DxWnd rel. 2.01.99 unfinished Manual

What is DxWnd?

DxWnd is a Win32 hooker that intercepts the window creation event for the configured tasks and alter the behaviour of several system call (from user32.dll, GDI32.dll and the DirectX libraries) 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.

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 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.

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.

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: 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 configuration file (dxwnd.ini vy default) 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.

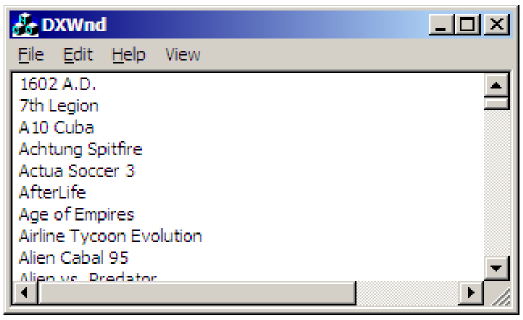
The command line arguments

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.

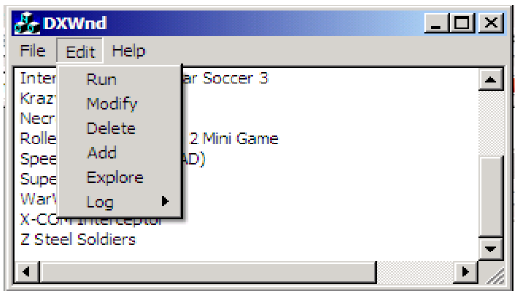
The Application GUI

DxWnd comes with a nice and simple Graphic User Interface: when started, it shows a form pretty muck 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 usually contain an example configuration files with some entries already set, but because of the program absolute path value, chances are that none of the existing entries will actually work without some corrections.

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

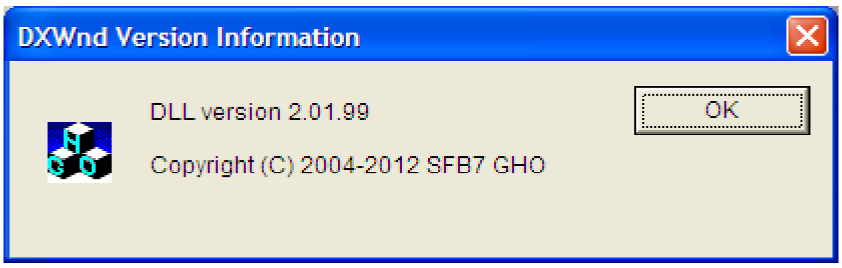


From release v2.1.68 on, DxWnd is also able to operate iconized in the System Tray, from where it shows its state (either IDLE, READY or RUNNING) and run a few useful commands.

From release v2.1.78 on, 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 now prompt the user 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.

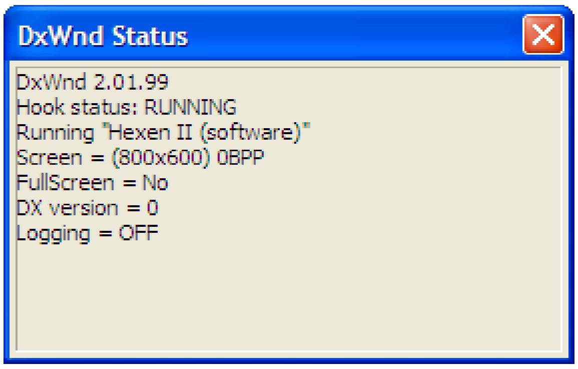
File → Sort program list	Arranges the program list in ascending alphabetical order (sort).
File → Clear all logs	Turns all tracing options off for all games in the list and deletes any dxwnd.log file.
File → Hook → Start / Stop	Hook Stop is a handy way to prevent DxWnd to do its job, pretty much the 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 → 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 → 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 (currently SFB7 whoever this might be, and GHO that is myself): see below.
View → Status	From v2.1.78 on, this command shows a status window with informations about DxWnd and the hooked program.



DLL version 2.01.99

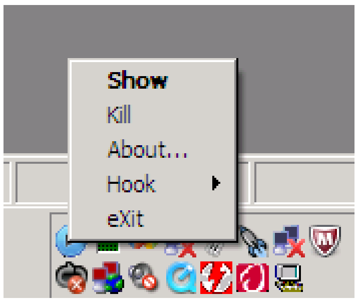
Copyright (C) 2004-2012 SFB7 GHO



```
DxWnd 2.01.99
Hook status: RUNNING
Running "Hexen II (software)"
Screen = (800x600) 0BPP
FullScreen = No
DX version = 0
Logging = OFF
```

DxWnd in the System Tray

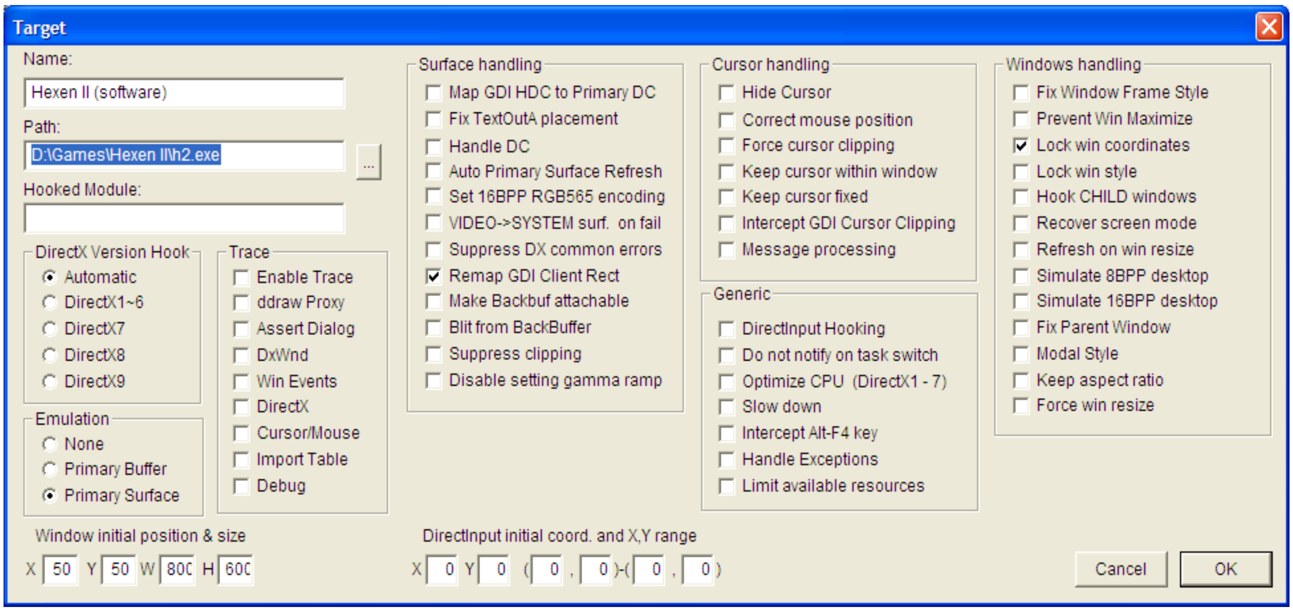
Once you move DxWnd in the system tray and until the program is terminated, an icon will be visible in the system tray. Right-clicking on the icon, you get a subset of the DxWnd commands, plus the Show command that shows the DxWnd window again. The Show command is the menu default, so you can activate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


Target

Name: Hexen II (software)

Path: D:\Games\Hexen II\h2.exe

Hooked Module:

DirectX Version Hook
- [✓ Automatic]
- DirectX1~6
- DirectX7
- DirectX8
- DirectX9

Emulation
- None
- Primary Buffer
- [✓ Primary Surface]

Trace
- Enable Trace
- ddraw Proxy
- Assert Dialog
- DxWnd
- Win Events
- DirectX
- Cursor/Mouse
- Import Table
- Debug

Surface handling
- Map GDI HDC to Primary DC
- Fix TextOutA placement
- Handle DC
- Auto Primary Surface Refresh
- Set 16BPP RGB565 encoding
- VIDEO->SYSTEM surf. on fail
- Suppress DX common errors
- [✓ Remap GDI Client Rect]
- Make Backbuf attachable
- Blit from BackBuffer
- Suppress clipping
- Disable setting gamma ramp

Cursor handling
- Hide Cursor
- Correct mouse position
- Force cursor clipping
- Keep cursor within window
- Keep cursor fixed
- Intercept GDI Cursor Clipping
- Message processing

Generic
- DirectInput Hooking
- Do not notify on task switch
- Optimize CPU (DirectX1 - 7)
- Slow down
- Intercept Alt-F4 key
- Handle Exceptions
- Limit available resources

Windows handling
- Fix Window Frame Style
- Prevent Win Maximize
- [✓ Lock win coordinates]
- Lock win style
- Hook CHILD windows
- Recover screen mode
- Refresh on win resize
- Simulate 8BPP desktop
- Simulate 16BPP desktop
- Fix Parent Window
- Modal Style
- Keep aspect ratio
- Force win resize

Window initial position & size: X 50 Y 50 W 800 H 600

DirectInput initial coord. and X,Y range: X 0 Y 0 (0 , 0)-(0 , 0)

Cancel OK

These are the settings:

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.
Hooked Module	A field in search for a purpose: since the last improvements in the call hooking, this is no longer required for telling extra dll to load, but plans are to use it for custom implementations of non standard libraries, like opengl custom releases.
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 will require an afterthought.
Emulation	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!).
Trace	In case of troubles and for debugging purposes you may ask DxWnd to trace the operations on a logfile, dxwnd.log that is opened in the game activation directory. You should turn these options OFF to play, and ON to send reports for a not working game to me. See notes below about the options meaning.
DirectInput Initial Coord. and relative X, Y range:	DirectInput hook settings about the cursor handling. By

	default, all set to 0.
Surface handling	These are a set of different tunable options referred to the video surface handling that will be explained later on in this manual
Cursor handling	These are a set of different tunable options referred to the the mouse handling that will be explained later on in this manual
Window handling	Some options about how to handle the main window and the child windows (if any)
Generic	These are a set of different tunable options referred to anything but the screen surface or the mouse that will be explained later on in this manual
Cancel button	Exits the form without saving any change (what else?)
OK button	Saves all changes and exits the form (what else?)

Emulation settings (very short description):

None	The program blits directly on top of the actual primary surface, that is simply moved and resized (and clipped) in order to be confined within a window.
Primary Buffer	The former “Pitch Bug Fix” flag, entirely reimplemented with a different technique. The pitch bug is a well known problem existing in old games that didn't consider the Pitch information returtned by the directdraw interface, then working correctly only when the scan lines were contiguous in memory. This is the case when you use either an old video card, or a OFFSCREEN memory surface, hence the trick: instead of blitting directly on primary video surface, the flag makes the program use a virtual primary surface then correctly blitted on video.
Primary Surface	When set, let the program work on fake primary surfaces. This allow to let the program use paletized and low color depth surfaces (e.g. 8bpp or 16bpp) while it uses an actual 32bpp surface as most desktops are currently set. This basically eliminates annoying screen color changes, but as a drawback is much more CPU consuming and often doesn't cope with 3D accelerated games.

Surface handling settings (very short description):

Map GDI HDC to Primary DC	Some games (the older, usually) mix GDI and DirectDraw screen handling. By default, DxWnd simply tries to associate the GDI desktop device contect to the window device context, making the game run in its window, but often loosing stretching capabilities and so on. This option set to ON make DxWnd try to associate the desktop device context to the DirectDraw primary surface device context, of course when one is available yet! Don't worry about this: most games don't even touch GDI, and this option is useless in these cases.
Fix TextOut Placement	Fixes the text placement of those games that uses GDI TextOut call to write text on screen without compensating for window frame offsets
Handle DC	Provides the program with an emulated GDI surface where it can write, and whose graphic is reverted back to the DirectDraw surfaces. It's necessary to see text in Age

	of Empires I & II, but it's a most CPU consuming option. You'd need a powerful CPU.
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.
Use 16BPP RGB565 encoding	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.
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.
Remap GDI Client Rect	Very powerful to manage the desperate cases with crazy mouse behaviour: it makes the program believe that its client rect area (see GetClientRect and GetWinRect APIs) is always at the top left corner of the screen.
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!
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.
Disable setting gamma ramp	Some games set the hardware gamma ramp feature, assuming that no one else has control of the desktop surface. In windowed mode, this will interfere with the luminosity of the whole screen with a possibly annoying effect (Jedi Outcast!). This flag disables the SetGammaRamp API, preserving the luminosity at the price of having the game maybe a little too dark.

Cursor handling settings (very short description):

Hide cursor	Forces hiding 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.
Force cursor clipping	Set hardware cursor clipping within the window's region. It greately 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	experimental
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 PeekMessag / 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”).

Window handling:

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 chenge 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 chenge its own windowstyle.
Hook CHILD Windows	Extends the DxWnd custom message processing to child windows.
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 chaanging 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 suppoed to ever be resized, actually). This flag is to force a refresh (useful for “Uprising”).
Simulate 8BPP desktop	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.
Simulate 16BPP desktop	Same as above, but declaring a 16BPP setting. These two flags should not be set together.

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.
Keep aspect ratio	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).
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.

Generic settings (very short description):

DirentInput hooking	Hooks DirectInput calls. Untested, so far.
Do not notify on task switch	Inhibits the task switch notification message that may hurt some games not designed to handle it properly
Optimize CPU load	Limits the framerate saving CPU time
Slow down	Interoduces some extra delay for old games that are not designed properly. Experimental.
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!
Handle exceptions	Intercepts some exceptions that have a reasonable workaround to fix them: so far, it works greately on some older version of Sonic-R (affected by a divide by zero exception) and Resident Evil (affected by a illegal instruction). In both cases, the offending instruction is eliminated from the assembly and the games work very well. In some cases, though, the fix doesn't prevent the game to show an exception pop-up dialogue.
Limit available resources	Unimplemented so far: this flag is meant to alter the result of system query API that may cause some short integer overflow when too many resources are available and the game interprets it as insufficient ones: for instance, too much space available on disk, too much RAM or video memory and so forth.

Window initial position & size (very short description):

X, Y, W, H	Obviously, these are the X, Y coordinates of the initial client area placement and its Width and Heigth. Because of the window border and titlebar, the window X, Y placement will be moved a little to the left and above, and its size will be a little grater. Whenever the X,Y coordinates are set to zero, a compensation is made to avoid that the window is pushed outside the screen. Most games can be easily moved and stretched on the desktop,

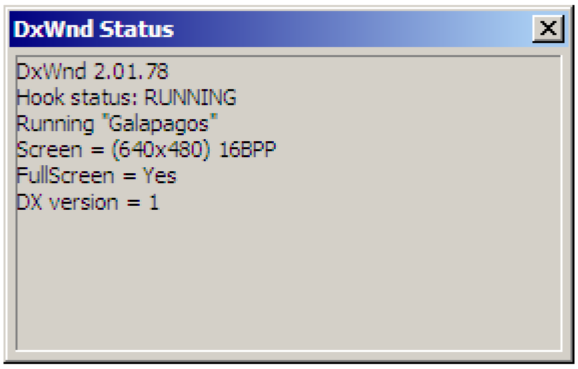
	but a few don't support this, so that setting initial proper coordinates is the only way to have the game running where you like.

Trace options (very short description):

Enable Trace	This works as a global flag that enables/disables all subsequent traces. If unchecked, no output is written. If checked, in output there are the error messages, plus the specific messages related to other flags (see below)
ddraw Proxy	** BEWARE ** Checking this option makes DxWnd hook a full set of DirectDraw APIs and COM methods proxy routines that are just meant to trace everything on disk, in the dxwnd.log file, when the "Operation Log" option is checked. Doing so, the normal DxWnd behaviour is inhibited, the game should run in full screen mode as originally planned, and a operation log for debugging purposes should be available.
Assert Dialog	Enables the generation of a dialogue message box to warn you about severe errors that prevent the correct behaviour of the program and shouldn't go unnoticed.
DxWnd	Enables the operation logging of all significant events that DxWnd performs to bring the fullscreen program in windowed mode.
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.
DirectX	Enables extended logging of all DirectX operations, no matter whether they are related to fullscreen / windowed mode or not.
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.

DxWnd Status

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



```
DxWnd 2.01.78
Hook status: RUNNING
Running "Galapagos"
Screen = (640x480) 16BPP
FullScreen = Yes
DX version = 1
```

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 / Direcr3D interface currently in use (namely, the one used to create the primary surface).

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-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.