WinDriver™ PCI/ISA Quick-Start Guide

A 5-Minute Introduction to Writing PCI Device Drivers

Version 12.7.0

Who Should Use WinDriver?

- Hardware developers — Use DriverWizard to quickly test your new hardware.
- Software developers — Use DriverWizard 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?

- Windows 10/8.1/Server 2012 R2/8/Server 2012/7/Server 2008 R2/Server 2008, Embedded Windows 8.1/8/7, and Linux.
  Check the Jungo Web site for updates on new operating systems support.
- WinDriver-based drivers are portable among all supported operating systems without any code modifications.

OS-specific support is provided only for operating systems with official vendor support.

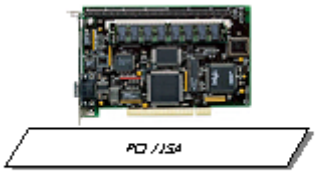
Where Can I Get More In-Depth Information?

- You can download a free, full-featured, 30-day evaluation of WinDriver, including documentation, from our web site: http://www.jungo.com/st/contact-form/?product=WinDriver.
- For user manuals, data sheets, FAQs, and other documentation, visit WinDriver Support page: http://www.jungo.com/st/support/windriver/.

7 Steps to Building Your Driver

1. **Set up**

   a. Connect your device to the PC.



   b. Install WinDriver.

2. **Select your device**

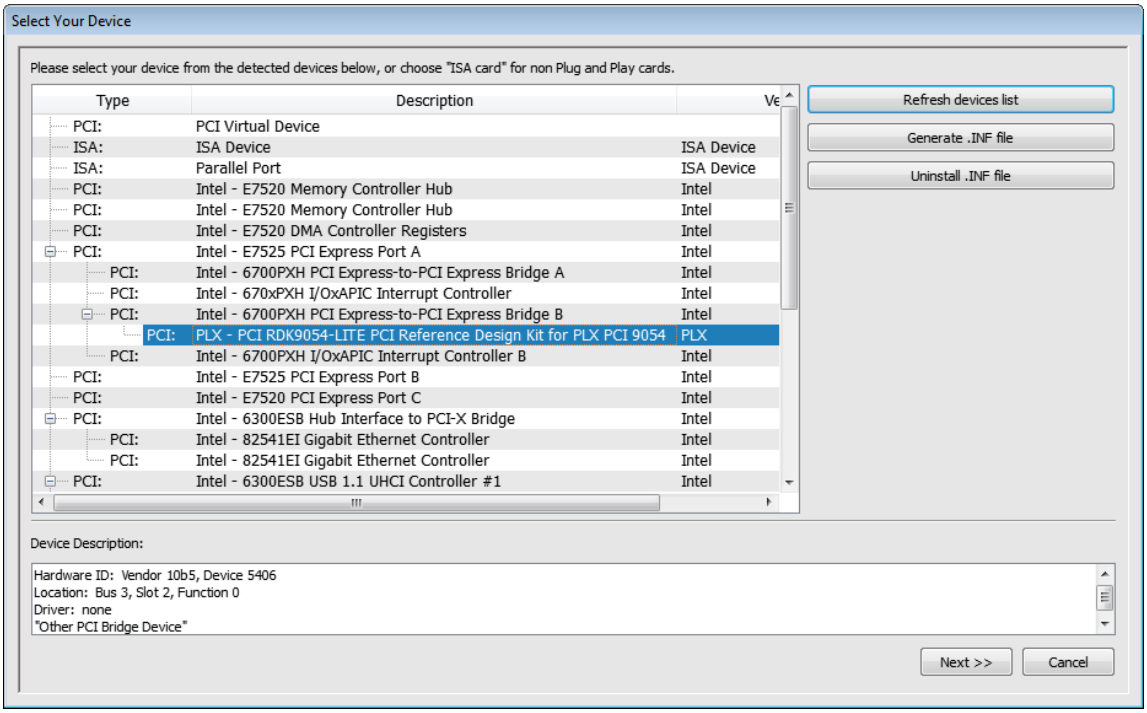
   a. Start DriverWizard — **<path to WinDriver>/wizard/wdwizard**. On Windows you can also run DriverWizard from the **Start** menu: **Start | Programs | WinDriver | DriverWizard**.

> On Windows 7 and higher you must run DriverWizard as administrator.

   b. In the dialogue box that appears, choose **New host driver project**.

   c. DriverWizard will show all Plug-and-Play cards plugged in your machine.

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

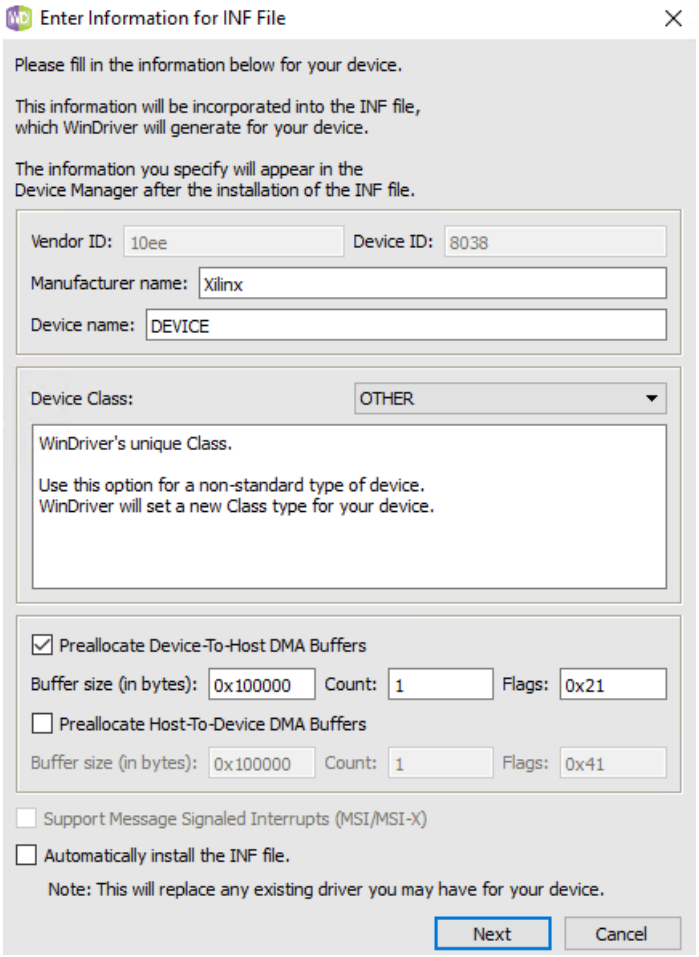


3. Install an INF File for Your Plug-and-Play Device (Windows)

When developing a driver for a Plug-and-Play device (PCI) 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:

a. 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.



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

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

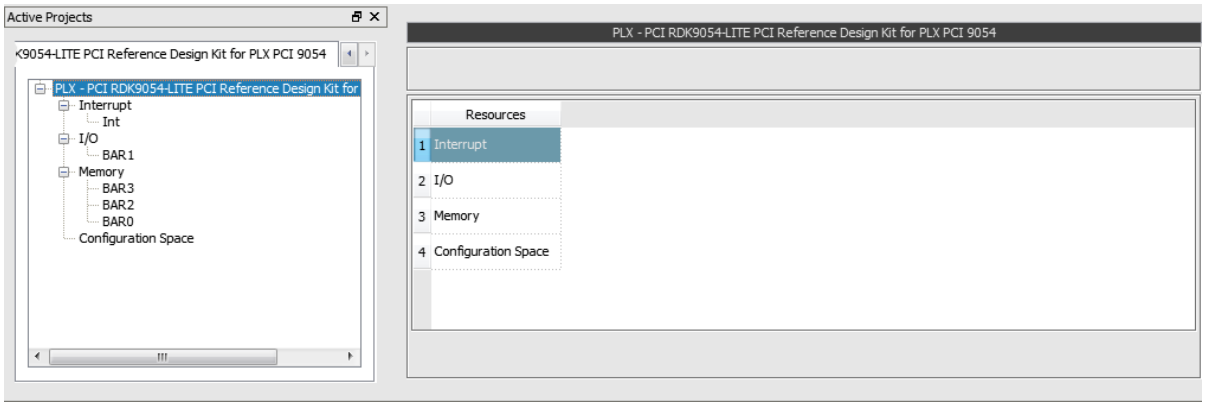
d. When the INF file installation completes, select and open your device from the list described in Step 2 above.

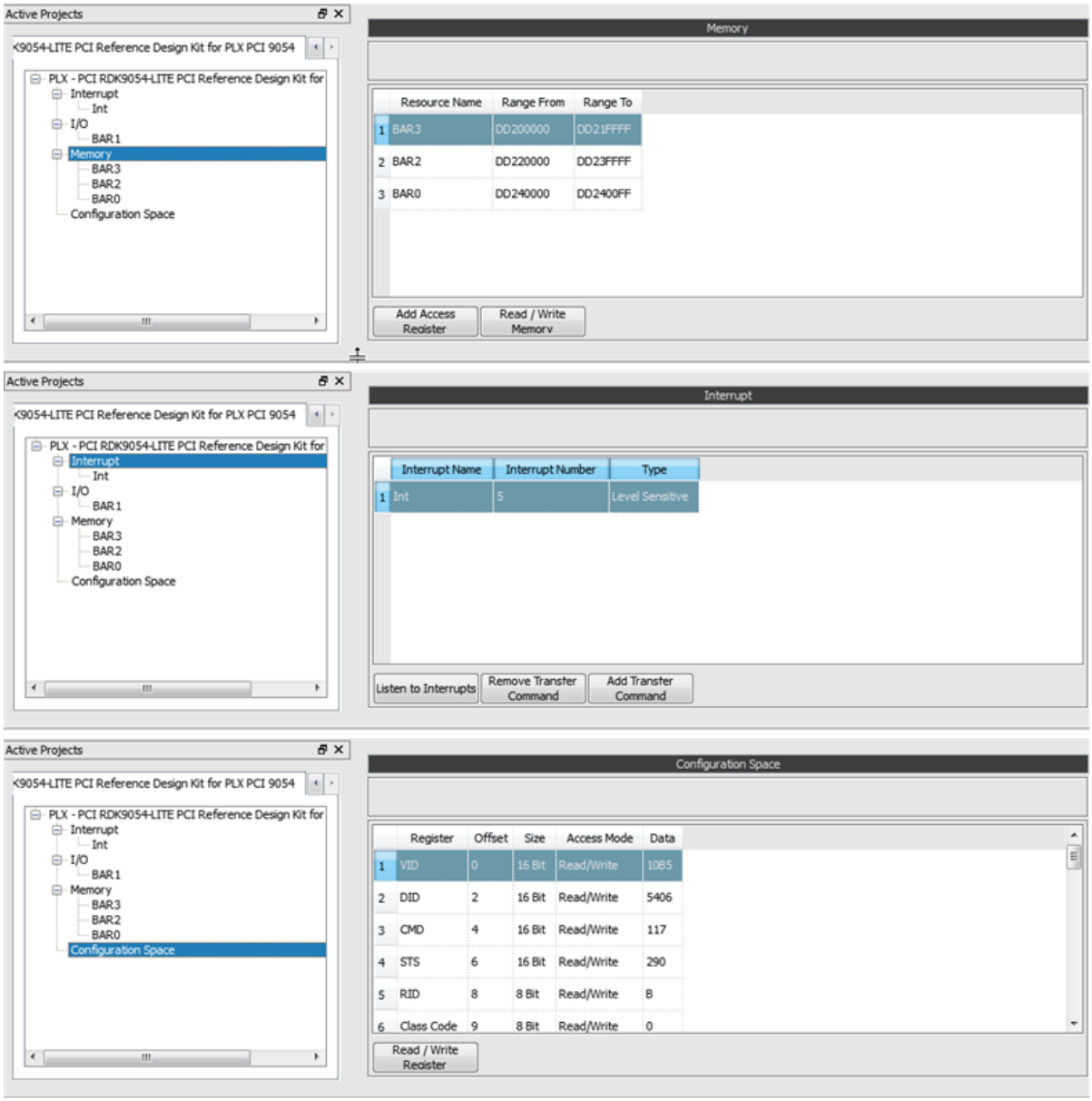
If the **Support Message Signaled Interrupts** option is enabled, you can use it to generate an INF file that supports handling of Message-Signaled Interrupts (MSI) or Extended Message-Signaled Interrupts (MSI-X). This is the default option when selecting to generate an INF file for a virtual PCI device, or when generating an INF file for a PCI device that supports MSI/MSI-X on Windows 7 and higher. For more information regarding MSI/MSI-X and the required INF file configuration, refer to the **WinDriver PCI Manual**.

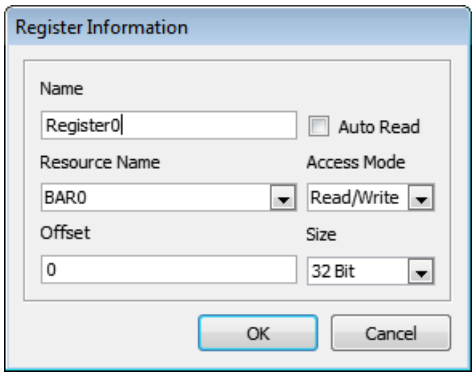
4. Detect/define your hardware's resources

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.





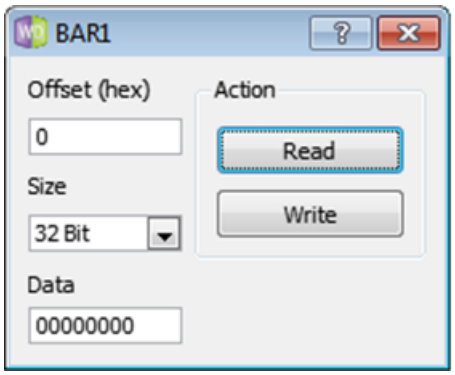


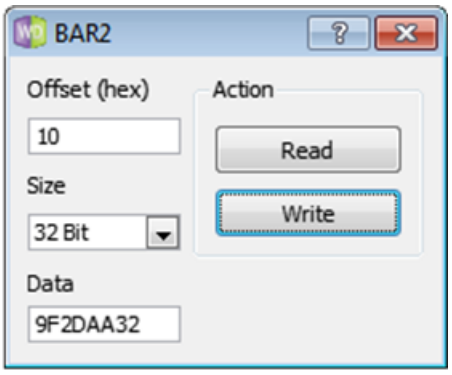
On Windows 7 and higher, you may need to register an IRQ with WinDriver before you can assign it to your non-Plug-and-Play hardware, as outlined in the WinDriver User's Manual.

5. Test your hardware

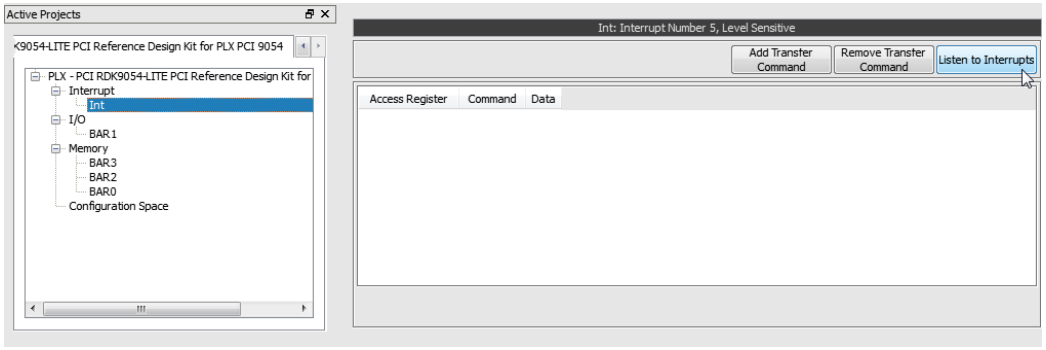
Before writing your device driver, it is important to make sure your hardware is working as expected. Use DriverWizard to diagnose your hardware:

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

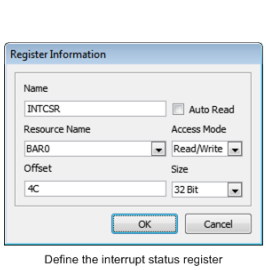




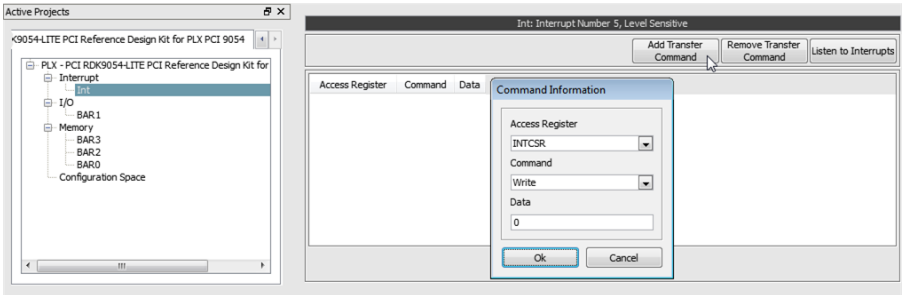
- "Listen" to your hardware's interrupts.



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

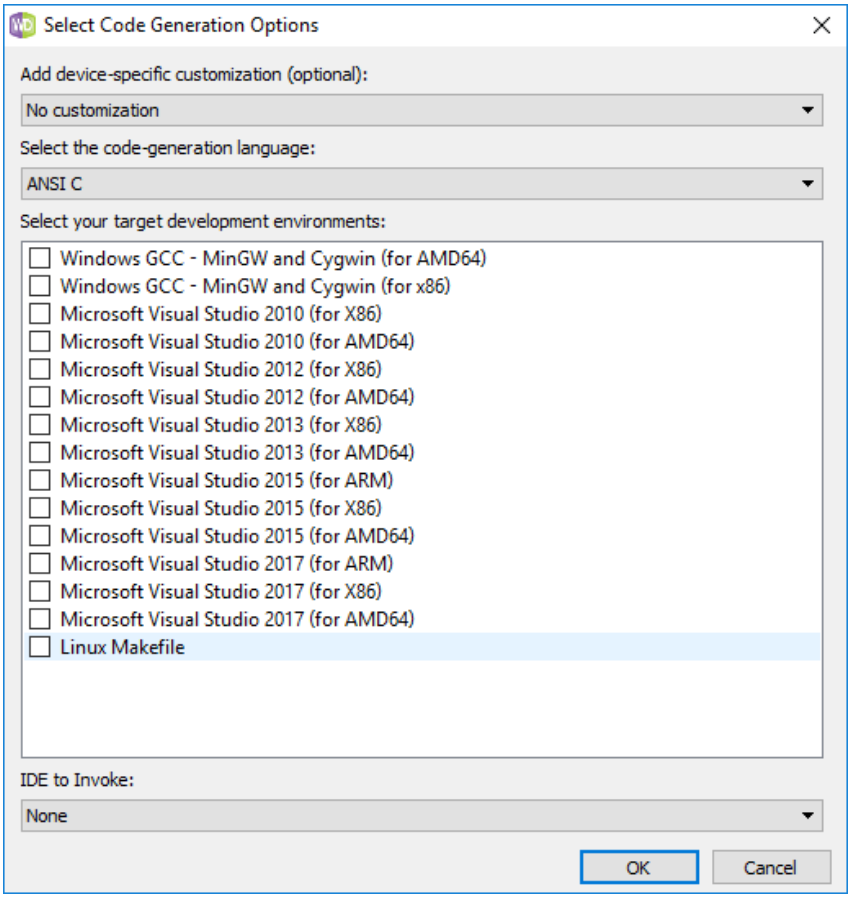


Define the transfer command(s) for acknowledging (clearing) the level-sensitive interrupt

6. Generate the driver code

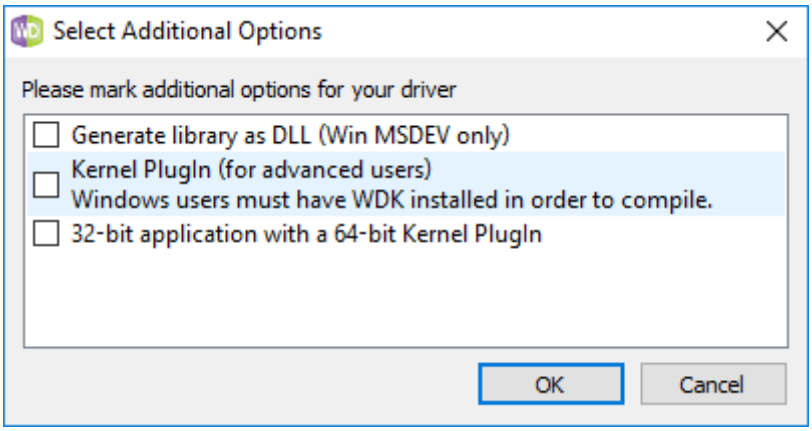
a. Select to generate code either via the **Generate Code** toolbar icon or from the **Project | Generate Code** menu option.

b. Optionally select to generate additional customized code for one of the supported devices, and choose the code language and target development environment(s) for the generated code:

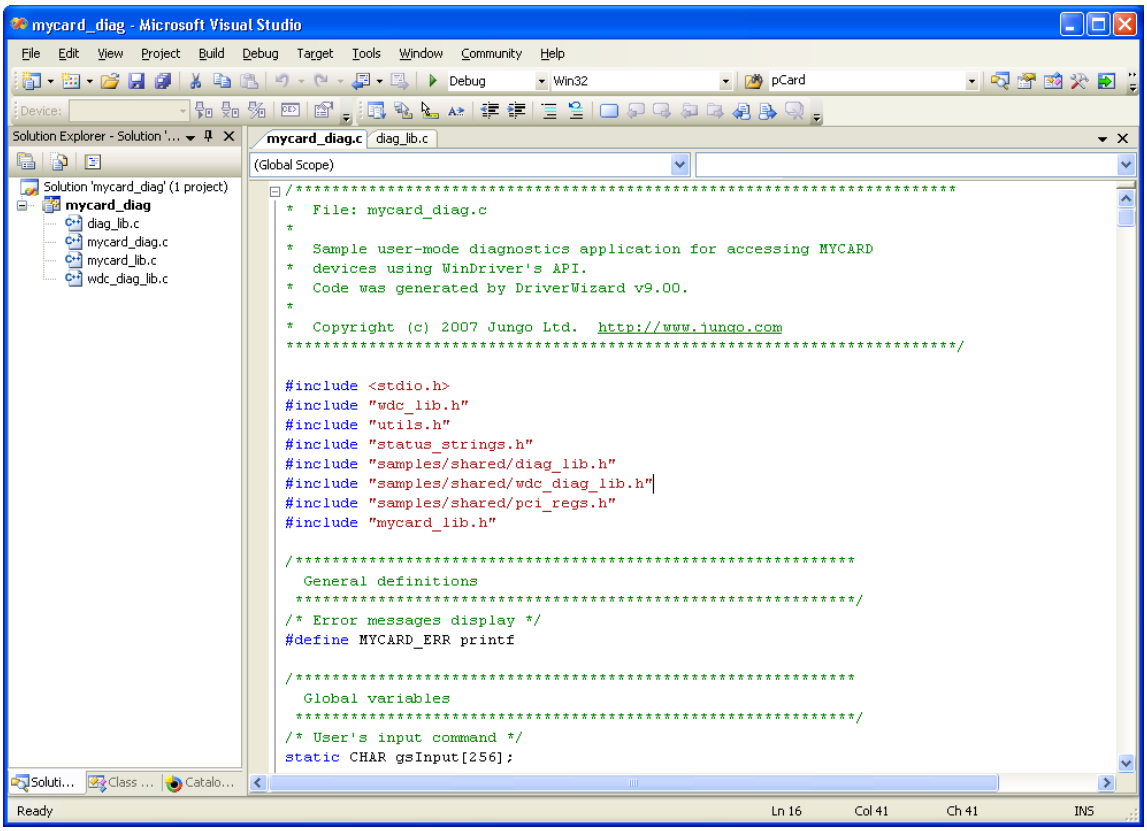


c. Select whether to handle Plug-and-Play and power management events from within your driver code, whether to generate Kernel PlugIn code (and what type of related application to create), and whether to build your project's library as a DLL (for MS Visual Studio Windows projects).

To build a Kernel PlugIn driver on Windows, the Windows Driver Kit (WDK) must be installed.



d. Click **OK**. DriverWizard will display a list of the generated files, and launch the development environment you selected to invoke in Step b above (if any).



DriverWizard generates the following:

- 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).

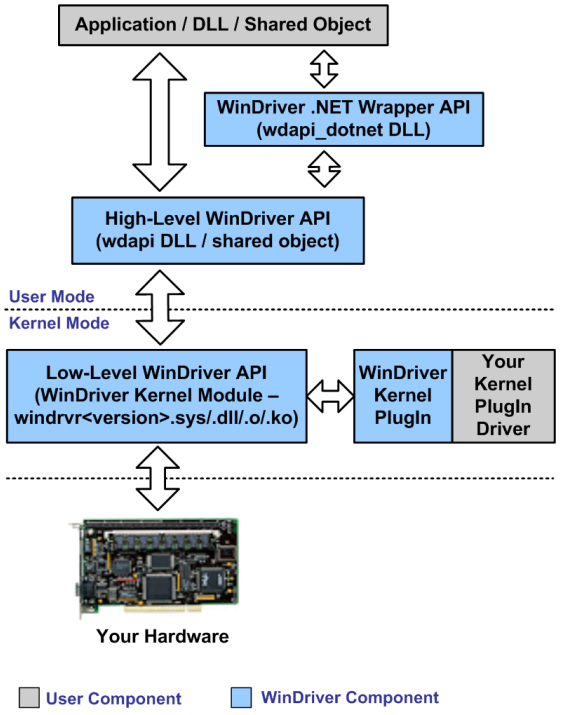
7. **Compile and run**

- 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.

Frequently Asked Questions

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 (MS Visual Studio, C++, GCC, Windows GCC, etc.) to develop and debug your driver.
The user-mode device-driver application/DLL that you develop with WinDriver accesses your hardware through the WinDriver kernel module (**windrvr1270.sys/.dll/.o/.ko** — depending on the OS) using the standard WinDriver functions.



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/contact-form/?product=WinDriver.

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 from/to your PCI card's memory using DriverWizard, 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:

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.

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.

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 gcWrite 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!


Writing to the wrong memory range can freeze the computer.

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).

5. Proceed to generate code via the **Generate Code** toolbar icon or by selecting the **Project | Generate Code** menu option. DriverWizard 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!

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!


You can now copy the sources of the project to any other supported operating system, 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):

```
<screencard_lib.h>

/* SCREENCARD run-time registers */
/* [The values should correlate to the register indexes in gSCREENCARD_Regs.] */
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:

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.

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.

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.

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**.

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).

6. Generate code by pressing the **Generate Code** toolbar icon or by selecting the **Project | Generate Code** menu option. DriverWizard 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!

7. Compile and run the sample application.

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!

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