OLE/OPC Memory Management White Paper

Al Chisholm, Intellution Inc

01/23/98

© Intellution Inc. 1998
ALL RIGHTS RESERVED

Abstract
Memory management in COM, Automation and OPC servers is a tricky issue. This paper will offer a concise explanation of the major issues and offer suggestions for creating servers and clients that are free of memory leaks. It assumes that you already know the basics of COM programming and memory allocation. All of these issues are also explained in various areas of the Microsoft Documentation. Also, experts may note that some of the issues are slightly simplified for clarity. That is, while it is always safe to follow the rules presented here, it is possible on rare occasions to break some of these rules but such situations are not discussed here.

When do I need to worry about this?
You need to think about these issues mainly when passing parameters to methods on interfaces. As a general rule you can check the IDL file and examine the parameter direction for each parameter.

All 'in' parameters are owned and managed by the program calling the method. In general no special rules apply and they are generally 'local' memory as discussed later. The called method can copy the data if it needs to be preserved but must never try to save a pointer to the data since the calling program will be reusing that memory after the method returns.

All 'out' parameters are 'shared' ownership. This is where you need to be careful. The called method allocates the memory which must be 'global' or 'string' memory as described below and passes a pointer back to the calling program. The calling program must 'free' the returned memory.

Note that this in/out rule applies to Caller/Callee, not to Client/Server. If the Server calls back into the Client (referred to as a 'Callback'), the same rules apply.

Memory Types
There are three types of memory you may want to allocate and manage in an OPC Server; local scratch memory, global memory to be returned to the client as 'out' parameters and BSTR (string) memory. You must take care to always allocate the right type of memory for the right use and also to free the memory using the proper function for that type of memory. Note that in many cases, calling the wrong 'free' function for a particular memory block will not generate any type of easily detectable runtime error but will still fail to free the memory resulting in leaks that can accumulate over time and eventually cause a system to fail.

Local Memory

Local scratch areas can be used within a function or can be allocated within an object as long as pointers to them are not passed outside the task or DLL. The memory can be allocated and freed using either of two techniques.

- malloc/free - the 'old' way.
- new/delete - the 'new' (recommended) way. If you allocate an array using new, be sure to use the **'delete []'** syntax when freeing the memory.

Note that although the two techniques both work, you CANNOT allocate a block with one technique and free it with the other.

Global Memory
Global memory is that which is allocated by the called method and passed back to the caller by way of 'out' parameters (except BSTR memory as noted below). The memory is then freed by the caller. There are two techniques for managing this type of memory.

- CoTaskMemAlloc/CoTaskMemFree - the 'easy' way.
- CoGetMalloc - the 'efficient' way.

These techniques use the same memory 'pool' and are interchangeable. For example you can allocate memory in a called function with CoTaskMemAlloc and free it in a calling method using the IMalloc obtained from CoGetMalloc. Note that in case of error it is also allowed for the called method to free the memory (for example where you get halfway through a method and determined that the data cannot be returned to the caller due to an error condition). In this case the 'out' pointer must be set to NULL and an E_xxx HRESULT must be returned.

String Memory
String memory is used with BSTRs. BSTRs must ALWAYS use this class of memory. The SysString… functions described in the OLE Automation Programmers Reference must always be used to allocate and free this sort of memory. For example if a client is freeing a BSTR that was returned from a server, it MUST use SysFreeString.

VARIANTs
OPC commonly makes use of variants. To summarize, all of the rules above still apply.

Depending on your needs, the VARIANT itself can be allocated from either **local** or **global** memory. Also depending on the context, data which the VARIANT points to can be allocated using **local**, **global** or **String** memory.

After you allocate one or more VARIANTs (e.g. by using CoTaskMemAlloc or new) you MUST call VariantInit() on each one. If you are storing a BSTR into the variant (VT_BSTR) you must allocate the BSTR from **String** memory. Note that there would be two separate memory blocks at this point; the block containing the VARIANT and the

block containing the BSTR and that the VARIANT (in block 1) would contain a pointer to the BSTR (in block 2).

Before freeing one or more VARIANTs (e.g. one returned as an 'out' to a caller) you MUST call VariantClear() for each one. Note that VariantClear is a very clever function. It will determine what type of memory (if any) the Variant points to and whether it is an array and it will then free that memory using the proper method. After calling VariantClear to free the contents of the Variant, you can free the memory containing the VARIANT itself (e.g. by using CoTaskMemFree or Delete). Note that it is safe and reasonable to call VariantClear on a variant that does not actually point to any other memory (e.g. a VT_R4 variant).

Testing
It is very strongly recommended that you test your clients and servers using a package (such as BoundsChecker) which can detect memory leaks.

CAUTION: OLE performs some very complex caching on String memory. For example it may group multiple BSTR strings within a single memory block. As a result, calling SysFreeString may not actually free the memory. This can make testing of BSTR related code very difficult. Not surprisingly, some of these diagnostic packages may report spurious memory leaks in such cases when in fact your code is correct.

It is often helpful to test a new client with a server that is known to be reliable (such as the OPC Sample Server) or to test a new server with a client that is known to be reliable (such as the OPC Sample Client).

Summary
Careful attention to these rules will result in reliable, leak free servers.