WinDriver™ PCI/PCI Express/PCMCIA Quick Start Guide

A 5-Minute introduction to writing device drivers

Who should use WinDriver?

1. Hardware developers – Use Driver Wizard to quickly test your new hardware.
2. Software developers – Use Driver Wizard to generate the device driver code to drive your hardware. Use the WinDriver tools to test and debug your driver.

Which operating systems does WinDriver support?

1. Windows Vista/Server 2003/XP/2000, Windows CE.NET, Windows Embedded CE v6.00, Windows Mobile 5.0/6.0 and Linux.

   Support for Windows NT 4.0, Solaris, and VxWorks is available in earlier versions. Check the Jungo web site for updates on new operating systems support.
2. WinDriver-based drivers are portable between all supported operating systems without any code modifications.

Where can I get more in-depth information?

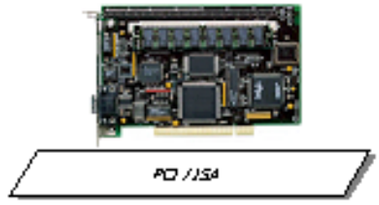
1. You can download a free, full-featured, 30 day evaluation of WinDriver, including documentation, from our web site: http://www.jungo.com/st/download.html.
2. For white papers, user manuals, and other documentation, visit the on-line WinDriver Support Page: http://www.jungo.com/st/support/support_windriver.html.

 Web: http://www.jungo.com.
Mailing Address: Jungo Ltd., 3031 Tisch Way, Suite 808, San Jose, CA 95128 U.S.A.
Tel: 1-877-514-0537 (USA); 972-9-885-9365 (worldwide). Email: sales@jungo.com.

7 steps to building your driver:

1. Set Up

(1) Plug your device to the PC



(2) Install WinDriver

2. Select your device

(1) Start DriverWizard by choosing: **WinDriver | DriverWizard** from Windows **Start** Menu (on Windows), or by running **WinDriver/wizard/wdwizard**.

Note: On Windows Vista you must run DriverWizard as adminstrator.

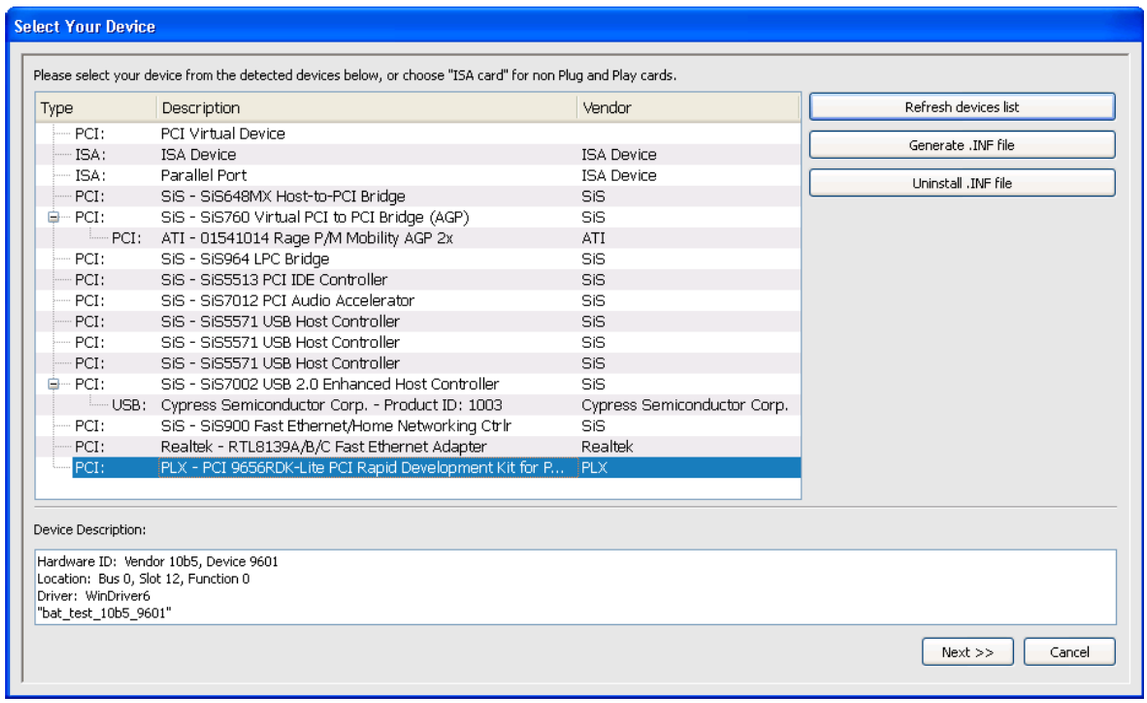
(2) In the dialogue box that appears, choose **New host driver project**.

(3) DriverWizard will show all Plug-and-Play cards plugged in your machine.

(4) For Plug-and-Play devices: select your device from the list of devices.
For non-Plug-and-Play (ISA) devices: select the **ISA card** option to define your device's resources.

 Web: http://www.jungo.com.
Mailing Address: Jungo Ltd., 3031 Tisch Way, Suite 808, San Jose, CA 95128 U.S.A.
Tel: 1-877-514-0537 (USA); 972-9-885-9365 (worldwide). Email: sales@jungo.com.

To generate code for a non-attached PCI device: select the **PCI: PCI Virtual Device** option.

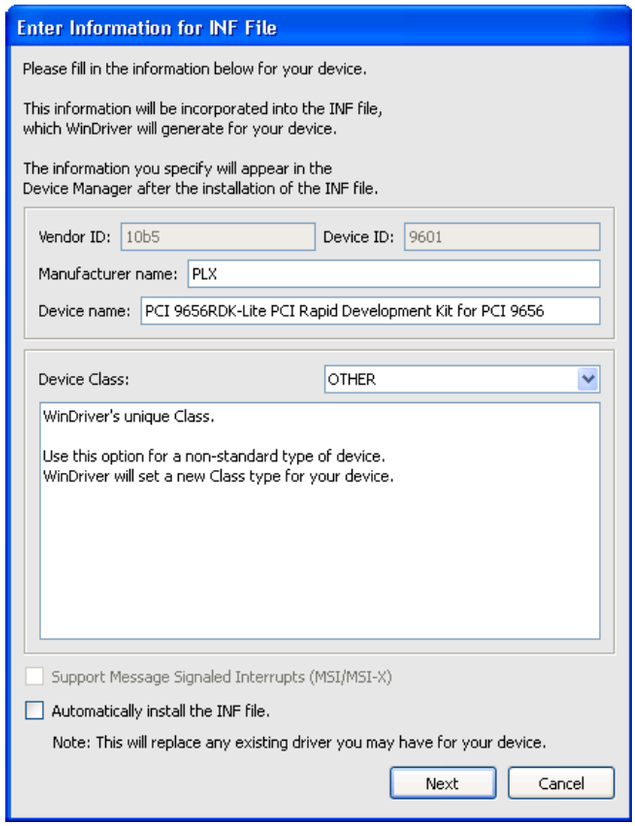


3. Install an INF File for Your Plug-and-Play Device (Windows)

When developing a driver for a Plug-and-Play device (PCI/PCI Express/PCMCIA) on Windows operating systems, in order to correctly detect the device's resources and communicate with the device using WinDriver, you need to install an INF file that registers your device to work with WinDriver.

DriverWizard automates the INF creation and installation process for you.
To generate and install an INF file with DriverWizard, follow these steps:

(1) Click the **Generate .INF file** button in the wizard's **Select Your Device** dialogue. DriverWizard will display information detected for your device – vendor ID, device ID, device class, manufacturer name and device name – and allow you to modify the manufacturer and device names and the device class information.



(2) When using DriverWizard on Windows, you can choose to automatically install the INF file by checking the **Automatically Install the INF file** option in the DriverWizard's INF generation dialogue.
If the automatic INF file installation fails, DriverWizard will notify you and provide manual installation instructions.

(3) Click **Next** in the INF generation dialogue in order to generate the INF file and install it (if selected).

(4) When the INF file installation completes, select and open your device from the list described in step 2 above.

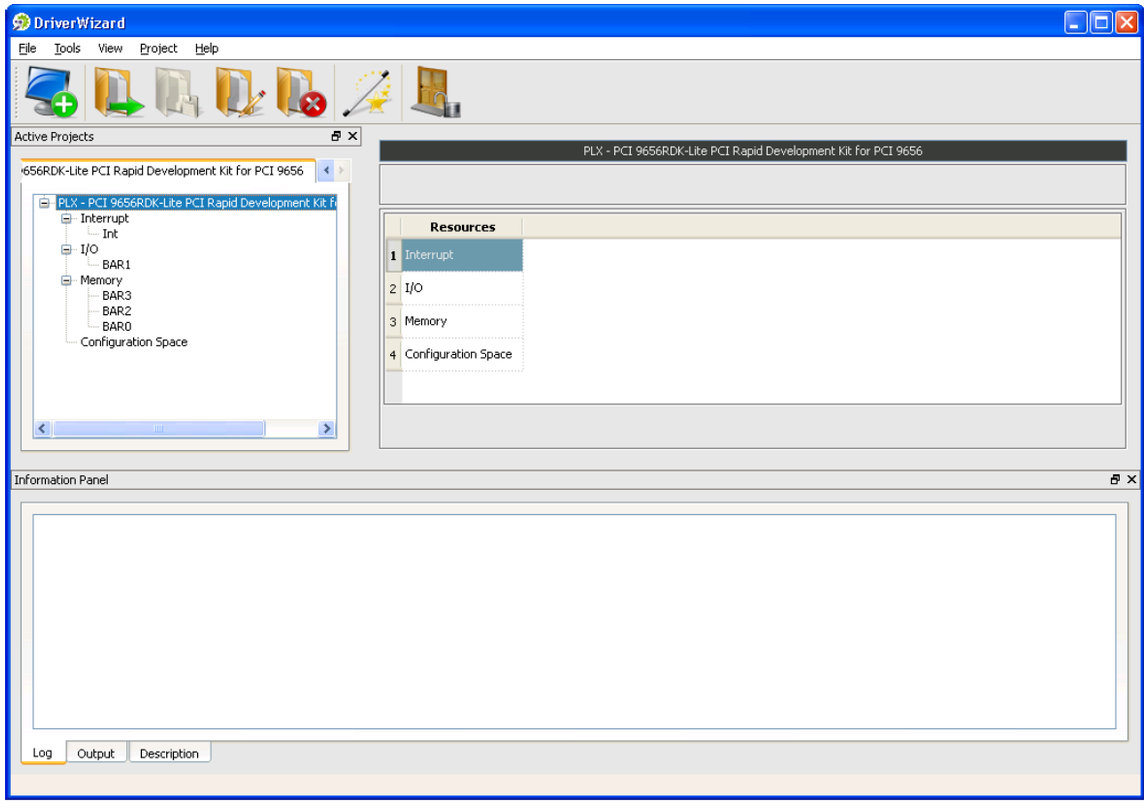
 Web: http://www.jungo.com.
Mailing Address: Jungo Ltd., 3031 Tisch Way, Suite 808, San Jose, CA 95128 U.S.A.
Tel: 1-877-514-0537 (USA); 972-9-885-9365 (worldwide). Email: sales@jungo.com.

** NOTE **
If the **Support Message Signaled Interrupts** option is enabled, you can use it to generate an INF file that supports handling of Message-Signaled Interrupts (MSI) or Extended Message-Signaled Interrupts (MSI-X). This is the default option when selecting to generate an INF file for a virtual PCI device, or when generating an INF file for a PCI device that supports MSI/MSI-X on Windows Vista. For more information regarding MSI/MSI-X and the required INF file configuration, refer to the **WinDriver PCI Manual**.

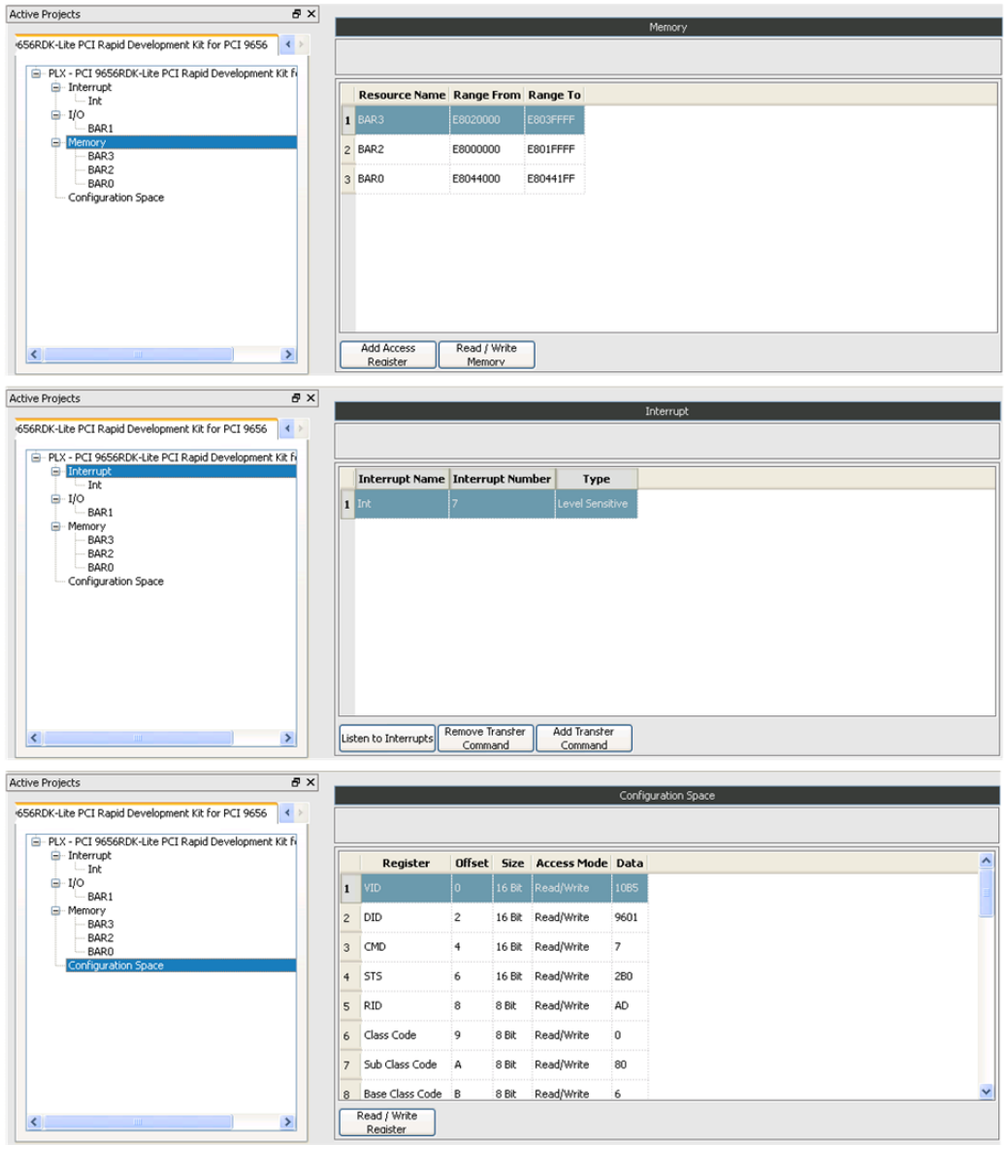
4. Detect / define your hardware's resources

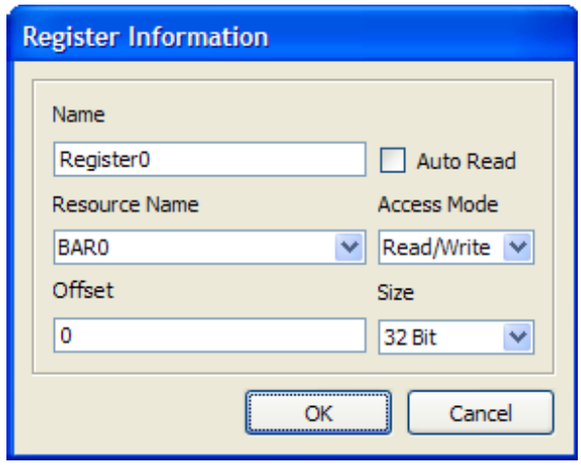
DriverWizard will automatically detect your Plug-and-Play hardware's resources (I/O ranges, Memory ranges, PCI configuration registers and Interrupts). You can define additional information yourself, such as defining registers for your device as well as assigning read/write commands for these registers to the interrupt.

For non-Plug-and-Play hardware (ISA) – define your hardware's resources manually.



 Web: http://www.jungo.com.
Mailing Address: Jungo Ltd., 3031 Tisch Way, Suite 808, San Jose, CA 95128 U.S.A.
Tel: 1-877-514-0537 (USA); 972-9-885-9365 (worldwide). Email: sales@jungo.com.

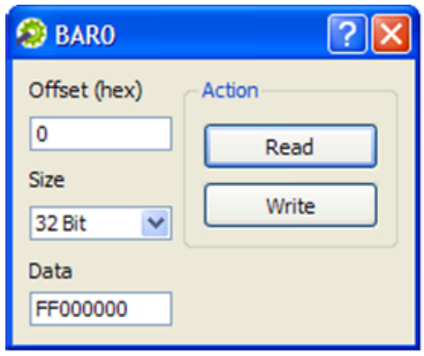


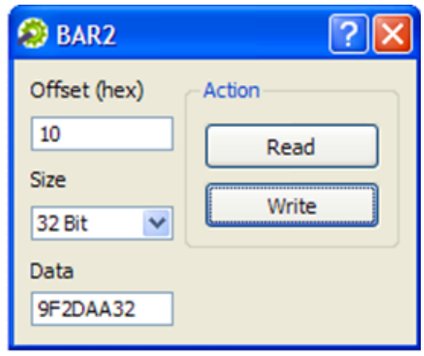


5. Test your hardware

Before writing your device driver, it is important to make sure your hardware is working as expected. Use DriverWizard to diagnose your hardware:

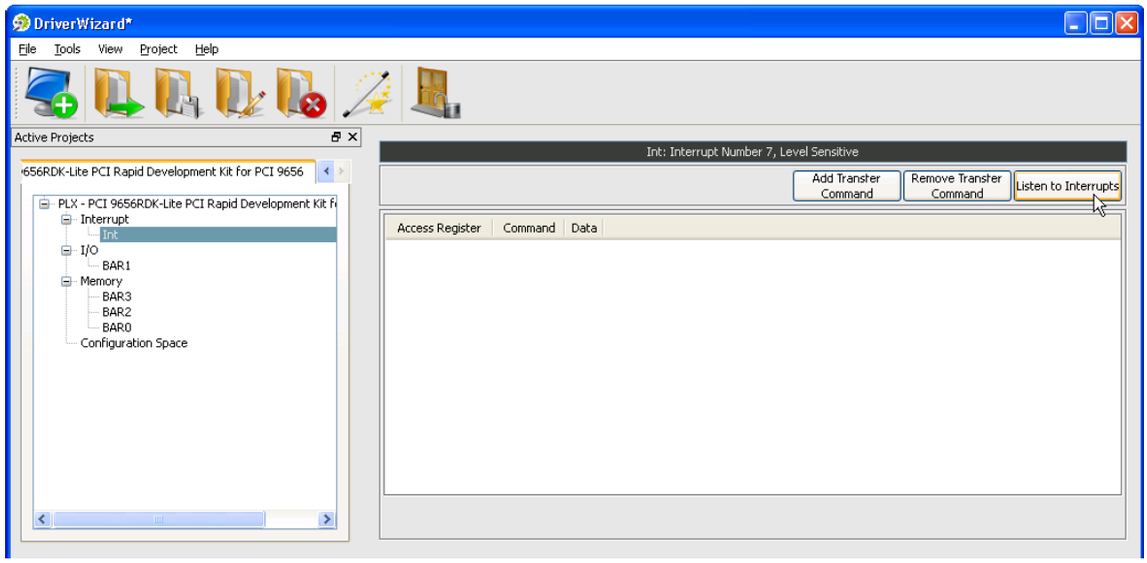
- Read and write to the I/O ports, memory space and your defined registers.





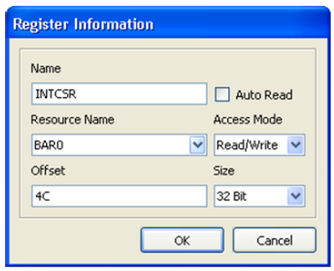
 Web: http://www.jungo.com.
Mailing Address: Jungo Ltd., 3031 Tisch Way, Suite 808, San Jose, CA 95128 U.S.A.
Tel: 1-877-514-0537 (USA); 972-9-885-9365 (worldwide). Email: sales@jungo.com.

- 'Listen' to your hardware's interrupts.

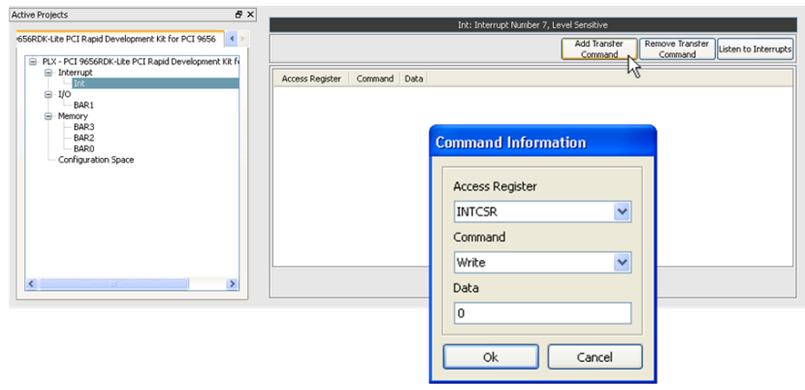


NOTE:

– For level-sensitive interrupts, such as legacy PCI interrupts, you must use DriverWizard to define the interrupt status register and assign the read/write command(s) for acknowledging (clearing) the interrupt, before attempting to listen to the interrupts with the wizard, otherwise the OS may hang! The specific interrupt-acknowledgment information is hardware-specific.



Define the interrupt status register

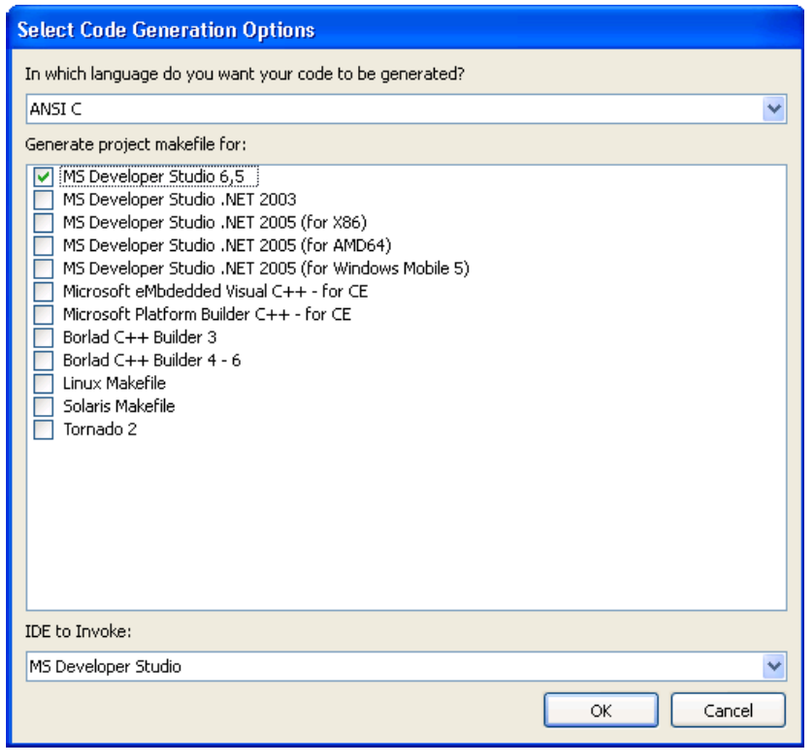


Define the transfer commands for acknowledging (clearing) the level sensitive interrupt

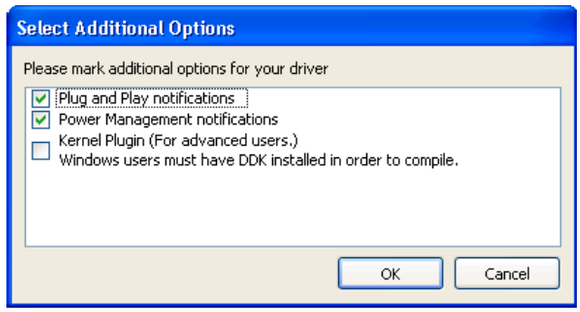
6. Generate the Driver Code:

(1) Select to generate code either via the **Generate Code** toolbar icon or from the **Project | Generate Code** menu.

(2) Choose the code language and indicate which development environments you would like to have project/make files for:



(3) Indicate whether you wish to handle Plug-and-Play and power management events from within your driver code, and whether you wish to generate Kernel PlugIn code. (Note: To build a Kernel PlugIn driver on Windows, you must have an appropriate Microsoft DDK installed):

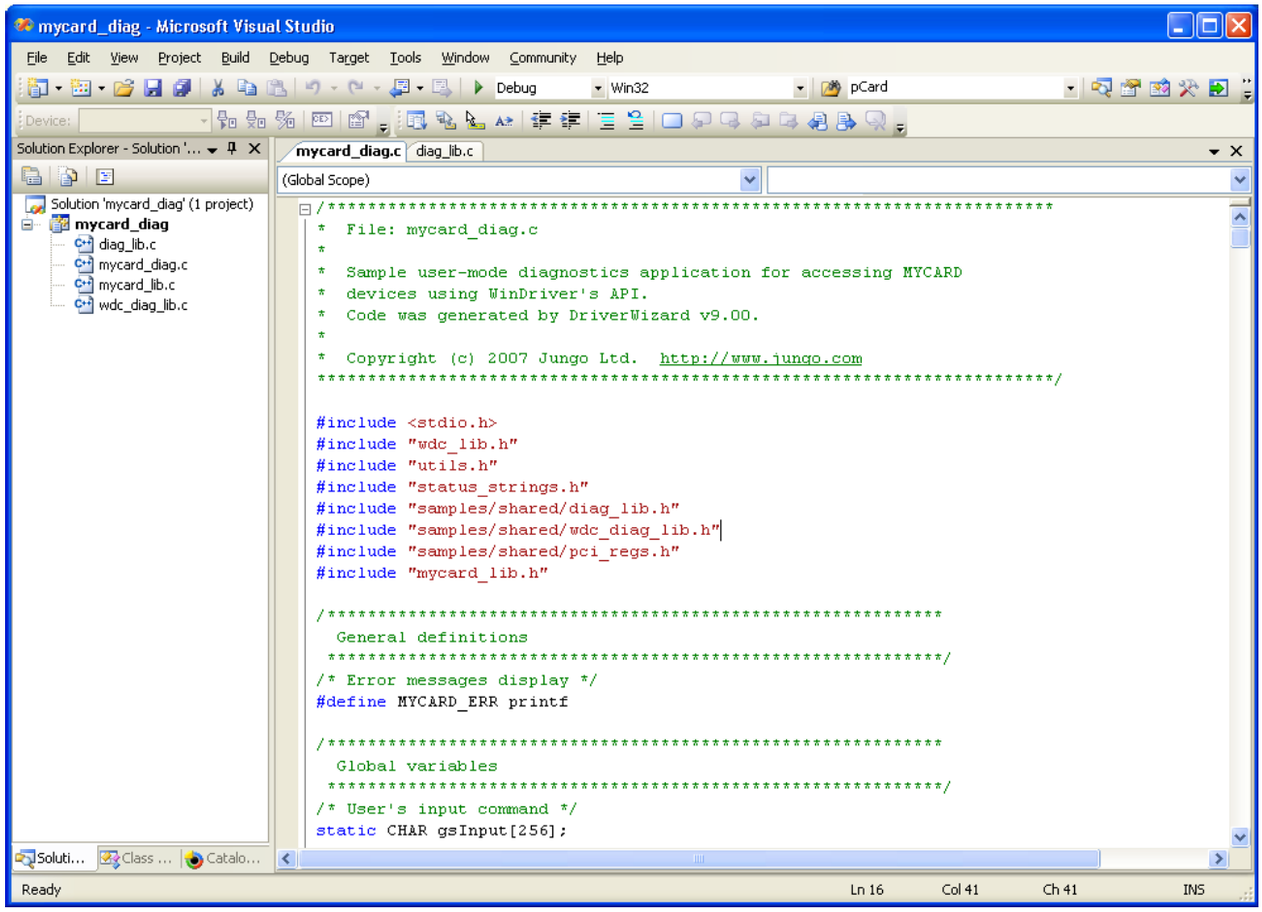


(4) Click **OK**. DriverWizard will launch the desired development environment you chose in step 6.2 above.

7. Compile and run

- The following code is generated:
  - API for accessing your hardware from the application level (and from the kernel).
  - A sample application that uses the above API to access your hardware.
  - Project/make files for all of the selected build environments.
  - An INF file for your device (for Plug-and-Play hardware on Windows Vista/Server 2003/XP/2000).
- Use the project/make file that DriverWizard generated with your selected compiler.
- Compile the sample diagnostics application, and run it! This sample is a robust skeletal code for your final driver.
- Modify the sample application to suit your application needs, or start from one of the many samples provided with WinDriver.

Mailing Address: Jungo Ltd., 3031 Tisch Way, Suite 808, San Jose, CA 95128 U.S.A.
Tel: 1-877-514-0537 (USA); 972-9-885-9365 (worldwide). Email: sales@jungo.com.



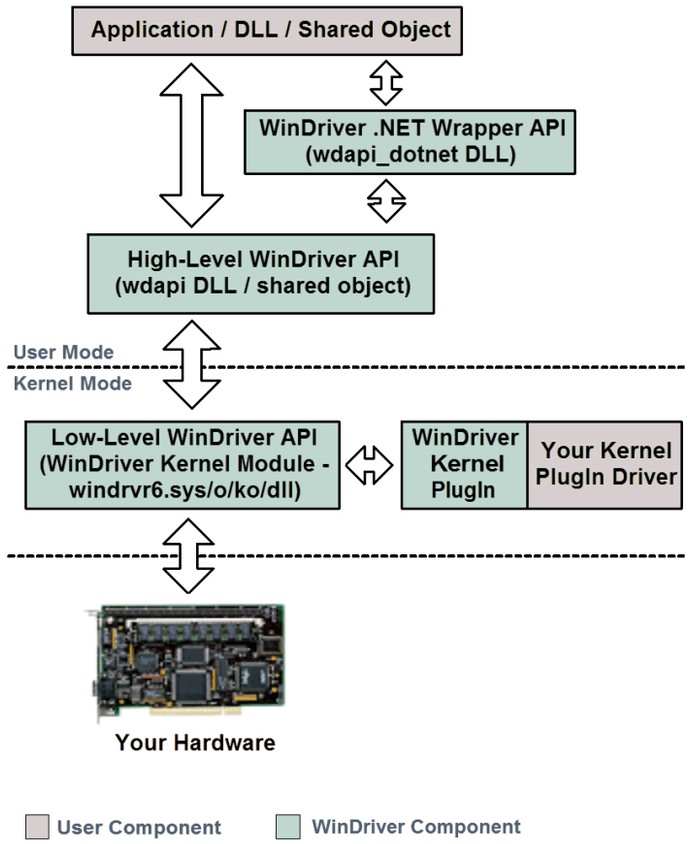
 Web: http://www.jungo.com.
Mailing Address: Jungo Ltd., 3031 Tisch Way, Suite 808, San Jose, CA 95128 U.S.A.
Tel: 1-877-514-0537 (USA); 972-9-885-9365 (worldwide). Email: sales@jungo.com.

Questions & Answers:

Q: How does WinDriver work?

A: With WinDriver, your device driver is developed in the user mode (as part of your application or as a separate DLL). This dramatically shortens development time by enabling you to use your standard development tools (MSDEV/Visual C/C++, MSDEV .NET, Borland C++ Builder, Borland Delphi, Visual Basic 6.0, MS eMbedded Visual C++, MS Platform Builder C++, GCC, etc.) to develop and debug your driver.

The device driver developed with WinDriver (YourApp.exe) accesses your hardware through the WinDriver kernel module (**windrvr6.sys/.dll/.o/.ko**) using the standard WinDriver functions.



 Web: http://www.jungo.com.
Mailing Address: Jungo Ltd., 3031 Tisch Way, Suite 808, San Jose, CA 95128 U.S.A.
Tel: 1-877-514-0537 (USA); 972-9-885-9365 (worldwide). Email: sales@jungo.com.

Q: How can I achieve optimal performance with WinDriver?

A: After your driver is complete, for optimal performance you can easily transfer the performance-critical portions of your driver code (Interrupt handlers, I/O handlers, etc.) to WinDriver's Kernel PlugIn, which runs at the kernel-mode level.

For example – write your interrupt handler code in the user mode. After debugging this code in the user mode, you can move the code into the Kernel PlugIn. This will cause your interrupt handler to be executed in the kernel level, thereby allowing it to operate at maximal performance.

This architecture enables you to develop and debug all of your driver code in the user mode, using WinDriver's API, and then migrate only the performance-critical portions of the code to the kernel mode via the simple Kernel PlugIn mechanism.

Practical Exercises:

The following exercises take you through some of WinDriver's functionality in a quick 5-minute session. You can try these exercises after downloading the WinDriver 30 days evaluation from: http://www.jungo.com/st/download.html.

Exercise #1: Reading and writing to PCI memory

Goal: Learn how to read and write to a PCI memory range, and how to define registers.

Overview: This exercise will demonstrate how you can read and write to your PCI card's memory through the Driver Wizard, and generate an application that does the same. You will do this by reading and writing to your PCI (or AGP) screen card.

Exercise Steps:

Step #1: Start DriverWizard and select to create a **New host driver project**.
When the wizard is already running, you can select to create a new project at any time by clicking the **New Device Driver Project** toolbar icon or selecting this option from the **File** menu.

Step #2: Choose your screen card from the list of Plug-and-Play cards displayed in the dialogue. Locate your screen card by identifying the name of your card's vendor in the displayed list.

Step #3: Click **Memory** in the left window. Your card's memory ranges will be displayed in the right window. One of these memory ranges is mapped to the screen – i.e. bytes in the memory range correspond to pixels on the screen (this is usually Bar 0; look for the largest memory range). Select the relevant BAR in the left window, then press the **Read / Write Memory** button in the write window and read from offset 0 of the selected BAR (the top left of the screen). Now move a window over the top left of the screen to change its color and read from the same offset again. If the value changed – than this is the correct memory range. Now write to this offset (try FFFFFFFF and 00000000 alternately), and see the color of the pixel change!

**** Note:** Writing to the wrong memory range can freeze the computer.

Step #4: Define a register called "TopLeft", which represents the top-left corner of the screen (i.e. the correct memory range and offset 0), and read and write from/to this register. Define another register called "Somewhere", whose offset is FF (i.e. some other pixel on the screen).

Step #5: Proceed to generate code via the **Generate Code** toolbar icon or by selecting the **Project | Generate Code** menu option. Driver Wizard will create the functions to access your hardware's resources. You can call these functions directly from your application in the user mode. DriverWizard will also create a sample application that uses these functions to access your device!

Step #6: Compile and run the sample application. Use it to read and write from your screen card.

That's how simple it is – try it!

**** Note:** You can now copy the sources of the project to any other supported operating system (Windows Vista/Server 2003/XP/2000, Windows CE.NET, Windows Embedded CE v6.00, Windows Mobile 5.0/6.0 and Linux), recompile, and run again!

A part of the API generated by DriverWizard in Exercise #1 (where screencard is the name selected for the driver project when generated the code with DriverWizard):

Mailing Address: Jungo Ltd., 3031 Tisch Way, Suite 808, San Jose, CA 95128 U.S.A.
Tel: 1-877-514-0537 (USA); 972-9-885-9365 (worldwide). Email: sales@jungo.com.

```
<screencard_lib.h>

/* SCREENCARD run-time registers */
/* [Values should correlate to the registers' indexes in the gSCREENCARD_Regs array] */
typedef enum {
    SCREENCARD_TopLeft,   /* TopLeft - This register represents the top left
        pixel on the screen */
    SCREENCARD_Somewhere, /* Somewhere - This register represents a pixel
        somewhere on the screen */
    SCREENCARD_REGS_NUM,  /* Number of run-time registers */
} SCREENCARD_REGS;

DWORD SCREENCARD_LibInit(void);
DWORD SCREENCARD_LibUninit(void);

WDC_DEVICE_HANDLE SCREENCARD_DeviceOpen(const WD_PCI_CARD_INFO *pDeviceInfo);
BOOL SCREENCARD_DeviceClose(WDC_DEVICE_HANDLE hDev);

DWORD SCREENCARD_IntEnable(WDC_DEVICE_HANDLE hDev,
    SCREENCARD_INT_HANDLER funcIntHandler);
DWORD SCREENCARD_IntDisable(WDC_DEVICE_HANDLE hDev);
BOOL SCREENCARD_IntIsEnabled(WDC_DEVICE_HANDLE hDev);
DWORD SCREENCARD_EventRegister(WDC_DEVICE_HANDLE hDev,
    SCREENCARD_EVENT_HANDLER funcEventHandler);
DWORD SCREENCARD_EventUnregister(WDC_DEVICE_HANDLE hDev);
BOOL SCREENCARD_EventIsRegistered(WDC_DEVICE_HANDLE hDev);

DWORD SCREENCARD_GetNumAddrSpaces(WDC_DEVICE_HANDLE hDev);
BOOL SCREENCARD_GetAddrSpaceInfo(WDC_DEVICE_HANDLE hDev,
    SCREENCARD_ADDR_SPACE_INFO *pAddrSpaceInfo);
```

```
<screencard_diag.c>

/* ---------------------------------------------
    SCREENCARD run-time registers information
   --------------------------------------------- */
/* Run-time registers information array */
const WDC_REG gSCREENCARD_Regs[] = {
    { AD_PCI_BAR1, 0x0, WDC_SIZE_8, WDC_READ_WRITE, "TopLeft",
        "This register represents the top left pixel on the" },
    { AD_PCI_BAR1, 0x50, WDC_SIZE_8, WDC_READ_WRITE, "Somewhere",
        "This register represents a pixel somewhere on the " },
    };
const WDC_REG *gpSCREENCARD_Regs = gSCREENCARD_Regs;
```

Exercise #2: Handling Interrupts

Goal: Learn how to test your hardware's interrupts, and write an interrupt handler.

Overview: In this exercise you will use DriverWizard to "Listen" to the interrupts that your floppy disk drive generates. You will then use DriverWizard to generate an application that listens to this interrupt and handles it with a user-mode interrupt handler.

Exercise Steps:

Step #1: Start DriverWizard and select to create a **New host driver project**.
When the wizard is already running, you can select to create a new project at any time by clicking the **New Device Driver Project** toolbar icon or selecting this option from the **File** menu.

Step #2: DriverWizard will display a list of the Plug-and-Play devices connected to your machine. Since we will be using the floppy disk drive, choose ISA from the menu.

Step #3: Click the **Add Resource** button in the right window.

- Define at least one memory range (it can be a dummy range). Select **Memory Resource** from the **Resource Type** combo box, and define a memory range for addresses 0x0-0x0, which will be enough for the purpose of successfully compiling and building the generated code and testing the floppy disk interrupt handling. Press **OK** when you are done.

Step #4: Select **ISA Device** in the left window, and click the **Add Interrupt** button in the right window.

- Choose 6 as the **Interrupt number**, select the **Type** to be **Edge Triggered**, check the **Shared** check box, and press **OK**.

Step #5: Select the Interrupt you just created, and press the **Listen to Interrupts** button.

- To see the floppy drive interrupts: Access the floppy drive (for example, by writing "a:" in a DOS console screen, or by choosing the floppy drive in your file browser).

Step #6: Generate code by pressing the **Generate Code** toolbar icon or by selecting the **Project | Generate Code** menu option. Driver Wizard will create the functions to access your resources and handle the interrupt you have defined. You can call these functions directly from your application in the user mode. DriverWizard will also create a sample application that uses these functions to access YOUR device!

Step #7: Compile and run the sample application.

Step #8: Enable and listen to your interrupt from within the sample application – see the floppy drive interrupts. Later, modify the interrupt handler and insert your own functionality into it.

That's how simple it is – try it!

**** Notes:**

1. Using WinDriver's Kernel PlugIn feature, your interrupts and I/O calls can be handled from kernel mode, thereby achieving optimal performance!

