Subject: PipeStream - bidirectional Stream
Posted by dolik.rce on Fri, 28 Dec 2012 17:46:35 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Tue, 10 August 2010 18:26* Stream is unidirectional per definition and should be used as such. In Contrast to other Stream implementations, Upp Stream brings in all to be used both as Input or as Output stream. these 2 modes are supported in one single instance, but should't be used at same time.
nevertheless, it does not produce ASSERT, Exception or sth. if one tries to Put and Get stuff from same Stream, it simply might not be logical or what you expect, because Stream uses only one ptr to represent current 'head' position for reading or writing. (thus it is not intrinsically possible to use a MemStream as a Circular Buffer, which would be nice. btw, how about implementing such one . These 2 Modes can be differed using the API functions IsStoring() / IsLoading(). The Modes are set using SetStoring() / SetLoading() and are normally set automatically, depending on how you created the stream instance.

Hi guys,

The above quote from Konstantin caught my eye when I was looking in the manual for a way to create a bidirectional Stream. It turns out his ideas are quite possible to implement In the attached archive is a PipeStream class that implements circular buffer with Stream interface. The circular buffer can be fixed size or automatic resizing. I believe this class might be useful for example as a temporary storage for data that need to be passed from one interface to another, where the rates at which those interfaces produce/consume data differ and buffering is needed (e.g. pumping data from stdin to external library).

Internally the class only adds second pointer to manage read and write positions and takes care of allocating/reallocating/deleting the buffer. The buffering inherited from Stream is intentionally deactivated, because the data are buffered by definition and it would be sub-optimal to buffer it twice. Seeking is not possible, most of other Stream capabilities is working as expected. Complete documentation is included.

Example usage:#include <PipeStream/PipeStream.h>
using namespace Upp;

```
CONSOLE_APP_MAIN {
 PipeStream s(8, true);       // create with PipeStream with
                        // smallish buffer to demonstrate resizing
 int n;
 String t;
 StringBuffer b;

 s.Put('a');                  // writing to stream
 s.Put("bcd");
 s.Put(String("efghijklm"));
 ASSERT(s.GetReserved() > 8);  // the internal buffer grew because
                        // strlen("abcd") + strlen("efghijklm") > 8
```

```
 t = s.Get(5);                // reading from stream
 ASSERT(t == "abcde");
 t = s.Get(10);
 ASSERT(t == "fghijklm");
 t = s.Get(10);
 ASSERT(t.IsEmpty());         // nothing more to read...


 s.Put(String("nopqrs"));
 n = s.Peek();                // peeking
 ASSERT(n == 'n');
 n = s.Get();                 // reading single byte
 ASSERT(n == 'n');


 b.Cat("tuvwxyz");            // writing from buffer
 s.Put(~b, b.GetCount());

 b.SetCount(s.GetLeft());     // reading to buffer
 s.GetAll(~b, s.GetLeft());
 ASSERT(String(b) == "opqrstuvwxyz");
}
```

It all works well with current version of Stream, but I noticed there is couple of things that would make it work even better: If Stream::SetLoading() and Stream::SetStoring were virtual, it would make PipeStream work with serialization too. Not sure if it would be good for anything though...
Stream::IsLoading() and Stream::IsStoring() should be const.
It might need a bit more work, but to me it looks quite useful and usable. What do you think?

Best regards,
Honza

File Attachments

1) PipeStream.zip, downloaded 479 times

Subject: Re: PipeStream - bidirectional Stream
Posted by kohait00 on Mon, 07 Jan 2013 14:37:34 GMT
View Forum Message <> Reply to Message

nice thing

i myself started sth similar. will check back the code (is some tim e ago now) and let you know..

happy new year btw.

Subject: Re: PipeStream - bidirectional Stream
Posted by mirek on Mon, 28 Jan 2013 19:04:15 GMT
View Forum Message <> Reply to Message

Looking at the code, I am not quite sure that you interpret loading/storing flag correctly. It really is just a flag for serialization routines...

I think I would not bother implementing SetLoading / SetStoring in PipeStream. I believe that in typical scenario, only either input or output will do serialization, so it makes sense to set in just for single end of pipe... In that case, what is already in Stream should be enough.

(consts added).

Mirek

Subject: Re: PipeStream - bidirectional Stream
Posted by dolik.rce on Mon, 28 Jan 2013 20:08:20 GMT
View Forum Message <> Reply to Message

Hi Mirek

mirek wrote on Mon, 28 January 2013 20:04Looking at the code, I am not quite sure that you interpret loading/storing flag correctly. It really is just a flag for serialization routines...I'm well aware that I give the loading/storing flag completely new meaning. A separate flag could be introduced to get the bidirectional access, but I don't think there would be much difference...

mirek wrote on Mon, 28 January 2013 20:04I think I would not bother implementing SetLoading / SetStoring in PipeStream. I believe that in typical scenario, only either input or output will do serialization, so it makes sense to set in just for single end of pipe... In that case, what is already in Stream should be enough.
Perhaps you are right about this. I can't really imagine any scenario where PipeStream would be used for serialization. It is intended for different tasks, mainly as buffer between various interfaces, as mentioned above.

Also, since the last version doesn't require explicitly calling SetLoading/SetStoring before reading/writing (first versions did treat incorrect state as error), and since using it for serialization is unlikely, it can be probably dropped without loosing any capabilities. Tomorrow I will update the code and than we can discuss it further...

Thanks for your feedback.

Honza

Subject: Re: PipeStream - bidirectional Stream

Posted by mirek on Mon, 18 Feb 2013 18:48:24 GMT
View Forum Message <> Reply to Message

After some more detailed code-review, there is one thing that makes me uneasy, and it is rdlim/wrlim.

First, you are comparing real pointers to NULL there, which is undefined in C/C++. Well, it will work in practice, but still...

More serious (but related) is the fact, that you are not using them at all Which in turn means that all the logic behind "fast" inlined Get/Put goes away. Perhaps I am not seeing everything right, but I think that you should be able to setup correct rdlim/wrlim in SetStatus and Get/Put... (if there is a reason, please tell, I am inclined to try myself, so if it is no-go, I would save my time

Somewhat related (in LZMA). In LzmaInStream::Read, how do you know that there is size elements available in PipeStream? I guess there is a reason hidden in the code, but I decided to ask first

Mirek

Subject: Re: PipeStream - bidirectional Stream
Posted by dolik.rce on Mon, 18 Feb 2013 21:08:58 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 18 February 2013 19:48After some more detailed code-review, there is one thing that makes me uneasy, and it is rdlim/wrlim.

First, you are comparing real pointers to NULL there, which is undefined in C/C++. Well, it will work in practice, but still...

Oups, forgot about that. I think it can be fixed by setting them to point on the beginning of the PipeStream buffer, to achieve the same intended effect of bypassing the Stream funcionality (as described below). Patch is attached.

mirek wrote on Mon, 18 February 2013 19:48
More serious (but related) is the fact, that you are not using them at all Which in turn means that all the logic behind "fast" inlined Get/Put goes away. Perhaps I am not seeing everything right, but I think that you should be able to setup correct rdlim/wrlim in SetStatus and Get/Put... (if there is a reason, please tell, I am inclined to try myself, so if it is no-go, I would save my time If I remember correctly the reason was that I wanted to use circular buffer, so there would be cases where wrlim or rdlim would be before the current ptr and that would be wrongly interpreted as a reason to read/flush more data. So I disabled the "buffering" in Stream completely. Perhaps I don't understand Stream correctly, but I got the impression that with the intended usage of PipeStream (reading and writing in fairly big chunks of data) most of the operations would be performed directly on the circular buffer without much performance penalty. The only methods where I see problem is Put(String), Put(const char*) and Put(int, int), where the iteration would probably make it bit slower. If you can figure out how to use the circular buffer with wrlim and rdlim always in

correct position, it would be the best, of course.

mirek wrote on Mon, 18 February 2013 19:48
Somewhat related (in LZMA). In LzmaInStream::Read, how do you know that there is size elements available in PipeStream? I guess there is a reason hidden in the code, but I decided to ask first That's simple - I don't The Read() function is not required to return size bytes, any number in range (0, size) is fine. Zero bytes returned means end of stream, which could make a problem here, but there is a condition in Put() that iteration of compression is only performed when there is at least 2^15 bytes of data available (which is the smallest chunk accepted by LzmaEnc_CodeOneBlock). Alternatively, it can be also called when the stream is closed with End(), but then it is correct to signalize end of stream when we run out of data in the PipeStream, because it is really an end.

I hope I explained it all well and correctly, it's been a while since I wrote it

Honza

File Attachments

1) PipeStream.patch, downloaded 426 times

Subject: Re: PipeStream - bidirectional Stream
Posted by mirek on Mon, 18 Feb 2013 21:42:31 GMT
View Forum Message <> Reply to Message

dolik.rce wrote on Mon, 18 February 2013 16:08mirek wrote on Mon, 18 February 2013 19:48After some more detailed code-review, there is one thing that makes me uneasy, and it is rdlim/wrlim.

First, you are comparing real pointers to NULL there, which is undefined in C/C++. Well, it will work in practice, but still...

Oups, forgot about that. I think it can be fixed by setting them to point on the beginning of the PipeStream buffer, to achieve the same intended effect of bypassing the Stream funcionality (as described below). Patch is attached.

mirek wrote on Mon, 18 February 2013 19:48
More serious (but related) is the fact, that you are not using them at all Which in turn means that all the logic behind "fast" inlined Get/Put goes away. Perhaps I am not seeing everything right, but I think that you should be able to setup correct rdlim/wrlim in SetStatus and Get/Put... (if there is a reason, please tell, I am inclined to try myself, so if it is no-go, I would save my time If I remember correctly the reason was that I wanted to use circular buffer, so there would be cases where wrlim or rdlim would be before the current ptr and that would be wrongly interpreted as a reason to read/flush more data. So I disabled the "buffering" in Stream completely. Perhaps I don't understand Stream correctly, but I got the impression that with the intended usage of PipeStream (reading and writing in fairly big chunks of data) most of the operations would be performed directly on the circular buffer without much performance penalty. The only methods where I see problem is Put(String), Put(const char*) and Put(int, int), where the iteration would probably make it bit slower. If you can figure out how to use the circular buffer with wrlim and rdlim always in

correct position, it would be the best, of course.

Even with circular buffer, there IMO is always range that can be expressed using rd/wrlim. And we do not know what the use of PipeStream is to be...

Quote:
That's simple - I don't The Read() function is not required to return size bytes, any number in range (0, size) is fine. Zero bytes returned means end of stream, which could make a problem here, but there is a condition in Put() that iteration of compression is only performed when there is at least 2^15 bytes of data available (which is the smallest chunk accepted by LzmaEnc_CodeOneBlock). Alternatively, it can be also called when the stream is closed with End(), but then it is correct to signalize end of stream when we run out of data in the PipeStream, because it is really an end.

Ah, I missed that size is pointer... OK then.

SetLoading is now unnecessary, right?

Mirek

Subject: Re: PipeStream - bidirectional Stream
Posted by dolik.rce on Mon, 18 Feb 2013 22:04:54 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 18 February 2013 22:42Even with circular buffer, there IMO is always range that can be expressed using rd/wrlim. And we do not know what the use of PipeStream is to be...Perhaps you're right. It would be a partial optimization, for the price of more complex code. I admit I was glad it just works, so I wasn't thinking about it much I'll try to look at it.

mirek wrote on Mon, 18 February 2013 22:42SetLoading is now unnecessary, right? Yes, managment of reading/writing state is now independent from SetStoring/Loading and hidden from user.

Honza

Subject: Re: PipeStream - bidirectional Stream
Posted by dolik.rce on Mon, 18 Feb 2013 23:36:17 GMT
View Forum Message <> Reply to Message

dolik.rce wrote on Mon, 18 February 2013 23:04mirek wrote on Mon, 18 February 2013 22:42Even with circular buffer, there IMO is always range that can be expressed using rd/wrlim.

And we do not know what the use of PipeStream is to be...Perhaps you're right. It would be a partial optimization, for the price of more complex code. I admit I was glad it just works, so I wasn't thinking about it much I'll try to look at it.
Ok, I found the showstopper... When you use wrlim/rdlim, the Stream functions know nothing about the fact that PipeStream is bidirectional and won't swap ptr and pptr when necessary. As far as I can tell at this late hour, it is only possible to overcome this problem by requiring user to explicitly set the state before each Get/Put operation, which is IMHO quite ugly Or do you have any other idea? I would prefer bypassing the Stream rather than having to put SetState() call to every place where it might be needed...

Honza

Subject: Re: PipeStream - bidirectional Stream
Posted by mirek on Mon, 18 Feb 2013 23:40:12 GMT
View Forum Message <> Reply to Message

dolik.rce wrote on Mon, 18 February 2013 18:36Or do you have any other idea? I would prefer bypassing the Stream rather than having to put SetState() call to every place where it might be needed...

Honza

Set lims to value that allows fast Get/Put only in actual mode and forces _Get/_Put otherwise.

E.g. in read mode, set wrlim to buffer begin. Then when Put is used, it will go to _Put, where the situation can be fixed (and rdlim set to buffer begin).

Mirek

Subject: Re: PipeStream - bidirectional Stream
Posted by dolik.rce on Wed, 20 Feb 2013 19:40:47 GMT
View Forum Message <> Reply to Message

mirek wrote on Tue, 19 February 2013 00:40Set lims to value that allows fast Get/Put only in actual mode and forces _Get/_Put otherwise.

E.g. in read mode, set wrlim to buffer begin. Then when Put is used, it will go to _Put, where the situation can be fixed (and rdlim set to buffer begin).
Well, that sounded easier than it was It also required to get rid of the internal counter of available bytes, because the Stream methods wouldn't update it. This lead to some hidden bugs which took me quite some time to figure out...

Anyway, I commited the changes to sandbox and it should be working now. The last thing I'm not sure about is the GetLeft() method, returning the number of bytes available for reading. It is not

virtual in Stream, but calculated as GetSize() - GetPos(). GetSize could be implemented in PipeStream to work correctly, but GetPos in it's current form will return wrong values Any ideas about this? What about making at least one of GetPos,GetLeft virtual too?

Honza

Subject: Re: PipeStream - bidirectional Stream
Posted by mirek on Wed, 20 Feb 2013 19:47:13 GMT
View Forum Message <> Reply to Message

dolik.rce wrote on Wed, 20 February 2013 14:40mirek wrote on Tue, 19 February 2013 00:40Set lims to value that allows fast Get/Put only in actual mode and forces _Get/_Put otherwise.

E.g. in read mode, set wrlim to buffer begin. Then when Put is used, it will go to _Put, where the situation can be fixed (and rdlim set to buffer begin).
Well, that sounded easier than it was It also required to get rid of the internal counter of available bytes, because the Stream methods wouldn't update it. This lead to some hidden bugs which took me quite some time to figure out...

Anyway, I commited the changes to sandbox and it should be working now. The last thing I'm not sure about is the GetLeft() method, returning the number of bytes available for reading. It is not virtual in Stream, but calculated as GetSize() - GetPos(). GetSize could be implemented in PipeStream to work correctly, but GetPos in it's current form will return wrong values Any ideas about this? What about making at least one of GetPos,GetLeft virtual too?

IMO, GetPos/GetLeft/GetSize etc... does not make much sense in non-seekable stream anyway (but we should update docs).

BTW, working on filter-streams, I have changed IsEof implementation to use Term instead. So as long as Term returns <0 value, sequential reading should work as expected.

Mirek

Subject: Re: PipeStream - bidirectional Stream
Posted by dolik.rce on Wed, 20 Feb 2013 20:41:45 GMT
View Forum Message <> Reply to Message

mirek wrote on Wed, 20 February 2013 20:47IMO, GetPos/GetLeft/GetSize etc... does not make much sense in non-seekable stream anyway (but we should update docs).
Looking in the docs now, it is really defined as difference between size and position... So the method in PipeStream should be renamed since it returns "number of bytes available for immediate read", which is definitely something else. What about calling it GetCount as everywhere else? Or something more descriptive like GetBytesAvailable?

Honza

Subject: Re: PipeStream - bidirectional Stream
Posted by mirek on Wed, 20 Feb 2013 20:52:23 GMT
View Forum Message <> Reply to Message

GetAvailable?

Subject: Re: PipeStream - bidirectional Stream
Posted by dolik.rce on Wed, 20 Feb 2013 21:49:05 GMT
View Forum Message <> Reply to Message

mirek wrote on Wed, 20 February 2013 21:52GetAvailable?
Ok, updated. Hopefully it is now ready for next round of your review