Image File Execution Options (IFEO)

By kexugit

Archived: 2026-04-05 13:23:53 UTC

Image File Execution options provides you with a mechanism to always launch an executable directly under the debugger. This is extremely useful if you ever need to investigate issues in the executable's startup code (services especially). You can set the IFEO options directly via the registry or indirectly using the Gflags tools (available with the Window debugging toolkit).

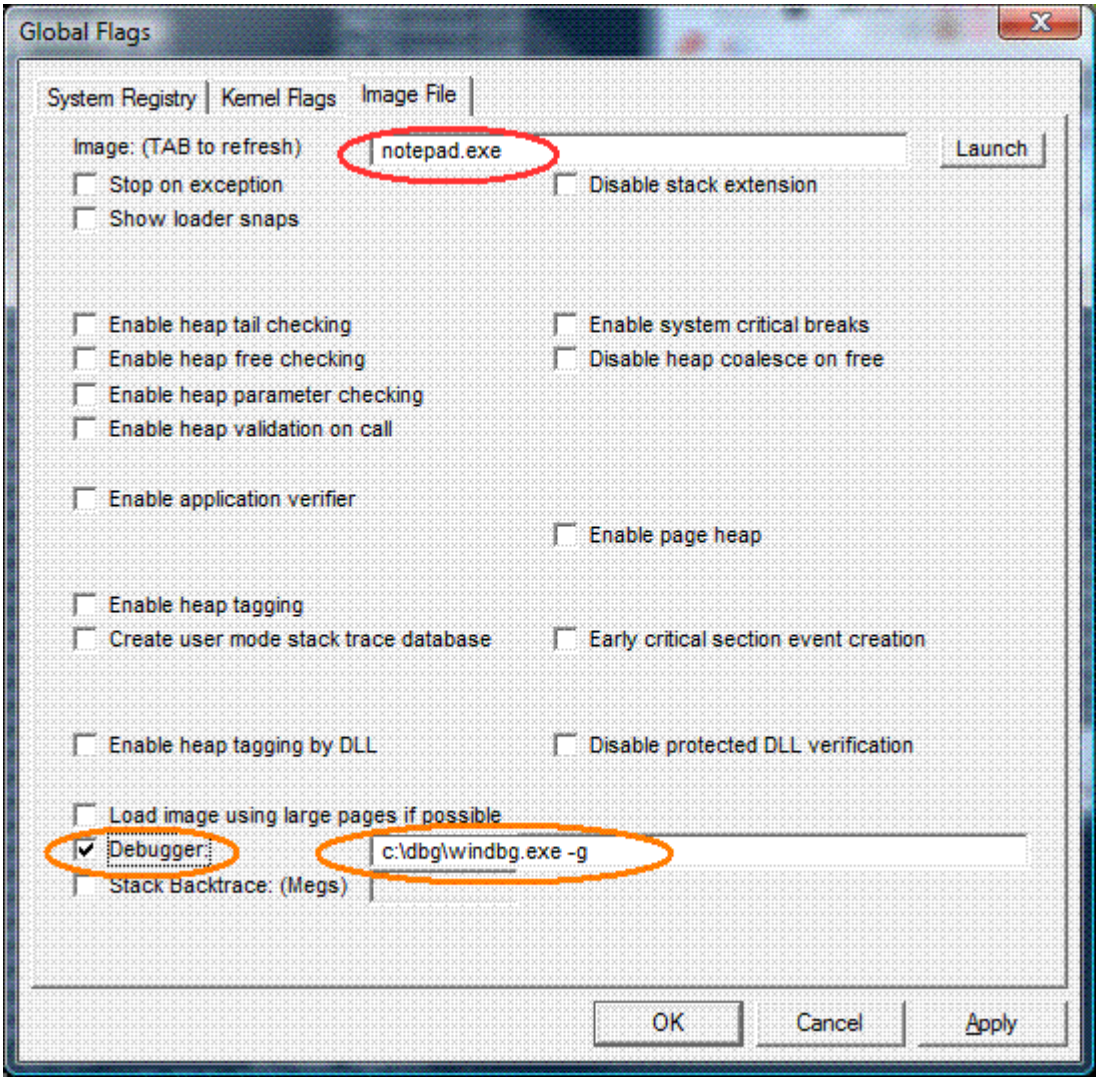
You need to create a registry key and populate it with a value as follows -

Key	**"HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\<executable>"**
Value	**Debugger : REG_SZ : <full-path to your favorite debugger>**

You do not need the full path to the application, only the exe name will suffice. However you do need the full path to the debugger. As an example, we look at launching notepad under ntsd, you would be creating the following -

Key	**"HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\notepad.exe"**
Value	**Debugger : REG_SZ : "c:\dbg\ntsd.exe -g"**

You can also use Gflags to set IFEO too -



How does IFEO work?

Kernel32!CreateProcess when called without the DEBUG_PROCESS or DEBUG_ONLY_THIS_PROCESS creation flags, checks the registry to see if IFEO has been set on the executable that it is launching. If yes, then it simply prepends the debugger path to the executable name, effectively getting the executable to launch under the debugger. If you do not specify the correct path to the debugger, then you'll probably get greeted with a "file not found" error. In our notepad/ntsd example above, Kernel32!CreateProcess ends up invoking -

"c:\dbg\ntsd.exe -g notepad.exe"

Now ntsd eventually launches notepad under the debugger by calling Kernel32!CreateProcess with one of the following creation flags - DEBUG_PROCESS or DEBUG_ONLY_THIS_PROCESS. The presence of any of these creation flags forces Kernel32!CreateProcess to bypass IFEO options this time around (else we would have been running into an endless loop) and actually launch the executable under the debugger.

IFEO and 64 bit -

A word of caution - For 32 bit executable running in the WOW on X64 machines, your natural tendency might be to create the registry key in the syswow node -

"HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\<executable>"

However Gregg Miskelly notes that you should set the IFEO corresponding to the bit-ness to the application calling into kernel32!CreateProcess to launch the executable -

"On Win 64, there are two copies of HKEY_LOCAL_MACHINE\Software (one for 32-bit apps, and one for 64-bit apps), and therefore there are two copies of these options. However, where the operating system looks isn't dependant on the bit-ness of the application that is going to be debugged (which is what you would probably expect). Instead, it is dependent on the bit-ness of the application that called CreateProcess."

Other IFEO caveats -

Raymond Chen notes the following caveat in his blog entry -

"If you passed special parameters via the STARTUPINFO structure, those parameters get passed to the debugger. And the PROCESS_INFO that is returned by the CreateProcess function describes the debugger, not the process being debugged."

IFEO and Managed debuggers -

IFEO can only be used for native or interop debugging, but not for managed debugging. Mike Stall has an excellent blog entry that describes in great detail exactly why. The gist is this - Managed debuggers like Mdbg/cordbg/VS.NET use ICorDebug::CreateProcess to launch managed executables under the debugger. However for managed debugging, the debugger should call ICorDebug::CreateProcess without the DEBUG_PROCESS or DEBUG_ONLY_THIS_PROCESS creation flags (this is publicly documented). This API internally ends up calling Kernel32!CreateProcess without the DEBUG_PROCESS or DEBUG_ONLY_THIS_PROCESS creation flags. This leads to the endless loop that I described above. Is this an ICorDebug API design flaw? Not really. Just an oversight in my opinion - The API designers missed one scenario. Maybe in the next version of the CLR this will be fixed (I do not know for sure).

Some excellent resources on IFEO -

Exercise for reader -

Does IFEO work with other Win32 APIs like ShellExecute, CreateProcessAsUser, CreateProcessWithLogonW and CreateProcessWithTokenW?

TIP of the day -

Question - System services can launch before the user has a chance to log on. So how do you debug the startup code of these system services?

Answer - Put the machine under kernel Debugger (KD), use IFEO to launch the service under NTSD (use ntsd's "-d" option to pipe the ntsd output to KD) and reboot the machine. When the system service launches, it will be launched under ntsd. The ntsd debugger will automatically cause it break into KD when it encounters the initial

loader breakpoint. The debugging session will begin in user mode automatically (yipee!). After you are done debugging, switch control to KD by issuing ".breakin" command.
