Go-HotSwap v7.02 User's Guide

Jungo Ltd

27th November 2005

COPYRIGHT

Copyright **©1997 - 2004 Jungo Ltd. All Rights Reserved**

Information in this document is subject to change without notice. The software described in this document is furnished under a license agreement. The software may be used, copied or distributed only in accordance with that agreement. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or any means, electronically or mechanically, including photocopying and recording for any purpose without the written permission of Jungo Ltd.

Windows, Win32, Windows 98, Windows Me, Windows CE, Windows NT, Windows 2000, Windows XP and Windows Server 2003 are trademarks of Microsoft Corp. WinDriver and KernelDriver are trademarks of Jungo. Other brand and product names are trademarks or registered trademarks of their respective holders.

Contents

List of Figures

Chapter 1

General Overview

1.1 Introduction

Jungo Go-HotSwap is a software infrastructure that adds the necessary software modules required to enable CompactPCI hot swapping, and provides the tools and development environment to develop hot swap aware drivers.

Go-HotSwap complies with PICMG 2.1 R2.0 requirements for CompactPCI General Use Full Hot Swap Software.

Go-HotSwap supports Linux, Solaris and Windows 2000/XP/Server 2003, utilizing the native Plug-and-Play capabilities of these operating systems.

Go-HotSwap software infrastructure consists of a low-level operating system extension, the Hot Swap Engine, that is responsible for the following activities:

- Identifying the insertion/removal of CompactPCI boards.
- Notifying the system services and the relevant drivers of a hot swap event.
- Notifying the relevant applications of a hot swap event.

Go-HotSwap includes Jungo's WinDriver, a driver development tool, as an integrated component. The driver development toolkit dramatically simplifies the very difficult task of developing a device driver from scratch. It provides a complete solution for creating high performance drivers, which handle interrupts and I/O at optimal rates.

It is not obligatory to use the included driver development toolkit in order to generate your driver's source code. You can create your driver from scratch, and simply add the Go-HotSwap API to your driver's source code to make it hot swap aware and to utilize the Go-HotSwap services. However, by using the included tools your development time is considerably reduced.

Go-HotSwap also includes the Configuration Manager, a graphical tool, which enables the Go-HotSwap engine to respond to hot swap events. The user can configure the Configuration Manager in order to execute tasks upon hot swap events such as running a batch file or starting/stopping a service, thus achieving hot swap capabilities without modifying the driver's source code. In addition, Go-HotSwap includes a sample in console-mode that has the same functionality as the Configuration Manager. The sample, **hs_activate**, also enables the Go-HotSwap engine to respond to hot swap events.

Go-HotSwap is an ideal solution for hardware vendors, as well as for system integrators and operating system vendors, aiming to provide hot swap capabilities to the end user.

We recommend you periodically visit Jungo's website at `http://www.jungo.com` for the latest information on Go-HotSwap and other development tools that Jungo offers.

1.2 Background

In order to take advantage of the hot swap capability of the CompactPCI hardware, a substantial amount of additional software is required at the operating system level.

The software aspect of the hot swap process involves:

- Hot swap event detection
- Dynamic allocation and de-allocation of system resources
- Configuration of newly inserted boards
- Notification to relevant subscribers

Normally, the system BIOS or the operating system assigns a memory address space, an I/O address space and IRQ to each of the CompactPCI devices when the system is powered up.

But when a CompactPCI board is inserted/removed when the system is already running, the system is required to detect the event and to dynamically reconfigure the memory, I/O address space and IRQ.
This is the role of the kernel hot swap modules. Basically, the kernel hot swap modules are required to detect the event of a CompactPCI card being inserted/removed, and to reconfigure the system resources accordingly.
The kernel hot swap modules are also required to inform the system of the event so that the system and applications running are not damaged, and the event of insertion/removal is transparent to the system user.

While many hardware products are already electrically and mechanically hot swap compatible, not all mass-market operating systems fully support hot swapping. Windows 2000/XP/Server 2003 operating systems, for example, do not recognize insertion/removal of CompactPCI boards.
The absence full hot swap support in mass-market operating systems is forcing companies to give up utilization of hot swap capabilities, purchase proprietary solutions or develop in-house solutions that comply with the CompactPCI standard. Any chosen path the companies choose to take, is undesirable due to the considerable resource consumption.

Developing an in-house hot swap software solution is not an easy task. It requires developing low-level services such as a hot swap system driver and a hot swap service[1].
These components involve hardware abstraction, hardware access and dynamic resource allocation. The development process involves learning the internals of the operating system, learning new tools for development/debugging in the kernel mode (such as DDK and ETK), writing the kernel-mode modules that do the basic hardware access and dynamic resource allocation (hot swap system driver and service), and repetition of the above steps for each operating system on which the code should run.

Go-HotSwap provides a complete off-the-shelf solution thus reducing the time-to-market factor and enabling users to focus on core technologies instead of going through all the effort involved in developing in-house hot swap software.

[1] See CompactPCI Hot Swap Specification PICMG 2.1 R2.

1.3 Feature List

Go-HotSwap architecture features the following advantages:

1.3.1 Multi Operating System Support

Go-HotSwap supports Solaris, Linux and Windows 2000/XP/Server 2003.

Benefit: Go-HotSwap provides maximum flexibility; the user is not restricted to only one operating system.

Example: By using Go-HotSwap hardware vendors can have software support for a multitude of operating systems, thus extending the market share of their product and complying with more customer requirements.

1.3.2 Driver Architecture

Go-HotSwap is not restricted to drivers that are hot swap compliant, such as drivers that fully support the Microsoft Windows 2000 PnP subsystem additions, and does not require the driver to be re-written in order to benefit from hot swap capabilities.

Benefit: Jungo's solution enables legacy drivers, as well as hot swap compliant drivers, to utilize the hot swap mechanism. This provides for comprehensive hot swap support, and enables quicker deployment of the hot swap solution.

Example: Go-HotSwap enables hardware vendors to use a driver written for a standard PCI device with CompactPCI hardware without having to modify the driver.

1.3.3 Driver Development Tools

A complete and robust hot swap solution should include the necessary tools to build hot swap device drivers from scratch. Jungo provides a complete solution for customers, which not only includes the additional software layers that are needed to support hot swap, but also includes WinDriver - a device driver development toolkit, which dramatically simplifies and automates the driver development process.

Benefit: Go-HotSwap includes the WinDriver development toolkit, which features DriverWizard, a graphical wizard that guides the user through the complicated task of creating a device driver.

NOTE:
WinDriver is integrated into the Go-HotSwap package, therefore, when you read the WinDriver documentation, all references to **WinDriver** should be read as references to **Go-HotSwap**. For example, WinDriver User's Guide refers to the PCI_SCAN utility, found at **\windriver\util\pci_scan.exe**.
A Go-HotSwap user will find the same utility at **\GoHotSwap\util\pci_scan.exe**, instead.

1.3.4 Cross Operating System Capability

The developed device drivers generated with the Jungo's driver development tools are binary compatible between Windows 2000, XP and Server 2003 and are source code compatible between all the above operating systems.

Benefit: major TTM (Time-to-Market) reducer. There is no need to repeat the development process over and over again, when porting to other operating systems. Just compile and run.

1.3.5 Hardware Compatibility

Go-HotSwap can be used to develop hot swap aware drivers for any CompactPCI device. Go-HotSwap supports any CompactPCI silicon/board/platform, thereby enabling you to use a combination of any vendors components in the system.

1.3.6 Flexible Driver Models

A complete hot swap solution should enable developers to create either user-mode or kernel-mode device drivers. Go-HotSwap supports both driver models, i.e. user-mode and kernel-mode.

Benefits: This enables hot swap aware user-mode drivers to be developed where possible, reducing the learning curve for the driver developer.

1.3.7 Stability

A robust hot swap solution should be stable and field-tested. Go-HotSwap is based on Jungo hardware access technology that has been field tested on thousands of platforms and hardware combinations by Jungo WinDriver and KernelDriver product lines.

1.3.8 Technical Support

Jungo offers excellent technical support directly by Jungo's R&D team - the same experts that wrote the software. During the evaluation period, Jungo provides 30 days of free technical support. Upon purchase, additional two months of free technical support are provided. Additional support programs can be purchased separately.

1.4 Evaluate Go-HotSwap Before Purchasing

Evaluation versions of Go-HotSwap for all supported operating systems are available at the Jungo website at `http://www.jungo.com`.

Jungo provides a fully featured evaluation version with which you can actually develop your hot swap capable driver, and purchase the software when satisfied with its functionality.

All evaluation versions of Go-HotSwap are fully featured. Their functionality is not lacking in any way. The following are the limitations of the evaluation versions:

- A message appears on each first use of Go-HotSwap saying that this version of Go-HotSwap is for evaluation purposes only and cannot be distributed.
- A dialog box, with a message stating that an evaluation version is being run, pops upon every interaction with the hardware.
- The Windows evaluation version expires 30 days from the date of installation.
- In the Linux and Solaris versions, the Go-HotSwap Hot Swap Engine (windrvr6) stops working after 60 minutes and has to be reloaded.

NOTE:
When the Go-HotSwap kernel stops working, you may not receive a warning message notifying you of this and you may mistakenly think that the kernel engine is malfunctioning. If your Go-HotSwap kernel stops responding, just unload it and then reload it again. It will then work for another 60 minutes.

Chapter 2

Architectural Overview

2.1 General

Go-HotSwap software is a combination of four components:

- **Hot Swap Engine** - This component serves as a generic hot swap aware driver that resides in the kernel. The Hot Swap Engine communicates directly with the hardware, as well as with the hardware specific driver (directly, or through the Configuration Manager). Its central task is to detect hot swap events, identify the hot swapped board's required resources and (optional) dynamically allocate them or de-allocate them upon removal of the board.
- **Kernel-Mode Messaging Mechanism** - This component is responsible for establishing and maintaining an updated log of the hot swap events according to the data received from the Hot Swap Engine. It then transfers the relevant messages to the subscribed drivers or applications.
- **Driver Development Toolkit** - This component, if you wish to use it, enables quick and simple development of your driver's hardware access code. The driver development toolkit enables the user to easily produce a user application aware of hot swap events.

> **NOTE:**
> It is not a requirement to use Jungo's driver development tools in order to develop your driver hardware access code; any third party driver development kit can be used.

- **Configuration Manager/hs_activate** - This component is an user-mode application which activates and stops the Hot Swap Engine and enables the Hot Swap Engine to respond to hot swap events. The **hs_activate** is a console-mode application suitable for all supported operating systems, while the Configuration Manager is a graphical implementation of **hs_activate**, suitable for Windows only. The user can configure the Configuration Manager/**hs_activate** in order to execute tasks upon hot swap events such as running a batch file or starting/stopping a service, thus achieving hot swap capabilities without modifying the driver's source code.

The following diagram illustrates two possible methods of working with Go-HotSwap:

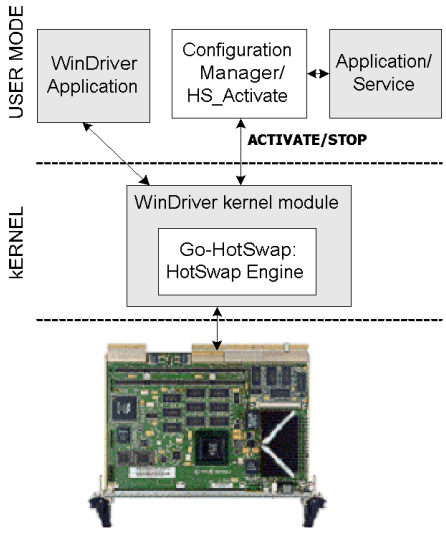


Figure 2.1: Go-HotSwap Architecture

In figure 2.1, the Configuration Manager/**hs_activate** is responsible for activating the Hot Swap Engine.
The Hot Swap Engine is then responsible for detecting hot swap events and (optional) allocating the boards' required resources. The Hot Swap Engine notifies either the WinDriver application or the Configuration Manager/**hs_activate** about hot swap events.

The Configuration Manager/**hs_activate** is also responsible for stopping the Hot Swap Engine.
You can use WinDriver to produce the user application, which includes automatic hot swap support. You have the option to use the WinDriver wizard to generate your driver's hardware access code. During this process, you are asked to select your driver's options, one of which is Plug and Play notifications, another is Power Management notifications. Select these options in order to use WinDriver's application with the Hot Swap Engine.
The alternative option is to use a different application/service via the Configuration Manager/**hs_activate**, which notifies/starts/stops the drivers or applications according to the way the user configured the Configuration Manager/**hs_activate**.

2.2 The Hot Swap Engine

The Hot Swap Engine acts as a generic Hot-Swap/Hot-Plug System Driver (as described in the Hot Swap Specification PICMG 2.1 R2.0). Its main task is to detect hot swap events and identify the hot swapped board's required resources. On Windows platforms you have the option to use the Hot Swap Engine to dynamically allocate the hot swapped board's resources (or de-allocate them upon the board removal).

2.2.1 Detecting Hot Swap Events

In order to detect live insertion/removal of CompactPCI devices from the bus, the Hot Swap Engine can use one of the following methods: polling the CompactPCI bus periodically, or polling the ENUM# signal[1].

Polling the CompactPCI Bus

When the polling mechanism is activated by `WD_WatchPciStart()`, the Hot Swap Engine starts polling the bus periodically. This is Go-HotSwap's default mechanism for detection of hot swap events, because it allows compliance with all CompactPCI system configurations and board types (including basic and full hot swap boards).
The Hot Swap Specification PICMG 2.1 R1.0 allows the ENUM# signal to either drive an interrupt or be polled by the system software at regular intervals. The implementation and support of the ENUM# signal requires platform specific support, making the bus polling mechanism a better choice as the default mechanism for detection of hot swap events. Moreover, this architecture allows Go-HotSwap to enable hot swap with boards that do not implement the ENUM# signal at all.

After initiation, the Hot Swap Engine scans the CompactPCI bus to determine which devices are actually present. To do this, the Hot Swap Engine accesses the PCI Configuration Space and reads the *Vendor ID* register in each one of the possible 32 CompactPCI slots on each of the CompactPCI buses. The bus reports an attempt to read the Vendor ID of non-existent devices by returning a value of all 1's.
When a board is found, the Hot Swap Engine retrieves additional information, such as:

[1] A signal provided by CompactPCI full hot swap boards and by hot swap friendly/ready CompactPCI silicon, in order to notify the system host that a board was either freshly inserted or is about to be extracted.

1. The board's device ID.
2. Whether the board is hot swap compatible or not (basic hot swap or full hot swap). If it is, the Hot Swap Engine continues to check:
   (a) Whether the status of the Handle Switch (activated by the lower ejector handle) is closed or open, by querying the Hot Swap Control and Status bits INS and EXT. Any alternations to the handle status detected in the next polling cycle will be used to determine whether the board was inserted (Handle Switch closed), or whether it is about to be removed (Handle Switch open).
   (b) Whether the board is a PCI bus master. A PCI bus master is a peripheral board containing a bridge that connects the parent bus on which it resides, to a child bus residing beyond the board, capable of carrying additional boards. The Hot Swap Engine checks this by accessing the *Class Code* register in the PCI configuration space of the board. If the board is a PCI bus master, the Hot Swap Engine locates the buses and devices residing beyond that bridge and adds them to the scan list. The same scan scheme is applied to a child bus as well.
   (c) Whether the board is a multifunction device, by querying the *Header Type* register in the PCI Configuration Space of the board. If it is, the Hot Swap Engine queries all the board's functions and adds them to the scan list.

At the end of this process the Hot Swap Engine constructs a list of existing buses, boards and functions.

At the next polling cycle the Hot Swap Engine will poll the listed buses, boards and functions, along with the PCI buses, updating the scan list according to changes that result from insertion/extraction of boards, into or out of the system.

From this point onward, the PCI buses are polled constantly at regular intervals, thus updating and constructing a new scan list whenever a change is made.

Once a new scan list has been constructed, the engine compares the new list to the old one. It sends a message concerning any modification it detected to the kernel-mode Messaging Mechanism, from where it can be extracted by the relevant processes (user-mode applications or device drivers).
The Hot Swap Engine will detect one of the following modifications:

- **The ejector handle of an existing board has been opened** - the Hot Swap Engine sends a message asserting that the board is about to be removed and

adds this data to its database. However, the engine does not allocate this board's resources to a newly inserted board until it has been physically removed from the chassis.

- **An existing board has been physically extracted from the chassis** - The Hot Swap Engine sends a message to the kernel-mode Messaging Mechanism, asserting that the board has been removed, and adds this data to its database. The message is sent whether this board was extracted after all subscribed applications and drivers had terminated their access to the extracted board, or before (surprise extraction). The Hot Swap Engine also frees the board's resources to be reallocated when a new board is inserted into the chassis.
- **A new board has been inserted** - If this board was found to be with hot swap abilities, but its ejector handle is still open, no resources are allocated to it. If the ejector handle of a hot swap aware board is closed, the engine allocates its required resources (see below) and sends the appropriate message to the kernel-mode Messaging Mechanism, including the vendor/device ID, bus, slot and function.

Monitoring the ENUM# signal

Go-HotSwap enables you to hook into the ENUM# signal in order to detect hot swap events. In this case you should not activate the polling mechanism, but instead instruct your application to listen to the ENUM# interrupt in order to detect the hot swap events. Listening to the ENUM# interrupt is done with WinDriver's interrupt handling API. Please refer to the WinDriver user's guide for detailed information.

2.2.2 Allocating Required Resources (Enumeration)

NOTE:
The resource allocation is performed by the Hot Swap Engine unless you specify to use the operating system's Plug and Play Manager for this purpose. In `WD_WatchPciStart` [A.1.3] the **dwOptions** field can be changed to specify that the operating system is responsible for resource allocation.

In non-hot swap systems, enumeration only takes place as the system boots.
The Go-HotSwap software enables live enumeration under a running system. When a hot swap board is inserted into the system, the Hot Swap Engine performs the process

of Dynamic Configuration, whereby it allocates system resources to the board[2] while the system is running.

When using the polling method, the Hot Swap Engine automatically allocates (or de-allocates) the required resources. When using the ENUM# signal to detect hot swap events, the Go-HotSwap function `WD_HsEventSend()` should be called in order to allocate (or de-allocate) the required resources. Please refer to the Go-HotSwap Function Reference for more information regarding this function.

The allocated system resources are some or all of the following:

I/O, Memory and Prefetchable Memory

The resource allocation process differs in the case of a simple board and a PCI Bus Master.

a. Allocating I/O, Memory and Prefetchable Memory for a simple board

   When a simple board is inserted into one of the slots, the Hot Swap Engine first queries its required resources by accessing the PCI configuration space registers. The Hot Swap Engine queries all 6 BARs (Base Address Registers) of the board, and identifies the required resources of each BAR. These resources could be I/O, memory and prefetchable memory. It then allocates the board's required resources accordingly.

b. Allocating I/O, Memory and Prefetchable Memory for a PCI bus master

   When a PCI bus master is inserted into one of the slots, resources should be allocated for the PCI bus master itself, as well as for the child bus and for any board residing on the child bus beyond the bridge. Again, these resources could be I/O, Memory and Prefetchable Memory. The Hot Swap Engine first queries the 2 BARs of the PCI bus master and identifies the resources required by each BAR; it then allocates the board's required resources accordingly. The Hot Swap Engine also allocates resources to the child bus, according to a predefined algorithm. The Hot Swap Engine then scans the child bus to determine what resources are to be allocated for any board located on the child bus. At this stage, the engine also sets a range of bus/slot numbers available at the child bus.

Interrupts

When the Hot Swap Engine detects a board that requires interrupts, it assigns the interrupts as follows:

[2] see PICMG Hot Swap Specification 2.1 R2.0.

1. On x86 machines, if the board is inserted into a slot that is listed on the interrupt routing table located in the BIOS, the engine finds the appropriate register in the APIC (Advanced Programmable Interrupt Controller) and writes the interrupt number to the board.

2. On PPC machines, or if the board is inserted into a slot that is not listed in the interrupt routing table, the engine recursively calculates the interrupt line and then finds the appropriate register in the APIC and writes the interrupt number to the board.

Command Register

In order to access the board, the engine sets the *Command* register in the PCI configuration space to a predefined value, as defined in the PCI specification R2.2.

When the allocation process is complete, the relevant drivers/applications are notified by the Hot Swap Engine of the boards'/ devices' newly allocated resources. This message is either transferred directly to the driver (if a hot swap aware driver had been developed), or via the Go-HotSwap Configuration Manager (if a legacy PCI driver had been used without any driver code programming or modifications).

2.3 Kernel-Mode Messaging Mechanism

Any application or driver that needs to be notified upon a hot swap event, detected by the Hot Swap Engine, registers itself to the kernel-mode Messaging Mechanism. The registration process also includes the set of event criteria that are relevant to this application or device. For example, an application can be registered to receive notifications whenever a CompactPCI board with a vendor ID of 0x1234 is removed. Each registration call creates a separate linked list of events.

As of this point, every hot swap event that occurs and meets the predefined set of criteria, is added to the linked list and sent to the relevant subscriber.

After receiving a notification, the subscriber retrieves additional information from the linked list that was not included in the original notification. In the example above, after the application received a notification that a CompactPCI board with a vendor ID of 0x1234 has been removed, it retrieves additional information regarding its device ID and physical location.

2.4 Driver Development Toolkit

Jungo offers two types of driver development tools that can be used in order to simplify the difficult task of developing a device driver:

- WinDriver
- KernelDriver

WinDriver is a toolkit for developing monolithic device drivers and KernelDriver is a toolkit for developing standard operating system internal drivers that require hardware access and that must communicate with the operating system or must be implemented in the kernel.
In most cases a monolithic device driver is required (i.e. a driver that drives a hardware by directly accessing the hardware). This is why WinDriver is an integral part of the Go-HotSwap package.

KernelDriver is not an integral part of the Go-HotSwap package but as a Go-HotSwap user you are entitled to receive Jungo's KernelDriver toolkit. If needed, please contact Jungo and KernelDriver will be provided to you immediately.

WinDriver provides a hardware-access API that is available to any application. The WinDriver API enables the application to access the CompactPCI card's registers, memory ranges, I/O ranges and handle interrupts, via WinDriver's kernel module. The kernel module is a generic device driver that can access any device, as instructed by the WinDriver API. When developing a hot swap aware driver using the WinDriver toolkit, WinDriver's kernel module includes the Hot Swap Engine.

WinDriver features the DriverWizard, a GUI-based diagnostics and driver generation tool environment, which can automatically generate the driver code that is specific to the particular hardware. Another WinDriver feature included is the Kernel PlugIn, which enables performance critical parts of the user-mode code to run in the kernel mode, thereby achieving optimal performance, limited only by system capabilities.

When using the WinDriver wizard to generate your driver's hardware access code, you are asked to select your driver's options, one of which is Plug and Play notifications, another is Power Management notifications. Select these options in order to create a hot swap aware application. Use the Configuration Manager/**hs_activate** to start/stop the Hot Swap Engine, which will then notify the WinDriver application about hot swap events.

NOTE:
WinDriver is integrated into the Go-HotSwap package; therefore when you read the WinDriver documentation, all references to the **WinDriver** directory should be read as references to the **Go-HotSwap** directory. For example - the WinDriver user's guide refers to the PCI_SCAN utility, which is found under **windriver\util\pci_scan.exe**; instead the utility is found under **GoHotSwap\util\pci_scan.exe**.

As mentioned before, it is not a requirement to use the included driver development tools in order to utilize the Go-HotSwap infrastructure, but by using WinDriver you will gain the following advantages:

- **Easy Development** - WinDriver enables programmers to create PCI/ISA/EISA/ISA PnP/PCMCIA/USB based device drivers in an extremely short time. WinDriver allows you to create your driver in **user mode**, in a familiar environment - using MSDEV, Visual C/C++, Borland, Delphi, Visual Basic or any other Win32 compiler. WinDriver eliminates the need for you to be familiar with operating system internals, kernel programming, the DDK or have any device driver knowledge.
- **Multi OS Support** - The driver created with WinDriver will run on Windows 98/Me/NT/2000/XP/Server 2003/CE, Linux, Solaris and VxWorks. The significance of it is that you write the code once and run on any of these platforms.
- **Cross OS capabilities** - The code generated with WinDriver will run on all the supported operating systems without any modifications.
- **Friendly Wizards** - The DriverWizard (included) is a Graphical diagnostics tool that lets you write to, and read from the hardware, before writing a single line of code. With a few clicks of the mouse, the hardware is diagnosed - memory ranges are read, registers are toggled and interrupts are checked. Once the device is operating to your satisfaction, the DriverWizard creates the skeletal driver source code, giving access functions to all of the resources on the hardware.
- **Kernel-Mode Performance** - WinDriver's API is optimized for performance. For drivers that need kernel-mode performance, WinDriver offers the **Kernel PlugIn**. This powerful feature enables you to create and debug your code in user mode, and run the performance critical parts of your code, (such as the interrupt handler, or access to I/O mapped memory ranges), in kernel mode,

thereby achieving kernel-mode performance (zero performance degradation). This unique feature allows the developer to run the user-mode code in the operating system kernel without having to learn how the kernel works. When working on Windows CE, there is no need to use the Kernel PlugIn since CE has no separation between user mode and kernel mode, thus enabling you to easily achieve optimal performance from the user-mode code.

WinDriver user's guide is included in this package, and we recommend that you refer to it when developing your driver.

2.5 Configuration Manager/hs_activate

The **Configuration Manager** and **hs_activate** are user-mode applications, which enable the Go-HotSwap engine to respond to hot swap events. The **hs_activate** is a console-mode application suitable for all supported operating systems, while the Configuration Manager is a graphical implementation of **hs_activate**, suitable for Windows only.

When developing a hot swap aware driver (using WinDriver or any other toolkit), the engine is able to interface directly with the driver within the kernel mode. However, Go-HotSwap enables you to use the Configuration Manager/hs_activate to initiate the engine and interface with the user to receive further instructions.

The Configuration Manager/hs_activate can be pre-configured to do any of the following upon insertion or extraction of a board: start/stop a PCI driver or a service or run any user-mode application or script. These instructions are kept in the Configuration Manager's database, as a configuration file.

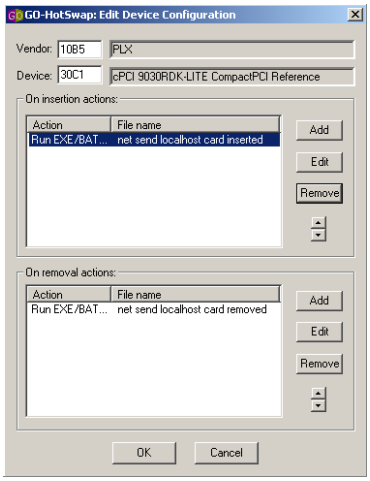


Figure 2.2: Configuration Manager - Device Configuration Screen

The Configuration Manager is not needed whenever a CompactPCI hot swap aware driver is developed with the WinDriver API, other than for activating and stopping the

Hot Swap engine. In this case, the Hot Swap Engine data is transferred directly to the newly developed CompactPCI driver.

2.6 Technical Overview Summary

The combination of these four components enables Go-HotSwap to add hot swap capabilities for CompactPCI hardware to all major operating systems, and with any available system board.

Go-HotSwap enables users to immediately add hot swap capabilities to their existing PCI drivers or to easily develop hot swap aware drivers from scratch. The resulting code will compile and run on all supported operating systems.

Go-HotSwap also transparently supports any CompactPCI hardware, i.e. basic or full hot swap, as well as silicon friendly and ready.

Chapter 3

Installation and Setup

3.1 Recommendations for Windows and Linux

In the CompactPCI system the devices are attached and detached dynamically from the system. Therefore the PCI devices tree, comprised of bridges and child devices, should be configured dynamically. The dynamic configuration includes in particular allocation of I/O and memory resources.

The bridge is configured (as *open*) once, either at boot time or when a device is attached at a later stage. If at boot time there are devices with I/O or memory attached to a child bus behind the bridge, the bridge is configured as open at this stage. Otherwise, the bridge is configured as closed and will be reconfigured as open, only when a device with I/O or memory is attached to the child bus. The bridge is configured to accommodate the resources of its child devices.

Therefore it is recommended to attach all child devices before boot time, so that the bridge's configuration will include sufficient resources for all the attached devices.

If only part of the devices are attached at the time of the bridge configuration, the allocated resources may not be sufficient for additional devices. This may result in a case where a device is attached and the operating system cannot assign the appropriate resources for this device.

3.2 System Requirements

For Windows Platforms

- An x86 processor
- Any 32-bit development environment supporting C (not required if working only with the Configuration Manager)

For Linux

- Linux 2.6
- An x86 processor
- A GCC compiler for WinDriver installation

  NOTE:
  The GCC compiler must be the same version as the running kernel.

- Any 32-bit development environment supporting C (such as GCC) for user mode
- On your development PC: **glibc2.3.x**

For Solaris

NOTE:
On Solaris, Go-HotSwap/WinDriver relies on System Configuration Administration support. Hot-swap is supported only for slots that are visible to the **cfgadm** system utility.

- Solaris 8.0 / 9.0 (for Solaris 8 it is recommended to use update 3 or higher)
- 64-bit or 32-bit kernel on SPARC platform
  or
  32-bit kernel on x86 platform
- Any 32-bit development environment supporting C (such as GCC).

- You should make sure that you have the following software installed:
  Solaris drivers:
  - **pcihp** - PCI nexus hotplug support
  - **hpcsvc** - hot-plug controller services
  - **sc** - System Management Controller Driver
  - **hsc** - Hotswap Controller module

  Applications may access the system hotplug services through the interfaces provided by **libdevice.so.1**. Once the system hotplug services and the **libdevice.so.1** library are properly installed, you should be able to obtain a list of CompactPCI slots by running **cfgadm**.

  A sample output from a correctly installed system:

```
$> cfgadm
Ap_Id              Type         Receptacle  Occupant     Condition
pci1:hsc0_slot2    unknown      connected   unconfigured unknown
pci1:hsc0_slot3    bridge/fhs   connected   configured   ok
pci1:hsc0_slot4    unknown      connected   unconfigured unknown
pci1:hsc0_slot5    unknown      connected   unconfigured unknown
pci1:hsc0_slot6    unknown      connected   unconfigured unknown
pci1:hsc0_slot7    unknown      connected   unconfigured unknown
pci1:hsc0_slot8    unknown      connected   unconfigured unknown
```

NOTE:
When hot-removing a device, it is required to issue an **`unconfigure`** command before physically removing the device, e.g.:

```
$> cfgadm -c unconfigure pci1:hsc0_slot3
```

There is no need to issue any command when hot-inserting a device.

3.3 Go-HotSwap Installation Instructions

The Go-HotSwap CD contains Go-HotSwap versions for all the different operating systems. The CD's root directory contains the Windows version. The installation for the Windows version automatically starts when you insert the CD into your CD drive. The other versions of Go-HotSwap are located in the sub-directories according to the operating system i.e. **\Solaris**.

3.3.1 Installation Instructions for Windows

This section refers to Go-HotSwap installation on Windows 2000, XP and Server 2003

NOTE:
You must have administrative privileges in order to install Go-HotSwap on Windows platforms.

1. Insert the Go-HotSwap CD into the CD drive. (When installing Go-HotSwap by downloading it from Jungo's web site, double-click the downloaded Go-HotSwap file (**HSxxx.exe**) in your download directory, and go to Step 3)
2. Wait a few seconds until the installation program automatically starts. If for some reason it does not start automatically, double-click the file **HSxxx.exe** (where xxx is the version number) and click the **Install Go-HotSwap** button.
3. Read the license text carefully, and click **Yes** if you accept its terms.
4. Choose the destination location in which to install Go-HotSwap.
5. In the **Setup Type** screen, choose one of the following:
   - Typical - to install all Go-HotSwap modules (generic Go-HotSwap toolkit + specific chipset APIs).
   - Minimal - to install only the generic Go-HotSwap toolkit.
   - Custom - to choose which modules of the Go-HotSwap to install; you may choose which APIs will be installed.

   We recommend that you choose **Typical** installation.
6. You will be prompted to reboot your computer.

The following steps are for registered users only:

In order to register your copy of Go-HotSwap with the license you received from Jungo, follow the steps below:

1. Activate the Go-HotSwap Configuration Manager by clicking
   Start | Programs | Go HotSwap | Go-HotSwap Configuration Manager.
   OR
   Activate the WinDriver wizard GUI by clicking
   Start | Programs | Go HotSwap | Wizard.

2. Select the **Register Go-HotSwap** option from the **File** menu and insert the license string you received from Jungo. Click the **Activate License** button.

3. To activate the source code you developed during the evaluation period, please refer to `WD_License` function reference.

   Calling the appropriate function with the license string you received from Jungo converts your existing samples from evaluation-only to registered.

The installation process installs the following:

- The kernel-mode element of the Go-HotSwap Engine
  - **windrvr6.sys**, in **\WINNT\system32\drivers**.
- Go-HotSwap API and the WinDriver API
  - **\GoHotSwap\include\windrvr.h**
- Sample applications that demonstrate the use of the Go-HotSwap API (user-mode - source code included)
  - **\GoHotSwap\hotswap\hs_detect**
  - **\GoHotSwap\hotswap\hs_activate**
  - **\GoHotSwap\hotswap\hs_reenum**
  - **\GoHotSwap\samples\...** - includes the following samples: BASIC_IO, INT_IO, PCI_DIAG, PCI_DUMP, PCI_SCAN, PCI_DMA, WDDEBUG, WDREG, ISAPNP_SCAN, SPEAKER, SPEAKER_GUI, CMOS and SHARED
- Utilities
  - **\GoHotSwap\hotswap\hs_extender.exe**
  - **\GoHotSwap\wizard\wdwizard.exe**
  - **\GoHotSwap\kerplug\...** includes Kernel PlugIn
  - **\GoHotSwap\util\...** - includes the following utilities: ISAPNP_SCAN, PCI_DIAG, PCI_DUMP, PCI_SCAN, WDREG, WDDEBUG and WDDEBUG_GUI
- Documentation

  - **GoHotSwap\Docs\...**

- Enhanced support for specific PCI chip-sets

  - **GoHotSwap\altera**
  - **GoHotSwap\amcc**
  - **GoHotSwap\marvell**
  - **GoHotSwap\plx**
  - **GoHotSwap\quicklogic**
  - **GoHotSwap\xilinx**

3.3.2 Installation Instructions for Linux

Preparing the System for Installation

In Linux, kernel modules must be compiled with the same header files that the kernel itself was compiled with. Since Go-HotSwap installs the kernel module **windrvr6.ko**, it must be compiled with the header files of the Linux kernel during the installation process. Therefore, before you install Go-HotSwap for Linux, verify that the Linux source code and the file **versions.h** are installed on your machine:

Install linux kernel source code

- If you have yet to install Linux, please choose **Custom** installation when performing the installation and then choose to install the source code.
- If Linux is already installed on the machine, you must check to see if the Linux source code was installed. You can do this by looking for linux in the **/usr/src** directory. If the source code is not installed, you can either reinstall Linux with the source code, as described above, or you can install the source code by following these steps:

  1. Login as super user.
  2. Type:
     /$ rpm -i /<source location>/ <Linux distributor>/RPMS/kernel-source-<version number>
     (For example: to install the source code from the Linux installation CD-ROM, for RedHat 7.1, type:
     /$ rpm -i /mnt/cdrom/RedHat/RPMS/

kernel-source-2.6.2.-2.i386rpm)

> **TIP**
> If you do not have an RPM with the source code, you may download it from: http://rpmfind.net/linux/RPM/.

Install version.h

- The file **version.h** is created when you first compile the Linux kernel source code. Some distributions provide a compiled kernel without the file **version.h**. Look under **/usr/src/linux/include/linux/** to see if you have this file. If you do not, please follow these steps:

  1. Type:
     /$ make xconfig
  2. Save the configuration by choosing **Save and Exit**.
  3. Type:
     /$ make dep.

Before proceeding with the installation, you must also make sure that you have a linux symbolic link. If you do not, please create one by typing:
/usr/src$ ln -s <target kernel>/ linux
(For example: for Linux 2.6 kernel type:
/usr/src$ ln -s linux-2.6/ linux)

Installation

1. Insert your CD into your Linux machine CD drive or copy the downloaded file to your preferred directory.
2. Change directory to your preferred installation directory (your home directory, for example):
   /$ cd ~
3. Extract the file **HSxxxLN.tgz** (where xxx is the version number):
   ~$ tar xvzf /<file location>/HSxxxLN.tgz
   For example:
   - From a CD:
     ~$ tar xvzf /mnt/cdrom/LINUX/HSxxxLN.tgz
   - From a downloaded file:
     ~$ tar xvzf /home/username/HSxxxLN.tgz

4. Change directory to Go-HotSwap.
5. Install Go-HotSwap for Linux:
   (a) `~/GoHotSwap/redist$ ./configure`

   > **NOTE:**
   > The **`configure`** script creates a **makefile** based on your specific running kernel. You may run the **`configure`** script based on another kernel source you have installed, by adding a flag **`--with-kernel-source=<path>`** to the configure script. The <path> is the full path to the kernel source directory. The default kernel source directory is **/usr/src/linux**.

   (b) `~/GoHotSwap/redist$ make`
   (c) Become super user:
   `~/GoHotSwap/redist$ su`
   (d) Install the driver:
   `~/GoHotSwap/redist# make install`

   The following three steps are optional:

6. Create a symbolic link so that you can easily launch the DriverWizard GUI
   `~/GoHotSwap$ ln -s <full path to GoHotSwap>/GoHotSwap/wizard/wdwizard/ usr/bin/wdwizard`
7. Change the read and execute permissions on the file **wdwizard** so that ordinary users can access this program.
8. Change the user and group ids and give read/write permissions to the device file **/dev/windrvr6** depending on how you wish to allow users to access hardware through the device.

The following steps are for registered users only

In order to register your copy of GoHotSwap with the license you received from Jungo, follow the steps below:

1. Activate the DriverWizard GUI:
   `~/GoHotSwap/wizard$ ./wdwizard`

2. Select the **Register Go-HotSwap** option from the **File** menu and insert the license string you received from Jungo.
3. Click the **Activate License** button.
4. To register source code you developed during the evaluation period, please refer to `WD_License` function reference.

Restricting Hardware Access on Linux

CAUTION:
Since **/dev/windrvr6** gives direct hardware access to user programs, it may compromise kernel stability on multi-user Linux systems. Please restrict access to the DriverWizard and the device file **/dev/windrvr6** to trusted users.

For security reasons the Go-HotSwap installation script does not automatically perform the steps of changing the permissions on **/dev/windrvr6** and the DriverWizard executable (**wdwizard**).

The Installed Components

This process installs the Go-HotSwap package, which includes the following components:

- windrvr6 - The Go-HotSwap kernel-mode element (in */kernel/drv/*)
- windrvr.h - Go-HotSwap and WinDriver API (in **GoHotSwap/include/**)
- Go-HotSwap samples (user-mode source code included):
  - **/GoHotSwap/hotswap/hs_detect**
  - **/GoHotSwap/hotswap/hs_activate**
  - **/GoHotSwap/hotswap/hs_reenum**
  - **/GoHotSwap/samples/...** - includes the following samples: BASIC_IO, INT_IO, PCI_DIAG, PCI_DUMP, PCI_SCAN, PCI_DMA, WDDEBUG, WDREG, ISAPNP_SCAN, SPEAKER, SPEAKER_GUI, CMOS and SHARED
- Utilities
  - **/GoHotSwap/wizard/wdwizard.exe**

  - **/GoHotSwap/kerplug/...**
  - **/GoHotSwap/util/...** - includes the following utilities: ISAPNP_SCAN, PCI_DIAG, PCI_DUMP, PCI_SCAN, WDREG, WDDEBUG and WDDEBUG_GUI

- Documentation
  - **GoHotSwap/Docs**
- Enhanced support for specific PCI chip-sets
  - **GoHotSwap/altera**
  - **GoHotSwap/amcc**
  - **GoHotSwap/marvell**
  - **GoHotSwap/plx**
  - **GoHotSwap/quicklogic**
  - **GoHotSwap/xilinx**

3.3.3 Installation Instructions for Solaris

Since Go-HotSwap installation installs the kernel module **windrvr6**, it should be installed by the system administrator logged in as root, or with root privileges.

1. Insert your CD into your Solaris machine CD drive or copy the downloaded file to your preferred directory.
2. Change the directory to your preferred installation directory (your home directory, for example):
   `/# cd ~`
3. According to your Solaris platform:
   - **HSxxxSLS32.tgz** (for SPARC 32-bit)
   - **HSxxxSLS64.tgz** (for SPARC 64-bit)
   - **HSxxxSL.tgz** (for x86 32-bit)

   Copy the file to the current directory (xxx is the version number):
   `~# cp /home/username /HSxxxSL.tgz /`

4. Extract the file:
   ~# **gunzip -c HSxxxSL.tgz | tar -xvf HSxxxSL.tar**

5. Change directory to Go-HotSwap.

6. Install Go-HotSwap for Solaris:
   The installation requires you to determine on which PCI card you will be working, by defining the device's Vendor ID and Device ID (hexadecimal values). There are two different ways to do this and install the driver:

   - Modify the installation file before performing the installation and then install Go-HotSwap:

     (a) Open the file **install_windrvr** for editing.
     In case of a single PCI device, modify the **DEFAULT_VENDOR_ID** and **DEFAULT_DEVICE_ID** existing variables to your PCI card's identification values.
     In case of multiple PCI devices, issue the **add_drv** command with the list of PCI devices, by modifying the **add_drv** line in the script. For example, for two PCI devices, PLX 9080 and PLX 9054, with vendor/device IDs 10b5/401 (PLX 9080) and 10b5/1860 (PLX 9054), change the **add_drv** command to:
     # **/usr/sbin/add_drv -v -m "* 0666 root sys" -i "pci10b5,401 pci10b5,1860" windrvr6**

     (b) Install Go-HotSwap:
     ~/GoHotSwap# **./install_windrvr**

   - Use the Command Line to change your device's identification values and install the driver in one step, by typing:
     ~/GoHotSwap# **VENDOR_ID=XXXX DEVICE_ID=XXXX ./install_windrvr**

7. Install the **libgcc** package, available for download from http://www.sunfreeware.com/.

8. Add an environment variable:

   - For SPARC 32-bit and x86 platforms:
     LD_LIBRARY_PATH=/usr/local/bin
   - For SPARC 64-bit platforms:
     LD_LIBRARY_PATH=/usr/local/lib:/usr/local/lib/sparcv9

The following three steps are optional:

9. Create a symbolic link so that you can easily launch the DriverWizard GUI:
   `~/GoHotSwap# ln -s ~/GoHotSwap/wizard/wdwizard /usr/bin/wdwizard`
10. Change the read and execute permissions on the file **wdwizard** so that ordinary users can access this program.
11. Change the user and group ids and give read/write permissions to the device file **/dev/windrvr6** depending on how you wish to allow users to access hardware through the device.

The following steps are for registered users only
In order to register your copy of Go-HotSwap with the license you have received from Jungo, please follow these steps:

- Activate the DriverWizard GUI:
  `~/GoHotSwap/wizard$ ./wdwizard`
- Select the **Register Go-HotSwap** option from the **File** menu and insert the license string you received from Jungo.
- Click the **Activate License** button.
- To register source code you developed during the evaluation period, please refer to `WD_License` function reference.

Restricting Hardware Access

CAUTION:
Since **/dev/windrvr6** grants direct hardware access to user programs, it may compromise kernel stability on multi-user Solaris systems. Please restrict access to DriverWizard and the device file **/dev/windrvr6** to trusted users only.
For security reasons the Go-HotSwap installation script does not automatically perform the steps of changing the permissions on **/dev/windrvr6** and the DriverWizard executable (**wdwizard**).

The Installed Components

This process installs the Go-HotSwap package, which includes the same installed components as the Linux distribution. For a detailed list, please refer to Section 3.3.2.

3.4 Checking Your Installation

You can check your installation using the wizard, as demonstrated here, or by using the Configuration Manager/**hs_activate**, as demonstrated in the following sections.

1. Start DriverWizard:
   On Windows, by choosing **Programs | Go-HotSwap | Wizard** from the **Start** menu, or using the shortcut that is automatically created on your Desktop. A third option for activating the Wizard on Windows is by running **wdwizard.exe** from a command prompt under the **wizard** sub-directory.
   On Linux and Solaris you can access the wizard application via the file manager under the **wizard** sub-directory, or run the wizard application via a shell.
2. Make sure that your Go-HotSwap license is installed. If you are an evaluation version user, you do not need to install a license.
3. Hot plug your board into the CompactPCI bus, and make sure that wizard detects it.

3.4.1 On Your Windows Machine

Start the Configuration Manager by choosing from the Windows Start menu:
Programs | GoHotSwap | HotSwap Configuration Manager.

- Make sure that your Go-HotSwap license is installed (see Section 3.3.1, Installing Instructions for Windows).
  If you are an evaluation version user, you do not need to install a license.
- Hot plug your board into the cPCI bus, and make sure that Go-HotSwap detects it.

3.4.2 On Your Linux and Solaris Machines

- Run the pre-compiled utility hs_detect
  (**.../GoHotSwap/hotswap/hs_detect/.../hs_detect**)
- Hot plug your board into the cPCI bus, and make sure that Go-HotSwap detects it.

3.5 Uninstalling Go-HotSwap

3.5.1 On Windows

1. Close any open Go-HotSwap applications, including Configuration Manager, DriverWizard, the DebugMonitor log and any user-specific applications.
2. If you created a Kernel PlugIn driver:
   - If your Kernel PlugIn driver is currently installed, uninstall it by running:
     `wdreg -name <Kernel PlugIn name> uninstall`

     NOTE:
     The Kernel PlugIn name should be specified without the *.sys extension.
   - Erase your Kernel PlugIn driver from the **%windir%\system32\drivers** directory.
3. On all Windows platforms on which **windrvr6.sys** was installed:
   - Uninstall any Plug-and-Play devices registered to work with Go-HotSwap/WinDriver via an INF file:
     `wdreg -inf <path to the device-specific INF file> uninstall`
   - Verify that there are no INF files that register your device(s) with Go-HotSwap/WinDriver (**windrvr6.sys**) in the **%windir%\inf** directory
4. Uninstall WinDriver's kernel module - **windrvr6.sys**.

   NOTE:
   It is recommended **not** to uninstall the WinDriver kernel module (**windrvr6.sys**), since WinDriver is designed as a generic driver module and may be used by other drivers in the system. To successfully terminate your usage of WinDriver without uninstalling the WinDriver kernel module, simply skip this step and proceed to the steps below.

   - To uninstall **windrvr6.sys** run:
     `wdreg -inf <path to windrvr6.inf> uninstall`

     NOTE:
     windrvr6.sys should reside in the same directory as **windrvr6.inf** when running this command.

- Erase the following files if they exist:
  %windir%\system32\drivers\windrvr6.sys
  %windir%\inf\windrvr6.inf

5. This step is required only for computers on which the entire Go-HotSwap tool-kit has been installed.

   - Uninstall the Go-HotSwap tool-kit using the uninstall shield:
     Start | Settings | Control Panel | Add/Remove Programs
   - Erase the **GoHotSwap** installation directory.
   - Erase Go-HotSwap's entry in the Start menu.
     For example: Right-click the mouse on **GoHotSwap** in **Start | Menu | Programs** and select **Delete**.

6. Remove the Go-HotSwap/WinDriver DLLs.

   NOTE:
   We recommend **not** to perform this step, since removing the DLLs will effect other Go-HotSwap/WinDriver based applications that may be running in the system.

   Erase the following DLL files if they exist:
   %windir%\system32\wd_utils.dll
   %windir%\system32\wdlib.dll

7. Reboot the computer.

3.5.2 On Linux

NOTE:
You must be logged in as root to perform the uninstall procedure.

1. Verify that the WinDriver module is not being used by another program:

   - View a list of modules and the programs using each of them:
     `/# /sbin/lsmod`
   - Close any applications that are using the WinDriver module.
   - Unload any modules that are using the WinDriver module:
     `/# /sbin/rmmod`

2. Unload the WinDriver module:
   `/# /sbin/rmmod windrvr6`
3. Remove the old device node in the **/dev** directory:
   `/# rm -f /dev/windrvr6`
4. If you created a Kernel PlugIn, remove it as well.
5. Remove the file **.windriver.rc** from the **/etc** directory:
   `/# rm -f /etc/.windriver.rc`
6. Remove the file **.windriver.rc** from **$HOME**:
   `/# rm -f $HOME/.windriver.rc`
7. If you created a symbolic link to DriverWizard, delete the link using the command:
   `/# rm -f /usr/bin/wdwizard`
8. Delete the Go-HotSwap installation directory, after changing the directory to the one above Go-HotSwap:
   `# rm -rf ~/GoHotSwap`

3.5.3 On Solaris

NOTE:
You must be logged in as root in order to perform the uninstall procedure.

1. Make sure no programs are using Go-HotSwap.
2. If you created a Kernel PlugIn, remove it by following these steps:
   (a) `# /usr/sbin/rem_drv kpname`
   (b) On 64-bit platforms (64-bit SPARC):
   `# rm /kernel/drv/sparcv9/kpname`
   On 32-bit platforms (32-bit x86/SPARC):
   `# rm /kernel/drv/kpname`
   (c) `# rm /kernel/drv/kpname.conf`
3. Run the following uninstallation script:
   `~/GoHotSwap# ./remove_windrvr`

4. Run the following command:

   - On 64-bit platforms (64-bit SPARC):
     # **rm -f /kernel/drv/sparcv9/windrvr6 /kernel/drv/windrvr6.conf**
   - On 32-bit platforms (32-bit x86/SPARC):
     # **rm -f /kernel/drv/windrvr6 /kernel/drv/windrvr6.conf**

5. Remove the **.windriver.rc** file in the **/etc** directory. To do this, run the following command:
   # **rm -f /etc/.windriver.rc**

6. Remove the **.windriver.rc** file in the **$HOME** directory. To do this, run the following command:
   # **rm -f $HOME/.windriver.rc**

7. If you created a symbolic link to DriverWizard, delete the link:
   # **rm -f /usr/bin/wdwizard**

8. Delete the Go-HotSwap installation directory, after changing the directory to the one above Go-HotSwap:
   # **rm -rf ~/GoHotSwap**

Chapter 4

Using Go-HotSwap

4.1 Overview

There are two basic methods in which Go-HotSwap can be used:

1. **Using the WinDriver/Go-HotSwap API:** Use Jungo's driver development tools to develop a driver from scratch, which will automatically support hot swap events.
2. **Using the Configuration Manager:** Configure the system to execute predefined tasks upon hot swap events according to a user-defined set of criteria. This method enables hot swap capabilities with legacy drivers without the need to modify the driver's source code[1]. Go-HotSwap provides a console-mode version of the Configuration Manager called **hs_activate**.

Having learnt what the Go-HotSwap package includes, we will further explore each of the above methods.

[1] Provided that the driver complies to PCI standard of dynamic loading/unloading.

4.2 What Does the Go-HotSwap Package Include?

4.2.1 Utilities

- **Go-HotSwap Configuration Manager** - Only on Windows versions; accessible through: **Start | Programs | GoHotSwap | Hot Swap Configuration Manager**. The Configuration Manager is a graphical configuration utility that enables defining of certain tasks to be performed upon future hot swap events (for example load/unload a legacy device driver).
- **WinDriver Utilities:** Go-HotSwap includes all the utilities that are provided with WinDriver. Below is a short description of these utilities, which provide tools for diagnostics, debugging and automatic code generation. For full documentation and implementation instructions please refer to the WinDriver User's Guide.
  - **DriverWizard** - A graphical development and debugging tool that guides you through the steps required to develop a device driver and collects debugging information on your driver as it runs. Diagnose your hardware using DriverWizard and let DriverWizard generate your driver's skeletal code.
  - **PCI_SCAN** (\GoHotSwap\util\) - A utility for obtaining a list of the CompactPCI devices installed and the resources allocated to each of them (memory ranges, I/O ranges and interrupts).
  - **PCI_DUMP** (\GoHotSwap\util\) - A utility for obtaining a dump of all the PCI configuration registers of the PCI/CompactPCI devices installed.
  - **PCI_DIAG** (\GoHotSwap\util\) - A utility for reading/writing PCI configuration registers, and accessing the PCI card's I/O and memory ranges. It can be used as a sample hardware access code.
  - **WDDEBUG** (\GoHotSwap\util\) - A utility that enables you to view debug messages (in protected operating systems these are kernel level messages). Debug mode slows down transfer operations and therefore it should be used only in the development process. Running this command without the on/off parameter prints the version of the installed Go-HotSwap.
  - **WDDEBUG_GUI** (\GoHotSwap\util\wddebug_gui.exe) - A Graphical User Interface of WDDEBUG.
  - **WDREG** (\GoHotSwap\util\) - Registers/removes Go-HotSwap in/from the Windows registry. To install Go-HotSwap on a new computer, copy

windrvr6.sys file to **C:\WINNT\SYSTEM32\DRIVERS** (WinNT). After copying Go-HotSwap to the driver's directory, add Go-HotSwap to the list of device drivers that Windows loads on boot by running **wdreg.exe** install. To install Go-HotSwap from within your own application, add the WDREG source to your own installation code.

– **Kernel PlugIn** (\GoHotSwap\kerplug) - The files and samples needed to create a **Kernel PlugIn**.

- **Distribution Package** (\GoHotSwap\redist) - The files specified in this section must be included in the driver you distribute to your customers.

4.2.2 Samples

The Go-HotSwap package provides source code for the utilities listed above, along with other samples, which demonstrate how the Go-HotSwap and WinDriver APIs can be used in various applications. Find the sample which is closest to the driver/application you need. Use it to jump-start your driver development process.

1. **Go-HotSwap Samples** (**\GoHotSwap\hotswap\...**)- Samples that demonstrate how various hot swap tasks are performed using Go-HotSwap API.

   - **hs_detect** (**\GoHotSwap\HotSwap\hs_detect**) - The **hs_detect** utility prints messages to the screen upon hot swap events.
     The application first activates the Go-HotSwap Hot Swap Engine, and registers itself to receive notifications regarding hot swap events according to a user predefined set of criteria. The default is to receive notifications regarding all hot swap events.
     When activated by the **hs_detect** utility, the Hot Swap Engine detects the insertion and removal of CompactPCI boards, dynamically allocates/deallocates resources required by these boards and sends the relevant notifications to the **hs_detect** application.
     When the **hs_detect** application receives such notifications, it prints a message to the screen accordingly. For example, if a cPCI board is removed, then **hs_detect** prints a message saying that a board of bus x, slot y, function z has been removed.
   - **hs_activate** (**\GoHotSwap\HotSwap\hs_activate**) - The **hs_activate** utility activates any desired application upon insertion or removal of a CompactPCI board, according to the user's pre-configuration.

This utility activates the Go-HotSwap Engine and upon each insertion/removal of a CompactPCI card, performs the actions defined in a special configuration file (**\GoHotSwap\hotswap\hs_activate\...\hs_conf.rul**).
The file **hs_conf.rul** can be edited and its configuration altered according to your requirements. This file contains the database of the actions to be carried out upon insertion/removal of each device. Please refer to **hs_activate** utility to see how **hs_conf.rul** is implemented.

- **hs_reenum** (**\GoHotSwap\HotSwap\hs_reenum**\) - The **hs_reenum** utility performs re-enumeration according to user permissions.
  It can be referred to as a sample for implementing the Go-HotSwap function `WD_HsEventSend()` for re-enumeration purposes.

2. **WinDriver Samples** (**\GoHotSwap\samples**)- Samples that demonstrate different common drivers. For more information, please refer to the relevant chapter in WinDriver user's guide.
3. **WinDriver for PLX/Altera/Marvell/AMCC/QuickLogic/Xilinx PCI chips** (for example - **\GoHotSwap\plx\p9054_diag etc**) - Source code of the diagnostic applications for the specific chipsets that WinDriver supports. For more information, please refer to the relevant chapter in WinDriver user's guide.

4.3 Using the API

Go-HotSwap and the included WinDriver toolkit provide an API with which you can achieve hot swap capabilities. From a process point of view, if you choose this method, there are two development challenges:

1. Generate the driver's hardware access code.
2. Add hot swap capabilities to the driver.

Here is a more detailed description of these steps:

4.3.1 Generate Driver's Hardware Access Code

If you have not yet developed hardware access code, we recommend you use Jungo's WinDriver toolkit, for it dramatically reduces development time; moreover, the

generated code has cross operating system capabilities. Detailed information on driver development issues (such as interrupt handling, DMA, improving performance, etc.), and the WinDriver tools, can be found in WinDriver User's Guide.

WinDriver's toolkit includes a wizard that helps generate your skeletal driver's hardware access code. During this process, you are asked to select your driver's options, one of which is Plug and Play notifications, another is Power Management notifications. Make sure to select these options in order to use WinDriver's application with the Hot Swap Engine, as demonstrated in figure 4.1.

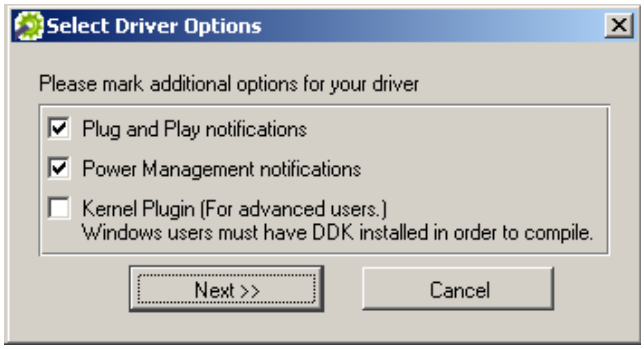


Figure 4.1: Notification Events

As explained earlier, it is not a requirement to have the driver's hardware access code generated with WinDriver; hot swap capabilities can also be achieved with drivers developed in other ways.

4.3.2 Add Hot Swap Capabilities to the Driver

In order to make your driver or application hot swap aware and to utilize the hot swap services provided by Go-HotSwap, you must add the Go-HotSwap API, as well as some relevant WinDriver API. A list of the API can be found in the Go-HotSwap Function Reference chapter, later on in this manual.
The API should be used according the following procedure:

1. Activate the Hot Swap Engine.
   The Hot Swap Engine should be activated in order for the application to receive Hot Swap services, which is done by calling the function

`WD_WatchPciStart()` [A.1.3]. Each application should call `WD_WatchPciStart()` before implementing another Go-HotSwap API. You can call it even if the Hot Swap Engine has already been activated; so there is no need to check if the Hot Swap Engine is activated or not.
After the Engine is activated, it starts detecting hot swap events, and accordingly, dynamically allocates/deallocates resources.

2. Register your application to receive notifications.
   In order for your application to utilize the hot swap services provided by the Hot Swap Engine and to become hot swap aware, you should register your application to receive notifications about hot swap events, according to a user defined set of criteria. Do this by using the API function `WD_EventRegister()` [A.3.2]. You can define whether to receive notifications about every hot swap event, or only about specific events, for example receive notifications only for a certain vendor ID or device ID on a specific cPCI slot.

3. Retrieve more information about a hot swap event.
   After your application registers itself to receive notifications, the Hot Swap Engine sends notifications to your application accordingly. When your application receives such notification, it can retrieve more information about the event by using the API function `WD_EventPull()` [A.3.4].

4. Perform any functionality pertinent to the event.
   When your application has all the information about the hot swap event, it can start performing any functionality that needs to be done in conjunction with the event. For example if a VoIP board has just been installed into the chassis, and a notification about this event has been received, once all relevant information has been retrieved, your application can start the VoIP board specific functionality. A simpler example would be to just print a message to the screen, describing the hot swap event that has just occurred. Actually this is what the utility **hs_detect** does; you can refer to it to see how the API is implemented.

The API can be also used for the following purposes:

- **Blue Led control:** The API can be used to light the blue led of a cPCI board when it is safe (from the application's perspective) to physically remove it from the backplane. Do this by using the API function `WD_EventSend()` [A.3.5]. The proper procedure is:

  1. Instruct the Hot Swap Engine to wait for an acknowledgement from the application before lighting the blue led.

2. Once the application is notified that the relevant board is about to be removed, it should follow the relevant procedure to ensure safe removal of the board, for example, stop communicating with the board.
3. Only at this stage may the application acknowledge the event to the Hot Swap Engine.
4. The Hot Swap Engine lights the blue led. The operator will then notice the blue led is lit, indicating that it is safe to physically remove the board.

- **Re-Enumeration:** This functionality is also achieved with `WD_EventSend()` [A.3.5]. Re-enumeration means re-allocation of the board's resources. The user can define whether to re-enumerate a bus, or a specific slot/device.
  The re-enumeration process should be called before the drivers of the re-enumerated boards start to load or after they have stopped, since as a result of the re-enumeration process, the originally assigned resources are reallocated and hence drivers may access invalid addresses.

The last stage of the hot swap procedure is to turn off the Hot Swap Engine. This is done by calling the `WD_WatchPciStop()` [A.1.4] function.

4.4 The Configuration Manager

4.4.1 Overview

When developing a hot swap aware driver, using the WinDriver driver development toolkit or any other development tool, Go-HotSwap is able to interface directly with the driver within the kernel mode. However, legacy PCI drivers were written without hot swap support, and are provided by vendors without the necessary software layers required to support hot swapping.
Go-HotSwap Configuration Manager enables immediate implementation of hot swap capabilities using a legacy PCI driver without any driver code programming or modifications. In this case, the Configuration Manager initiates the Hot Swap Engine and interfaces with the user to receive further instructions.
The user can instruct the Configuration Manager to load/unload a PCI driver during the insertion/extraction of a board, or to run any user-mode application or script. These instructions are kept in the Configuration Manager's database, as a configuration file. The Configuration Manager's database is constructed of a list of hot swap enabled

cards, and a list of instructions for each card, to be performed upon insertion or removal of the card.
The Configuration Manager is not needed when a CompactPCI hot swap aware driver is developed with the included Go-HotSwap driver development toolkit, other than for activating/stopping the Hot Swap Engine. In this case, the data from the Hot Swap Engine is transferred directly to the newly developed CompactPCI driver using the kernel-mode Messaging Mechanism. The Hot Swap Engine can also be activated by adding `WD_WatchPciStart` [A.1.3] to the hot swap aware driver developed with Go-HotSwap/WinDriver, instead of using the Configuration Manager.

NOTE:
The Configuration Manager is currently included only in the Windows versions. In the Linux and Solaris versions you may use **hs_activate**, which is the console version of this utility.

4.4.2 Configuration Manager Database

The Database can be viewed by opening the Configuration Manager screen.

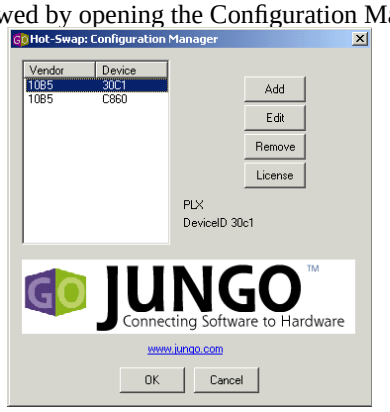


Figure 4.2: The Configuration Manager Database

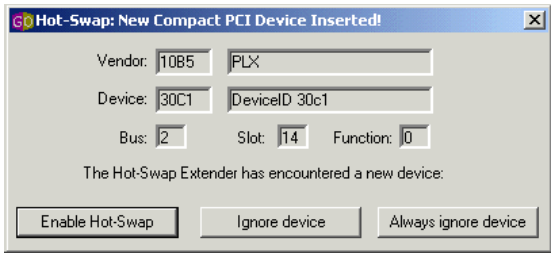
In order for the Configuration Manager to support a card's hot swap functionality, the card must be registered to the Configuration Manager's database.
The registration can be initiated manually by the user, otherwise it is automatically registered by Go-HotSwap, when a card that is not listed in the database is hot swapped.

Manual Registration Initiation

In order to manually add a board to the database, simply open the Go-HotSwap Configuration Manager's device configuration screen, press **Add** and fill in the relevant details in the appropriate fields: device ID, vendor ID and if necessary, bus, slot and function.

Automatic Registration Initiation

When a board that is not listed in the Configuration Manager's database is hot swapped, a pop-up message appears, asking the user whether to enable the board hot swap abilities, ignore the device or always ignore the device:

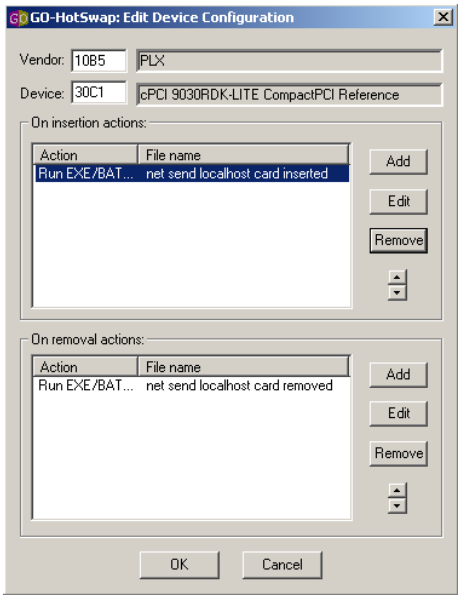


Any of the options will result in an automatic registration of the card to the Configuration Manager database.

4.4.3 Configuring the Configuration Manager

Go-HotSwap user-mode Configuration Manager enables the user to configure the system to perform certain actions upon card insertion and/or card removal.
The following actions are supported:

- Running a *batch* file (a file that contains a sequence of commands to be executed together)
- Running an *EXE* file (an executable file with an .EXE extension, file in a format that the computer can directly execute)
- Starting/stopping a service (this includes starting/stopping NT drivers)

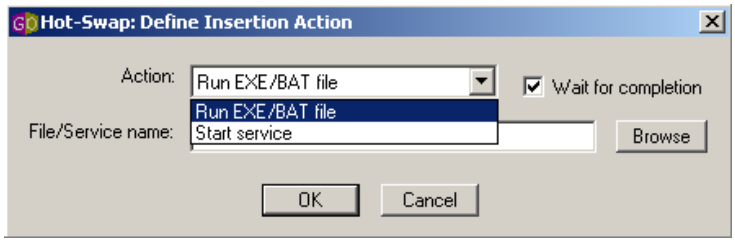


The Configuration Manager can be configured easily using the **Edit Device Configuration** screen to execute any of the above listed tasks, upon insertion or removal of the card.
In the **Edit Device Configuration** screen, you can easily add or remove tasks, and edit each one of the tasks as described later.
The Configuration Manager can be configured to run any of the tasks in parallel or serially (i.e. complete one task before starting the next one).

4.4.4 Running a Batch File

Open the Go-HotSwap Configuration Manager, press **Edit**, and the **Edit Device Configuration** screen will be displayed. Press **Add** to open the **Define Insertion/Removal Action** screen:

Specify the batch file name in the **File/Service** box along with any other parameters required, if any. The parameters must be entered as shown below:
<Batch File> #bus# #slot# #function# #vendor# #device#
Where:



- **<Batch File>** is the name of your batch file with or without the extension. If the absolute path of the file is not specified, then the batch file is assumed to be in the current directory.
- **#bus#** - instructs Go-HotSwap to send the bus number on which the board resides to the batch file.
- **#slot#** - instructs Go-HotSwap to send the appropriate slot number in which the board resides to the batch file.
- **#function#** - instructs Go-HotSwap to send the appropriate function number of the card to the batch file.
- **#vendor#** - instructs Go-HotSwap to send the relevant Vendor ID of the card to the batch file.
- **#device#** - instructs Go-HotSwap to send the relevant Device ID of the card to the batch file.

NOTE:
The parameters are meta parameters; do not replace them with real values. For example, if the name of the batch file is MyFile.bat and it requires the bus and function numbers to be sent to the batch file, then type exactly:
MyFile.bat #bus# #function#

4.4.5 Running an EXE File

An EXE file can also be run automatically during the card insertion and removal. The format of the parameters passed for the EXE file is the same as that for a batch file.

4.4.6 Starting and Stopping Services

In Windows 2000, Windows XP and Windows Server 2003, a service can be dynamically started and stopped using the following commands:

- `\> net start <driver name>`
- `\> net stop <driver name>`

4.4.7 Using Legacy Drivers on Windows 2000/XP/Server 2003

Generally legacy drivers are not hot swap aware, specifically they are incapable of detecting hot insertion or removal. Go-HotSwap adds hot swap awareness to PnP compliant legacy drivers. Go-HotSwap detects hot insertion/removal of all CompactPCI devices and uses the standard operating system's PnP support to notify the legacy driver of PnP events.

In order to use Go-HotSwap with legacy drivers, you must first install the legacy driver's INF file supplied by the vendor. Then activate the Go-HotSwap Configuration Manager, which starts the Hot Swap Engine. Once the INF is installed and the Hot Swap Engine is activated, the Windows PnP Manager will notify the legacy driver of PnP events.

Appendix A

Function Reference

A.1 Go-HotSwap

A.1.1 General

Add the Go-HotSwap functions described in this chapter to your code in order to make your application/driver hot swap aware and to utilize the Go-HotSwap services: register your driver or application to receive notifications and information about hot swap events (insertion/removal of your cPCI board), acknowledge receipt of the hot swap notification (to light the Blue LED) and perform re-enumeration according to user permissions.
This section describes the calling sequence used in **hs_activate**, but you may write a new application based on the given samples. You can refer to the **hs_detect**, **hs_activate** and **hs_reenum** utilities, (**\GoHotSwap\hotswap\...** - source code included) as samples for implementation of the Go-HotSwap API. You may also refer to the files **windrvr.h**, **windrvr_events.h** and **windrvr_int_thread.h** for API definitions.

NOTE:
This section describes the Go-HotSwap calling sequence and the Go-HotSwap API functions. In addition to the Go-HotSwap API functions, the Go-HotSwap calling sequence includes WinDriver API functions, which are described in detail in the WinDriver Function Reference.

A.1.2 Calling Sequence

(1) All Go-HotSwap (and WinDriver) functions should receive a HANDLE parameter as the first argument, which is a value returned from the WinDriver

function `WD_Open()`. Therefore this function must be called before any other Go-HotSwap or WinDriver function is called. For more information about the `WD_Open()` function, please refer to the WinDriver Function Reference.

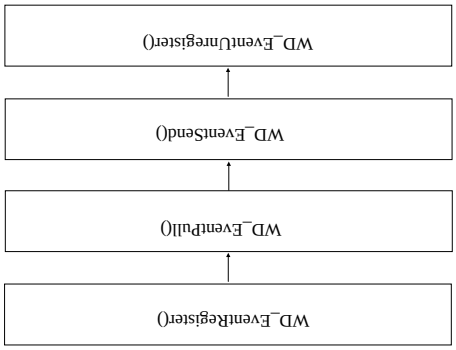
(2) We recommend you call the WinDriver `WD_Version()` function after calling `WD_Open()` and before calling any other Go-HotSwap/WinDriver function. Its purpose is to return the Go-HotSwap/WinDriver kernel module (windrvr6) version number, thus providing the means to verify that your application is version compatible with the Go-HotSwap/WinDriver kernel module.

(3) We recommend you use Go-HotSwap function wrappers in order to conveniently handle hot swap events.
The `EventRegister()` function wrapper calls a callback function upon hot swap events. It wraps the calls to the WinDriver `WD_EventRegister()`, `WD_EventPull()` and `WD_EventSend()` functions.
The `EventUnregister()` function wrapper wraps the calls to the WinDriver `WD_EventUnregister()` function.

(4) You must call `WD_WatchPciStart` to activate the Hot Swap Engine, and `WD_WatchPciStop` to stop the Hot Swap Engine.

The functions that compose the wrapper functions can be called separately instead of using the wrappers, according to the following sequence:

(5) For more details regarding the Go-HotSwap functions, please refer to the function reference below. For more details regarding the WinDriver functions, please refer to WinDriver user's guide (included in Go-HotSwap package).



A.1.3 WD_WatchPciStart()

PURPOSE

• Starts the Go-HotSwap Hot Swap Engine polling cycle, with the option to exclude devices from the polling cycle.

PROTOTYPE

```
void WD_WatchPciStart(HANDLE hWD, WD_HS_WATCH *pHsWatch)
```

PARAMETERS

Name	Type	Input/Output
➢ hWD	HANDLE	Input
➢ pHsWatch	WD_HS_WATCH *	
❑ handle	DWORD	Output
❑ dwOptions	DWORD	Input
❑ dwNumMatchTables	DWORD	Input
❑ matchTables	WD_HS_MATCH_TABLE	
❖ wVendorId	WORD	Input
❖ wDeviceId	WORD	Input
❖ wSubVendorId	WORD	Input
❖ wSubDeviceId	WORD	Input
❖ dwOptions	DWORD	Input

DESCRIPTION

hWD	Handle received from `WD_Open`.
pHsWatch	WD_HS_WATCH elements.
handle	To be used in WD_WatchPciStop() [A.1.4]. Returns INVALID_HANDLE_VALUE if the operation failed.
dwOptions	If 0, use the Go-HotSwap Hot Swap Engine for the hot-swapped board's resource allocation. If WD_CPCI_PNP_SUPPORT, use the operating system's Plug and Play support .
dwNumMatchTables	Number of match tables.
matchTables	WD_HS_MATCH_TABLE elements:
wVendorId	PCI vendor ID to exclude from polling cycle.
wDeviceId	PCI device ID to exclude from polling cycle.
wSubVendorId	PCI subvendor ID to exclude from polling cycle.
wSubDeviceId	PCI subdevice ID to exclude from polling cycle.

dwOptions	If 0, the Go-HotSwap Hot Swap Engine polls all devices. If WD_MATCH_EXCLUDE, the Go-HotSwap Hot Swap Engine polls all devices excluding the devices specified in the match table. You have the option of excluding specific devices or all devices of a specified vendor or subvendor, or any specified combination.

REMARKS

When the polling cycle is initiated by `WD_WatchPciStart()`, the Hot Swap Engine starts detecting hot swap events and dynamically allocates (or deallocates) the required resources accordingly. After calling `WD_WatchPciStart`, you may call the Go-HotSwap functions `WD_EventRegister` in order to register your application to receive notifications about hot swap events and WD_EventPull to retrieve additional information about the hot swap event.

A.1.4 WD_WatchPciStop()

PURPOSE

• Stops the Go-HotSwap Hot Swap Engine polling cycle.

PROTOTYPE

```
void WD_WatchPciStop(HANDLE hWD, WD_HS_WATCH *pHsWatch)
```

PARAMETERS

Name	Type	Input/Output
➢ hWD	HANDLE	Input
➢ pHsWatch	WD_HS_WATCH *	
❑ handle	DWORD	Input
❑ dwOptions	DWORD	Input
❑ dwNumMatchTables	DWORD	Input
❑ matchTables	WD_HS_MATCH_TABLE	
❖ wVendorId	WORD	Input
❖ wDeviceId	WORD	Input
❖ wSubVendorId	WORD	Input
❖ wSubDeviceId	WORD	Input
❖ dwOptions	DWORD	Input

DESCRIPTION

Name	Description
hWD	Handle received from `WD_Open`.
pHsWatch	WD_HS_WATCH elements.
handle	Returned handle from WD_WatchPciStart [A.1.3].
dwOptions	For future use.
dwNumMatchTables	Number of match tables.
matchTables	WD_HS_MATCH_TABLE elements. For details, refer to `WD_WatchPciStart` [A.1.3].

EXAMPLE

```
WD_WatchPciStart and WD_WatchPciStop Implementation

WD_HS_WATCH hWatch;
BZERO(hWatch);

WD_WatchPciStart(hWD, &hWatch);
if (!hWatch.handle)
{
   printf("Failed Watching CompactPCI bus\n");
   return -1;
}
// Use the rest of the Hot Swap API in this part of the code
// When completed, stop the Hot Swap Engine
if (hWatch.handle)
   WD_WatchPciStop(hWD, &hWatch);
```

A.2 WinDriver Functions Used by Go-HotSwap

A.2.1 WD_Open()

PURPOSE

• Opens a handle to access the WinDriver kernel module. The handle is used by all WinDriver APIs, and therefore must be called before any other WinDriver API is called.

PROTOTYPE

```
HANDLE WD_Open();
```

PARAMETERS

None

RETURN VALUE

The handle to the WinDriver kernel module.
If device could not be opened, returns INVALID_HANDLE_VALUE.

REMARKS

If you are a registered user, please refer to `WD_License()` function reference to see an example of how to register your license.

EXAMPLE

```
HANDLE hWD;

hWD = WD_Open();
if (hWD==INVALID_HANDLE_VALUE)
{
   printf("Cannot open WinDriver device\n");
}
```

A.2.2 WD_Version()

PURPOSE

• Returns the version number of the WinDriver kernel module currently running.

PROTOTYPE

```
DWORD WD_Version(HANDLE hWD, WD_VERSION *pVer);
```

PARAMETERS

Name	Type	Input/Output
➢ hWD	HANDLE	Input
➢ pVer	WD_VERSION *	
❑ dwVer	DWORD	Output
❑ cVer[100]	CHAR	Output

DESCRIPTION

Name	Description
hWD	The handle to WinDriver's kernel-mode driver received from `WD_Open` [A.2.1].
pVer	`WD_VERSION` elements:
dwVer	The version number.
cVer[100]	Version info string.

RETURN VALUE

Returns WD_STATUS_SUCCESS (0) on success, or an appropriate error code otherwise.

EXAMPLE

```
WD_VERSION ver;

BZERO(ver);
WD_Version(hWD, &ver);
printf("%s\n", ver.cVer)
if (ver.dwVer<WD_VER)
{
   printf("Error - incorrect WinDriver version\n");
}
```

A.2.3 WD_Close()

PURPOSE

• Closes the access to the WinDriver kernel module.

PROTOTYPE

```
void WD_Close(HANDLE hWD);
```

PARAMETERS

Name	Type	Input/Output
➢ hWD	HANDLE	Input

DESCRIPTION

Name	Description
hWD	The handle to WinDriver's kernel-mode driver received from `WD_Open` [A.2.1].

REMARKS

This function must be called when you finish using WinDriver kernel module.

EXAMPLE

```
WD_Close(hWD);
```

A.2.4 EventRegister()

PURPOSE

• Register your application to receive Plug and Play and power management event notifications, according to a predefined set of criteria, and call a callback function upon event receipt.

PROTOTYPE

```
DWORD EventRegister(HANDLE *phEvent, HANDLE hWD,
   WD_EVENT *pEvent, EVENT_HANDLER pFunc, void *pData);
```

PARAMETERS

Name	Type	Input/Output
➢ phEvent	HANDLE *	Output
➢ hWD	HANDLE	Input
➢ event	WD_EVENT *	Input
❑ handle	DWORD	Output
❑ dwAction	DWORD	Input
❑ dwStatus	DWORD	N/A
❑ dwEventId	DWORD	N/A
❑ dwCardType	WD_BUS_TYPE	Input
❑ hKernelPlugIn	DWORD	Input
❑ dwOptions	DWORD	Input
❑ u	union	
❖ Pci	struct	
✧ cardId	WD_PCI_ID	
♦ dwVendorId	DWORD	Input
♦ dwDeviceId	DWORD	Input
✧ pciSlot	WD_PCI_SLOT	
♦ dwBus	DWORD	Input
♦ dwSlot	DWORD	Input
♦ dwFunction	DWORD	Input
❖ Usb	struct	
✧ deviceId	WD_USB_ID	
♦ dwVendorId	DWORD	Input

♦ dwProductId	DWORD	Input
✧ dwUniqueID	DWORD	Input
❖ Pcmcia	struct	
✧ deviceId	WD_PCMCIA_ID	
♦ wManufacturerId	WORD	Input
♦ wCardId	WORD	Input
✧ pcmciaSlot	WD_PCMCIA_SLOT	
♦ uBus	BYTE	Input
♦ uSocket	BYTE	Input
♦ uFunction	BYTE	Input
♦ uPadding	BYTE	N/A
➢ func	EVENT_HANDLER	Input
➢ data	void	Input
➢ dwEventVer	DWORD	Internal use
➢ dwNumMatchTables	DWORD	Input
➢ matchTables[1]	WDU_MATCH_TABLE	Input

DESCRIPTION

Name	Description
phEvent	If successful, phEvent will hold the handle to be used in `EventUnregister` [A.2.5].
hWD	The handle to WinDriver's kernel-mode driver received from `WD_Open` [A.2.1].
event	The criteria set for registering to receive event notifications.
handle	Optional handle to be used by `WD_EventUnregister`. Returns 0 when event registration fails.

dwAction	A bit mask field indicating which events to register to. **Plug and Play events:** • WD_INSERT - Device inserted • WD_REMOVE - Device removed **Device power state:** • WD_POWER_CHANGED_D0 - Full power • WD_POWER_CHANGED_D1 - Low sleep • WD_POWER_CHANGED_D2 - Medium sleep • WD_POWER_CHANGED_D3 - Full sleep • WD_POWER_SYSTEM_WORKING - Fully on **Systems power state:** • WD_POWER_SYSTEM_SLEEPING1 - Fully on but sleeping • WD_POWER_SYSTEM_SLEEPING2 - CPU off, memory on, PCI/PCMCIA on • WD_POWER_SYSTEM_SLEEPING3 - CPU off, Memory is in refresh, PCI/PCMCIA on aux power • WD_POWER_SYSTEM_HIBERNATE - OS saves context before shutdown • WD_POWER_SYSTEM_SHUTDOWN - No context saved
dwCardType	Can be either WD_BUS_PCI, WD_BUS_USB or WD_BUS_PCMCIA (from the WD_BUS_TYPE options).
hKernelPlugIn	Handle to Kernel PlugIn returned from `WD_KernelPlugInOpen` (when using the Kernel PlugIn to handle the events).
dwOptions	Can be either WD_ACKNOWLEDGE or zero. If WD_ACKNOWLEDGE, the user can perform actions on the requested event before acknowledging it. The OS waits on the event until the user calls WD_EventSend(). If the EventRegister() [A.2.4] wrapper is called, WD_EventSend() [A.3.5] will be called automatically after the callback function exits.
cardId.dwVendorId	PCI Vendor ID to register to. If zero, register to all PCI vendor ID's.
cardId.dwDeviceId	PCI Device ID to register to. If zero, register to all PCI Device ID's.

pciSlot.dwBus	PCI bus number to register to. If zero, register to all PCI buses.
pciSlot.dwSlot	PCI slot to register to. If zero, register to all slots.
pciSlot.dwFunction	PCI function (on the device) to register to. If zero, registers to all functions.
deviceId.dwVendorId	USB Vendor ID to register to. If zero, register to all USB vendor ID's.
deviceId.dwProductId	USB Product ID to register to. If zero, register to all USB Product ID's.
dwUniqueID	Unique ID of the USB device to register to. If zero, register to all unique ID's.
deviceId.wManufacturerId	PCMCIA Manufacturer ID to register to. If zero, register to all PCMCIA manufacturer ID's.
deviceId.wCardId	PCMCIA card ID to register to. If zero, register to all PCMCIA card ID's.
pcmciaSlot.uBus	PCMCIA bus number to register to. If zero, register to all PCMCIA buses.
pcmciaSlot.uSocket	PCMCIA socket to register to. If zero, register to all sockets.
pcmciaSlot.uFunction	PCMCIA function (on the card) to register to. If zero, registers to all functions.
pcmciaSlot.uPadding	1 byte padding (reserved).
func	The callback function to call upon receipt of event notification.
data	The data to pass to the callback function.
dwEventVer	For internal use only.
dwNumMatchTables	For USB only. (*) **NOTE:** For USB it is recommended to use WDU_Init()/WDU_Uninit() (see USB function reference, instead of directly using EventRegister()/EventUnregister().
matchTables[1]	For USB only; See NOTE above (*).

RETURN VALUE

Returns WD_STATUS_SUCCESS (0) on success, or an appropriate error code otherwise.

REMARKS

This function wraps `WD_EventRegister`, `WD_EventPull`, `WD_EventSend` [A.3.5] and `InterruptEnable`.

EXAMPLE

```
HANDLE *event_handle;
WD_EVENT event;
DWORD dwStatus;
BZERO(event);
event.dwAction = WD_INSERT | WD_REMOVE;
event.dwCardType = WD_BUS_PCI;
dwStatus = EventRegister(&event_handle, hWD, &event,
   event_handler_func, NULL);
if (dwStatus!=WD_STATUS_SUCCESS)
{
   printf("Failed register\n");
   return;
}
```

A.2.5 EventUnregister()

PURPOSE

• Un-registers from receiving Plug and Play and power management event notifications.

PROTOTYPE

```
DWORD EventUnregister(HANDLE hEvent);
```

PARAMETERS

Name	Type	Input/Output
➢ hEvent	HANDLE *	Input

DESCRIPTION

Name	Description
hEvent	Handle received from `EventRegister` [A.2.4].

RETURN VALUE

Returns WD_STATUS_SUCCESS (0) on success, or an appropriate error code otherwise.

REMARKS

This function wraps `WD_EventUnregister` and `InterruptDisable`.

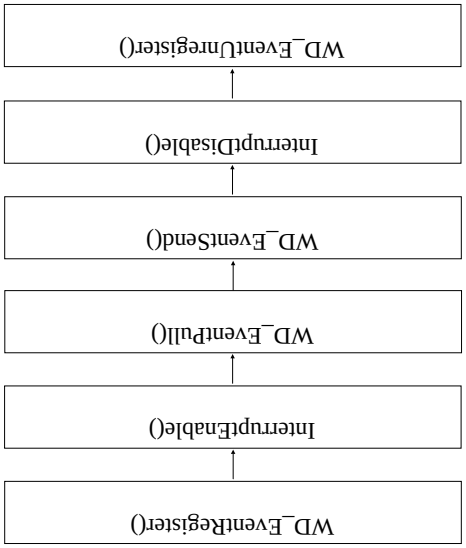
EXAMPLE

```
EventUnregister(event_handle);
```

A.3 Plug and Play and Power Management - Low Level Functions

A.3.1 Calling Sequence

The following is a typical calling sequence of the WinDriver API, used for handling Plug and Play and power management events. We recommend that you use `EventRegister` [A.2.4] and `EventUnregister` [A.2.5], or `WDU_Init` and `WDU_Uninit` (for USB), instead of these low level functions, in order to handle Plug and Play and power management events in a more convenient manner.



A.3.2 WD_EventRegister()

PURPOSE

• Register your application to receive Plug and Play and power management event notifications, according to a predefined set of criteria.

PROTOTYPE

```
DWORD WD_EventRegister(HANDLE hWD, WD_EVENT *pEvent);
```

PARAMETERS

Name	Type	Input/Output
➢ hWD	HANDLE	Input
➢ pEvent	WD_EVENT *	
❑ handle	DWORD	Output
❑ dwAction	DWORD	Input
❑ dwStatus	DWORD	N/A
❑ dwEventId	DWORD	N/A
❑ dwCardType	WD_BUS_TYPE	Input
❑ hKernelPlugIn	DWORD	Input
❑ dwOptions	DWORD	Input
❑ u	union	
❖ Pci	struct	
✧ cardId	WD_PCI_ID	
♦ dwVendorId	DWORD	Input
♦ dwDeviceId	DWORD	Input
✧ pciSlot	WD_PCI_SLOT	
♦ dwBus	DWORD	Input
♦ dwSlot	DWORD	Input
♦ dwFunction	DWORD	Input
❖ Usb	struct	
✧ deviceId	WD_USB_ID	
♦ dwVendorId	DWORD	Input
♦ dwProductId	DWORD	Input
✧ dwUniqueID	DWORD	Input

✥ Pcmcia	struct	
✧ deviceId	WD_PCMCIA_ID	
✦ wManufacturerId	WORD	Input
✦ wCardId	WORD	Input
✧ pcmciaSlot	WD_PCMCIA_SLOT	
✦ uBus	BYTE	Input
✦ uSocket	BYTE	Input
✦ uFunction	BYTE	Input
✦ uPadding	BYTE	N/A

DESCRIPTION

Name	**Description**
hWD	The handle to WinDriver's kernel-mode driver received from `WD_Open` [A.2.1].
pEvent	WD_EVENT elements:
handle	Handle to be used by `WD_EventUnregister` [A.3.3]. Returns 0 when event registration fails.
dwAction	A bit mask field indicating which events to register to. **Plug and Play events:** • WD_INSERT - Device inserted • WD_REMOVE - Device removed **Device power state:** • WD_POWER_CHANGED_D0 - Full power • WD_POWER_CHANGED_D1 - Low sleep • WD_POWER_CHANGED_D2 - Medium sleep • WD_POWER_CHANGED_D3 - Full sleep • WD_POWER_SYSTEM_WORKING - Fully on **Systems power state:** • WD_POWER_SYSTEM_SLEEPING1 - Fully on but sleeping • WD_POWER_SYSTEM_SLEEPING2 - CPU off, memory on, PCI/PCMCIA on • WD_POWER_SYSTEM_SLEEPING3 - CPU off, Memory is in refresh, PCI/PCMCIA on aux power • WD_POWER_SYSTEM_HIBERNATE - OS saves context before shutdown • WD_POWER_SYSTEM_SHUTDOWN - No context saved

dwCardType	Can be either WD_BUS_PCI, WD_BUS_USB or WD_BUS_PCMCIA (from the WD_BUS_TYPE options).
hKernelPlugIn	Optional handle to Kernel PlugIn returned from `WD_KernelPlugInOpen`.
dwOptions	Can be either WD_ACKNOWLEDGE or zero.
dwVendorId	PCI Vendor ID to register to. If zero, register to all PCI vendor ID's.
dwDeviceId	PCI Device ID to register to. If zero, register to all PCI Device ID's.
dwBus	PCI bus number to register to. If zero, register to all PCI buses.
dwSlot	PCI slot to register to. If zero, register to all slots.
dwFunction	PCI function (on the device) to register to. If zero, registers to all functions.
dwVendorId	USB Vendor ID to register to. If zero, register to all USB vendor ID's.
dwProductId	USB Product ID to register to. If zero, register to all USB Product ID's.
dwUniqueID	Unique ID of the USB device to register to. If zero, register to all unique ID's.
wManufacturerId	PCMCIA Manufacturer ID to register to. If zero, register to all PCMCIA manufacturer ID's.
wCardId	PCMCIA card ID to register to. If zero, register to all PCMCIA card ID's.
uBus	PCMCIA bus number to register to. If zero, register to all PCMCIA buses.
uSocket	PCMCIA socket to register to. If zero, register to all sockets.
uFunction	PCMCIA function (on the card) to register to. If zero, registers to all functions.
uPadding	1 byte padding (reserved).

RETURN VALUE

Returns WD_STATUS_SUCCESS (0) on success, or an appropriate error code otherwise.

REMARKS

In order to receive the desired notifications you must also call `InterruptEnable`. When the callback function sent to `InterruptEnable` is called it means that a new event has occurred.

NOTE:
If WD_ACKNOWLEDGE is set in the dwOptions field, you must call `WD_EventPull` [A.3.4] and `WD_EventSend` [A.3.5] to acknowledge the event in order to allow the system to handle the event normally. If you do not call `WD_EventPull` and `WD_EventSend`, the system might hang while waiting for your application to acknowledge the event.

EXAMPLE

```
WD_EVENT Event;
BZERO(Event);
Event.dwAction = WD_INSERT | WD_REMOVE;
Event.dwCardType = WD_BUS_PCI;
WD_EventRegister(hWD, &Event);
if (Event.handle)
   printf("successfully registered to receive Plug and Play events\n");
else
   printf("WD_EventRegister failed\n");
```

A.3.3 WD_EventUnregister()

PURPOSE

• Un-registers from receiving Plug and Play and power management events notifications.

PROTOTYPE

```
DWORD WD_EventUnregister(HANDLE hWD, WD_EVENT *pEvent);
```

PARAMETERS

Name	Type	Input/Output
➢ hWD	HANDLE	Input
➢ pEvent	WD_EVENT *	
❑ handle	DWORD	Input
❑ dwAction	DWORD	N/A
❑ dwStatus	DWORD	N/A
❑ dwEventId	DWORD	N/A
❑ dwCardType	WD_BUS_TYPE	N/A
❑ hKernelPlugIn	DWORD	N/A
❑ dwOptions	DWORD	N/A
❑ u	union	
❖ Pci	struct	
✧ cardId	WD_PCI_ID	
♦ dwVendorId	DWORD	N/A
♦ dwDeviceId	DWORD	N/A
✧ pciSlot	WD_PCI_SLOT	
♦ dwBus	DWORD	N/A
♦ dwSlot	DWORD	N/A
♦ dwFunction	DWORD	N/A
❖ Usb	struct	
✧ deviceId	WD_USB_ID	
♦ dwVendorId	DWORD	N/A
♦ dwProductId	DWORD	N/A
✧ dwUniqueID	DWORD	N/A
❖ Pcmcia	struct	

✧ deviceId	WD_PCMCIA_ID	
♦ wManufacturerId	N/A	Input
♦ wCardId	WORD	N/A
✧ pcmciaSlot	WD_PCMCIA_SLOT	
♦ uBus	BYTE	N/A
♦ uSocket	BYTE	N/A
♦ uFunction	BYTE	N/A
♦ uPadding	BYTE	N/A

DESCRIPTION

Name	Description
hWD	The handle to WinDriver's kernel-mode driver received from `WD_Open` [A.2.1].
pEvent	WD_EVENT elements:
handle	Handle received by `WD_EventRegister` [A.3.2].

RETURN VALUE

Returns WD_STATUS_SUCCESS (0) on success, or an appropriate error code otherwise.

EXAMPLE

```
WD_EVENT Event;
BZERO(Event);
Event.handle = handle;
WD_EventUnregister(hWD, &Event);
```

A.3.4 WD_EventPull()

PURPOSE

• Retrieves information regarding a Plug and Play or power management event that occurred.

PROTOTYPE

```
DWORD WD_EventPull(HANDLE hWD, WD_EVENT *pEvent);
```

PARAMETERS

Name	Type	Input/Output
➢ hWD	HANDLE	Input
➢ pEvent	WD_EVENT *	
❑ handle	DWORD	Input
❑ dwAction	DWORD	Output
❑ dwStatus	DWORD	N/A
❑ dwEventId	DWORD	Output
❑ dwCardType	WD_BUS_TYPE	Output
❑ hKernelPlugIn	DWORD	N/A
❑ dwOptions	DWORD	Output
❑ u	union	
❖ Pci	struct	
✧ cardId	WD_PCI_ID	
♦ dwVendorId	DWORD	Output
♦ dwDeviceId	DWORD	Output
✧ pciSlot	WD_PCI_SLOT	
♦ dwBus	DWORD	Output
♦ dwSlot	DWORD	Output
♦ dwFunction	DWORD	Output
❖ Usb	struct	
✧ deviceId	WD_USB_ID	
♦ dwVendorId	DWORD	Output
♦ dwProductId	DWORD	Output
✧ dwUniqueID	DWORD	Output
❖ Pcmcia	struct	

✧ deviceId	WD_PCMCIA_ID	
♦ wManufacturerId	WORD	Output
♦ wCardId	WORD	Output
✧ pcmciaSlot	WD_PCMCIA_SLOT	
♦ uBus	BYTE	Output
♦ uSocket	BYTE	Output
♦ uFunction	BYTE	Output
♦ uPadding	BYTE	N/A

DESCRIPTION

Name	Description
hWD	The handle to WinDriver's kernel-mode driver received from `WD_Open` [A.2.1].
pEvent	WD_EVENT elements:
handle	Handle received from `WD_EventRegister` [A.3.2].
dwAction	A bit mask field indicating which events to register to. **Plug and Play events:** • WD_INSERT - Device inserted • WD_REMOVE - Device removed **Device power state:** • WD_POWER_CHANGED_D0 - Full power • WD_POWER_CHANGED_D1 - Low sleep • WD_POWER_CHANGED_D2 - Medium sleep • WD_POWER_CHANGED_D3 - Full sleep • WD_POWER_SYSTEM_WORKING - Fully on **Systems power state:** • WD_POWER_SYSTEM_SLEEPING1 - Fully on but sleeping • WD_POWER_SYSTEM_SLEEPING2 - CPU off, memory on, PCI/PCMCIA on • WD_POWER_SYSTEM_SLEEPING3 - CPU off, Memory is in refresh, PCI/PCMCIA on aux power • WD_POWER_SYSTEM_HIBERNATE - OS saves context before shutdown • WD_POWER_SYSTEM_SHUTDOWN - No context saved
dwEventId	An ID to identify the event in the complementary `WD_EventSend` [A.3.5] function.

dwCardType	Can be either WD_BUS_PCI, WD_BUS_USB or WD_BUS_PCMCIA (from the WD_BUS_TYPE options).
dwOptions	Return WD_ACKNOWLEDGE if it was used in `WD_EventRegister` [A.3.2].
dwVendorId	PCI Vendor ID.
dwDeviceId	PCI Device ID.
dwBus	PCI bus number.
dwSlot	PCI slot.
dwFunction	PCI function (on the device).
dwVendorId	USB Vendor ID.
dwProductId	USB Product ID.
dwUniqueID	Unique ID of the USB device.
wManufacturerId	PCMCIA Manufacturer ID.
wCardId	PCMCIA type and model ID.
uBus	PCMCIA bus number.
uSocket	PCMCIA socket.
uFunction	PCMCIA function.
uPadding	1 byte padding (reserved).

RETURN VALUE

Returns WD_STATUS_SUCCESS (0) on success, or an appropriate error code otherwise.

REMARKS

Your application should call `WD_EventPull` [A.3.4], after receiving an event notification, in order to retrieve additional information identifying the event. (For example: your application can register to receive a notification about every Plug and Play or power management event that occurs, and after receiving a notification, it can retrieve the exact details of the event i.e., insertion/removal, vendor ID, device ID, etc.).

EXAMPLE

```
WD_EVENT Event;
BZERO(Event);
Event.handle = handle;
WD_EventPull(hWD, &Event);
if (Event.dwCardType==WD_BUS_PCI)
{
   printf("got PCI event %d.%d.%d vid %04x/%04x action 0x%x\n",
      Event.u.Pci.pciSlot.dwBus, Event.u.Pci.pciSlot.dwSlot,
      Event.u.Pci.pciSlot.dwFunction, Event.u.Pci.cardId.dwVendorId,
      Event.u.Pci.cardId.dwDeviceId, Event.dwAction);
}
else if (Event.dwCardType==WD_BUS_USB)
{
   printf("got USB event unique %x vid %04x/%04x action 0x%x\n",
      Event.u.Usb.dwUniqueID, Event.u.Usb.deviceId.dwVendorId,
      Event.u.Usb.deviceId.dwProductId, Event.dwAction);
}
else if (Event.dwCardType==WD_BUS_PCMCIA)
{
   printf("got PCMCIA event %d.%d.%d vid %04x/%04x action 0x%x\n",
      Event.u.Pcmcia.pcmciaSlot.uBus,
      Event.u.Pcmcia.pcmciaSlot.uSocket,
      Event.u.Pcmcia.pcmciaSlot.uFunction,
      Event.u.Pcmcia.deviceId.wManufacturerId,
      Event.u.Pcmcia.deviceId.wCardId, Event.dwAction);
}
```

A.3.5 WD_EventSend()

PURPOSE

• Acknowledges a Plug and Play or power management event.

PROTOTYPE

```
DWORD WD_EventSend(HANDLE hWD, WD_EVENT *pEvent);
```

PARAMETERS

Name	Type	Input/Output
➢ hWD	HANDLE	Input
➢ pEvent	WD_EVENT *	
❑ handle	DWORD	Input
❑ dwAction	DWORD	Input
❑ dwStatus	DWORD	N/A
❑ dwEventId	DWORD	Input
❑ dwCardType	WD_BUS_TYPE	N/A
❑ hKernelPlugIn	DWORD	N/A
❑ dwOptions	DWORD	Input
❑ u	union	
❖ Pci	struct	
✧ cardId	WD_PCI_ID	
♦ dwVendorId	DWORD	N/A
♦ dwDeviceId	DWORD	N/A
✧ pciSlot	WD_PCI_SLOT	
♦ dwBus	DWORD	N/A
♦ dwSlot	DWORD	N/A
♦ dwFunction	DWORD	N/A
❖ Usb	struct	
✧ deviceId	WD_USB_ID	
♦ dwVendorId	DWORD	N/A
♦ dwProductId	DWORD	N/A
✧ dwUniqueID	DWORD	N/A
❖ Pcmcia	struct	
✧ deviceId	WD_PCMCIA_ID	

♦ wManufacturerId	WORD	N/A
♦ wCardId	WORD	N/A
✧ pcmciaSlot	WD_PCMCIA_SLOT	
♦ uBus	BYTE	N/A
♦ uSocket	BYTE	N/A
♦ uFunction	BYTE	N/A
♦ uPadding	BYTE	N/A

DESCRIPTION

Name	Description
hWD	The handle to WinDriver's kernel-mode driver received from `WD_Open` [A.2.1].
pEvent	WD_EVENT elements:
handle	Handle to be used by `WD_EventUnregister` [A.3.3]. Returns zero when event registration fails.
dwEventId	Event ID received from `WD_EventPull` [A.3.4].
dwOptions	Should be WD_ACKNOWLEDGE.

RETURN VALUE

Returns WD_STATUS_SUCCESS (0) on success, or an appropriate error code otherwise.

REMARKS

You must use `WD_EventSend` [A.3.5] to acknowledge Plug and Play or power management events, if you registered to receive notifications of such events with the WD_ACKNOWLEDGE flag set in `WD_EventRegister` [A.3.2].

EXAMPLE

```
WD_EVENT Event;
BZERO(Event);
Event.handle = handle;
WD_EventPull(hWD, &Event);
if (Event.dwOptions & WD_ACKNOWLEDGE)
   WD_EventSend(hWD, &Event);
```

Appendix B

CompactPCI Hot Swap Overview

B.1 What is CompactPCI?

CompactPCI is an adaptation of the PCI specification for industrial and/or embedded applications requiring a more robust mechanical form factor then desktop PCI. CompactPCI uses industry standard mechanical components and high performance connector technologies to provide an optimized system intended for rugged applications. CompactPCI provides a system that is electrically compatible with the PCI specification, allowing low cost PCI components to be utilized in a mechanical form factor suited for rugged environments.
CompactPCI is an open specification supported by the PICMG (PCI Industrial Computer Manufacturers Group), which is a consortium of companies involved in utilizing PCI for embedded applications.
CompactPCI exists to provide a standard form factor for those applications requiring the high performance of PCI as well as the small size and ruggedness of a rack mount system. CompactPCI provides the mechanism to directly apply PCI components and technology to a new mechanical form factor while maintaining capability with existing operating systems and application software available for desktop PCI.

B.2 CompactPCI Features

The following list illustrates the main features of CompactPCI:

- 33 and 66 MHz PCI performance
- 32- and 64-bit data transfers
- 8 CompactPCI slots per bus segment at 33 MHz
- 5 CompactPCI slots per bus segment at 66 MHz
- Industry standard software support
- 3U and 6U form factor
- IEEE (1101.1, 1101.10, and 1101.11) Eurocard packaging
- Wide variety of available I/O
- System management bus
- Hot swap support

A CompactPCI system is composed of one or more CompactPCI bus segments. Each segment is composed of up to eight CompactPCI slots (at 33MHz). Each CompactPCI segment bus consists of one System slot, and up to seven Peripheral slots. Physically the system's slot can be located at any position in the backplane.
The Peripheral slots may contain simple boards, intelligent slaves, or PCI bus masters.

B.3 Why CompactPCI?

CompactPCI is attractive to industrial applications, such as high-speed telecom and datacom systems for 3 reasons:

1. CompactPCI has an extremely high bandwidth; this makes it particularly well suited for applications such as servers, routers, converters and switches.
2. CompactPCI boards are inserted/removed from the front of the chassis, and I/O can break out either from the front or through the rear; this feature is well suited for industrial use where insertion/removal is a common routine.
3. The Hot Swap feature supported by the CompactPCI hardware specifications makes it possible to insert or remove circuit boards to or from a running system without causing damage to the other boards or crashing system software, thus enabling the system to be up and running while hardware is being replaced, maintained or upgraded.

B.4 Hot Swap

General

Hot swap is the ability to insert and remove boards, without adversely affecting a running system. CompactPCI Hot Swap Specification allows vendors to add hot swap capabilities to CompactPCI system.
The hot swap feature that has been established in the CompactPCI Specification is particularly well suited, and has indeed been a critical requirement, for application designs in the telecom and datacom markets. In systems that do not support hot swapping of peripheral boards, each process of board insertion or removal, requires a complete shut-down of the entire system until the process is complete, in order to prevent damage to other boards or to the system software. In time critical systems such as telecom and datacom systems, system downtime is a major financial and quality of service factor; any downtime translates into financial loss, and disconnecting service to active lines. Hence, reduction or even elimination of system downtime is critical. The hot swap facility reduces the system downtime, which may be required for replacement of boards and making changes in the system configuration. The hot swap feature is one of the key reasons why CompactPCI architecture is so attractive for telecom and datacom vendors.

CompactPCI hot swap is not limited to peripheral boards, but at this time the CompactPCI Hot Swap Specification (PICMG 2.1 R1.0) does not address hot swapping of the System Board Computer (SBC).

CompactPCI Boards (other than the system host) can be one of three types:

- **Non Hot Swap:** Boards that do not have all the features required for hot swap. They cannot be inserted or extracted from an operating CompactPCI system.
- **Basic Hot Swap:** Boards that have the minimum features required for hot swap.
- **Full Hot Swap:** Boards that have the minimum features plus the additional resources for software connection control. Full hot swap boards allow the full range of system capabilities.

NOTE:
Go-HotSwap architecture allows it to fully support both **basic** and **full** hot swap boards.

CompactPCI Silicon can be one of four types:

- **Non Hot Swap:** The silicon is not compatible with the minimal requirements of the CompactPCI Hot Swap Specification.
- **Hot Swap Capable:** Silicon that contains the minimum requirements to operate in a hot swap environment. These minimum requirements are attributes that a board designer would not be able to compensate for with external circuitry.
- **Hot Swap Friendly:** Silicon that contains support for software connection control (as defined in PICMG 2.1 R1.0). Namely providing a support for a Control and Status Register in the PCI configuration space, and providing resources for ENUM#, hot swap Switch and the Blue Led.
- **Hot Swap Ready:** Silicon that contains all the desirable features for hot swap; which are all the features supported by hot swap friendly silicon, plus Bias Voltage Support, Early Power Support, and 64 Bit Initialization. These chips add the pre-charge and early power support, so that no external circuitry is required, except for the onboard power control.

NOTE:
Go-HotSwap architecture, allows it to fully support hot swap **capable**, **friendly** and **ready** silicon.

Enumeration and Dynamic Configuration

Enumeration is defined by the CompactPCI Specification (PICMG 2.1 R1.0) as the action taken by the system host to poll the configuration spaces of all the PCI devices and assign (or release) the necessary resources (memory and/or I/O address space, interrupts, and software drivers). In systems that do not support hot swap, enumeration only takes place as the system boots. In hot swap capable systems, dynamic configuration is required, whereby a hot swap board is allocated system resources (enumerated) by the system software, following insertion of the board. These same resources are released prior to extraction of the board.
In order to take advantage of the hardware's hot swap capability, a substantial amount of additional software is required - at the device driver level, at the system services level and at the applications level.

The Software Aspect

The software aspect of the hot swap process involves detection of the hot swap event, dynamic configuration and resource allocation. It also involves informing the system of the event so that the system and applications running are not damaged, and the insertion/removal event is transparent to the system user.
While many hardware products are already electrically and mechanically hot swap compatible, until now little software existed that allowed utilization of the new capabilities. Mass-market operating systems, such as Windows 98, Me, NT, 2000, XP, Server 2003, Solaris (until version 8), Linux and VxWorks, currently lack the additional software layers needed to support the new hot swap capabilities. The absence of such software standards has retarded market growth, and is forcing companies either to give up utilization of hot swap capabilities, or to develop/buy proprietary solutions. Either way, the situation is a considerable resource consumer.
Another problem involves the hot swap standard itself, which is somewhat shady in some of the software areas. The gray areas in the specification have resulted in different implementations of some of the hot swap mechanisms provided by the system board manufacturers. As a result, a CompactPCI hardware vendor that desires to ship hot swap aware drivers with the hardware needs to develop and provide a different driver for each different system board needed to be supported.
In order to provide a complete hot swap aware solution, the hardware vendor needs to either use proprietary operating systems, or implement the hot swap mechanism from

scratch, and separately, for each different system hardware that needs to be supported as well as for each different operating system that needs to be supported.

PHONE / FAX

Phone:
USA (Toll-Free): 1-877-514-0537
Worldwide: +972-9-885-9365
Fax: USA (Toll-Free): 1-877-514-0538
Worldwide: +972-9-885-9366

E-MAIL

Support: support@jungo.com
Sales: sales@jungo.com

WEB SITE

`http://www.jungo.com`

ADDRESS

Jungo Ltd.
P.O.B 8493
Netanya, 42504
ISRAEL