Subject: better Stream manual....more info needed
Posted by kohait00 on Tue, 10 Aug 2010 16:26:26 GMT
View Forum Message <> Reply to Message

hi guys..

as i stumbled over need of Stream stuff, i am trying to understand it having had a look on the current docu, i realized it to be behind current development state (OutStream, TeeStream not existing in docu..)

but even some general things should be know about Stream, trying to sum up what i mean to have learned so far. please correct where errored. (this might go into Stream docu later)

* Stream represents only a logical 'cut out piece' or a finite snapshot of a per definition concurrent, infinite data stream, beeing processed, handled or available to some extent. hence the pos variable, indicating the current offset or pos'ition of the data chunk represented by your Stream instance, from the logical start or beginning of stream.

* Stream is basicly only an interface class with some pointers to ref some memory somewhere. it usually does not contain the data itself. Thats why it may have MemStream, StringStream, FileStream etc.. 'they' access or even contain (StringBuffer) the buffer and are using the pointers from Stream base to handle it. The buffer is a current chunk of data, *entirely* accessable in your stream.

* 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.

* in both modes, the extension of the buffer marks the accessible space for reading or writing. for reading, it means the current available, *already read* data chunk, from buffer to rdlim. ptr meaning the read position currently processing it. the space from ptr till rdlim meaning the still to read data.
for writing, it means the *already allocated* data for beeing able to write to. buffer to wrlimit is its extension. buffer till ptr meaning the data, already written to the space, ptr till wrlim the space free to fill.

* Serializing stuff to Stream is quite cool. in other implementations, Stream has a split interface for serializing and deserializing stuff, so the user had to keep track which order the elements go and use 2 different functions basicly, for Serializing and Deserializing. upp stream puts away this head

ache. it uses 1 interface, and handles the difference about serialize/deserialize internally & implicitly with the help of IsLoading() / IsStoring()..
the user benefits from this only having to specify *one* functionional place, that determines the order of serialization and that is maintained the same on both directions, hurray. (drawback, one cant deserialize from a 'const Stream &', because the function needs a 'Stream &', but the case where this happens is to be neglacted). since this is somehow unusual, one needs to get familiar with the Stream.

* a global template operator%(Stream &, T &) enables your class to be serialized as well by the user, simply calling stream % yourclass. to make this happen, provide a 'void Serialize(Stream& s)' function in your class, where you define the the behaviour and order of de/serialization of your content (other data types also beeing serializable). after deserializing your class, finishing init of your class should be done. this makes handling of data transport using Socket quite easy, but is not yet implemented

* you can easily implement own buffered Streams, it provides a mainly protected interface. Stream provides a rich default serialization interface to serialize all kind of stuff, including NTL containers...(recursively calling serialize on them then as well..)

* one should know pretty well what kind of Stream to use what for.. i.e. a MemStream cant be used as a self growing Buffer, it handles only a previously provided chunk of memory which cant grow just like that (because the memory chunk cant grow). for this purpose, use StringStream.

* the basic handling of a Stream is this: a Stream refers to a chunk of memory, represented by 'buffer' pointer. it implicitly stores its size with the rdlim and wrlim pointers, representing the extension of the buffer for reading and writing operations. every API function asumes that it can store or read its data directly in this chunk of data, at ptr location, normally whitout invoking any kind of flush or the like, advancing the ptr though.
but sooner or later it will have consumed it's space (reacing respective rdlim, means having read all, or wrlim, meaning having written all). then it will claim some 'upper level' action to either provide more data, done by advanceing the snapshot position in the read case, or writing out stored data and mark it as free again. this is done invoking _Get(..) or _Put(..). in other words... _Put normally takes care of processing the full buffer by flushing it somehow, and rewinding the ptr and adjusting wrlim, declaring buffer empty. _Get typically claims some more data to be made available inside the Stream, maybe by copying some data in buffer and again rewinding the ptr and adjusting rdlim. this behaviour is to be defined somehow, and is special for any kind of stream.

Flushing behavior is not invoked by generic Stream implementation by default. but higher level Streams use it in to do exacely this. either providing more data or flushing it to the underying destination.

* there are several helper functions around handling Streams, even copying, which is normally not possible just like that, or stuff like providing a version for streamed data and some constraints on version (min, max) when serializing. take a look in Stream.cpp.

* OutStream uses a small internal buffer to accumulate things, before it forwards them to a to be user implemented Out(..) function, which should take care of processing it somehow (sending

somewhere or what ever..)

* TeeStream is an OutStream that uses internal buffer again, and when time has come, pushes it to 2 other streams..(so a little 'data latency' is expected..if you want to make the data be available at once, call Flush() after your operations..

this was some info i accumulated needed it to serialize here to understand it myself... extend if something is missing.

cheers

Subject: Re: better Stream manual....more info needed
Posted by cbpporter on Wed, 11 Aug 2010 06:22:16 GMT
View Forum Message <> Reply to Message

Well the manual page is a Reference page, i.e. documenting the interface of the class. with a little reformatting something similar to your post could be included as a separate Documentation page.

Would you care to add a few more details and submit a new page for the manual?

Subject: Re: better Stream manual....more info needed
Posted by kohait00 on Wed, 11 Aug 2010 06:38:37 GMT
View Forum Message <> Reply to Message

it was meant like that, this page claims no copyright, it is BSD licence...

anyone who has more details on that please add to this..all together is welcome to go into an info page about Stream.

often in upp, i found that, despite of a clear interface description, i could not properly use the class, because i didnt know the 'preasumptions' taken while conception of class. it's the missing big picture, i've often spoken about in other threads, that makes usage of upp classes sometimes sticky. though, it invites one to dig into code... but rookies like i am could learn quite a lot from conceptional preasumptions from profs

so any contributions gladly welcome

Subject: Re: better Stream manual....more info needed
Posted by mirek on Fri, 13 Aug 2010 08:26:37 GMT
View Forum Message <> Reply to Message

It is now in Core documentation - srcdoc/AboutStream.

I believe that you have rights to edit this.

One comment - while there are some unusual things about U++ Stream, it does not differ in basic concept from standard library streams or even streams in other languages.

Subject: Re: better Stream manual....more info needed
Posted by kohait00 on Fri, 13 Aug 2010 09:00:09 GMT
View Forum Message <> Reply to Message

the overall concept of a stream is the same. this is clear case. but the concept of serializing / deserializing with ONE function is really quite unusual..could lead to misinterpretation

---