen and needs simply to be documented as a standard requirement.

The resolution proposed here differs from that in the text of US138, which propsed a new set of constructors and a new function to perform the same actions as copy-construction, move-construction, and move-assignment, but do not move or copy the rdbuf() pointer.

Proposed Wording

In section 27.5.4 [ios], rename `basic_ios` members `move` and `swap` to avoid confusion with functions that actually move and swap:

```
protected:
  basic_ios();
  void init(basic_streambuf<charT,traits>* sb);
  void partial_move(basic_ios& rhs);
  void partial_move(basic_ios&& rhs);
  void partial_swap(basic_ios& rhs);
  void set_rdbuf(basic_streambuf<charT, traits>* sb);
```

Make the same changes to the descriptions of the above functions in 27.5.4.2 [basic.ios.members] starting at paragraph 20 and add the "no touch" guarantee for the stream buffer:

```
void partial_move(basic_ios& rhs);
void partial_move(basic_ios&& rhs);
```

> *Postconditions*: *this shall have the state that `rhs` had before the function call, except that `rdbuf()` shall return 0. rhs shall be in a valid but unspecified state, except that rhs.rdbuf() shall return the same value as it returned before the function call, and `rhs.tie()` shall return 0. A call to this function does not modify the stream buffers pointed-to by either `this->rdbuf()` or `rhs.rdbuf()`.

```
void partial_swap(basic_ios& rhs);
```

> *Effects*: The states of *this and rhs shall be exchanged, except that rdbuf() shall return the same value as it returned before the function call, and rhs.rdbuf() shall return the same value as it returned before the function call. A call to this function does not modify the stream buffers pointed-to by either `this->rdbuf()` or `rhs.rdbuf()`.
>
> *Throws*: Nothing.

In section 27.7.1.1 [istream], make the move constructor, move-assignment operator, and swap members public, and add a public swap function:

```
  protected:
    basic_istream(basic_istream&& rhs);
    // assign/swap
    basic_istream& operator=(basic_istream&& rhs);
    void swap(basic_iostream& rhs);
  };

  void swap(basic_istream& lhs, basic_istream& rhs);
}
```

Change 27.7.1.1.1 [iostream.cons]/3 as follows:

```
basic_istream(basic_istream&& rhs);
```

Effects: Move constructs from the rvalue `rhs`. This is accomplished by default constructing the base class, copying the `gcount()` from `rhs`, calling `basic_ios<charT, traits>::move(rhs)` to initialize the base class, setting the stream buffer with `set_rdbuf(rhs.rdbuf())`, and setting the rdbuf() and the gcount() for rhs to 0.

Change 27.7.1.1.2 [istream.assign]/3 as follows:

```
void swap(basic_istream& rhs);
```

Effects: Calls basic_ios<charT, traits>::swap(rhs). Exchanges the values returned by gcount() and rhs.gcount() and uses `set_rdbuf()` to exchange the values returned by `rdbuf()` and `rhs.rdbuf()`.

```
void swap(basic_istream& lhs, basic_istream& rhs);
```

Effects: calls `lhs.swap(rhs)`

In section 27.7.1.5 [iostreamclass], make the move constructor, move-assignment operator, and swap members public, and add a public swap function:

```
  protected:
    basic_iostream(basic_iostream&& rhs);
    // assign/swap
    basic_iostream& operator=(basic_iostream&& rhs);
    void swap(basic_iostream& rhs);
  };

  void swap(basic_iostream& lhs, basic_iostream& rhs);
}
```

Change the move constructor in section 27.7.1.5.1 [iostream.cons]/3:

```
basic_iostream(basic_iostream&& rhs);
```

Effects: Move constructs from the rvalue `rhs` by constructing the `basic_istream` base class with `move(rhs)` then setting `rdbuf()` with `set_rdbuf(rhs.rdbuf())` and clearing the `rhs` with `rhs.rdbuf(0)`.

Change swap in section 27.7.1.5.3 [iostream.assign]/2

`void swap(basic_iostream& rhs);`

Effects: `Calls basic_ios<charT, traits>::swap(rhs)` and uses `set_rdbuf()` to exchange the values returned by `rdbuf()` and `rhs.rdbuf()`.

`void swap(basic_iostream& lhs, basic_iostream& rhs);`

Effects: calls `lhs.swap(rhs)`

Change 27.7.2.1 [ostream] , make the move constructor, move-assignment operator, and swap members public, and add a public swap function:

```
~~protected:~~
   basic_ostream(basic_ostream&& rhs);
   // assign/swap
   basic_ostream& operator=(basic_ostream&& rhs);
   void swap(basic_ostream& rhs);
};

void swap(basic_ostream& lhs, basic_ostream& rhs);
```

Change 27.7.2.2 [ostream.cons]/5 as follows:

`basic_ostream(basic_ostream&& rhs);`

Effects: Move constructs from the rvalue rhs. This is accomplished by default constructing the base class, ~~and~~ calling `basic_ios<charT, traits>::move(rhs)` to initialize the base class, then setting `rdbuf()` with `set_rdbuf(rhs.rdbuf())` and clearing the `rhs` with `rhs.rdbuf(0)`.

Change swap in section 27.7.1.5.3 [iostream.assign]/2

`void swap(basic_ostream& rhs);`

Effects: `Calls basic_ios<charT, traits>::swap(rhs)` and uses `set_rdbuf()` to exchange the values returned by `rdbuf()` and `rhs.rdbuf()`.

`void swap(basic_ostream& lhs, basic_ostream& rhs);`

Effects: calls `lhs.swap(rhs)`

References

N3102: ISO/IEC FCD 14882, C++0X, National Body Comments