Memory Leak Detection

By **David A. Jones** | 4 Jul 2009

VC6VC7VC7.1Win2KWinXPWin2003VS.NET2003DevIntermediate

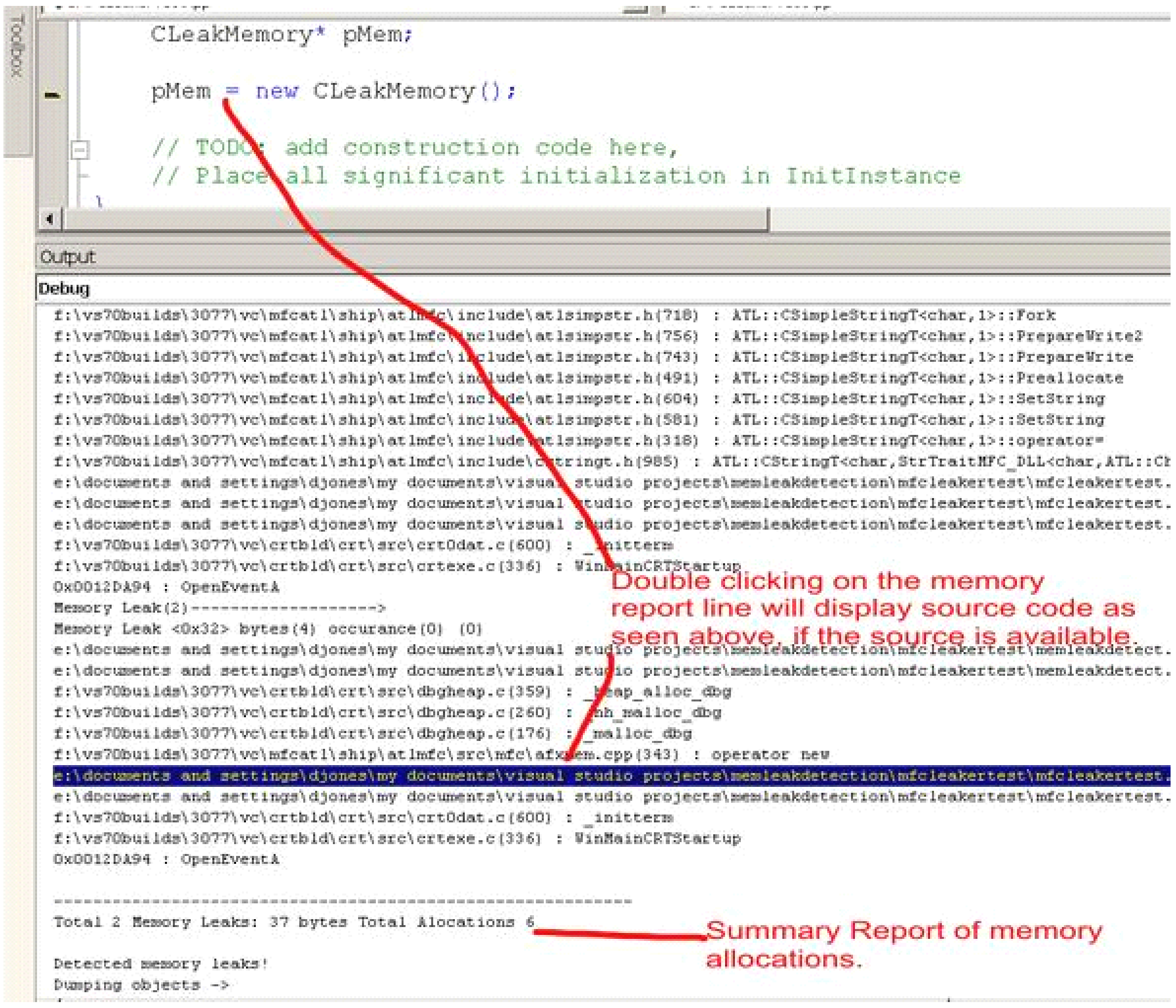
Adding Memory Leak Detection in your applications

See Also

- More like this
- More by this author

Sponsored Links

- Download demo project - 35 Kb
- Download source - 5 Kb
- Download non-MFC demo project - 12 Kb
- User Contributed Latest Source - 7.11 KB
- Latest User Contributed Code Updates


Toolbox
Output
Debug
Double clicking on the memory report line will display source code as seen above, if the source is available.
Summary Report of memory allocations.

```
    CLeakMemory* pMem;

    pMem = new CLeakMemory();

    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
```

```
f:\vs70builds\3077\vc\mfcatl\ship\atlmfc\include\atlsimpstr.h(718) : ATL::CSimpleStringT<char,1>::Fork
f:\vs70builds\3077\vc\mfcatl\ship\atlmfc\include\atlsimpstr.h(756) : ATL::CSimpleStringT<char,1>::PrepareWrite2
f:\vs70builds\3077\vc\mfcatl\ship\atlmfc\include\atlsimpstr.h(743) : ATL::CSimpleStringT<char,1>::PrepareWrite
f:\vs70builds\3077\vc\mfcatl\ship\atlmfc\include\atlsimpstr.h(491) : ATL::CSimpleStringT<char,1>::Preallocate
f:\vs70builds\3077\vc\mfcatl\ship\atlmfc\include\atlsimpstr.h(604) : ATL::CSimpleStringT<char,1>::SetString
f:\vs70builds\3077\vc\mfcatl\ship\atlmfc\include\atlsimpstr.h(581) : ATL::CSimpleStringT<char,1>::SetString
f:\vs70builds\3077\vc\mfcatl\ship\atlmfc\include\atlsimpstr.h(318) : ATL::CSimpleStringT<char,1>::operator=
f:\vs70builds\3077\vc\mfcatl\ship\atlmfc\include\cstringt.h(985) : ATL::CStringT<char,StrTraitMFC_DLL<char,ATL::Ch
e:\documents and settings\djones\my documents\visual studio projects\memleakdetection\mfcleakertest\mfcleakertest.
e:\documents and settings\djones\my documents\visual studio projects\memleakdetection\mfcleakertest\mfcleakertest.
e:\documents and settings\djones\my documents\visual studio projects\memleakdetection\mfcleakertest\mfcleakertest.
f:\vs70builds\3077\vc\crtbld\crt\src\crt0dat.c(600) : _initterm
f:\vs70builds\3077\vc\crtbld\crt\src\crtexe.c(336) : WinMainCRTStartup
0x0012DA94 : OpenEventA
Memory Leak(2)------------------->
Memory Leak <0x32> bytes(4) occurance(0) (0)
e:\documents and settings\djones\my documents\visual studio projects\memleakdetection\mfcleakertest\memleakdetect.
e:\documents and settings\djones\my documents\visual studio projects\memleakdetection\mfcleakertest\memleakdetect.
f:\vs70builds\3077\vc\crtbld\crt\src\dbgheap.c(359) : _heap_alloc_dbg
f:\vs70builds\3077\vc\crtbld\crt\src\dbgheap.c(260) : _nh_malloc_dbg
f:\vs70builds\3077\vc\crtbld\crt\src\dbgheap.c(176) : _malloc_dbg
f:\vs70builds\3077\vc\mfcatl\ship\atlmfc\src\mfc\afxmem.cpp(343) : operator new
e:\documents and settings\djones\my documents\visual studio projects\memleakdetection\mfcleakertest\mfcleakertest.
e:\documents and settings\djones\my documents\visual studio projects\memleakdetection\mfcleakertest\mfcleakertest.
f:\vs70builds\3077\vc\crtbld\crt\src\crt0dat.c(600) : _initterm
f:\vs70builds\3077\vc\crtbld\crt\src\crtexe.c(336) : WinMainCRTStartup
0x0012DA94 : OpenEventA

-------------------------------------------------------------
Total 2 Memory Leaks: 37 bytes Total Alocations 6

Detected memory leaks!
Dumping objects ->
```

Introduction

OK, So you want a memory leak detector and don't want to pay thru the nose for it! You've gone ahead and read all the articles on Memory Leak Detection (whew!!) and are totally confused and frustrated with all the technical details on how to hook memory, walk a stack, display symbols and still get the performance you need to run your application. You've Looked at lots of code and found that it's kind of a big mess to add your own memory leak detection. Well I hope that I can help out and clear the air. I have managed to create a single class that you can add to your code and find those pesky memory leaks. You can do this in debug mode or final release mode and customize your own memory leak detection.

Background

Ok, I worked for a big corporation and demanded they buy me all those expensive development toys :-) such as memory leak detection tools that I needed. Who cares how much it costs, I said! It's worth every penny! You don't want your application crashing on your customer premises do you? You don't want to consume our clients computing resources do ya? How embarrassing that would be being a top software development company? Well that was all good and said until I left that big corporation and started my own company! Wow those development tools I needed are sure expensive! But the fact is I must have a memory leak detector to make my applications bullet proof. To give me the confidence that my application is working at it's best! Not that I write leaky code or anything, ooooh Nooooo not me :-(. I point the finger at all the other people that I link into my code. From poorly or undocumented API's to complex source code I'd rather not look at. It's their fault for not telling me to free this or free that or delete this or delete that. hmmm. But it's my fault if I don't even check my application before I distribute it to others and it's eats all their memory!

There's no excuse to have leaky apps, your reputation is on the line, you need to test your applications and make sure they are not leaking and consuming unnecessary resources.

Before you Use the Code

Make sure that the *dbghelp.dll* file is an up to date version 6.1.17.2 or later. You can distribute this file with your application if you wish. If you don't know where to get this file you can download it from the Microsoft site

Using the code

C**MemLeakDetect**() class is the only class you need to worry about. In your code just include the "**MemLeakDetect**.h" file and then create a global instance of the class with CMemLeadDetect **memLeakDetect**;

This has the effect of catching any memory leaks that are present in your application before "theApp" starts execution and after it has exited. It will also work for non-MFC apps, Win32 apps, Console Apps. It's too easy! Well isn't that what it's all about! Making your life easier?

```
// MFCLeakerTest.cpp : Defines the class behaviors for the application.
//
#include "stdafx.h"
#include "MFCLeakerTest.h"
#include "MFCLeakerTestDlg.h"
#include "MemLeakDetect.h"
// CMFCLeakerTestApp
BEGIN_MESSAGE_MAP(CMFCLeakerTestApp, CWinApp)
```

```
 ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

// CMFCLeakerTestApp construction
CMFCLeakerTestApp::CMFCLeakerTestApp()
{
 CLeakMemory* pMem;

 pMem = new CLeakMemory();

 // TODO: add construction code here,
 // Place all significant initialization in InitInstance
}
// Detect Memory Leaks
#ifdef _DEBUG
CMemLeakDetect memLeakDetect;
#endif

// The one and only CMFCLeakerTestApp object
CMFCLeakerTestApp theApp;

// CMFCLeakerTestApp initialization
BOOL CMFCLeakerTestApp::InitInstance()
{
 // InitCommonControls() is required on Windows XP if an application
 // manifest specifies use of ComCtl32.dll version 6 or later to enable
 // visual styles.  Otherwise, any window creation will fail.
 InitCommonControls();
 CWinApp::InitInstance();
 AfxEnableControlContainer();
  ...
 }
```

Understanding the Code

This class really looks pretty straight forward. The public methods are very simple to understand. Init() initializes the `CMemLeadDetect` class. `End()` does the memory report of unfreed memory when the destructor method is called. **AddMemoryTrace()** - is when an alloc memory orcurrs,**RemoveMemoryTrace()** - is a free memory event, **RedoMemoryTrace()** - is a re-alloc memory event. I must point out that the fileName is limited to`MLD_MAX_NAME_LENGTH(256)` characters. If you anticipate more than this as a file name then please increase this limit. Also the same for traceinfo`MLD_MAX_TRACEINFO(256)` . This means that if you expect that there will be more than 256 functions deep when a memory allocation occurs then by all means increase this limit also. Personally I've seen it go to 230 entries but it was really unusual but not impossible. I would say that 256 entries is kind of ordinary but 512 entries is very large. Somewhere in between maybe good but it's up to you.

The `AllocBlockInfo()` class is a sub-class of `CMemLeakDetect()` because this is what keeps track of all of the allocations and free's. It's indexed by the request number that allocates, reallocates or frees memory in a `CMap` list which is a hash list. Because it's a hash list it's important to initialize it so that all the allocations in the hash list are done before the application initializes. This will keep your application from being sluggish with the **CMemLeakDetect**. But don't just stick any number in the hash list because you can seriously degrade a hash list performance if it starts dividing even slots which in turn are called collisions. To avoid these collisions (as many as we can) it's best to initialize it with a prime number such as`AllocatedMemoryList.InitHastTable(10211, TRUE)` as I have done. When your application ends and the memory report dumps all the unfreed memory and a summary you will see a total allocations that occurred in your application. If you have more than a total of 10211 allocations you should bump up the prime number to the one that is more than your maximum. If you want more entries find a bigger prime number. This pre-allocation of hash table entries will help your performance tremendously during runtime. The `STACKFRAMEENTRY` is a structure that is an array of traceinfo starting at location [0] which is always the **CMemLeakDetect**`()` methods and it walks the stack to all the callers that exist on the stack [1], [2].... and so on. The last traceinfo entry is [n] = 0;

```
//#define MLD_MAX_NAME_LENGTH   256
#define MLD_MAX_TRACEINFO   256

typedef struct _STACKFRAMEENTRY
{
  ADDRESS   AddrPC;
  ADDRESS   AddrFrame;

} STACKFRAMEENTRY;

class CMemLeakDetect
{
 public:
  class AllocBlockInfo
  {
   public:
    inline AllocBlockInfo() {};
    inline ~AllocBlockInfo() {};
    inline AllocBlockInfo(AllocBlockInfo& abi)
    {
     address  = abi.address;
     size  = abi.size;
     lineNumber = abi.lineNumber;
     occurance = abi.occurance;
     memcpy(&traceinfo[0], &abi.traceinfo[0], sizeof(traceinfo));
     memcpy(fileName, abi.fileName, sizeof(fileName));
    };

   void*    address;
   DWORD    size;
   char    fileName[MLD_MAX_NAME_LENGTH];
   DWORD    lineNumber;
   DWORD    occurance;
   STACKFRAMEENTRY  traceinfo[MLD_MAX_TRACEINFO];
  };

 public:
  CMemLeakDetect();
  ~CMemLeakDetect();
  void Init();
  void End();
  void addMemoryTrace(void* addr,  DWORD asize,  char *fname, DWORD lnum);
  void redoMemoryTrace(void* addr,  void* oldaddr,
   DWORD asize,  char *fname, DWORD lnum);
  void removeMemoryTrace(void* addr);
  void cleanupMemoryTrace();
  void dumpMemoryTrace();
  //
 CMap< LPVOID, LPVOID, AllocBlockInfo, AllocBlockInfo > m_AllocatedMemoryList;
 DWORD memoccurance;
 bool  isLocked;
 // private:

  BOOL initSymInfo(char* lpUserPath);
  BOOL cleanupSymInfo();
  void symbolPaths( char* lpszSymbolPaths);
  void symStackTrace(STACKFRAMEENTRY* pStacktrace);
  BOOL symFunctionInfoFromAddresses(ULONG fnAddress,
   ULONG stackAddress, char* lpszSymbol);
  BOOL symSourceInfoFromAddress(UINT address, char* lpszSourceInfo);
  BOOL symModuleNameFromAddress(UINT address, char* lpszModule);

  HANDLE    m_hProcess;
  PIMAGEHLP_SYMBOL  m_pSymbol;
  DWORD    m_dwsymBufSize;
};
```

The private members are more interesting because that is where a lot of the tricky work has been done! My goal was the keep all the sym* routines as simple as possible and I tried not to do too many system calls. `symStackTrace()` is the function that captures the stack at the time of any memory operation.

Points of Interest

`catchMemoryAllocHook()` this is a memory hook routine which is the at the heart of this `CMemDetect` class. It's where the most complex and convoluted part of the code is. Documentation on this area is sparse and mostly undocumented by our friends at Microsoft. I'm not even sure I have caught all the situations. If there's going to be a major bug it's most likely going to show up here. Also I'm only using a simple locking mechanism because I didn't want to do any system calls for a Critical Section, Semaphores, or `CSimpleLock`. All the documentation I read says that only one thread can access this memory hook at a time and it reentrant safe! Based on that premise I just used the simplest mechanism.

Credits

I must give credit to Code Glow http://www.codework.com/glowcode/order.html for the inspiration because I didn't want to pay for it at $299.00 US (a fair price though), Compuware Corporation http://www.compuware.com/products/devpartner/bounds.htm again too expensive for my cheap budget! :-) so I wrote my own. Thanks to Erwin Andreasen & Henner Zeller for LeakTracer http://www.andreasen.org/LeakTracer/ Johan Lindh for MemWatchhttp://www.linkdata.se/sourcecode.html R. Walker http://www.codeproject.com/script/Submit/memleackcheckarticle.asp and Jochen Kalmbachhttp://www.codeproject.com/tools/leakfinder.asp in which I found inspiration to provide my own. I did a lot of research and found an overwhelming amount of information on the subject but nothing was done in a comprehensive and clear manner. So finally I give credit to myself :-) for all the late nights and hard work getting this class all debugged and cleaned up for this article. :-)

History

- Initial Version **July 30, 2004**, by David A. Jones
- Added Non MFC Version **October 6, 2004**, by David A. Jones
- Fixed User Issues **October 11, 2004** by David A. Jones
    - fixed buffer overruns now using `MLD_MAX_NAMELEN`
    - add symStackTrace2 that is an alternative to `StackWalker64()`, in non-MFC version & untested
    - fixed up some lint issues in both versions
- Update October 18, 2004 by David A. Jones
    - added an alternative stackwalker using `MLD_CUSTOMSTACKWALK` to select which stackwalker to compile
    - consolidated the MFC version and the non MFC version. They are both the same code now
    - fixed more UNICODE issues, should be UNICODE compliant
- Update July 3, 2009 by Tim Stevens
    - Tim Stevens, July 3rd 2009: See this and future updates at Tim Steven's Windows Memory leak detection (update to existing article)

License

About the Author

David A. Jones

Web Developer

Canada

Member