Subject: StringBuffer size [BUG]
Posted by amando1957 on Thu, 27 Nov 2008 17:54:19 GMT
View Forum Message <> Reply to Message

Writing text like this:
```
FileOut f;
 StringBuffer sb(128);
 sb  << "asdf" << ", "
  << 123 << ", " << 3.14;
 f.Put( (const char*) sb);
 f.Close();
```

it will save to text-file
FreeFreeFreeFreeFreeFreeFreeFreeFreeFreeFreeFreeFreeFree
FreeFreeFreeFreeFreeFreeFreeFreeFreeFreeFreeFreeFreeFree
FreeFreeFreeasdf, 123, 3.14

Obvious its filled with an arbitrary auto-text.
But I can declare an empty "StringBuffer sb;", then it does not.

Martin

Subject: Re: StringBuffer size [BUG]
Posted by bytefield on Thu, 27 Nov 2008 19:09:18 GMT
View Forum Message <> Reply to Message

I think it's not a bug. You create a StringBuffer which store 128 chars and they are initialized with FreeFreeFree... to make you know when you use it without being initialized with something. It's easier to debug. Operator << is appending to the end of buffer, so that's why you get FreeFree stuff and also your chars.
If you want to make a StringBuffer initialized for 128 chars and want then to put your chars in it you have to call sb.Clear() to clear the initial content.

Also note that the buffer is filled with FreeFree... just when you have a debug build of your application.

Subject: Re: StringBuffer size [BUG]
Posted by leidner on Tue, 19 May 2009 02:31:59 GMT
View Forum Message <> Reply to Message

Hi,

While it is true that the default text may simplify debugging,

I agree with the poster that this behavior violates
developers expectations (I don't know of any other API where
you have to call clear() before using an object the first time),
and because of that, it makes it more likely that bugs get
introduced in the first place.

Would you consider changing this so as to initialize with the
empty string?

Best regards
Jochen

Subject: Re: StringBuffer size [BUG]
Posted by mirek on Tue, 19 May 2009 05:27:58 GMT
View Forum Message <> Reply to Message

leidner wrote on Mon, 18 May 2009 22:31Hi,

While it is true that the default text may simplify debugging,

I agree with the poster that this behavior violates
developers expectations (I don't know of any other API where
you have to call clear() before using an object the first time),
and because of that, it makes it more likely that bugs get
introduced in the first place.

Would you consider changing this so as to initialize with the
empty string?

Best regards
Jochen

But that would defeat the purpose of StringBuffer. You are using the constructor variant with len parameter mostly if you need to "convert" external data to String, for example in LoadFile.

If you want to start with empty string, please do not ask for non-empty one... (just do not put the length into the constructor).

Mirek

Subject: Re: StringBuffer size [BUG]
Posted by piotr5 on Tue, 26 May 2009 12:44:05 GMT
View Forum Message <> Reply to Message

luzr wrote on Tue, 19 May 2009 07:27But that would defeat the purpose of StringBuffer. You are using the constructor variant with len parameter mostly if you need to "convert" external data to String, for example in LoadFile.
what does this mean? how do I call the constructor of StringBuffer at the exact location where my data is already stored in another container? if it were stored in a string-buffer then I could just pick this, then I don't need a constructor with len-parameter. are there any examples of re-interpreting some memory-position as a string-buffer of certain size? I really do not understand why StringBuffer actually needs a constructor with length-parameter!

a related idea, a possibility which is missing in U++. it is something similar to the array-type in c. we have vector and similar containers which basically are just pointers to some data along with the necessary beaurocrazy. what I am missing is a class which I could just sort-of reinterpret_cast to and from char[]. I imagine a templated class like:

```
template <class T,int L> class StaticArray
{...
T t[L];
...};
StaticArray<int,5> primes={{2,3,5,7,11}};
```

with all the pick-constructors, conversions and stuff. is there any library providing such a basic storage-class? I can imagine this would provide some opportunity for range-checking, and if the class T would provide a constructor alike T(0), then even a length could be defined as either zero-terminated or in case of zero-less StaticArray as L. I think in some badly written book on C I read that a "char s[50]" passed as a parameter to string-functions will be treated as zero-terminated, and in case of no zero it will default to the size 50. I've never seen this to work for C but it would be a nice feature for U++. is anyone interested in that kind of thing? but I guess I'm just curious how a pick-constructor could be implemented here. just imagine:

```
StaticArray<char,11> f1() {
StaticArray<char,11> out={"hello world"};
return out;
}

main() {
StaticArray<char,11> h=f1();
h[4]=' ';
Cout()<<f1()<<"\n"; //output: "hell  world"
}
```

Subject: Re: StringBuffer size [BUG]

Posted by cbpporter on Tue, 26 May 2009 13:04:01 GMT
View Forum Message <> Reply to Message

AFAIK StringBuffer is meant to be a helper when one needs to interoperate with C like API that expects a preallocated buffer and a size for it. So instead of doing:

```
char* temp = (char *)malloc(MAXSIZE);
c_api_call(temp, MAXSIZE);
...
free(temp);
```

we can use StringBuffer to make it more safe. String Buffer also has pick semantics relative to String, so you can be convert it with zero cost if let's say you want to return the value of the buffer as a string from a function.

StringBuffer is not a class meant to build strings more efficiently like StringBuffer in Java or StringBuilder in C++. In U++ String plays the role of such classes, that's why it has a capacity.

It is also not meant to handle a raw pointer as a vector, though I find the idea very interesting. I think I saw somewhere a similar idea, but I don't remember where.