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 359 times