



What is DxWnd?

DxWnd is a Win32 hooker that intercepts and alter the behaviour of window messages and APIs by means of event interception and code injection directed to the configured tasks in order to get a proper behaviour of fullscreen programs, but in a windowed environment..... too complicated? Well, actually DxWnd is a tool that does its best to let you run fullscreen applications in a window.

Is that all? Well, no, not really..... Taking advantage of the hooked logic injected in the application, DxWnd does some more little tricks, that fall in two general categories: making your program more compatible to different environments and altering its time flow. So, that makes some old programs able to run on modern platforms (well, at least sometimes...) and to increase or decrease the game speed at your will. Have you always been a complete nerd with FPS? Here comes your revenge: bullet time wherever you need it!

Why DxWnd?

This is not a silly question. Actually, you may think to two different questions:

- 1) Why should I use DxWnd?
- 2) Why someone should spend his time to develop it?

Why should I use DxWnd?

Let's start with the first one. Once upon a time (in the IT clock, that means a few years ago) people used to have clumsy PCs that in a slow and single-threaded environment tried to do their best to impress their owner with strength demonstrations, usually trying to move colored pixels on the screen in the fastest possible way. That custom was named videogaming, and implied using all tricky ways to improve the system performances. One common way to do that, was to highjack all hardware resources and dedicate them for this single purpose, of course disabling any attempt to run parallel tasks in other portions of the screen: the classic example being any videogame developed for Win95 and further.

Now, you guys may ask yourselves why should all this time be passed away and PC increased their power by a 100x times more, to keep playing the very same game in the very same environment. Someone is calling you on a chat? A new email message is arrived? You want to browse the net meanwhile? Something is happening on your favourite social network or MMPORPG? Forget about it! You're currently dealing with a task that wants 100% of your attention, even if it uses 1% of your PC power. So, why not attempting to push this old and invasive application within a window of its own ?

That's what DxWnd is mainly for: let fullscreen applications run pretending they're still in a fullscreen context, but actually within their own separate window. And, taking advantage of the code hooking needed to do so, in some case it may even happen that things are further improved, but we'll see this later.

A second undoubtful fact is the technological evolution that has turned games from different windows operating systems (through Windows 95 to current Windows 8), CPU architectures (16, 32, 64 bits) and from old and tricky directdraw support to recent 3D libraries like direct3d 8/9/10/11, and OpenGL. This evolution has left some victims behind: a lot of wonderful games are no longer supported in the current environment, even despite the efforts Microsoft is making to support legacy. In some cases, DxWnd is crucial to recover these old glories of the past.

Why someone should spend his time to develop it?

Now the second question: why someone should ever bother to develop a thing like this? This is different story. I started looking for a window-izer for a specific purpose: not having a dual monitor PC at home, I was looking for a way to debug fullscreen videogames.

Looking in the net resources, I got references to an asian (japanese?) DxWnd project that seemed discontinued, but left an old copy of the C++ sources (unfortunately, not the most recent release) to be downloaded. After that, there were several attempts to translate and improve the program, but none shared the sources again. When I opened the project trying to understand the basic principle, I found that it was incredibly simple and yet sophisticated, acting I think in a very close way as virus or anti-virus programs do.

So I just thought it was such a pity that this incredible piece of artwork of C++ programming could be left discontinued, and then I decided to "adopt" the project and continue it, even if in the meanwhile I bought a second monitor for my domestic PC. And for the same reason, I published the source code on sourceforge, a proper location for any open source piece of coding, and I encourage anyone to join the project and extend it further on. And let me thank again the mysterious coder whose only trail left to make a reference is SFB7: whoever you are, SFB7 (if this was your nick), thank you.

From the time I published the first DxWnd releases, then, a lot of improvements have been made, most of them involving sophisticated techniques that I learnt from several great teachers, coming from SourceForge, CodeProject and anywhere else in the net. Thank you all, open source supporters!

How does DxWnd work?

Well, actually there are several different ways you may write a fullscreen application, and that's why there are corresponding different ways to handle it hence some annoying configuration to do before.

Please, bear in mind that DxWnd is still an experimental program, and then its configuration is still a little clumsy. This aspect will be improved and simplified at proper time, later on.

Anyway, these are the basic principles of the DxWnd behaviour:

1. DxWnd DOES NOT alterate in any way the behaviour of your software (either system or applications) when not active. When turned off, everything behaves as if DxWnd never run on your machine, or never existed at all.
2. DxWnd DOES alterate the behaviour of your application software when running: it hooks custom code that changes the applications' behaviour, hopefully in a positive way, but you never know. It's possible that because of hacks to the directdraw or other system code there might be annoying effects such as frozen screen, unresponsive keyboard and so on. Be patient and maybe you'll find a good game setting to play without side effects.
3. This is tricky: unless you need code injection support (this will be explained later) when running, DxWnd affects ALL games in the shown list, no matter whether the cursor is highlighting a particular one, or if you started your game outside the DxWnd interface. That's why you need not activate the game from the DxWnd menu, but you could keep managing it as usual (clicking on desktop icons, shortcuts or whatsoever). So, remember this: whenever DxWnd is running, it impacts on ANY game it is configured on its game list, no matter if you didn't strt it from DxWnd interface.
4. Again, DxWnd is currently coded to make ONE SINGLE game working at a time, even if it could be possible to start and intercept more than one in parallel. In some cases, the games work together, but unpredictable things happen for instance when you try to control more than one game at a time. Maybe one day it will make it possible to play more games in parallel, but so far that feature is unsupported, so DxWnd is operating on one game (the fist started up) while the others will not be effected and should start normally in their original fullscreen mode.

DxWnd stores ALL its settings on a couple of configuration files (dxwnd.ini for almost everything, and dxwnd.reg for altered registry keys configuration) in the very same folder where dxwnd.exe and the hooker dxwnd.dll are located. No info is written in the registry or anywhere else in the system. No installation procedure is required, just copy the files where you like better,

create your own shortcut entries wherever you like and, whenever you're satisfied with some DxWnd setting, just back-up the configuration by simply copying the dxwnd.ini file somewhere else. Also, keep in mind that ALL changes are written on disk just when DxWnd exits safely, so whenever it crashes your configuration changes are certain to be lost.

From release 2.02.22, then, it's also possible to export and import single pieces of configuration to separate files, so that people are encouraged to share working game configurations by sharing these files only.

The command line arguments

DxWnd accepts a few command line arguments, that can all be combined together to alter his behaviour:

/T	Starts DxWnd iconized in the System Tray (see DxWnd in the System Tray)
/I	Starts DxWnd initially in the IDLE state, so that it doesn't effect the programs until you manually issue a Hook▶Start command
/C:<filename>	Uses the <filename> configuration file instead of the default config.ini file. In any case, the configuration file must be located in the same DxWnd execution folder.
/lang=<xx>	If you need a localized version of DxWnd, this argument causes DxWnd to load all resources text from external dlls, whose name contains the <xx> pattern.

The Application GUI

DxWnd comes with a nice and simple Graphic User Interface: when started, it shows a form pretty much like the one in the following picture:

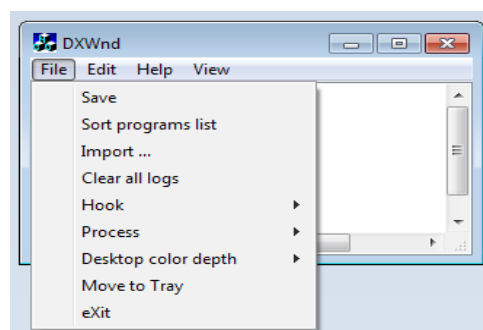


In the main window there is the list of hooked programs: DxWnd can currently handle up to 256. Trying to add more than that will give an error message. Keep in mind that DxWnd bundles contain an export subfolder where all supported games have their own default configuration ready to be imported, but because of the program absolute path value, these entries will actually work only after updating the path with the proper local value.

As shown in the picture, each configuration line includes an icon which color tells the general status of the program, as follows:

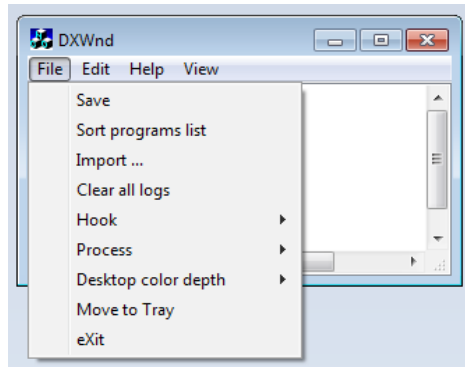
<input type="radio"/> Age of Empires III	blank icon: the configuration refers to a not existent program's path, so that the entry won't work unless the path is corrected.
<input type="radio"/> AfterLife	grey icon: the hook is not enabled: this program can be activated by the DxWnd menu, but won't be windowed.
<input checked="" type="radio"/> Age of Empires 2	green icon: the hook is enabled and the program will be windowed when run either from the DxWnd interface or however else.
<input checked="" type="radio"/> Age of Empires	red icon: the program requires code injection, then it will need to be activated from the DxWnd interface only.

You can activate command either via the top menu, or by right clicking on a row in the application list. These are the commands:



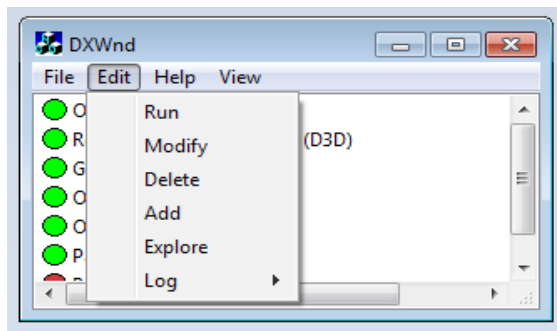
DxWnd is also able to operate iconized in the System Tray, from where it shows its state (either IDLE, READY or RUNNING) and run a few useful commands.

DxWnd detects the video settings when is started, and compares it to the current value after killing a task or terminating itself: in case it finds differences, it prompts you asking whether you want the previous screen setting to be restored. This is quite useful to handle all the games that terminates without restoring the previous setting, as it may happen when they die abnormally.

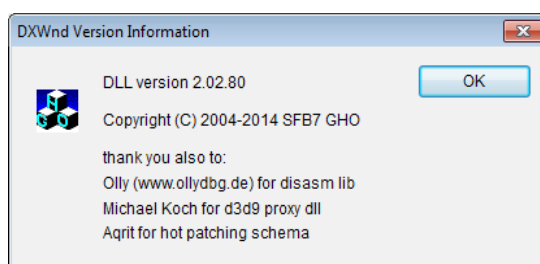
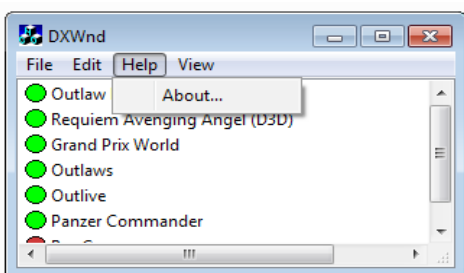


File → Save	Saves the current configuration to disk.
File → Sort program list	Arranges the program list in ascending alphabetical order (sort).
File → Import ...	Imports one program configuration from an external .dxw exported file
File → Clear all logs	Turns all tracing options off for all games in the list and deletes any dxwnd.log file.
File → Hook → Start / Stop	Hook Stop is a handy way to prevent DxWnd to do its job, pretty much the same to stop the program, but leaving it running (in the IDLE state). Hook Start restores the default behaviour (the READY state, or RUNNING when operating on a task).
File → Process → Pause	Tries to pause the program by lowering each thread priority to the minimum.
File → Process → Resume	Restore threads priorities.
File → Process → Kill	Kills the last process activated by the DxWnd interface. Very useful to get rid of games gone crazy because of DxWnd that refuse to terminate themselves.
File → Desktop color depth → 8 / 16 / 24 / 32 BPP	On recent platforms, low color depths are no longer supported for the desktop, though still working. This menu let you ask the system to set the current color depth to 8, 16, 24 or 32 bits per pixel. Of course, it is possible that some color / resolution compinations are not supported.
File → Move to Tray	Move DxWnd in the System Tray, where a dedicated icon will show its state and allow a few commands, including the possibility to show the application window again. Note that once DxWnd goes in the System Tray, it always stays there also when it is made visible again.
File → eXit	Exits DxWnd. Beware that if a game was activated while DxWnd was active, it will very likely crash

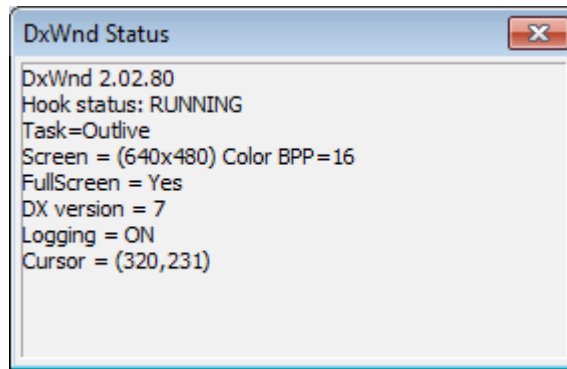
after the DxWnd termination, so a check is made and you'd be prompted to confirm the operation.



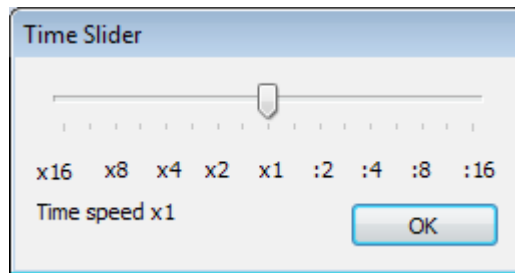
Edit → Run	Starts the currently selected application
Edit → Modify	Opens the configuration panel to set/change the selected program settings
Edit → Delete	Deletes the selected application entry (asking a Yes/No confirmation)
Edit → Add	Inserts a new application entry in the list. The configuration panel is opened to let you define the initial settings.
Edit → Explore	Opens Microsoft Explorer to the folder where the application is located. This is a shortcut to something usually useful.
Edit → Log → View	Opens the dxwnd.log logfile of the selected application, if existing. Beware that in order to do so, you should "associate" the log file extension to your preferred text editor before.
Edit → Log → Delete	Deletes the logfile of the selected application, if existing.



Help → About	Shows the program version and references the development team and project supporters: currently SFB7 whoever he/she might be, GH0 (that is myself), Olly for having developed both OllyDBG and the disassembly library I'm using and Aqrit for so many hints, informations and pieces of code.
--------------	--

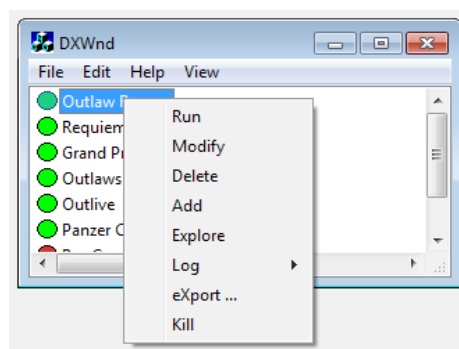


View → Status	This command shows a status window with informations about DxWnd and the hooked program.
---------------	--



View → Time Slider	Shows a time slider window that can be used to know and dynamically alter the time flow speed
--------------------	---

When right-clicking on the program's list, instead, the following menu will be shown:

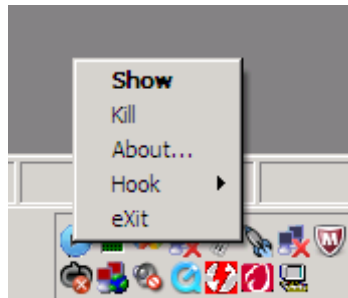





Run	Activates (run) the selected task. Same as double-clicking on the list entry.
Modify	Shows and let you change the program's configuration.
Delete	Deletes (after a confirmation command) the selected

	entry.
Add	Prompts for all data needed to define a new task in the list.
Explore	Open a window explore session pointing to the program's configured install path.
Log → View	Same as Edit → Log → View
Log → Delete	Same as Edit → Log → Delete
Export ...	Exports the highlighted program configuration to a file
Kill	Kills the corresponding program. Differently from the File → Process → Kill command, this command would not kill the currently active program, but the selected one, no matter whether it was managed by dxWnd or not. This is the reason why sometimes the first Kill command may fail and this one is in general more reliable, at the cost of selecting the proper entry.

DxWnd in the System Tray

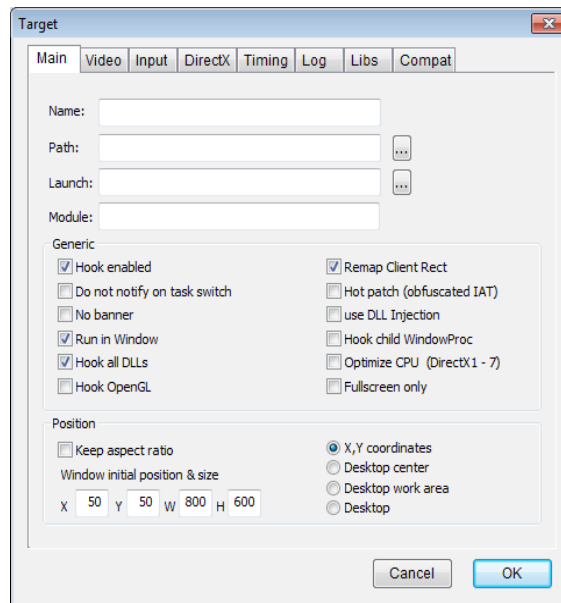
Once you move DxWnd in the system tray and until the program is terminated, an icon will be visible in the system tray. Right-clicking on the icon, you get a subset of the DxWnd commands, plus the Show command that shows the DxWnd window again. The Show command is the menu default, so you can activate it also by double-clicking on the DxWnd tray icon.



	READY state: DxWnd is ready to hook a program
	IDLE state: DxWnd is running, but will not affect any program
	RUNNING state: DxWnd is currently operating on a program

The configuration panel

Through the Add or Modify command, this tabbed panel will be shown:



Each panel defines the configuration for any given program's characteristics.

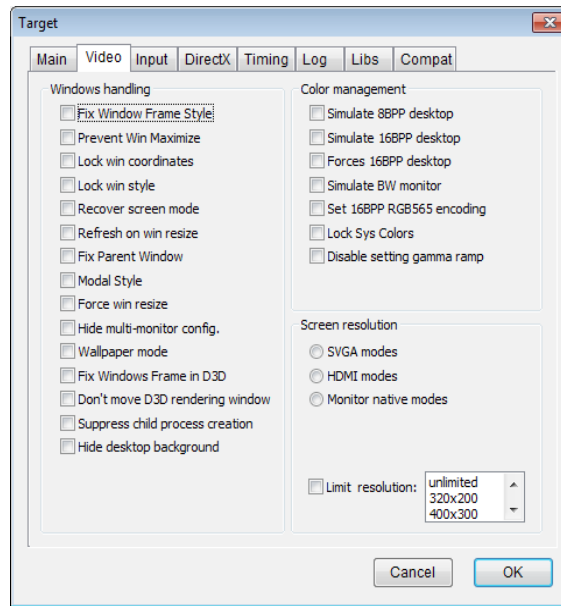
the Main tab:

Name:	The user defined program name, to allow you to label your application with an evocative naming, possibly including qualifiers, versioning etc. If unset, DxWnd will insert here the task filename.
Path:	The pathname of the task to be activated / hooked.
Launch:	In some cases, the program to be hooked can't be run by launching itself directly, but needs arguments or other environment elements provided by some frontend father program. To simplify the program activation, the frontend pathname can be written here: when set, the run command does not run the path in the field above, but this one.
Module:	In some occasional cases, some dlls may get unnoticed to the DxWnd hooking logic. In these fortunately rare cases, you have a chance to make the program working by referencing here one or more module names to be added at the DxWnd search algorithm.
Hook enabled	If this flag is not set, DxWnd ignores the task - see gray icon
Do not notify on task switch	Inhibits the task switch notification message that may hurt some games not designed to handle it properly
No banner	Well, DxWnd celebrates itself a little by showing a very short splash screen at the beginning. If you're not happy with this, checking this flag will disable the show.
Run in window	Checked by default, tells DxWnd to try to run the program in windowized mode, that is the essential reason why DxWnd exists. But if not checked DxWnd still performs all other functions not related to the screen size, such as time stretching, compatibility options and so forth.
Hook all DLLs	The original DxWnd behaviour was limited to search & hook calls made by the main program directly. Checking this flag cause DxWnd to recurse in all non-system DLLs address space to hook calls there. It's necessary in all cases where the graphic engine is not directly coded in the program, but it's implemented in a separated engine DLL.
Hook OpenGL	Enables OpenGL API hooking
Remap Client Rect	Enabled by default, makes DxWnd remap the window client coordinates so that the program receives the same values as if the program was running in fullscreen mode.
Hot patch (obfuscated IAT)	The original DxWnd used IAT patching to redirect API calls to the altered routines. This method has its advantages, but fails when not all API are reached because they are located in unconnected dlls, referenced by ordinal number or referenced by programs with obfuscated IAT. IAT obfuscation is a sophisticated but common enough technique to make hacker's life harder: for instance, the game executable of Doom III has an obfuscated IAT. Checking this flag cause DxWnd to use an alternate patching technique, that is the "hot patching" that creates a detour assembly code right at

	<p>the beginning of the API implementation. Once done EVERY SINGLE CALL gets intercepted no matter from where the call is made, but it isn't always possible to apply this technique. Luckily, in the vast majority of cases, they both work.</p>
Use DLL injection	<p>The basic hook technique intercepts the first window creation event. At that time, the program may have done unwanted actions already, such as changing video mode or detecting bad conditions or crashing. Checking this flag cause the DxWnd logic to be "injected" right at the beginning of the task execution, making DxWnd able to intercepts all events. The drawback is that this only works when the task is activated from the DxWnd interface - see red icon.</p> <p>Another drawback is that the injection process resembles pretty much of an activation from a debugger, increasing the chances for game protections to intercept this situation and stop the program.</p>
Hook child WindowProc	<p>By default, DxWnd intercepts the WindowProc routine of the main window, and this is enough for its purposes. In some cases, though, this is not enough and this flag tells DxWnd to intercept and redirect the WindowProc routines of all child windows as well.</p>
Optimize CPU (DirectX 1-7)	<p>Optimizes the CPU load, but only for ddraw operations (DirectX1 to DirectX7)</p>

Keep aspect ratio	<p>When the window is resized, the aspect ratio set by the window initial size is preserved (by default the 4:3 aspect ratio such as 800x600).</p>
Window initial position & size	<p>Four values for the initial X, Y coordinates of the upperleft window corner and the window width and height. All values are referred to the window client area rather than the outside border. The values are used depending on the Position selection</p>
Position	<p>A selection of 4 possible cases:</p> <ul style="list-style-type: none"> • X, Y coordinates: the window is placed at the chosen coordinates • Desktop center: the window is centered on the screen, and only the width and height fields are used. • Desktop work area: the window occupies the whole screen but the bottom taskbar. • Desktop: the window occupies the whole screen, as if it was fullscreen (a.k.a. Fake-fullscreen mode)

the Video tab:



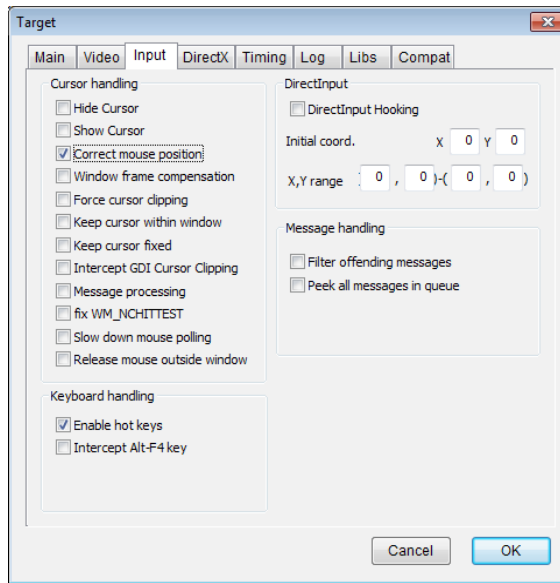
Fix Window Frame Style	Initializes the game window with a title bar and resizeable borders.
Prevent Win Maximize	Some modern games don't actually go in fullscreen mode, but just make the window occupy the whole screen. The option intercepts Windows messages and user32 calls to avoid changing the window position and size to make it a full-screen window.
Lock win coordinates	Intercepts messages and calls that the program makes to himself to change its own window coordinates. In this way, though, the game window becomes fixed in position and size.
Lock win style	Intercepts messages and calls that the program makes to himself to change its own windowstyle.
Recover screen mode	Sets the screen mode to registry default settings. In general, DxWnd intercepts any attempt to change display settings and prevents unwanted operations, but still some programs have display settings instructions before DxWnd could possibly intercept them (e.g. before the window is created and the windows hook is invoked), so that changing the display settings right after is the only possible solution. Try this when other options don't work.
Refresh on win resize	Any decently written windows application should take care of refreshing the screen primary surface when resized, and most fullscreen games do it. Some don't (they were not supposed to ever be resized, actually). This flag is to force a refresh (useful for "Uprising").
Fix Parent Window	Typically, a game is started with an invisible program window, and then created a separate and child window for handling the graphic. Some games don't use the child window, but they rather use the parent one. In this case, the parent window becomes visible, then needs to be properly resized. It's experimental, for now, but seems to be able to manage successfully several tough games: Solaris, SleepWalker, Sid Meier's Sim Golf, the Worms serie....
Modal Style	When "Fix Window Frame Style" is set, a borderless and titleless modal style is chosen instead of the default one.
Force win resize	Experimental (and not working very well so far): should force the processing of window resizing messages so that the window can be resized by dragging borders.
Hide multi-monitor config	Makes the program ignore that there are multiple monitors in your system configuration, giving informations about the primary monitor only.
Wallpaper mode	Experimental - forces the program Z-order to the lowest level so that it runs below any other task, like an active wallpaper.
Fix window frame in D3D	Tries to avoid D3D to render on the whole window surface including the

	<p>window border. It activates a small trick that cause the program to render to a child modal surface within the main window borders.</p>
Don't move D&D rendering window	
Suppress child process creation	<p>Suppressing the birth of child processes is necessary whenever the task is running child processes as video players, splash screens or similar things. In this case, hooking more than one process may be difficult and not worth the result.</p>
Hide desktop background	<p>Starts the windowized program together with four black borderless windows that surround it entirely giving a better feeling of concentration. The whole idea was borrowed from "Mr. Hide": https://sourceforge.net/projects/mrhyde/</p>

Simulate 8BPP desktop	<p>Some games pretends you switch the video mode to 8BPP before you activate them, making it useless the 8BPP emulated mode. This flag just let the program believe that the desktop setting is in 8BPP mode already.</p>
Simulate 16BPP desktop	<p>Same as above, but declaring a 16BPP setting. These two flags should not be set together.</p>
Forces 16BPP desktop	<p>In some cases, the game really needs a 16BPP desktop, but does not contain the code for activate this color depth, and on modern platforms, though supported, it is very difficult (Win7) when not impossible (Win8) to do it manually. This flag switched the desktop to 16BPP before the game would complain.</p>
Simulate BW monitor	<p>Activate a tweak in the palette handling that causes all colors to be replaced with the corresponding grayscale color. It works only on 8BPP palettized games or emulating 16BPP on a 32BPP desktop.</p>
Set 16BPP RGB565 encoding	<p>By default, DxWnd emulates 16BPP color with RGB555 encoding. The option forces RGB565. Thi option, of course, impacts the video only in emulation mode and for 16BPP color depth.</p>
Lock Sys Colors	
Disable setting gamma ramp	<p>Disables API trying to alterate the default gamma ramp making the screen lighter or darker. Since there API affect the whole screen, this flag is mainly useful to avoid the background desktop to be affected.</p>

Screen resolution	<p>Affects the resolutions detected by the application. There is a choice of the following values:</p> <ul style="list-style-type: none"> • SVGA modes: the classic 4:3 screen resolutions starting from 320x200 up to 1280x800 • HDMI modes: the typical 16:9 resolutions from 640x360 up to 1980x1080 • Monitor native modes: whatever returned from the video card <p>Notice that 320x200 resolution is typically no longer supported, but it is not a problem to emulate it in window and this resolution is necessary to run some older games.</p>
Limit resolution	<p>Disables any resolution higher than the selected value. By default, the choice is "unlimited", that means no resolution is disabled.</p>

the Input tab:



Cursor Handling

Hide cursor	Forces hiding the hardware cursor.
Show Cursor	Forces showing the hardware cursor.
Correct mouse position	Compensate for X,Y mouse coordinates when the window is moved or resized. It should be typically set for most games.
Window frame compensation	In some cases, mouse X,Y coordinates appear displaced of the same amount as the window border (top and left). Checking this flag causes DxWnd to subtract this value and recover the error.
Force cursor clipping	Set hardware cursor clipping within the window's region. It greatly improves the game playability in some cases (namely, the Dungeon Keeper series)
Keep cursor within window	Avoid moving the cursor outside the window area. Doing so was used as "Cursor OFF" directive in some games.
Keep cursor fixed	Inhibits the SetCursorPos() API: in some cases, it affects the program's behaviour (e.g. "Necrodrome").
Intercept GDI cursor clipping	Disables GDI clipping, avoiding unpleasant effects such as the mouse that is not free to move within the whole window.
Message processing	Most programs get X,Y mouse coordinates from the mouse messages or from the specific API. One tricky way to get the same info, though, is to listen from the windows message queue using PeekMessage / GetMessage, and retrieve the X,Y coordinates from ANY received message in the pt field. Checking this box make DxWnd to fix the X,Y coordinates on this uncommon situation as well (see "Uprising").
Fix WM_NCHITTEST	
Slow down mouse polling	Some old programs have this bad habit to continuously loop through the mouse status polling with no delay, using 100% of CPU time. This flag introduces a minimal and unnoticeable delay between mouse polls, saving most of CPU time.
Release mouse outside window	Normally, when the mouse is placed outside the window and the window keeps receiving mouse messages, the mouse cursor is placed on the corresponding window border and the program performs video scrolling or so forth. There are some cases in which you don't want this to happen, for instance when you want to use two programs alternatively, such as a game and a keyboard simulator. Checking this flag causes DxWnd to detect the mouse outside window condition, and in this case it places the cursor right in the middle of the screen, where it is supposed to make no harm.

Keyboard Handling

Enables hot keys	DxWnd can set some special keys (Alt-Fn) to trigger special actions, storing the key association in the DxWnd.ini file. This flag enables the hot keys definitions. If unchecked, all hot keys are disabled for this program.
Intercept Alt-F4 key	Intercepts the Alt-F4 key in the message processing loop to immediately terminate the program, avoiding any programmed exit procedure (out-tro, savegame warnings, ads...). Of course, IF the game is doing the message processing loop!

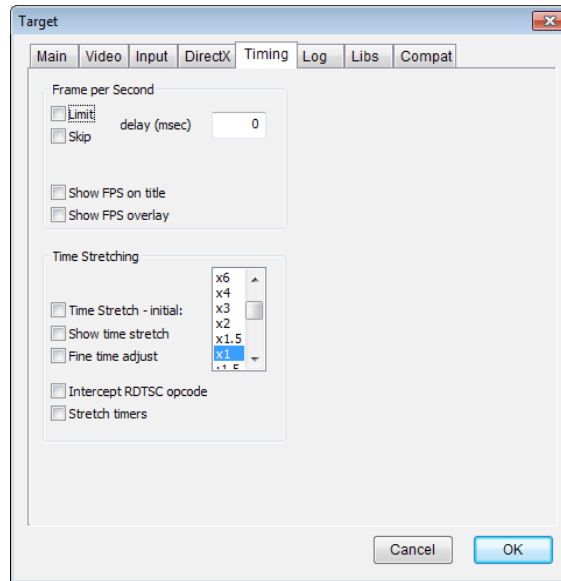
Message Handling

Filter offending messages	In theory, a fullscreen message should not expect several messages that are impossible to receive, such as border sizing, taskbar events and so forth. Some programs are not properly coded to react to such "impossible" messages and may show falfuncions. This flag causes potentially harmful and meaningless messages to be suppressed.
Peek all messages in queue	

DirectInput

DirectInput Hooking	Enables hooking on DirectInput methods
Initial coord.	Initial coordinates for the DirectInput mouse coordinates
X, Y range	Range of X, Y values (min & max) that is permitted for the mouse coordinates when detected by DirectInput methods.

the Timing tab:



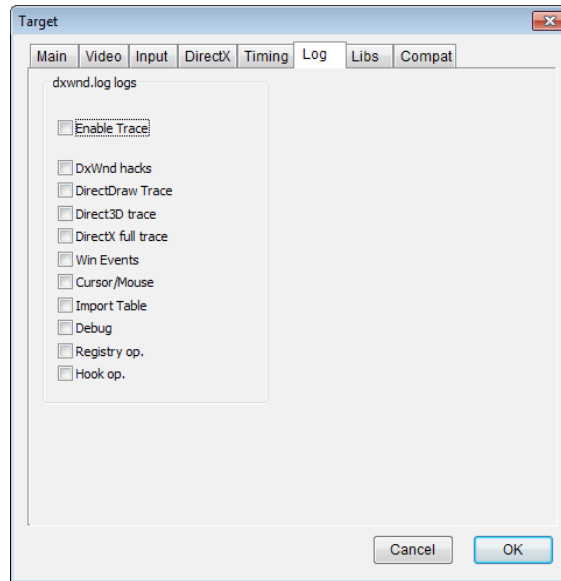
Frame per Second

Frame per Second - Limit	Introduces a configurable delay between screen refresh operations, so that the FPS is limited accordingly
Frame per second - Skip	Introduces no delay in the screen operations, but skips several screen updates so that the actual FPS value is limited without the program noticing it
Frame per second - delay (msec)	The delay expressed in milliseconds to be used either for Limit or Skip operations
Show FPS on title	When checked, the FPS counter is appended to the window title
Show FPS overlay	When checked, the FPS counter is drawn as an overlay of the program client area, in a corner of the screen and periodically and randomly moved to other corners to avoid accidentally overlap an important screen region.

Time stretching

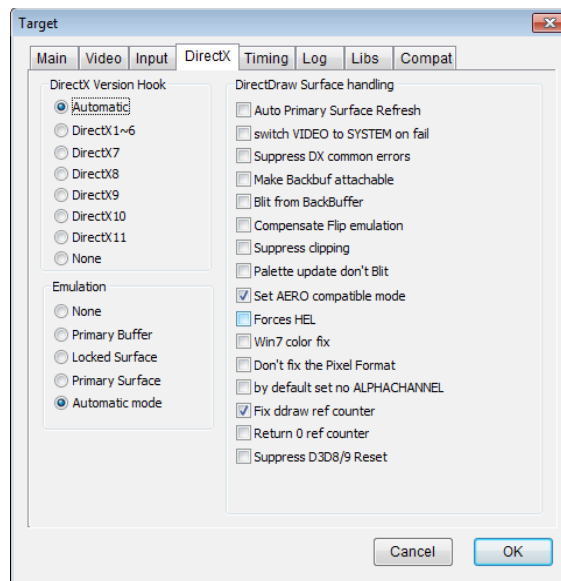
Time Stretching - initial:	If this option is checked, DxWnd tries to emulate an accelerated (xn) or decelerated ($:n$) time flow. The value set in the listbox is the initial value, that can be altered by means of the time control slider in the time panel. When checked, DxWnd stretches time in timing API such as <code>GetSystemTime()</code> , the query performance APIs and the <code>Sleep</code> APIs, that are the most used ways to control a program timing.
Show time stretch	When checked, the time stretch ratio is drawn as an overlay of the program client area, in a corner of the screen and periodically and randomly moved to other corners whenever it is updated.
Fine time adjust	When checked, the time stretch ratio coarse becomes finer, from a 1.5 (50%) ratio to a 1.1 (10%) ratio for each tick. This allows a better and finer control of timing, despite the more limited range (about 0.5x up to 2.0x)
Intercept RTDSC opcode	The platform performance counter should be determined by calling the <code>QueryPerformanceCounter()</code> API. But there is a more direct way, that is calling the assembler RTDSC family instructions. When checked, DxWnd looks for RTDSC and RTDSCP instructions by disassembling the program text segment and replaces them with stretchable time counters.
Stretch timers	When checked, the window timers are stretched, namely the <code>user32.dll</code> timers set by <code>SetTimer</code> and the multimedia timers in <code>winmm.dll</code> set by <code>timeSetEvent()</code>

the Log tab:



Enable Trace	This works as a global flag that enables/disables all subsequent traces. If unchecked, no output is written. If checked, error messages, plus the specific messages related to other flags (see below) are written in the dxwnd.log file in the program's execution directory.
DxWnd hacks	Enables the operation logging of all significant events that DxWnd performs to bring the fullscreen program in windowed mode.
DirectDraw trace	
Direct3D trace	
DirectX full trace	Enables extended logging of all DirectX operations, no matter whether they are related to fullscreen / windowed mode or not.
Win Events	Enables logging of all Window messages intercepted in the application's queues, together with events that are generated or processed internally by the Peek/GetMessage APIs.
Cursor / Mouse	Enables extended logging of all cursor or mouse related operations. ** BEWARE ** some old games don't mind the possibility of concurrent use and perform mouse/cursor operations in close loops, so that this type of log can quickly grow quite big in size. In this case, consider the possibility to slow down the program by using the "Slow Down" flag.
Import Table	Enables extended logging of the Import Table as seen by the DxWnd program. This can be quite useful to analyse and troubleshoot uncommon executables (e.g. when copy protections are applied).
Debug	Writes some more detailed information for diagnostic purposes.
Registry op.	
Hook op.	

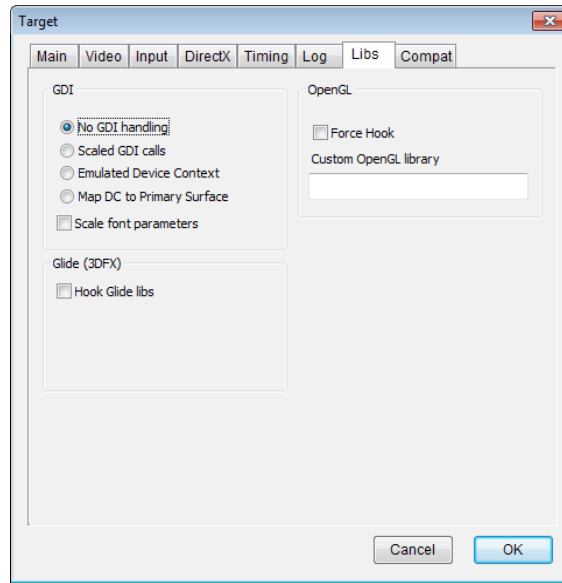
The DirectX tab:



DirectX Version Hook	Sets the basic intervention strategy: depending on the game technology, a different hooking technique should be adopted. Automatic tries to find it by itself, but it doesn't always succeed! OpenGL handling require a dedicated setting (see libs tab)
Emulation	<p>Defines the basic surface emulation strategy: either "None" (the program controls the desktop color depth), "Primary Buffer" (same as 'none', but blit operations are made against a memory surface and then transferred to the real primary surface - this handles the otherwise known pich-bug problem) or "Primary Surface" (the virtual primary has a different color depth of the real primary, and DxWnd takes care of the color transformation internally: slower but no screen mode changes!).</p> <p>The "Automatic mode" uses a very simple strategy for Direct3D games: it tries the "Primary Surface" mode first and, if that doesn't succeed, switches to "Locked surface". It is the best strategy I came across, but it doesn't always work!</p>
Auto primary surface refresh	Some badly programmed games (namely the "Cossaks" series) don't follow the specification to blit changes on screen, they just get the primary surface memory address and keep writing there. The option forces a periodic blitting of the primary surface on screen even if the game doesn't request it. You want a second example? It has not been easy to find, but "Crush! Deluxe" suffers the same problem.
VIDEO → SYSTEM Surface on fail	When this option is set and a CreateSurface fails because of video memory shortage, DxWnd backs this up by creating the surface on memory. Oddly enough, some games expect to notice this by themselves and work correctly only when the option is NOT set.
Suppress DX common errors	Some games running in windowed mode generate sporadic errors that wouldn't prevent the game to work, but terminate the application. This option makes directx methods return OK condition in such common cases.
Make Backbuf attachable	Alters the size specification of the created backbuffer so that it copes with the actual primary surface, so that it may be attachable to a ZBUFFER surface. It makes "Dave Mirra Freestyle BMX" playable.
Blit from Backbuffer	Some games (the Sims, the only one so far....) read graphic data from the primary surface. When the game runs windowed, the approximation introduced in a scaled window brings cumulative error that appear as a progressive "smearing" effect. In this case, it might be better to read the data from the backbuffer surface that is not scaled, even if in such a way you get other troubles when scrolling (see it by yourself...). The only reasonable alternative: write game code in a better way, in my opinion!

Compensate Flip emulation	
Suppress clipping	DxWnd sets clipping on the primary surface. If the game does it as way, there might be interferences. As a matter of fact, setting this flag is the only way to make "Pax Imperia Eminent Domain" working correctly.
Full RECT blit	Causes every blit operation to primary surface to be applied to the full surface (NULL rect coordinates). It could be handy to recover wrong surface handling, but this situation should be classified more properly as a DxWnd BUG!
Palette update don't blit	Avoid executing a blit operation in case of palette update. This could be used to fix conflicts between GDI and ddraw palette updates
Set AERO compatible mode	Quite useful in Windows Vista to Win8 platforms, forces the program to declare its compatibility with AERO desktop mode, avoiding then the switch from AERO to standard desktop mode.
Forces HEL	
Win7 color fix	
Don't fix the Pixel Format	
By default set no ALPHACHANNEL	
Fix ddraw ref counter	
Return 0 ref counter	
Suppress D3D8/9 reset	

the Libs tab:



GDI

No GDI handling	Disables video-related GDI / user32 API hooking
Scaled GDI calls	
Emulated Device Context	
Map DC to Primary Surface	
Scale font parameters	

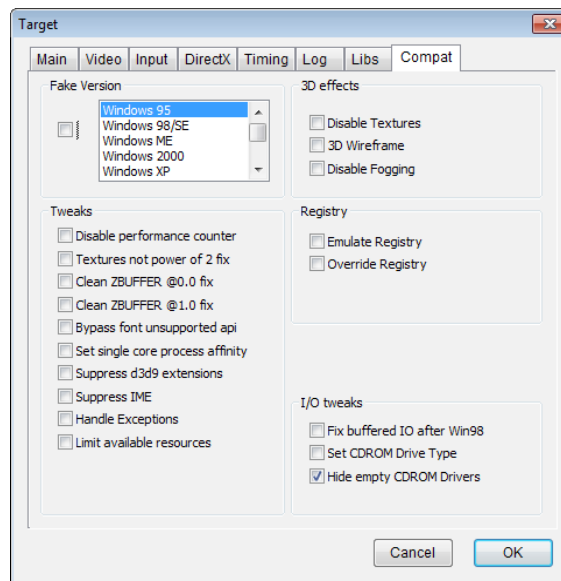
Glide (3DFX)

Hook Glide libs	Enables Glide API hooking
-----------------	---------------------------

OpenGL

Hook OpenGL	Enables OpenGL API hooking
Force Hook	Experimental - probably unnecessary
Custom OpenGL library	By default, DxWnd detects OpenGL APIs within the standard OpenGL32.dll library. The field allow to set a different filename for any custom OpenGL implementation that may refer to a different library name.

the Comaptibility tab:



Fake Version	Causes the program to detect the chosen Windows release. Notice: this is not the same thing as the compatibility setting of the Windows properties panel, that also adjust the system's behaviour to emulate the chosen platform.
--------------	---

Tweaks

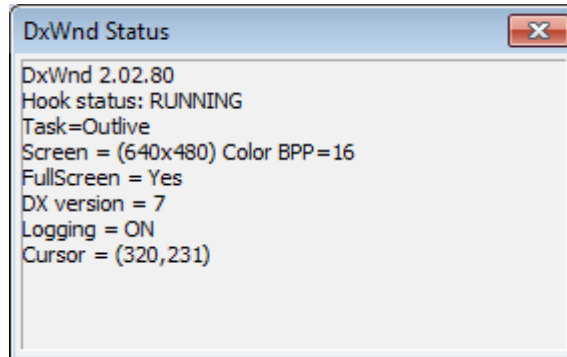
Suppress d3d9 extensions	D3d9.dll comes in different versions depending on the platform: on older windows releases it exports the Direct3DCreate API, while in more recent versions there are many further exported calls. The flag suppresses the additional entries.
Suppress IME	The flag tries to suppress IME windows, though that is not fully working yet!
Handle exceptions	Setting this flag causes DxWnd to set its own exception handler that tries (and often succeeds!) to fix several common exceptions such as the divide by zero exception of old games that were trying to calculate the CPU speed.
Limit available resources	Makes the query for available resources (either RAM, hard disk free space and so on) to return a limited value: some old games can't handle a very big integer number and see it as a negative value, refusing to start.

3D Effects

3D wireframe	As a fancy and easy action that DxWnd may implement on D3D and OpenGL programs (not ddraw ones!!), checking this flag will show the graphic in wireframe mode.
--------------	--

DxWnd Status

The DxWnd status shows the following information, refreshing them periodically each one second:



DxWnd version: in the picture, the current one: 2.01.78

Hook status: either IDLE, READY or RUNNING (see tray icons)
when running:

Running: the task name (see the configuration panel)

Screen = (width x height) colordepth, as seen by the task

FullScreen = Yes/No depending whether the task has set the cooperative level to EXCLUSIVE or not

DX Version = version of the DirectDraw / Direct3D interface currently in use (namely, the one used to create the primary surface).

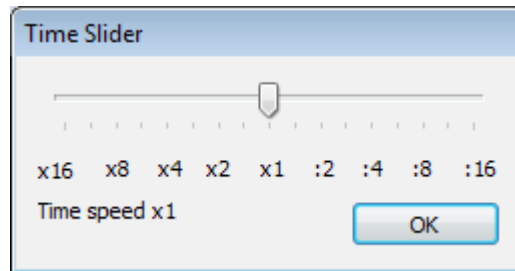
Logging = the logging flag (either ON or OFF)

Cursor = the X,Y cursor coordinates as intercepted and fixed by DxWnd

FPS = frame per second value calculated by DxWnd

Time Slider

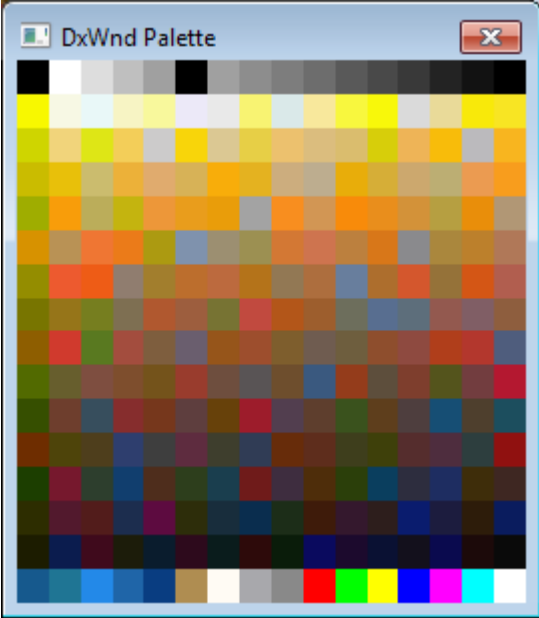
The Time Slider shows the current time stretching value, refreshing it periodically each one second. It also allow the user to alter the time stretching factor by grabbing the slider (click on the slider with the left button and keep it pressed) and moving it to left or to right:



The time stretching factors shown below the slider are related to the coarse (default) time resolution. If the fine time adjustment flag is set, the leftmost slider position corresponds approx. To x2 and the rightmost to approx. :2

DxWnd Palette

The DxWnd Palette shows the 256 colors in the current emulated palette, refreshing them periodically each one second:



Special keys

DxWnd injects in the controlled application some special keys that might be useful:

Alt-F12	When the "Force cursor clipping" option is ON, this key toggles the clipping region ON and OFF so that you can exit the game area and control other tasks or move/resize your game window.
Alt-F11	Forces a surface repaint. Some old games didn't even consider the possibility of a task overriding the game area, so they don't repaint when they should. I know this sounds a little "technicality", but if your game screen gets dirty, try this key to fix it.
Alt-F10	Toggles logging ON/OFF. Since painting operations can be quite verbose, toggling the log can be a useful trick to get information about a specific program's activity without having to browse tons of log lines.
Alt-F9	Toggle position locking: when the <i>Windows - Lock win Coordinates</i> flag is set, the window can't be moved or resized, unless you toggle this behaviour OFF, do the change and lock the position again to ON.
Alt-F8	Toggle Handle DC mode: since this DC emulation is very CPU intensive, it may be convenient to switch it OFF and back ON to make a game more fluid.
Alt-F7	Toggle the FPS display ON and OFF.
Alt-F6/Alt-F5	Increase / decrease the timeshift multiply factor when the time stretching option is set.
Alt-F4	This key is the well known quit command for any task. If the application doesn't react quickly enough to your command, you could set the "Intercept Alt-F4 key" option to cause DxWnd to immediately quit the program.