

## Re: How do you inherit DOS console in Win32 application?

**Source:** <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2004-03/0441.html>

---

**From:** Beth (*BethStone21\_at\_hotmail.NOSPICEDHAM.com*)

**Date:** 03/18/04

Date: Thu, 18 Mar 2004 04:45:38 -0000

e^ln(x) wrote:

- > *How do you inherit and continue using the existing command-line console in a*
- > *Win32 assembly console application?*

All "console" Win32 programs automatically inherit the "standard handles" of the parent console process that spawned it...which, in most cases, is a "DOS box" prompt...

Get these handles using the "GetStdHandle" API, passing one of the three constants `STD_INPUT_HANDLE` (-10; same as 0FFFFFFFF6h), `STD_OUTPUT_HANDLE` (-11; same as 0FFFFFFFF5h) or `STD_ERROR_HANDLE` (-12; same as 0FFFFFFFF4h)...the "HANDLE" DWORD value that Windows returns is then a valid "file handle" to pass to either "WriteConsole" or "WriteFile" (the difference between these two APIs is only that "WriteConsole" knows it's writing text to a console and, hence, there's a "WriteConsoleW" and "WriteConsoleA" – which handle UNICODE and ANSI respectively – whereas "WriteFile" is a general "please dump this data in the file" API and so might `_not_` be writing text but could be used to write raw binary data...note that for writing ASCII text, there's not really any practical difference between the two...basically, "WriteFile" literally dumps what you provide "as is", "WriteConsoleA" and "WriteConsoleW" know that they are dealing with ANSI / UNICODE text and behave accordingly...the difference is subtle and minor but turns up with things like CR / LF and so forth :)...

- > *Borland C and MVC both do this automatically and hide the startup code from*
- > *the debugger. Every Windoze console application inherits the DOS console*
- > *when invoked from the command line, but I can't figure out how to do it.*

Windows is actually the one most involved in this process and it "inherits" these "standard handles" from the "DOS box" prompt, whenever the application is marked as being a "console"

## alt.lang.asm: Re: How do you inherit DOS console in Win32 application?

application...it depends on your linker how exactly you specify this but something like "/SUBSYSTEM:CONSOLE" is what you'd find with MS's LINK.EXE, for example...once you do this, the "DOS box" itself automatically "spawns" your console application passing over the "standard handles" for that console...

Alternatively, what really happens at a lower-level is that the "DOS box" uses "CreateProcess" and fills out a STARTUPINFO structure with the various details...if you use the "GetStartupInfo" API then it returns the structure that the "DOS box" used when it called "CreateProcess"...and, inside that structure, you'll discover that the three "standard handles" are provided...basically, the "GetStdHandle" API which grabs the standard handle values is more or less a "convenience function" in this context for pulling out the right data from this structure...you can save yourself a few API calls and also have the opportunity to get a lot more information (as the STARTUPINFO structure has information such as the position and size of the console window, the title of the window, etc....look it up in your Win32 reference and there's some interesting stuff there :), by simplying grabbing the information yourself in a more "manual" way with "GetStartupInfo" (as there are strings inside STARTUPINFO, then there's actually a "GetStartupInfoA" and "GetStartupInfoW" API for ANSI and UNICODE respectively...with ASM, though, data types are usually unimportant and you'd just declare the strings as general "pointers" rather than specifically "constant pointer to BYTE" and "constant pointer to WORD"...in other words, if you're not fussy about "types" then you can create a "STARTUPINFO" structure in ASM where the strings are just "pointers" (DWORD or PTR :) and then it's up to you to remember to look at the strings in an ANSI or UNICODE way, according to whether you used "GetStartupInfoA" or "GetStartupInfoW"...what I'm trying to get at here is that under HLLs like C / C++ or whatever, they tend to make more of a fuss about "data types" that they need a separate "STARTUPINFOA" and "STARTUPINFOW"...most ASM coders tend to not care about this and I thought I'd just point out that the only difference between these two structures is the "data type" of the strings inside it...hence, if you don't care about "data types" and a simple "DWORD" will serve your purposes, then it's worth nothing that the two structures are actually identical except for this (same order, same offsets...a "pointer" to an ANSI or UNICODE string is the same size, of course, as a pointer to, well, anything ;)...you DO, though, need to call the right "GetStartupInfoA" or "GetStartupInfoW" to tell it whether to fill in the string details with ANSI or UNICODE versions :))...

Personally – just to get my usual Microsoft gripes in – this isn't really "UNICODE" because each "code point" is now 32-bit these days (well, 24-bit or something like that is "significant" but it gets stored in 32-bit DWORDs...there's a heck of a lot of those Far Eastern ideographic characters that they realised not even 65,536 characters covers it completely ;)...this is really just the "BMP" part ("Basic Multilingual Plane", not "bitmap"...an unfortunate coincidence in

## alt.lang.asm: Re: How do you inherit DOS console in Win32 application?

abbreviations there ;)...but unless you want ancient Egyptian hieroglyphics (have you even got the font for that? I know I haven't! ;) or you speak Mandarin or some other Far Eastern language, the first 16-bits are actually good enough for all other typical uses...they cover Latin, Cyrillic, Greek, Hebrew, Arabic and so forth, as well as a whole bunch of dingbats and symbols (hey, there's even the good old "box drawing characters" in there somewhere! Useful, perhaps, if you're converting some old DOS code that used those "IBM ASCII" things ;)...it's just the Far Eastern and historical stuff that "overflowed" the 16-bit UNICODE that strictly it's no longer 16-bits in size anymore...Microsoft can be forgiven on the point that this wasn't the case when they started calling their "wide character" support "UNICODE" (they were right at the time but they have since become slightly incorrect afterwards...but, hey, can't totally blame them as they did sensibly put "W" (for "wide character") rather than "U" (for "UNICODE") after the API names, suggesting they kind of guessed it might change later...

But the main gripe here is technical...as you've noticed, a GUI application – even when started from a "DOS box" – can't get access and doesn't "inherit" the standard handles...that's poor design in my view...a case of Microsoft not thinking about any uses for what they do and, therefore, didn't think of adding it and then they've gone and made it impossible...why would you want to do this, anyway? Simple...how about a "dual mode" application that, for example, can accept a command-line and operates with the "standard handles" in a typical command-prompt utility way...but if there's no command-line given, it launches a GUI text editor or dialogue box instead? Such a thing would be highly useful for programmers that like to use MAKE and would like a GUI "interactive" interface too...

Microsoft may think this is "mutually exclusive" but I do not and see no reason – but their programmers simply never thinking of it and only coding to their own uses – for why this support should be particularly absent for GUI applications...actually, in a sense, Microsoft did it all wrong by having this strange "flag in the header says it's a console or GUI" format in the first place...this makes them mutually exclusive when, logically, there's no particular reason why they should be whatsoever...there should just be "Win32 applications" and a "console" application could some "GetConsole" API to "inherit" the "standard handles" (you have to call a similar API to get the values of those handles, anyway, this way would, therefore, be no extra effort because both processes are merged into one :)...)

They've needlessly – through very typical "oh, oops, didn't think of that!" Microsoft design – made such an application impossible...the best you can manage – which is incredibly ugly and very amateur-looking – is to mark it a "console" application to "inherit" the handles but if it turns out that you want the "GUI mode" instead then you can call the "FreeConsole" API (which shuts down the link to the console it inherited :) and use Win32 GUI API from then on

## alt.lang.asm: Re: How do you inherit DOS console in Win32 application?

instead...a solution which works in a bizarre way but is terribly ugly...as Windows' code is so slow then you can clearly see the "console" window appear and disappear should you launch it into GUI mode directly from Explorer or something...that'll just confuse the user something rotten to have strange console windows flashing about all over the place: "Huh? What was that? Has something gone wrong? Oh no! I must have caught a virus!!" ...blah–blah–blah...

Over on a real OS like Linux with X–Windows, for example, it's perfectly possible to have an application choose whether to use the "standard handles" of an Xterm window...or to use a GUI interface...or to use BOTH (nice idea for debugging purposes, for instance...if it's a GUI application, you can have it feed "debugging output" as it is running to the "standard output" handle and view it in an Xterm window that you launch it from...don't care? Launch it directly without Xterm and there's no "standard handles" and the output goes nowhere so it runs as if this stuff wasn't there at all...add some "conditional compilation" to include / exclude the debugging output from the code so that you can instantly drop it when it comes to the "release version" of the program...and, hey presto, you've gotten yourself a pretty neat way to debug GUI applications...also, as we're sending to "standard handles" then there's the possibility to also "redirect" the output to a "log file" and you can look at that later should a problem arise :)...also, of course, X–Windows itself allows an application to "connect" to multiple X servers all over the globe without any change in the actual source code at all...and, basically, unlike Microsoft – who clearly spend a total of ten minutes "designing" their entire OS and make up 90% of that as they are going along – the options are numerous and made in no way "mutually exclusive" from each other for no particularly logical reason other than that Microsoft want to stuff more "proprietary data" into the executable headers to isolate it off in their usual "de–commoditising" way..."POSIX compliant"? NOT in a spiritual way whatsoever...

> *I've been through the Windoze SDK from one end to the other and have found no*

> *suggestions or example code. Does anybody have any hints?*

Look up "GetStdHandle" and also "GetStartupInfo" (look inside the STARTUPINFO structure that this API loads for you with the start–up information that the "DOS box" created your "process" with ;)...

Also, more generally, the official Microsoft name for this stuff is "character mode support" (they avoid "DOS box" or "DOS program" due to the fact that these aren't actually DOS applications whatsoever...they only look like DOS applications but are fully–fledged Win32 applications that can call all of the API...hence, by their terminology, you're writing a "character mode" Win32 application :)...as hinted in one of the other replies, this "character mode support" does actually go beyond DOS prompts only and you can do things like give the window a title, have "consoles" with

Re: How do you inherit DOS console in Win32 application?

## alt.lang.asm: Re: How do you inherit DOS console in Win32 application?

irregular sizes (something other than 80 x 25 or 80 x 50 :), deal with mouse stuff, edit the "screen buffer" directly (which is somewhat like writing bytes to B800h in DOS but via Win32 API :), etc., etc....there's much to see and experiment with...but, yeah, first you need to know that Microsoft call it "character mode support" or you'll easily wander passed it in the SDK reference, wondering why you can't find anything about "DOS box support"...jargon and terminology, indeed, is a double-edged sword...fantastically useful when you know it...utterly useless when you don't ;)...

> *Alternatively, does anyone have c0nt.asm from the Borland CBuilder package?*

> *That is alleged to be the startup code used by the compiler.*

Knowing some of the typical Borland "naming conventions" for these things, then, yeah, the "c0" bit means "C start-up"... "nt" presumably refers to Windows NT...although, actually, with my older Borland compiler, there's two separate startups (well, three actually, if you include the DLL start-up code too...unlike EXEs, there only needs to be one start-up code for DLLs because "console" and "GUI" is actually "decided" by the main EXE that launches things...the DLLs it loads must suffer under whatever the EXE set as its flags in the headers, making underhand reference again, of course, to my personal gripe that this whole "flags in headers" approach is bad design and inherently flawed because it makes things that aren't actually "mutually exclusive" in any logical sense, mutually exclusive in a practical sense...that's just Sir Bill NOT thinking when he was designing that part of Windows about the possibility of an application that might want to select what kind of application it is at run-time...due to command-line options or whatever...you can imagine, for example, an assembler that assembles when provided a command-line to tell it what to assemble – useful for batch files and MAKE – as a "console" application but when provided nothing (or some "-editor" switch), instead launches a GUI text editor...something like RosAsm or FASM does...it would, in fact, be excellent to have, say, FASM work in such a "dual mode" way with its GUI text editor...but was still perfectly compatible with putting into MAKE files and using it in a "batch" way instead, where any "errors" would turn up at the console, where it would be expected for a typical command-line utility...it's due to stupid unnecessary "mutually exclusive" nonsense from Microsoft and others that, indeed, this is probably why we don't see more applications designed to work "batch" or "interactive" depending on what command-line (or no command-line) they are given...in logical and practical terms, there's plenty of useful stuff that could be made of such things...FASM for Windows coming as one download\_ and the GUI / text mode option is selected by a "switch" like "-editor", which could be incorporated into a "shortcut" icon for Explorer that a double-click there gets you the "interactive" GUI editor but using it as a command-line tool means it'll work in that way just as well...in fact, this would provide both but be entirely "seamless" from the way things are done normally that you wouldn't perhaps even notice that

## alt.lang.asm: Re: How do you inherit DOS console in Win32 application?

it's the same executable providing both...and, in development terms, put both into one executable also guarantees that the GUI and command-line "versions" are, in fact, the same version – just different "interfaces" to it – that you won't develop any strange "this doesn't seem to work in the GUI version but does in the other version" bugs...a case of "code re-use" that makes things better rather than just for the sake of it...it's a great idea when done properly...but as we can see from Microsoft, it's a nightmare of bugs and development troubles when done improperly ;)...

Anyway, gripes aside, you'd have to check it up directly with the version of Borland's compiler you've got there (they might have deliberately "combined" the various start-up codes into one start-up file that's more "generic"...if they have, mind you, that's trading "efficiency" for "convenience" so whether that's a good idea will vary from person to person...it'd probably throw an extra few KB onto the EXE size for sections of code for the other type of "mode" that it actually doesn't really need and never actually runs in practice...terribly inefficient but that's the "bloatware" direction that everything in the HLL world is headed, unfortunately)...but look for files that start with "c0", as that's the "signal" that it's "C start-up"...and you might find that there's more than one "version" depending on the type of application...for example, with my quite old Borland compiler, there's "c0x32.obj", "c0w32.obj" and "c0d32.obj" (where "x" is for "console", "w" is for "(native) Windows" and "d" is for "DLL" ;)...this makes sense, though, because it's "convention" for a console application to start at "main", while GUIs start at "WinMain" and DLLs start at "DllEntryPoint"...all of which are completely different ways for the application / DLL to start up that they deserve to have different start-up code (which is why I'm thinking that you should probably check to see if there's other "c0xxxx" files in your Borland LIB directory and to find out what each one is...because it's likely – due to the different "entry-points" for these types – that the start-up code has to be different to support that...they'll actually all want to "link" to different "entry point symbols" with the linker, you see :)...

[ You'll also commonly see different versions of the "standard library" too...as another example, "cw32.lib" for "C Windows 32-bit standard library", "cw32mt.lib" which is the same library made "multi-thread safe" (looking at my C++ Builder, it looks like Borland have simply dumped "cw32.lib" and now only supply the "multi-thread safe" version...which is an understandable decision as that's the "safe" version which should work for all occasions, multi-threaded or not...but, well, if you're NOT multi-threaded then there's probably some minor "inefficiency" in that library over the older non-multi-threaded one because it includes checks and semaphores and critical sections and so forth, no doubt, to make it "multi-thread safe" which are actually redundant when you're NOT writing a "multi-threaded" application in any way...but this shouldn't amount to a great deal of "inefficiency" ...at least, NOT in light of the fact

that you're using C / C++ in the first place...it's not nice for a "speed demon" ASM coder but it's positively "angelic" in "HLL bloatware" terms...I've seen start-up code that requires `_20KB_` (!!!) just to process a "GetCommandLineA" string into the usual C "argc, argv" stuff (that is, separating them out into an array of strings, split according to the spaces in the command line...why on Earth someone needs 20KB to do this simple task is beyond me but, clearly, someone out there does because the files that do things this badly exist for all to "bloat" their code unnecessarily with...usually, I "detach" the "c0x32.obj" or "c0w32.obj" start-up code and simply replace it with a hand-coded routine that loads up the parameters into the right places and it all works just fine but without taking up even close to 1KB, let alone 20KB...there's an awful lot of "nonsense" in these files and, sorry, but the HLLers can't harp on about their "optimising compilers" in this case because the ".obj" start-up files are merely `_linked_` "as is" so the "optimiser" has nothing whatsoever to do with this and can't make it any better than whatever crap they supply ;)...

Beth :)