Subject: Heap errors behavior is dependent on target machine.
Posted by jfranks on Sat, 28 Nov 2015 13:03:33 GMT
View Forum Message <> Reply to Message

Brief Description: Memory heap errors don't happen for development
machine 'A'. However, they do occur for the same
executable image on the embedded target machine
'B'.

Full Description of the issue:

Description of development machine -- i.e., machine 'A'

The development machine is a Dell desktop computer that has an Intel CORE i5, and a host operating system Windows 7. A virtual machine was installed using Oracle VM Virtual Box to run a guest operating system Linux Mint 17.2. Our application development was done on this virtual machine as a Linux based application using U++ nightly snapshot upp-x11-src-9200.

Description of the embedded target machine -- i.e., machine 'B'

This is proprietary custom hardware that has a touchscreen, custom keypad entry device, a commercial power supply, commercial single board computer, and a hard drive. The operating system is the same as that used on the development machine. The CPU is compatible to run the executable image produced on the development machine.

Description of the problem we are having with U++ memory diagnostic:

1. We developed and debugged our graphical U++ application on machine 'A'. All memory heap errors were located and corrected. The executable image developed on this machine is compatible for running on machine 'B'.

2. We installed the debug version of our executable image on machine 'B'. Everything runs great except when we exit our application. The behavior is different on machine 'B' in that there are memory heap errors, followed by a segfault on X11.

```
Heap leaks detected!
Segmentation fault
```

3. We enabled machine 'B' to have development capability and installed U++ IDE based on upp-x11-src-9200. We did a code checkout into this machine from our SVN server. We compiled the code. Then we ran the debug executable built on this machine. The result was the same as item 2 above.

The debug executable image built on machine 'B' was copied to machine 'A' and it exhibited a different behavior -- it worked correctly on exit from the application, i.e., there were no memory heap errors, nor segfault. That is odd.

Next, we decided to start debugging on machine 'B' in earnest. We modified our application code and inserted MemoryIgnoreLeaksBegin(); and MemoryIgnoreLeaksEnd(); so as to exclude all of our application code from leak detection. The result was the same as in item 2 above.

We more aggressively applied the U++ memory ignore function by reworking the GUI_APP_MAIN macro and explicitly replaced it with the following.

```
//GUI_APP_MAIN {
void GuiMainFn_();
int main(int argc, const char **argv, const char **envptr)
{
MemoryIgnoreLeaksBegin();
 UPP::AppInit__(argc, argv, envptr);
 UPP::Ctrl::InitX11(NULL);
 UPP::AppExecute__(GuiMainFn_);
 UPP::Ctrl::ExitX11();
 UPP::AppExit__();
MemoryIgnoreLeaksEnd();
 return UPP::GetExitCode();
}

void GuiMainFn_()
{
 ... our application code starts here ....
}
```

The results on machine 'B' did not change -- still a memory heap issue on exit and a segfault. A large log file was produced with many memory breakpoints.

Next, we compiled a release version of the application on machine 'B' without any debug flags. Everything works great because the U++ memory diagnostics are disabled. As we run the release version, there is nothing that indicates a problem at any time, even when we exit.

The log file generated from running the debug was over 2400 items. I've attached a snapshot of the call-stack while in the debugger for the lowest numbered memory break-point #1.

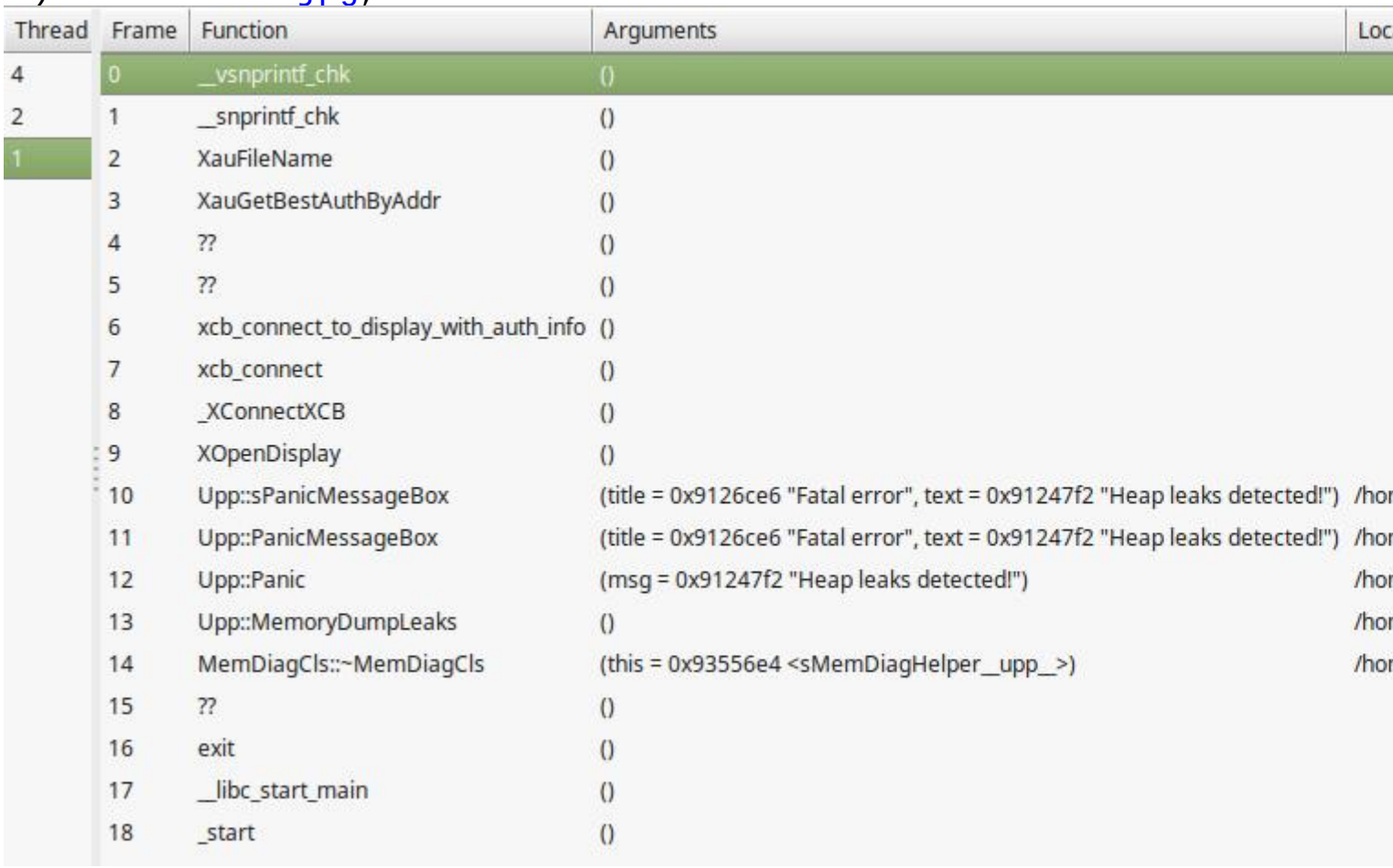
We are having a difficult time sorting this out and are asking

for help or ideas of where we go from here.

-- Jeff

File Attachments

1) call-stack.jpg, downloaded 982 times



Thread	Frame	Function	Arguments	Loc
4	0	__vsnprintf_chk	()	
2	1	__snprintf_chk	()	
1	2	XauFileName	()	
	3	XauGetBestAuthByAddr	()	
	4	??	()	
	5	??	()	
	6	xcb_connect_to_display_with_auth_info	()	
	7	xcb_connect	()	
	8	_XConnectXCB	()	
	9	XOpenDisplay	()	
	10	Upp::sPanicMessageBox	(title = 0x9126ce6 "Fatal error", text = 0x91247f2 "Heap leaks detected!")	/hor
	11	Upp::PanicMessageBox	(title = 0x9126ce6 "Fatal error", text = 0x91247f2 "Heap leaks detected!")	/hor
	12	Upp::Panic	(msg = 0x91247f2 "Heap leaks detected!")	/hor
	13	Upp::MemoryDumpLeaks	()	/hor
	14	MemDiagCls::~MemDiagCls	(this = 0x93556e4 <sMemDiagHelper__upp__>)	/hor
	15	??	()	
	16	exit	()	
	17	__libc_start_main	()	
	18	_start	()	

Subject: Re: Heap errors behavior is dependent on target machine.
Posted by mirek on Sat, 28 Nov 2015 17:43:27 GMT
View Forum Message <> Reply to Message

Hi,

I would try/suggest these:

Does the problem occur with e.g. examples/UWords too?

Have you tried memory breakpoint? http://www.ultimatepp.org/srcdoc$Core$Leaks$en-us.html

From the information given, it looks like the issue with global constructor, e.g. some INITBLOCK.

Can you post a couple of lines of log with leak with smallest breakpoint number?

Mirek

Subject: Re: Heap errors behavior is dependent on target machine.
Posted by jfranks on Sat, 28 Nov 2015 21:37:23 GMT
View Forum Message <> Reply to Message

Thank you for helping us.

Q1. Does the problem occur with e.g. examples/UWords too?

A1. No, the problem does NOT occur with examples/UWord. I compiled
and ran this on machine 'B' and everything worked correctly.
I was able to enter some text, save it to a qtf file, exit the
program without any issues.

Q2. Have you tried memory breakpoint?
http://www.ultimatepp.org/srcdoc$Core$Leaks$en-us.html

A2. Yes we have done that. Memory breakpoint #1 was used to generate
the snapshot of the call-stack (uploaded previously), while in
the debugger.

It seems strange memory break-point #1 was not hit immediately
when the application was run. Instead, break-point #1 did not
engage in the debugger until we tried to exit the application. I
expected it to be the other way around.

Q3. Can you post a couple of lines of log with leak with smallest
breakpoint number?

A3. Yes, I have done that on this response. Also, I was in error when
I said that there were more than 2200 items in the log file.
Actually, there are only 355 items each time we run the
application and then exit. I ran wc on the log-file erroneously
thinking that each line was a memory leak (too many long hours).

174 items + <size 828>
174 items + <size 812>
7 items <various sizes>

I've included the log-file with comments that show where patterns
repeat until the final 7 items are reported. I have not been able
to figure out anything relating to the repeating patters, however,
the last 7 items have to do with a shared library that manages

the serial ports. Each one of the 7 items is caused by a stdc++
string that is part of that library.

As an experiment, I modified that shared library to use const char*
instead of stdc++ strings. For example:

```
#if 0
   const std::string ERR_MSG_PORT_NOT_OPEN     = "Serial port not open." ;
   const std::string ERR_MSG_PORT_ALREADY_OPEN = "Serial port already open." ;
   const std::string ERR_MSG_UNSUPPORTED_BAUD  = "Unsupported baud rate." ;
   const std::string ERR_MSG_UNKNOWN_BAUD      = "Unknown baud rate." ;
   const std::string ERR_MSG_INVALID_PARITY    = "Invalid parity setting." ;
   const std::string ERR_MSG_INVALID_STOP_BITS = "Invalid number of stop bits." ;
   const std::string ERR_MSG_INVALID_FLOW_CONTROL = "Invalid flow control." ;
#else
   const char* ERR_MSG_PORT_NOT_OPEN     = "Serial port not open." ;
   const char* ERR_MSG_PORT_ALREADY_OPEN = "Serial port already open." ;
   const char* ERR_MSG_UNSUPPORTED_BAUD  = "Unsupported baud rate." ;
   const char* ERR_MSG_UNKNOWN_BAUD      = "Unknown baud rate." ;
   const char* ERR_MSG_INVALID_PARITY    = "Invalid parity setting." ;
   const char* ERR_MSG_INVALID_STOP_BITS = "Invalid number of stop bits." ;
   const char* ERR_MSG_INVALID_FLOW_CONTROL = "Invalid flow control." ;
#endif
```

I compiled and installed the modified serial port shared library
and then re-tested the application. Those last 7 items
disappeared! Also, the other items in the log file that repeated
174 times now repeat only 124 times. I don't know why that changed.

There must be a clue here.

-- Jeff

File Attachments

1) p101-dbg-log.txt, downloaded 501 times

Subject: Re: Heap errors behavior is dependent on target machine.
Posted by mirek on Sun, 29 Nov 2015 06:07:11 GMT
View Forum Message <> Reply to Message

So you are using C++ shared library using global constructors? I would say we have a winner...

Subject: Re: Heap errors behavior is dependent on target machine.
Posted by mirek on Sun, 29 Nov 2015 11:53:40 GMT
View Forum Message <> Reply to Message

I have 'revisited' relevant code and after a while digging through GCC docs found a possible solution to the problem (problem being non-U++ C++ library with global objects allocating memory). It is now in trunk, please check!

Mirek

Subject: Re: Heap errors behavior is dependent on target machine.
Posted by jfranks on Sun, 29 Nov 2015 17:34:51 GMT
View Forum Message <> Reply to Message

Thank you for your help.

I did what you suggested.
Nightly build upp-x11-src-9242.tar.gz was downloaded,
installed, and used to rebuild our application.

I reverted the modifications in the serial port
shared library and restored the original stdc++ strings,
and then recompiled and re-installed it.

The test results have not changed. We still have
memory heap errors on application exit.
Examination of the log file showed that U++ memory
diagnostic is still tagging the stdc++ strings from
the serial port shared library. Also, the pattern
of leaks with size 812 and 828 has not changed except
for the total number of each, which is now 162.

Conclusion: testing with nightly build 9242 shows that the
issue is not resolved.

What else that can be done?

-- Jeff

Subject: Re: Heap errors behavior is dependent on target machine.
Posted by mirek on Sun, 29 Nov 2015 17:48:03 GMT
View Forum Message <> Reply to Message

svn trunk, not nightly build (not enough nigths between since the change

Anyway, there is now manually created 9246 'nightly' build with new code included...

Subject: Re: Heap errors behavior is dependent on target machine.
Posted by jfranks on Mon, 30 Nov 2015 01:50:33 GMT
View Forum Message <> Reply to Message

Thank you for continuing to help us on this issue.

The nightly build 9246 was downloaded, installed, and used
to build our application.

The result is the same as previous ...
Heap leaks detected!
Segmentation fault

The log file contains the same pattern of 164 pairs of size = 812 & 828.
This is followed by the last 7 breakpoints
that correspond to stdc++ strings in the serial port shared library.

Is there something else that can be done to try and fix?

-- Jeff

Subject: Re: Heap errors behavior is dependent on target machine.
Posted by mirek on Mon, 30 Nov 2015 09:03:01 GMT
View Forum Message <> Reply to Message

Getting out of options.

The main hypothesis here is that we are detecting leaks too early.

Still, we can check this:

In the file with those std::string globals, put something like

```
struct MyInitChecker {
    MyInitChecker() { printf("Module initialized"); }
    ~MyInitChecker() { printf("Module deinitialized"); }
};

static const MyInitChecker myinitchecker;
```

then at the end of Core/heapdbg.cpp change destructor:

```
MemDiagCls::~MemDiagCls()
{
```

```
 if(--sMemDiagInitCount == 0) {
            printf("Now checking for leaks");
  UPP::MemoryDumpLeaks();
      }
}
```

Also, there are some details not yet provided:

- what is that "compatible" CPU?
- is the system updated to current version and it is exactly the same?
- are there any peripherals using serial communication that are not on Dell?
- what is that shared library?

Last but not least, it is entirely possible that the library leaks by design. In that case, it can be just bad luck and not really fixable. Well, in reality, leaving some global leaks is still considered "normal" in mainstream C++.

Subject: Re: Heap errors behavior is dependent on target machine.
Posted by mirek on Mon, 30 Nov 2015 09:05:59 GMT
View Forum Message <> Reply to Message

P.S.:

You might also try this, after the first check:

```
MemDiagCls::~MemDiagCls()
{
 if(--sMemDiagInitCount == 0) {
            printf("Now checking for leaks");
 // UPP::MemoryDumpLeaks();
      }
}
```

(That will turn off leaks checking, but will be helpful to check that what the shared library does after checking for leaks).

Another thing to try, nearby:

```
static const MemDiagCls sMemDiagHelper __attribute__ ((init_priority (101)));
```

change to

static const MemDiagCls sMemDiagHelper __attribute__ ((init_priority (0)));

- compiler will warn there, but it is worth try

Subject: Re: Heap errors behavior is dependent on target machine.
Posted by mirek on Mon, 30 Nov 2015 09:45:34 GMT
View Forum Message <> Reply to Message

I have found a way how to disable leaks checking in some 'external' cases like this one, it is now in _trunk_.

You can also just replace heapdbg.cpp from here

https://github.com/ultimatepp/mirror/blob/1ce7608b2fb7571902917401d4215fb76f03eafd/uppsrc/Core/heapdbg.cpp

Please try and report (after those other tests .

Mirek

Subject: Re: Heap errors behavior is dependent on target machine.
Posted by jfranks on Tue, 01 Dec 2015 04:58:39 GMT
View Forum Message <> Reply to Message

Great work!! Thank you so much for your help.

1. Regarding your instructions -- Mon, 30 November 2015 10:03 . . .

   The results of that test is as follows:

```
    $ ./p101-dbg
Module initialized
Now checking for leaks
    Heap leaks detected!
    Segmentation fault
```

   The destructor for the shared serial library was not yet called
   when heap leaks were being evaluated.

   Regarding details that you requested . . .

Q1. What is that "compatible" CPU?

A1. Celeron J1900 on an IMB-151 single board computer.
Reference: http://www.asrock.com/ipc/overview.asp?Model=IMB-151

Also, on machine 'B', "uname -a" provides the following:

```
Linux administrator-desktop 3.19.0-28-generic #30~14.04.1-Ubuntu \
 SMP Tue Sep 1 09:32:55 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
```

Q2. Is the system updated to current version and is it exactly the same?

A2. Machines 'A' and 'B' are using Linux Mint 17.2 distros. There is one serial port sharable library (32-bit) that is not part of the distro that was installed on both machines.

Q3. Are there any peripherals using serial communications that are not on Dell?

A3. Yes. The installation of our application on the embedded target machine 'B' uses serial port #1 for logging, and serial port #2 for Modbus communications. So, two serial ports exist in hardware for machine 'B'. However, the application can run without these serial ports, since logging and Modbus are optional items.

The development Machine 'A' is just a common desktop computer. Serial ports have not been made available to the guest virtual machine, which is where we are developing and testing. When the app runs and discovers no serial logging port and/or no Modbus port, these functions are just turned off.

Also, the hardware for the embedded system machine 'B' has additional devices and Linux drivers to handle a custom keypad HID device, and a touchscreen.

Q4. What is that shared library?

A4. libserial-0.5.2 ... attached to this response. We have compiled this as a 32-bit library so that it matches our 32-bit application for both machine 'A' and 'B'.
libserial-0.5.2/src/SerialPort.cpp is where those stdc++ strings are located the U++ heap diag was flagging as a memory leak.

---

2. Regarding your instructions on Mon, 30 November 2015 10:05 . . .

The results of testing with heap dump disabled:

```
$ ./p101-dbg
Module initialized
Now checking for leaks
Module deinitialized
```

This confirms that the destructor for the serial port shared
library is called after heap leaks has been evaluated.

static const MemDiagCls sMemDiagHelper __attribute__ ((init_priority (0)));
caused a compilation error (out-of-range).
I changed the 0 to a 1, and the compiler gave a warning.

The results of testing with init_priority (1) was the same as previous:

```
$ ./p101-dbg
Module initialized
Now checking for leaks
Module deinitialized
```

---

3. Regarding your instructions on Mon, 30 November 2015 10:45 . . .

The modified Core/heapdbg.cpp file was downloaded and replaced
the previous one in my copy of nightly build 9246.

This file was obtained from

https://github.com/ultimatepp/mirror/blob/1ce7608b2fb7571902917401d4215fb76f03eafd/uppsrc/Core/heapdbg.cpp

Results: There is an improvement !! Heap errors related to stc++
strings disappeared from the log file. However, we are
still experiencing heap errors related to something else.

The pattern of heap leaks with sizes of 812 and 828 are
still there. The number of these varies.

Conclusion:

- memory heap leaks related to stdc++ strings
are resolved with the modifications that you made
to Core/heapdbg.cpp

- memory heap leaks are still being detected
  from some other source.

BTW: The above result was double checked.

- Core/heapdbg.cpp was reverted to the original. Retesting showed
the last 7 items in the log file related to stdc++ strings
reappeared. This is what is expected for this test scenario.

- Core/heapdbg.cpp was again replaced with the new modified
version. Retesting showed for this case that heap errors
previously related to stc++ strings disappeared.
This is good and is an improvement.

Conclusion:

The memory heap diagnostic is improved so that stdc++ strings
in shared library are ignored (that is good).

There is another source of heap errors. Let's say that these
are all from a similar source because they always come in pairs and
the total number for each size (812 and 828) are always the same.

Is there a strategy that can be applied to be able to identify
something about where these originate?

Summary:

One down, one to go.

-- Jeff

File Attachments

1) SerialPortLib.tar.gz, downloaded 476 times

Subject: Re: Heap errors behavior is dependent on target machine.
Posted by mirek on Tue, 01 Dec 2015 18:48:48 GMT
View Forum Message <> Reply to Message

Have you tried memory breakpoint on some remaining block?

Next things to try:

- try again with IgnoreMemoryLeaks around whole GUI_APP_MAIN

- try with empty GUI_APP_MAIN

- try empty project with serial library linked in

I have digged through serial library, so far found nothing suspicious... Any chance you are using global/static objects of this serial library?

Subject: Re: Heap errors behavior is dependent on target machine.
Posted by jfranks on Wed, 02 Dec 2015 15:44:40 GMT
View Forum Message <> Reply to Message

Those are good ideas to try.

I tried a shortcut to determine where memory alloc was being done for size 812 and 828 by making a temporary change to ASSERT for those sizes in U++ memory allocator.

Both of those traps pointed back to our application code related to software that is handling custom hardware. That hardware was not available to machine 'A', but machine 'B' is dependent on this.

I have to do more testing in order to determine if these were valid allocations that are not leaks, or if they are the leaks we've been looking for all along, or if there are other places where these sizes were allocated.

At this time, the focus has shifted to our application code and not on U++ heap debug diag.

This will take a little while to sort out, but I'll report back my findings.

-- Jeff

Subject: [RESOLVED] Re: Heap errors behavior is dependent on target machine.
Posted by jfranks on Fri, 04 Dec 2015 14:29:39 GMT
View Forum Message <> Reply to Message

Mirek,

Thank you so much for improving U++ memory diagnostic so that stdc++ strings from shared library are ignored.

As promised, here is the rest of the story to a successful completion of this effort.

We were not able to use the breakpoint feature in the U++ heap diag because each run of the application produced different breakpoint serial numbers. This happened because memory leak in our application occured on different threads and events plus scheduling produced a non-deterministic pattern for assigning serial numbers to each leak detected.

That is why I resorted to temporarily putting an assert into heap diag memory allocator for size 812 and 828. This pointed right back to our application code in a place we did not expect. The memory heap leak ignore function did not work on this because our code implemented a factory class for creating event messages inside our application, which the linker put into a special section (by design) as the application was loaded into memory. These event messages were related to custom hardware that was available only to machine 'B' and not available for machine 'A' due to the lack of a driver in the VM. It turned out that some of these messages were mishandled in our application, which caused the heap diag error only on machine 'B', and not on machine 'A'.

We could have never completed our mission without your help and the improvements to heap diag regarding the stdc++ strings ignore outside our application. This is great work. U++ and theIDE have become impressive over the years. Again, thank you for your help.

Our application now works perfectly on exit and does not have memory heap errors anymore !!

-- Jeff

Subject: Re: [RESOLVED] Re: Heap errors behavior is dependent on target machine.
Posted by mirek on Fri, 04 Dec 2015 16:09:07 GMT
View Forum Message <> Reply to Message

Thanks for reporting. One thing less to worry about

Mirek

Subject: Re: [RESOLVED] Re: Heap errors behavior is dependent on target machine.
Posted by Mindtraveller on Fri, 04 Dec 2015 18:36:34 GMT
View Forum Message <> Reply to Message

Guys, I've been reading this topic lately like a crime story.
Glad to hear it's finally a happy end.
Just to mention, some time ago I've posted serial i/o library based on the same serial library (with little improvements).
http://www.ultimatepp.org/forums/index.php?t=msg&th=8894 &goto=43100&#msg_43100
May be it will be handy for some of your future projects, Jeff.