Subject: Re: SHMMAP and structures
Posted by Peter Mason on Thu, 10 Aug 2006 23:10:04 GMT
View Forum Message <> Reply to Message

Mike Wallace wrote:
> Hi Everyone,
>
> In the documentation for SHMMAP, I ran across this following regarding
> the kind of data that can be used in SHMMAP: "The array can be of any
> type except pointer, object reference, or string. (Structure types are
> allowed as long as they do not contain any pointers, object
> references, or strings.)"
>
> This is the only place where I read of using a structure with SHMMAP.
> Has anyone ever mapped structures before? If so, can you mix and
> match the data types within the structure or are you restricted to
> using only one data type in the structure? Also, how is the data
> laid out in memory if you write out a structure this way? (I'm
> interested in the data layout because a non-IDL program needs to use
> the memory space as well.)
>
> I know that all these questions are ones that I could answer myself
> if I had enough time to play with the commands, but I don't have
> enough time at the moment to do a thorough investigation, but I was
> hoping that someone out there may have worked with this before. TIA,
>
> -Mike

I haven't actually used SHMMAP but some time back I wrote something similar myself so I think I know how it works.

As long as the structure doesn't contain any string, pointer or objref members, it is a complete block of data. It stays the same size for all its life. (Well, all IDL structures do that.) All of the members have their storage space right there in the structure. As far as reading and writing structures is concerned, there is no control information buried in amongst the data. The knowledge of what datatype each member is resides elsewhere. Even array members have no control info in the structure data block. They just have room for the declared number of elements (of the declared datatype).

If you write out an array of structures using IDL's SHMMAP then you can read it from an external application quite easily. You just need to define the structure properly in the external app. This is easy in C as there's mostly a C datatype corresponding to each IDL datatype. There are some things to be wary of, of course. With 32-bit windows, you might have to go digging a bit to find the 64-bit integer types but they're there. If you

aren't using 32-bit Windows then you have to be careful with your 32-bit and 16-bit integers. If you have a choice of what "float" and "double" mean then be sure to pick the 4-byte and 8-byte IEEE formats. C doesn't have complex numbers but they are just FLTARR(2)s and DBLARR(2)s.

One thing you have to pay attention to is padding bytes. Each member in an IDL structure can have padding bytes before it, in order to align it in memory. There might be padding bytes at the end of the structure as well, to align the first member of the next structure in an array. (Even if you have just one structure element, IDL sets things up for an array.) Unlike other IDL I/O routines, SHMMAP doesn't apply any smarts to its data and these padding bytes will go along for the ride.
I think I read somewhere that IDL uses the default padding rules of the host platform. I've been using Win32 exclusively for many years so I don't know about other platforms any more, but on Win32 the padding rules seem quite simple. The start of the structure can be assumed to be aligned on at leat an 8-byte boundary and then each structure member gets 0 or more padding bytes to align it to its byte count. So a 32-byte integer or single-precision float could be preceded by up to 3 padding bytes, for example. Whatever it takes to have it on a 4-byte boundary. As far as arrays are concerned, it's just the datatype that matters to padding.

The reason that SHMMAP doesn't tolerate string, pointer and objref structure members is because these types do not have their data within the structure's data block. A string just has a descriptor in the data block. The guts of it - the string itself - resides elsewhere on a heap somewhere. A pointer or objref just has an index in the data block. This index ultimately refers to the actual data - again, elsewhere on a heap somewhere. If you were allowed to do SHMMAP I/O with structure members like these then the best that you could possibly hope for is that your IDL program would crash. You'd be reading in string descriptors and internal indexes that referred to arbitrary locations in memory or things that didn't exist (if you were lucky).

HTH
Peter Mason
