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 318 times