RapidDriver Manual

Table of Contents

1 What's new?

RapidDriver 2.0, new features and changes:

- The user interface was completely redesigned to be clean, modern, and visually appealing.
- Now RapidDriver manipulates with the "Devices" instead of "Projects".
- Serial port support was added.
- Added macro support.
- Added the ability to run examples directly from the GUI.
- The interrupt handling technique was improved.
- Many small bugs was fixed.

Caution: ***Windows 98 and Windows ME are obsolete and no more supported in RapidDriver!***

2 RapidDriver

RapidDriver

Ver. 2.0

http://www.RapidDriver.com

tools@entechtaiwan.com

2.1 Introduction

The RapidDriver is a tool that was created to support newly developed hardware devices without being the DDK expert. RapidDriver provides advanced support to a wide range of ISA, PCI, USB, and

Parallel Port devices under Windows 2000, XP, and 2003.

Caution: ***Windows 98 and Windows ME are obsolete and no more supported in RapidDriver!***

Three editions of the RapidDriver toolkit are available:

- Test and debug your hardware with ***RapidDriver Explorer***
- Manage hardware from your application with ***RapidDriver Developer***
- Build your own device driver with ***RapidDriver Source Builder***
- Take advantages of additional debugging features of built-in ***RapidDebugger***

RapidDriver Explorer

With RapidDriver Explorer you may do work with entire PC as a single "device", or with the separate PCI, USB, ISA/PC-104, or Parallel Port device. Simply install your new hardware, create a new RapidDriver "device project", and then select your device from the list of PnP devices that are auto-detected by the RapidDriver application. RapidDriver extracts all required hardware resource information directly from the device, leaving you free to begin developing and testing your device specifics immediately. For PCI, ISA and LPT devices, you can read/write hardware ports and memory-mapped registers, and listen for interrupts. For USB devices, you can obtain all the USB descriptors, perform read/write operations on "bulk" or "interrupt" pipes, and more. RapidDriver even allows you to describe and test non-PnP ISA or PC-104 devices.

RapidDriver Developer

The ***Developer Edition*** of RapidDriver includes all the features of the ***Explorer***, and additionally allows you to create and distribute your own applications incorporating integrated ISA/PCI/USB/LPT drivers and DLLs to control your hardware.

RapidDriver Source Builder

The ***Source Builder Edition*** of RapidDriver includes all the features of both the ***Explorer*** and the ***Developer Editions***, plus the ability to generate the source code to a fully-featured driver for your device which has to conform to the Windows Driver Model (WDM). The resulting source code can be directly edited and compiled with MS Visual C/C++ or with the DDK "build" utility.

2.2 Installation and System Requirements

Installation

To install RapidDriver just run the installation program and follow on-screen instructions.

Disk space requirement for RapidDriver is about 6 MBytes.

System Requirements

To execute the RapidDriver application, you must be running one of following operating systems:

- Windows 2000
- Windows XP
- Windows 2003

To build the drivers, you should have installed the Microsoft Driver Development Toolkits (DDK). You

can order DDK CD directly from Microsoft: http://www.microsoft.com/ddk. It is free with the small shipping&handling fee . The Windows DDK includes various tools that are useful in developing drivers, including build utilities (*build.exe* and *nmake.exe*), a C/C++ compiler (*cl.exe*), and a linker (*link.exe*). The Build utility automatically invokes NMAKE, the compiler, and the linker according to the command-line options you specify. But you can edit and build your drivers by much more comfortable way if you have installed the Microsoft Visual C/C++ compiler.

To see debug output from your drivers you can use free **DbgMon** utility from Open System Resources, Inc. Visit http://www.osronline.com to download this very useful software.

2.3 License Agreement

RapidDriver License Agreement
Copyright (c) 2005
EnTech Taiwan

EnTech Taiwan is willing to license the accompanying software to you only if you accept all of the terms in this license agreement. please read the terms carefully before you install the software, because by installing the software you are agreeing to be bound by the terms of this agreement. if you do not agree to these terms, ***EnTech Taiwan*** will not license this software to you, and in that case you should immediately delete all copies of this software you have in any form.

1. Ownership of the software

1.1. The enclosed ***EnTech Taiwan RapidDriver*** program ("***Software***") and the accompanying written materials are owned by ***EnTech Taiwan*** or its suppliers and are protected by international laws.

2. Grant of license

2.1. ***EnTech Taiwan*** grants to you as an individual, a personal, nonexclusive "one-user" license to use the ***Software*** on a single computer in the manner provided below at the site for which the license was given. If you are an entity, ***EnTech Taiwan*** grants you the right to designate one individual within your organization to have the right to use the ***Software*** on a single computer in the manner provided below at the site for which the license was given.

2.2. If you have not yet purchased a license to the ***Software***, ***EnTech Taiwan*** grants to you the right to use the ***Software*** for an evaluation period of 30 days. If you wish to continue using the ***Software*** and accompanying written materials after the evaluation period, you must register the ***Software*** by sending the required payment to ***EnTech Taiwan***. For the payment details please visit http://www.entechtaiwan.com/store.shtm

3. Restrictions on use and transfer

3.1. You may not distribute any of the headers, source files, or libraries which are included in the ***Software*** package.

3.2. RapidDriver may not be used to develop a development product, an API, or any products which will eventually be part of a development product or environment, without the written consent of the ***EnTech Taiwan***.

3.3. You may make printed copies of the written materials accompanying ***Software*** provided that they used only by users bound by this license.

3.4. You may not distribute or transfer your licensed ***Software***.

3.5. You may not rent or lease the ***Software*** or otherwise transfer or assign the right to use the ***Software***.

3.6. You may not reverse engineer, decompile, or disassemble the ***Software***.

4. Disclaimer of warranty

4.1. This ***Software*** and its accompanying written materials are provided by ***EnTech Taiwan*** "AS IS" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, are disclaimed.

4.2. In no event shall ***EnTech Taiwan*** its suppliers be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, savings, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this ***Software***, even if advised of the possibility of such damage.

If you have any questions concerning this Agreement or wish to contact ***EnTech Taiwan*** for any reason, please write e-mail at tools@entechtaiwan.com.

EnTech Taiwan (c) 2005
Web site: http://www.RapidDriver.com
E-mail: tools@entechtaiwan.com

2.4 Ordering and Contact Information

RapidDriver maintained at the following internet address:

http://www.RapidDriver.com , where you can find the latest release and links to support and on-line ordering sites.

To order an appropriate ***RapidDriver*** edition and license through one of our resellers please visit RapidDriver ordering page.

Comments, questions and suggestions regarding ***RapidDriver*** can be directed by e-mail to the RapidDriver support group.

RapidDriver toolkit is constantly being updated and changed. Your feedback is very important for us for continual improvement of this product. If you need additional features on some areas not covered in RapidDriver, simply let us know. Please help us improve our toolkit to meet your needs!

2.5 Evaluation Version Limitations

- On each RapidDriver Explorer load/unload nag screen appears which cannot be closed for 5 seconds.
- Only one device of each hardware type (USB, PCI, ISA, LPT) can be installed at the same time.

- Evaluation period is 30 days from the date of installation.
- You must keep RapidDriver application running in background to evaluate the RapidDriver Developer functionality.

3 RapidDriver Explorer

RapidDriver Explorer

Ver. 2.0

http://www.RapidDriver.com

tools@entechtaiwan.com

3.1 Introduction

RapidDriver Explorer is intelligent tool for PC hardware analysis and debugging. RapidDriver Explorer allows you to start work with hardware after a couple of clicks. You do not have to perform any additional steps (write kernel mode driver and create test applications). Everything is already done!

Simply install your new hardware, create a new RapidDriver project, and then select your device from the list of PnP devices that are auto-detected by the RapidDriver application. RapidDriver extracts all required hardware resource information directly from the device, leaving you free to begin developing and testing your device specifics immediately.

For PCI, ISA and LPT devices, you can read/write hardware ports and memory-mapped registers, and listen for interrupts. For USB devices, you can issue USB-specific requests , perform read/write operations on "bulk" or "interrupt" pipes, and more. RapidDriver even allows you to describe and test non-PnP ISA or PC-104 devices.

The process of working with any hardware device includes several steps:

1. Create new RapidDriver device project or open an existing one from the list.

2. Install the driver for ISA, PCI, LPT, or USB device.

3. Test your device with help of GUI interface or with the built-in RapidDebugger

4. Run a sample application (**Menu | Examples | Run Example**).

Before begin your work with a device, you create a new RapidDriver device project. This is a file with the extension ".rdp" and is the source for all RapidDriver output. A RapidDriver project is a small text file, that looks like standard Windows Ini-file. The project file will be saved in the projects folder, which is set by default to the <RapidDriver>\Projects directory. You can change a project folder by selecting of the **Project | Change Working Folder...** main menu alternative or in process of creating new RapidDriver device project.

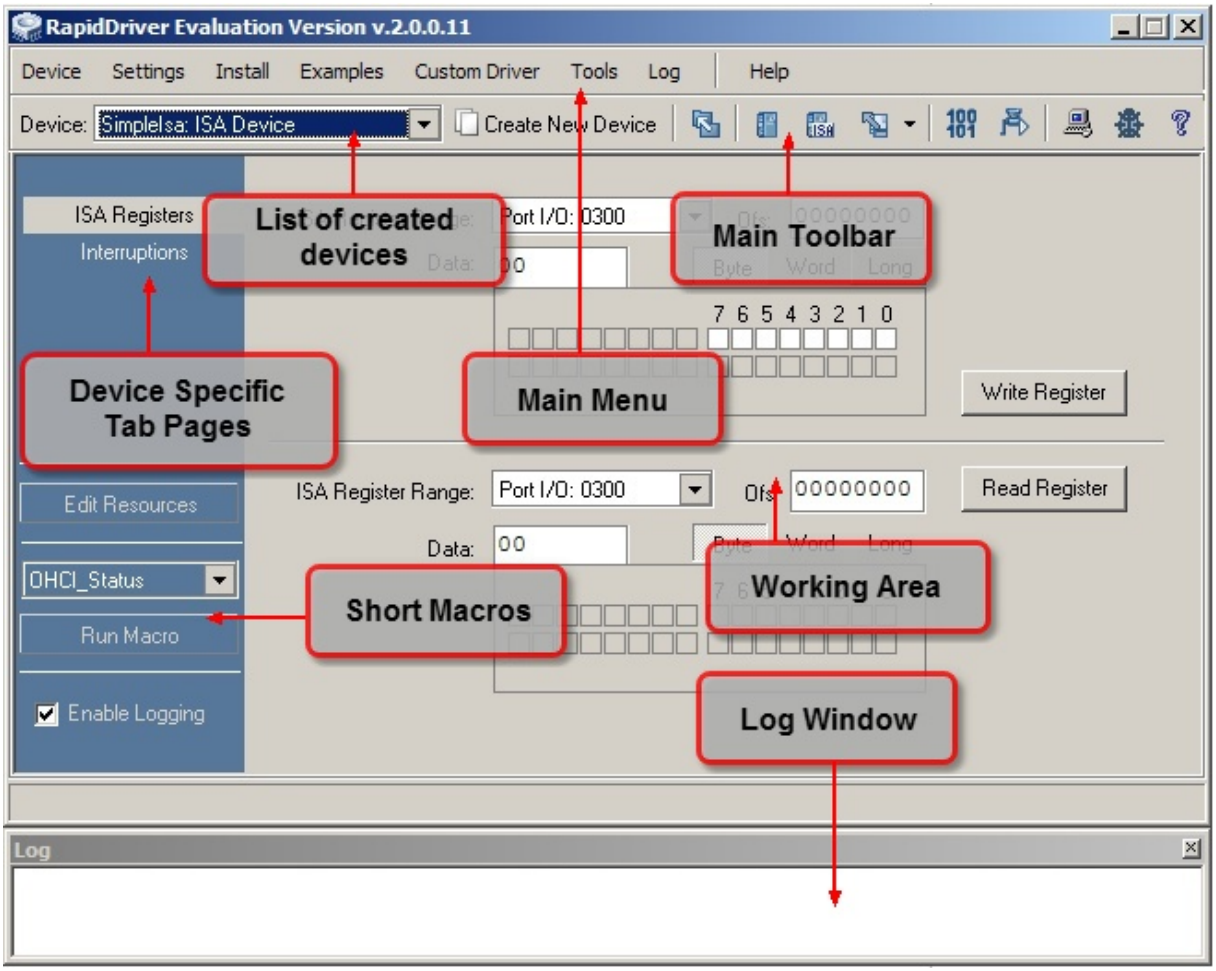
You can create a new device project from scratch for a legacy ISA/PC-104 device, or import all hardware resources from an existing PnP device that are auto-detected by the RapidDriver. The RapidDriver can not extract the hardware resources from a non-PnP ISA or PC-104 device, so you should describe all hardware resources for this kind of devices "manually".

3.2 Getting Started

3.2.1 The User Interface (Overview)

The user interface of RapidDriver Explorer is neat and intuitive (see picture). When you start RapidDriver, you are immediately placed within the integrated development environment (IDE). This IDE provides all the tools you need to test and debug your device by many ways:

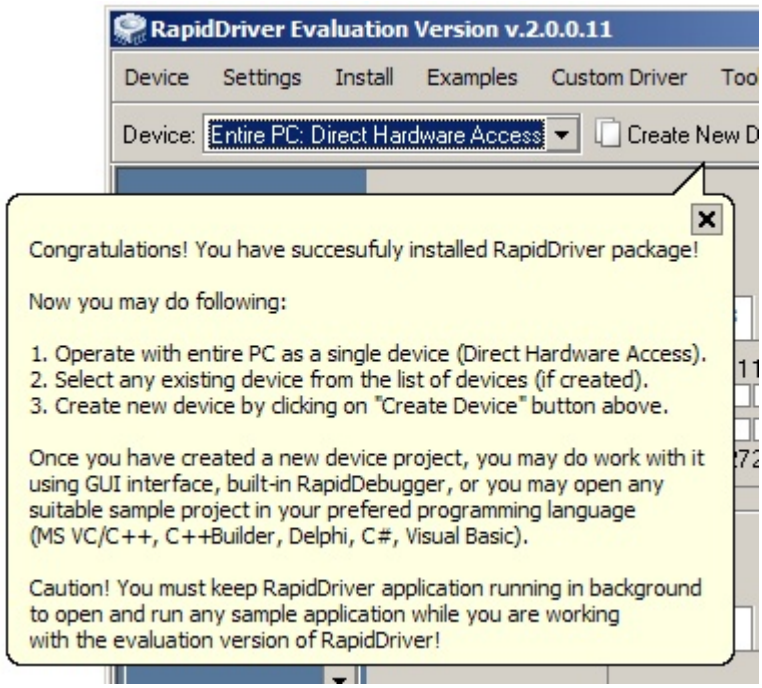
- directly from GUI interface;
- you may create then run a short macro;
- with the help of built-in command-line and scripting debugger (see RapidDebugger)



The main dialog window that appears will vary depending on the type of device selected as well as on the RapidDriver edition.

3.2.2 What to Do After Installing RapidDriver?

When you run RapidDriver for the first time, you will get a "Congratulations...' splash screen:



I. If you are new to RapidDriver, we recommend you to begin your tour from the "***Entire PC: Direct Hardware Access***" pseudo-device. In this case you can operate with the PC hardware without been "linked" to a specific device. Read more...

II. As a next step, you may create new device project - simply click the "New Device" toolbar button, then follow all on-screen instructions. After the driver have been successfully installed, your device should appear under the "RapidDriver Devices" class branch in Device Manager (clicking on the toolbar button will open the Device Manager window):

Now you may do work with your device:

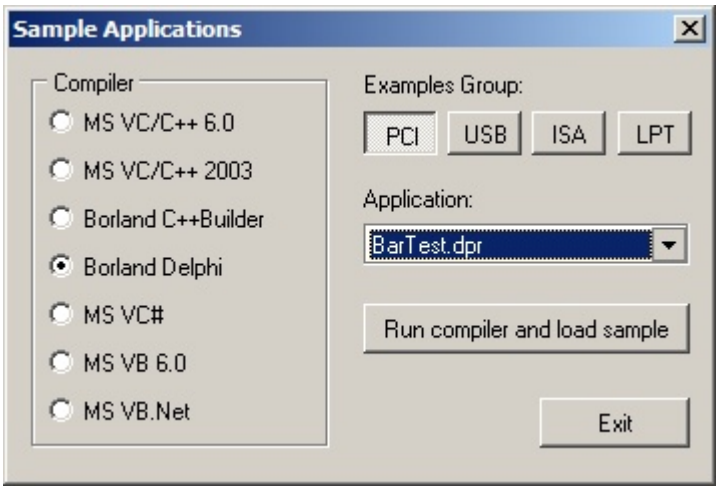
- directly from GUI interface;

- open and run any test example in your preferred programming environment (MS VC/C++, BC++Builder, Delphi, C#, VB, VB.Net).

- with the help of built-in command-line and scripting debugger (see RapidDebugger)

3.2.3 Running Samples

RapidDriver includes a lot of various samples written for various programming languages. All sample projects are located in <RapiDriver>\Examples directory. You may open the sample project directly from the GUI: **Menu | Examples | Run Example** or click the corresponding toolbar button.
The "Sample Applications" dialog will appear:

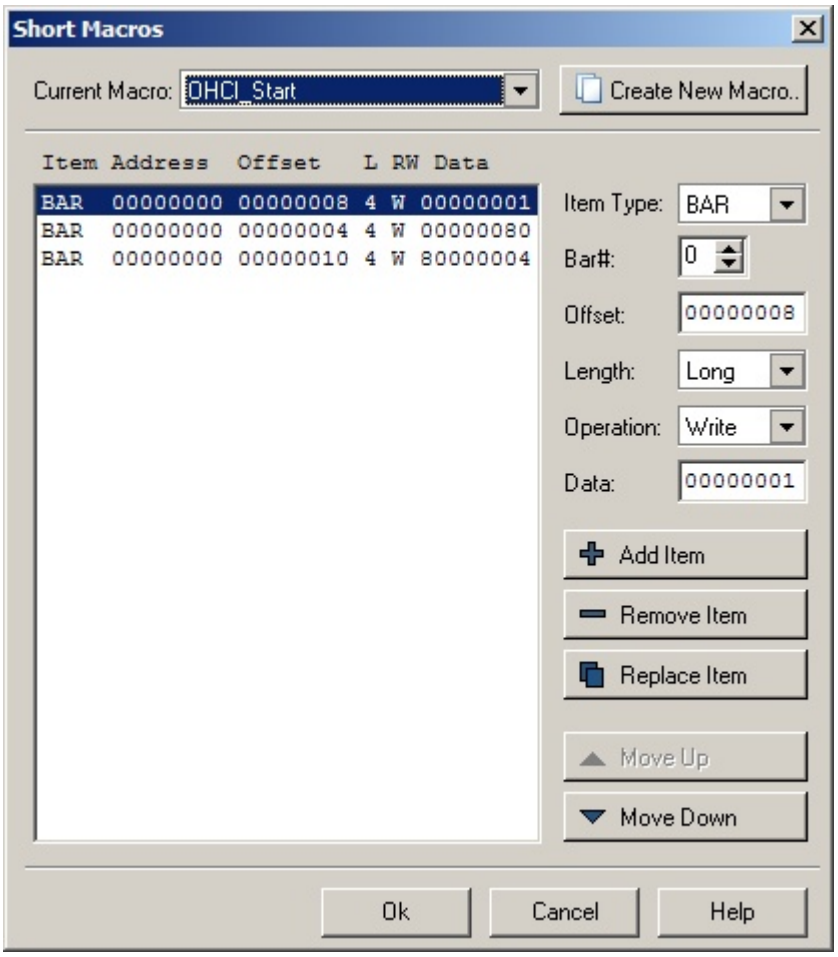


You should select your preferred compiler, the group (PCI, USB, ISA, or LPT), and the application, then click the "Run compiler and load sample" button.

Be noticed that you must keep RapidDriver application running in background if you have an evaluation version of RapidDriver!

3.2.4 Creating and Using Short Macros

A "macro" is a way of automating tasks, particularly repetitive tasks. If you are debugging a device, defining a macro-sequence of simple I/O operations is the easiest and most convenient method, simplifying your work. The creation of a macro is a relatively simple process: select the **Settings | Short Macros** menu item. The "Short Macros" dialog will appear:



Click the "Create New Macro" button then enter the name of macro.

To create every macro item, you should select:

- macro item type (port, memory, or PCI BAR),
- port or memory address, or BAR number
- address offset,
- length of operation (Byte, Word, or Long),
- type of i/o operation (read/write),
- data for the "write" operation.

After that, you should click the "Add Item" button. You may edit every item, delete it, move up and down.

Later you may select and run a macro from the practically any device project by clicking on the "Run Macro" button:

3.3 New RapidDriver Device Project

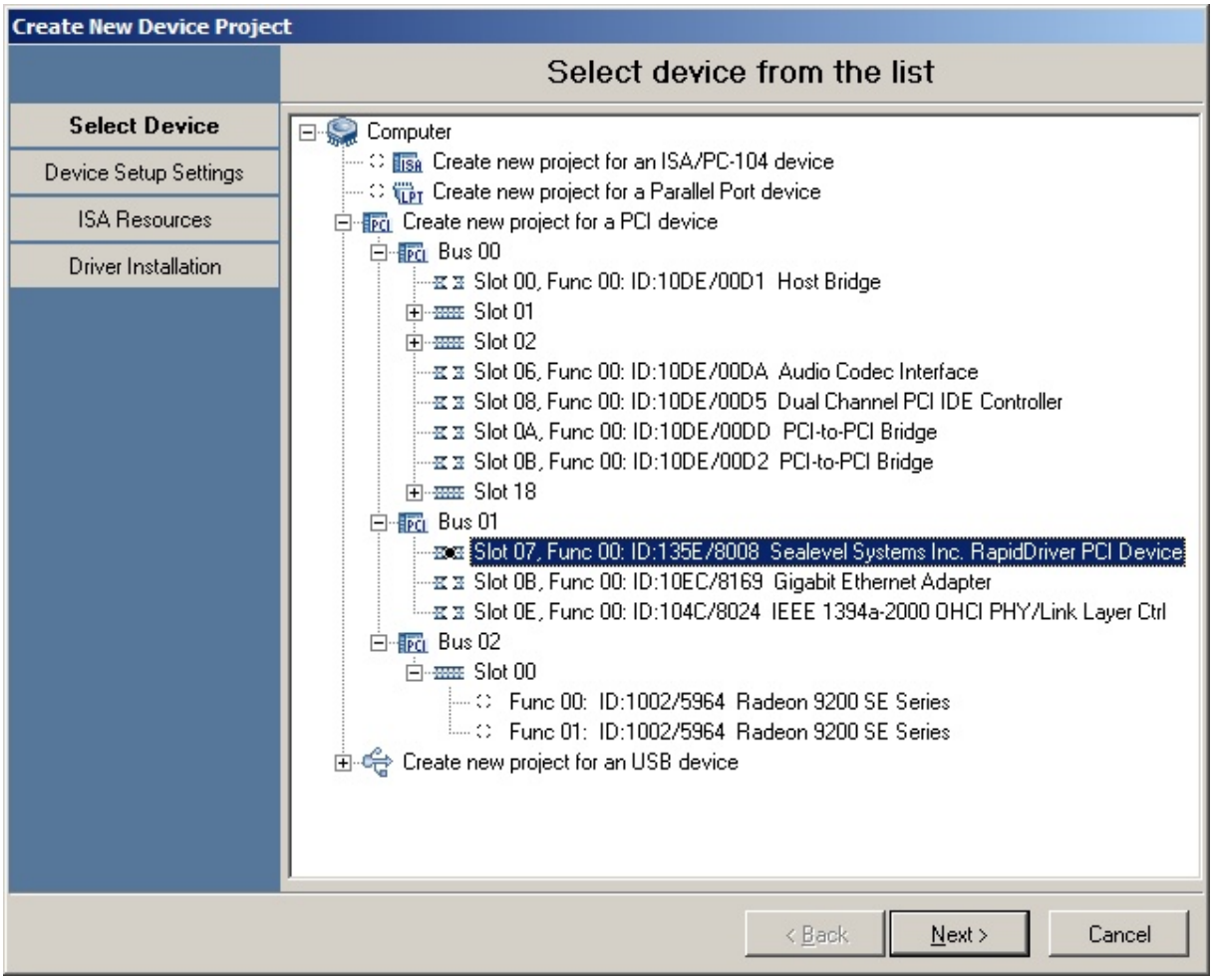
Caution! ***To prevent your system from being inoperative, you should not even try to create new device project for the system devices like videoadapter or mainboard devices! If this is really necessary, try to work with these devices from the "Entire PC: Direct Hardware Access" device project, or with the help of RapidDebugger.***

To create a new RapidDriver device project open New Device Project Wizard window by clicking on the "Create Device" toolbar button or select menu item **Device | Create New Device...**. The wizard will walk you through the basic steps of creating new deivce:

- Step 1: Select Device
- Step 2: Device Setup Settings
- Step 3: ISA Resources
- Step 4: Driver Installation

3.3.1 Step 1: Select Device

At this step you must choose a project type and/or device from the list:



You will then be given the opportunity to specify the project type as follows:

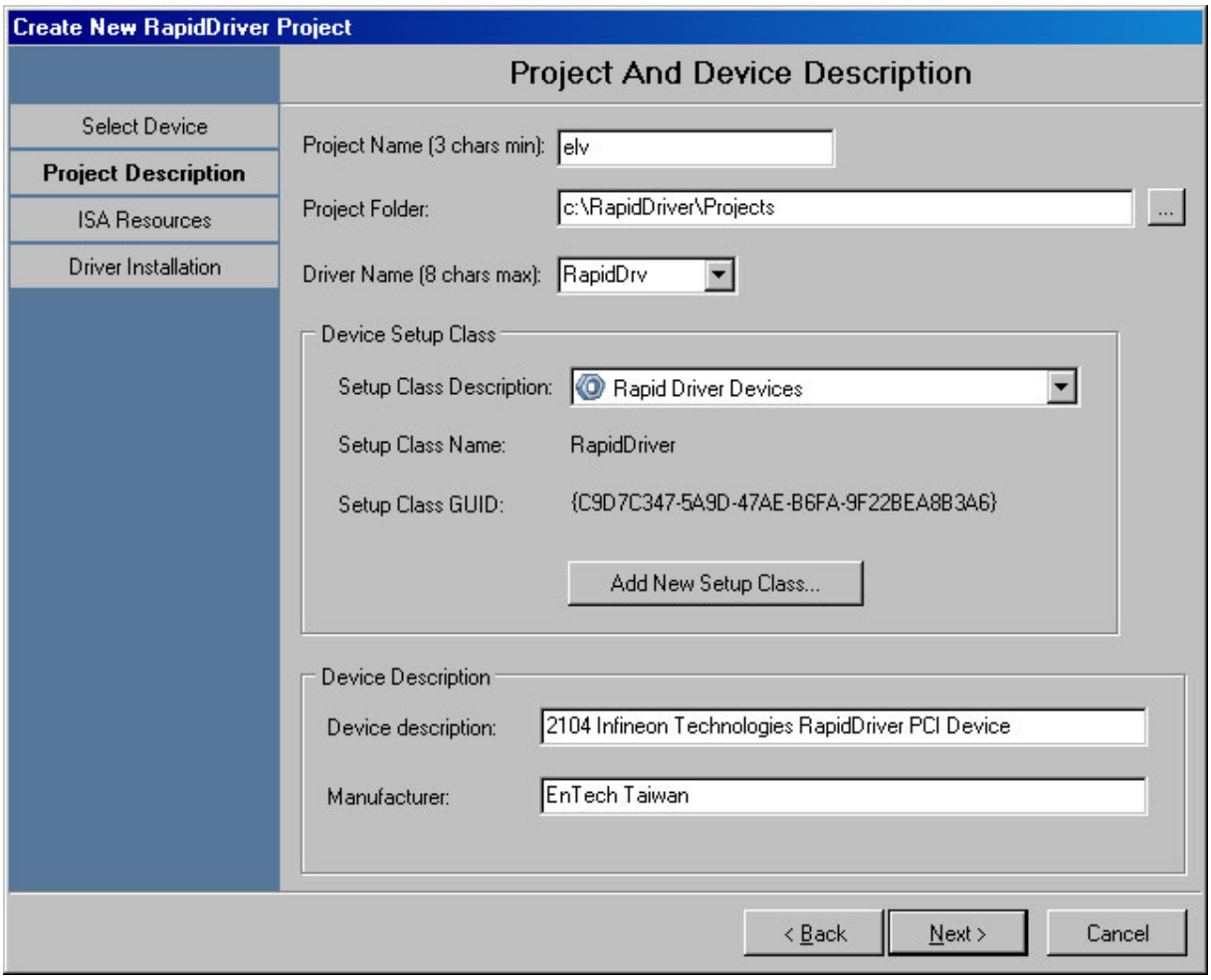
- Create new project for a legacy ISA/PC-104 device
- Create new project for a Parallel Port device
- Create new project for a PCI device (click on '+' to expand the PCI devices branch then select a device from the list)
- Create new project for an USB device (click on '+' to expand the USB devices branch then select a device from the list)

3.3.2 Step 2: Device Setup Settings

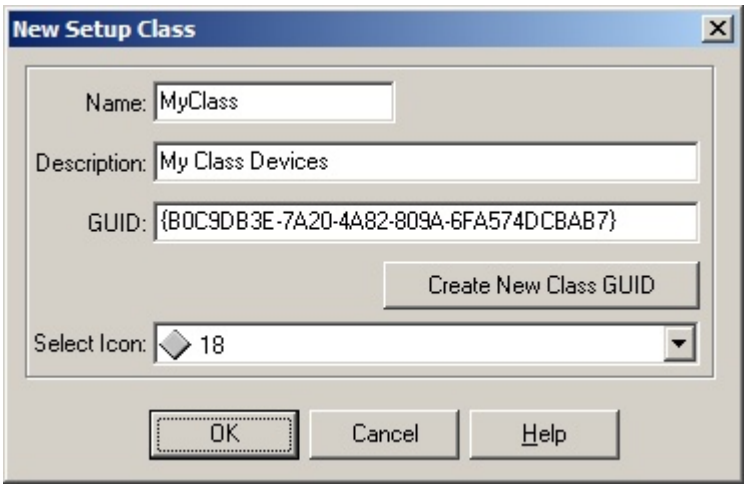
First of all, let's have a look at the process of installing a new device into the computer that operates under Windows and supports Plug&Play. To do this, apart from the driver you need a so-called INF-file that contains all the necessary information about the hardware and WDM driver being installed, including a list of files to be copied, description of keys created in the registry, etc.

It goes without saying that the above-mentioned INF-file should be created in advance, as well as all other files that it points to (the driver and any optional files). The RapidDriver creates an INF-file in "silent" mode when the **Install | Install Device using ...** menu item have been selected. You should provide all necessary installation information you install the driver for your new device. This information usually has been created during the new RapidDriver device project process. But you can edit this information later (Menu **Settings | Setup Settings**).

This page asks you for basic information about the project and device you want to work with:



All entries on this page are self documented, so all you need to do is to fill in the necessary fields with information about your company and also about the class of your device. For the **RapidDriver Explorer** edition this class should be always "RapidDriver Devices". With the **RapidDriver Developer or Source Builder** editions you can select another installation classes as well. The only situation that requires special comments is the one when a device doesn't belong to any of the enumerated classes. In this case you should click the "Add New Setup Class..." button, which shows dialog intended for new setup class creation. In the "New Setup Class" dialog supply name, description and index of predefined icon of the new device class, and give it a unique identifier (GUID):



After pressing OK new setup class is added to the system registry and from this moment can be easily used by your device driver.

- Project Name

Type a unique project name here. The project name must be at least 3 characters long.

- Working Folder

A current working folder location for the new device project is supplied automatically, but you can change the drive and path. You also can change the current working folder location by selecting of **Device | Change Working Folder**... main menu item.

- Driver Name

Type or select a driver name. This name must be at least 3 characters long, but not more than 8 characters.
Note: The driver name is always "RapidDrv" for the RapidDriver Explorer. For the RapidDriver Developer you can also select one of the following:

- RapidLpt for a Parallel Port device,
- RapidIsa for an ISA/PC1-4 device,
- RapidPci for a PCI device,
- RapidUsb for a USB device.

You are free to choose any driver name if you are working with the RapidDriver Source Builder.

- Setup Class Name

Select the standard device class to which your device belongs, or click "Add New Setup Class" and create your own setup class. The setup class name is always "*RapidDriver Devices"* for the RapidDriver Explorer and Developer editions.

- Device Description

Insert the description of your device here. This description will appear in Device Manager after you install a device:

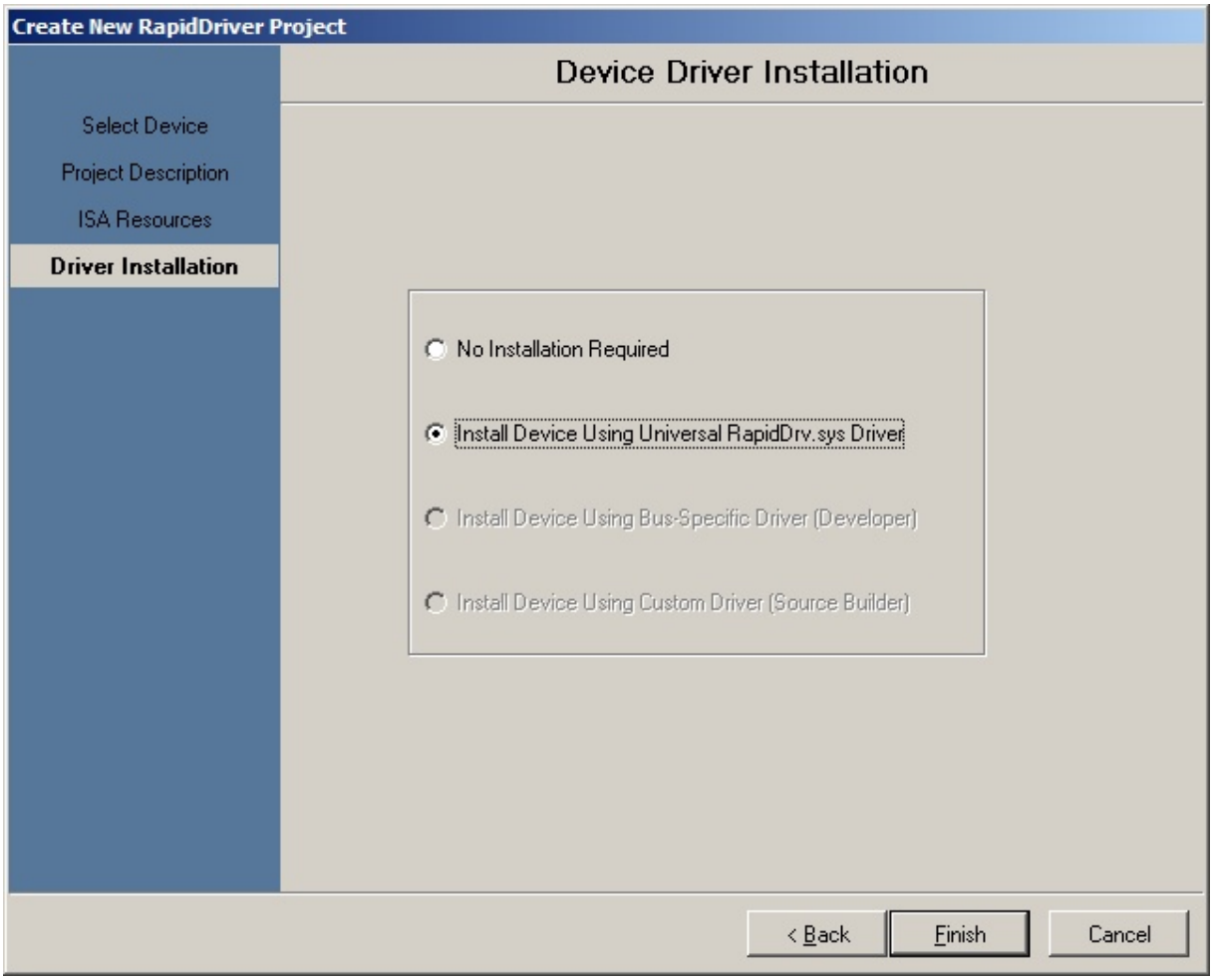
- Manufacturer Name

Name of the hardware manufacturer.

3.3.3 Step 3: ISA Resources

This step is specific for the ISA/PC-104 devices and will be discussed later.

3.3.4 Step 4: Driver Installation



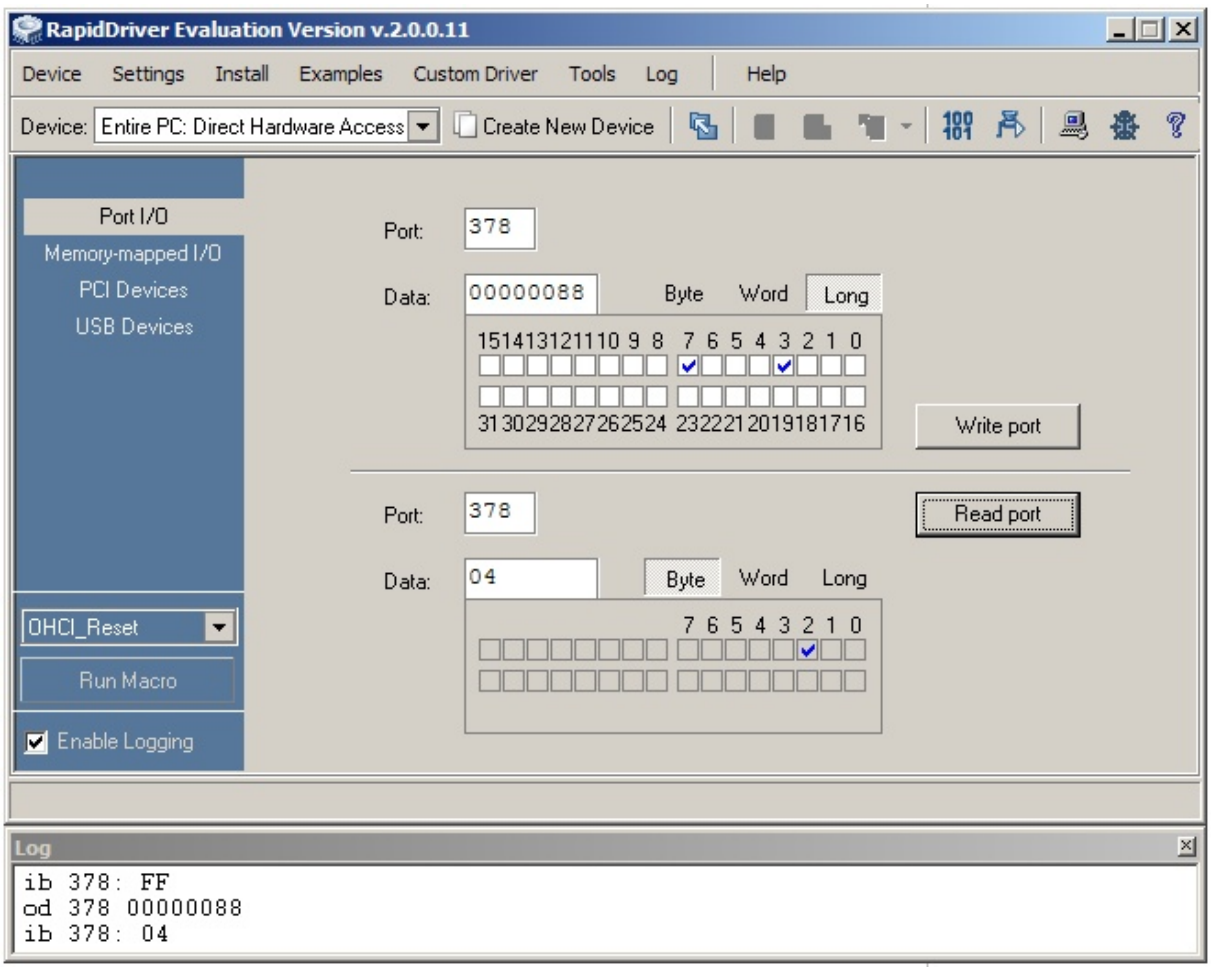
Select a driver which should be installed after you press the "Finish" button. You may also select the "No installation required" alternative and install the driver later by pressing of the toolbar button, or by selecting of **Install | Install using..** main menu item**.**

3.4 Entire PC - Direct Hardware Access

- direct access to any port of your PC
- read/write the physical memory addresses (memory-mapped registers)
- accessing PCI devices
- limited accessing USB devices

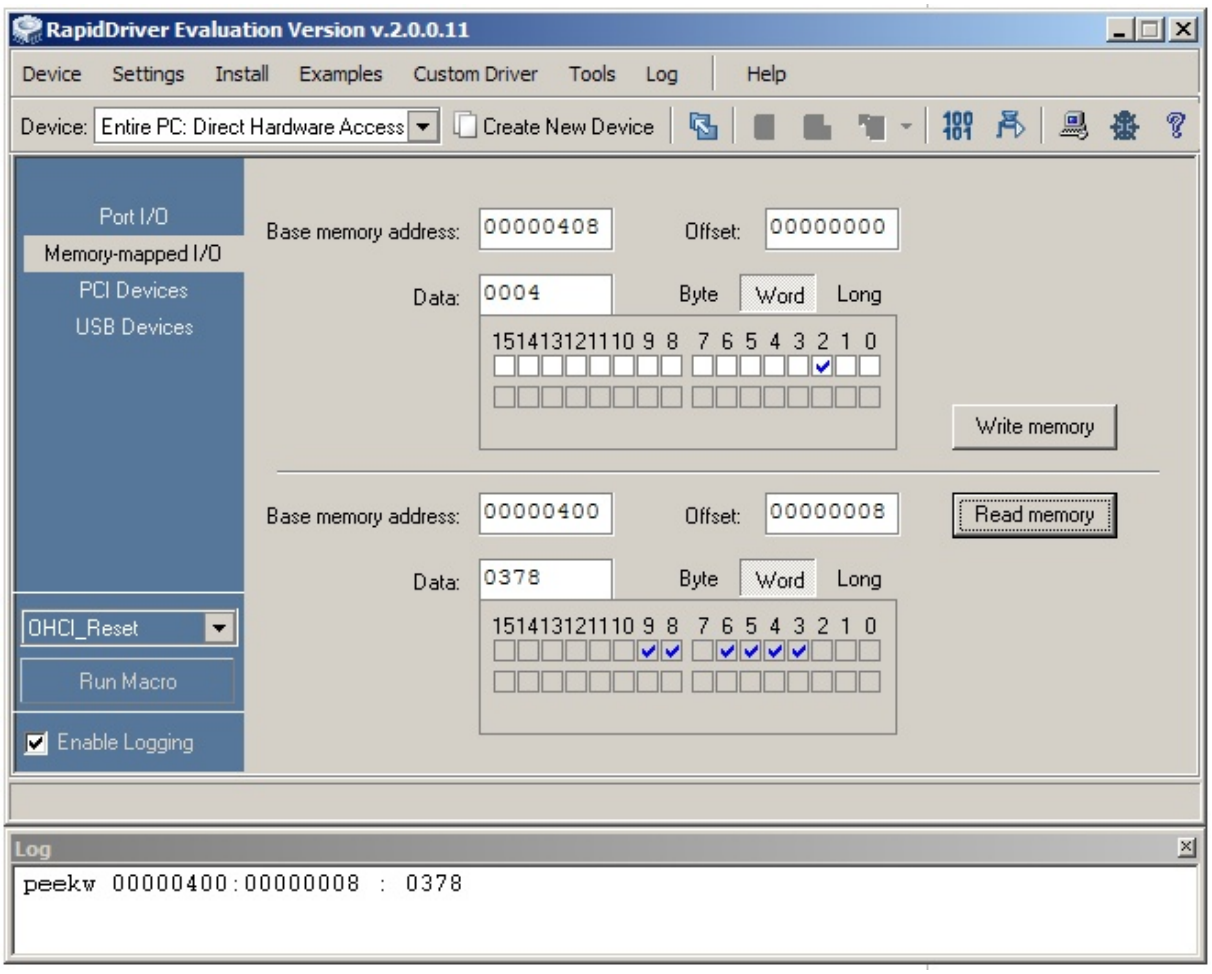
Also you may test your device with the help of built-in RapidDebugger.

3.4.1 Port I/O



Here you may select the port address then perform the read/write operations. The size of register is specified as a byte, word, or double word (long). Also you may run the group of operations with the help of selected macro.

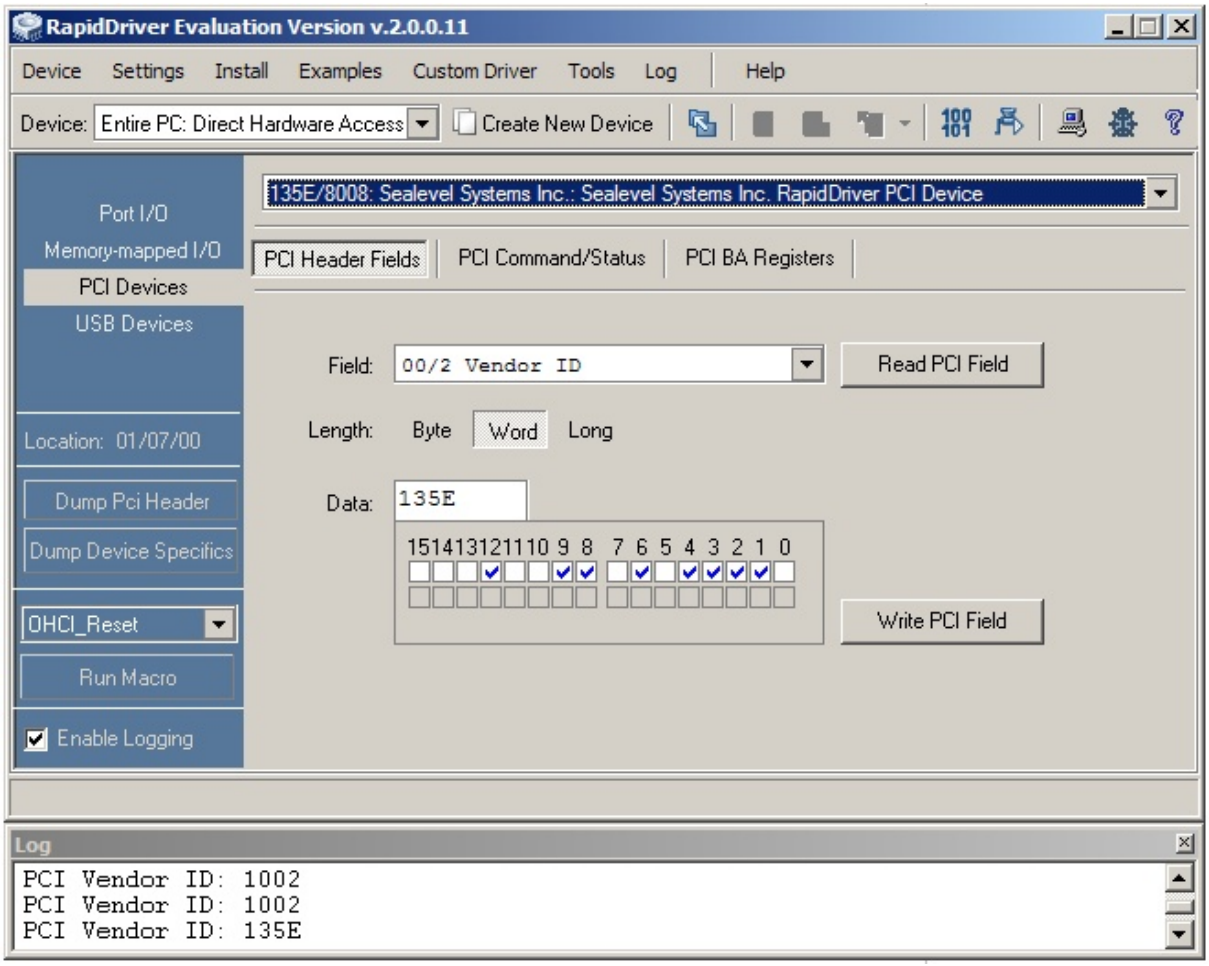
3.4.2 Memory-mapped I/O



Specify the base physical memory address and offset, then perform the read/write operation for a byte, word, or double word value.

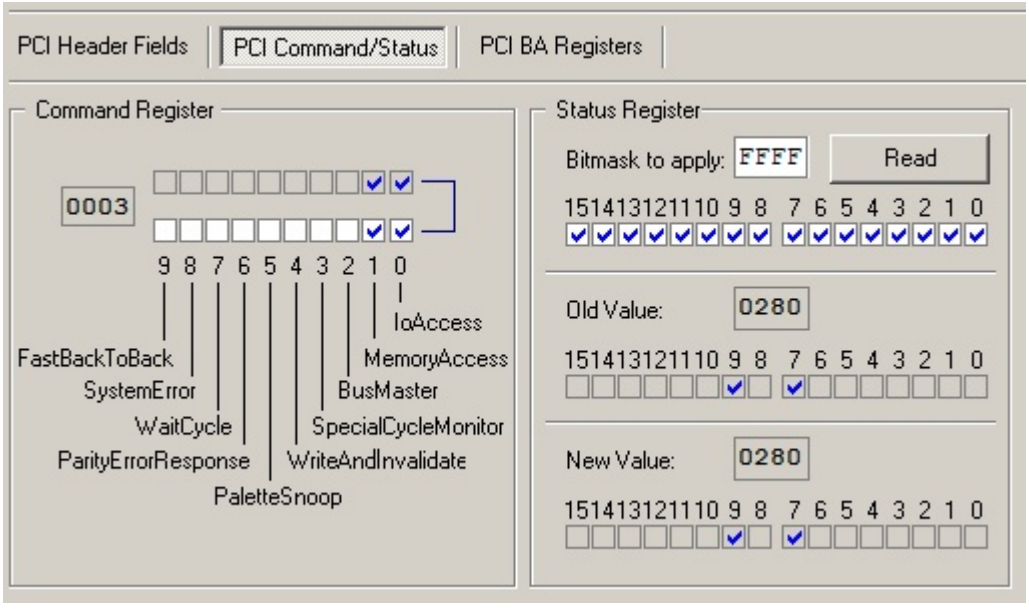
3.4.3 PCI Devices

First, select a device from the drop down list of autodetected PCI devices. Then you may read or write (if possible) any field of PCI configuration space:

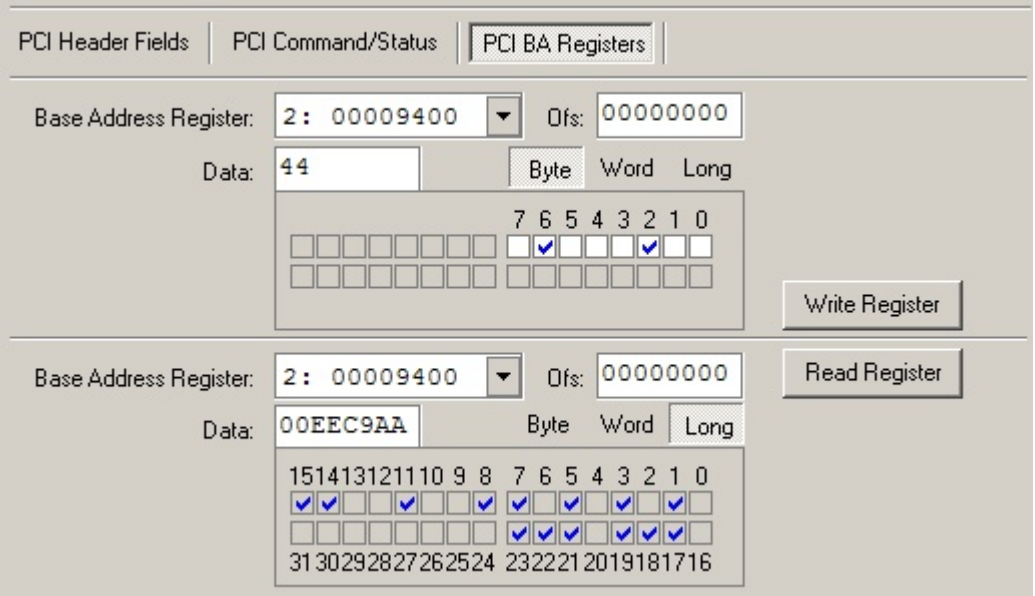


.

You may also operate on the PCI Command and Status registers,



as well as to manipulate with the PCI Base Address Registers (BARs):



3.4.4 USB Devices

Enter topic text here.

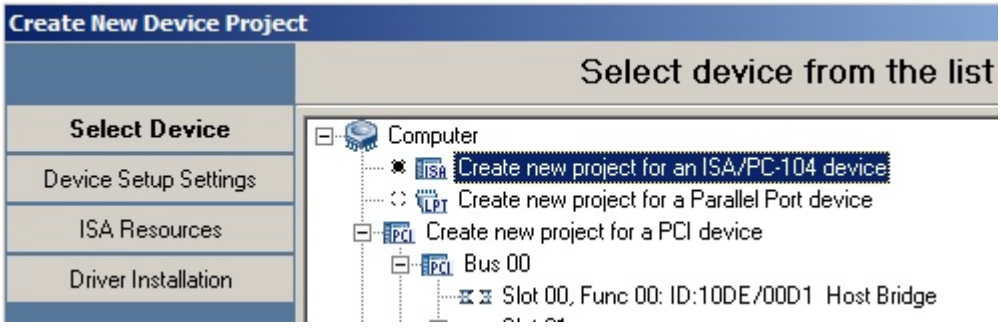
3.5 ISA/PC-104 Device Project

- New ISA/PC-104 Device Project
- Test ISA Device

Also you may test your device with the help of built-in RapidDebugger or run any ISA test example (**Menu | Examples | Run Example**).

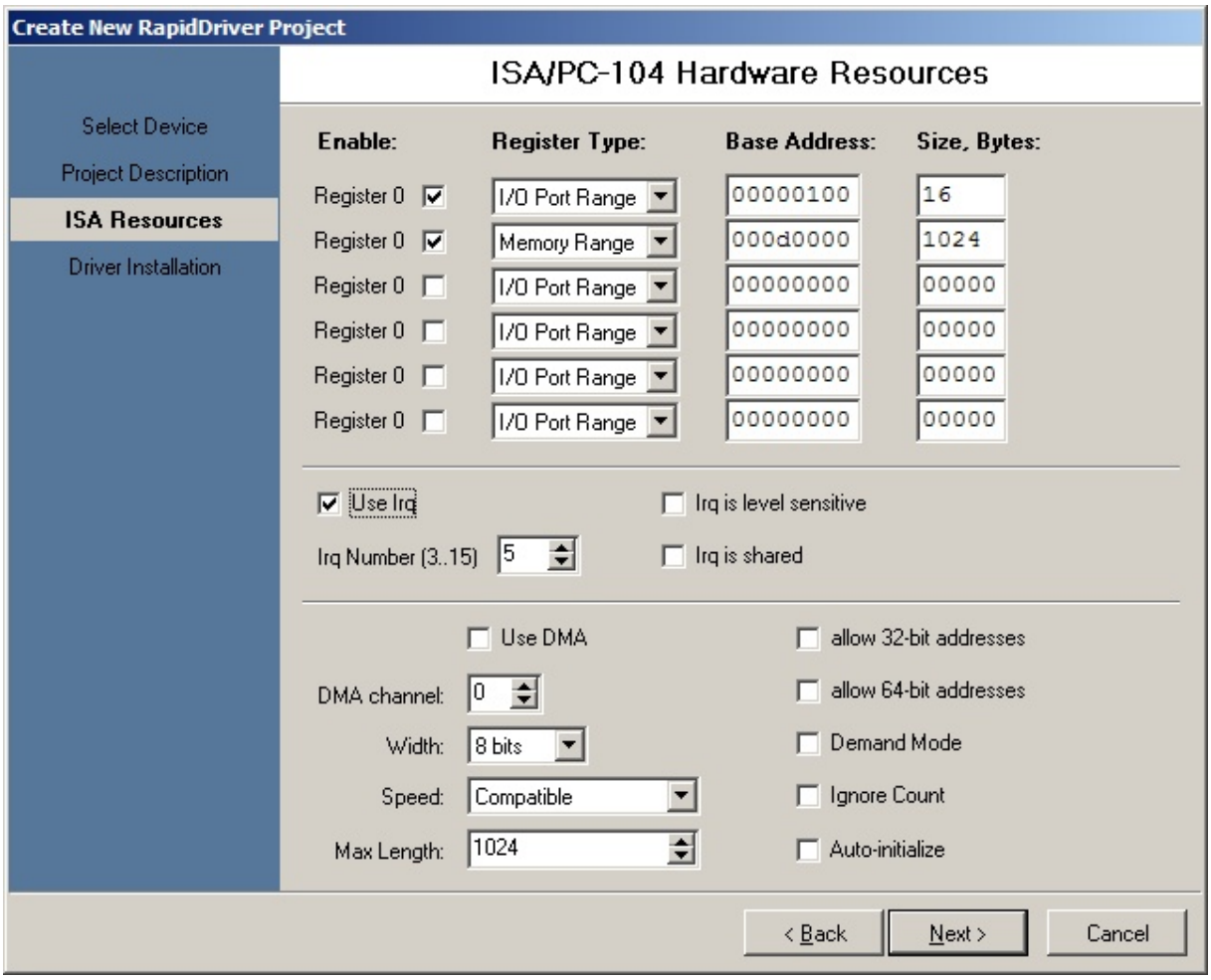
3.5.1 New ISA/PC-104 Device

To create a new ISA or PC-104 RapidDriver device project you must run the New Device Project wizard where you should select "Create new project for a legacy ISA/PC-104 device" at Step1:



Step 2 has nothing different with any other device type.

Step3: ISA Resources



Here you should describe up to 6 port/memory ranges. Also you can define if your device utilizes Irq and/or DMA channel.

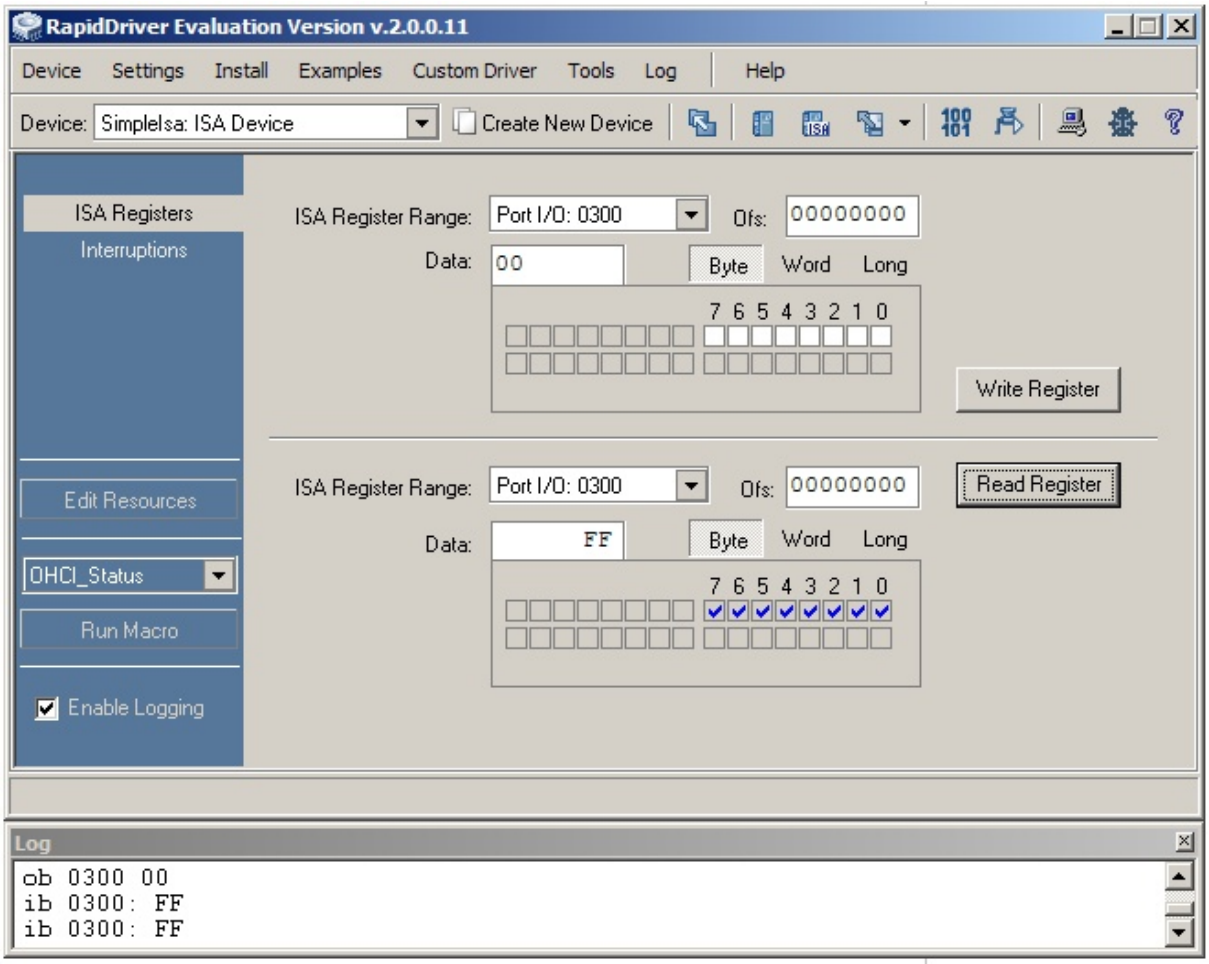
Step4: All the same as described above

3.5.2 Test ISA/PC-104 Device

Attention! You should install your device before you can test it. You can do it through the main menu (**Actions | Install Device Using RapidDrv.sys**) or by clicking on the toolbar button.

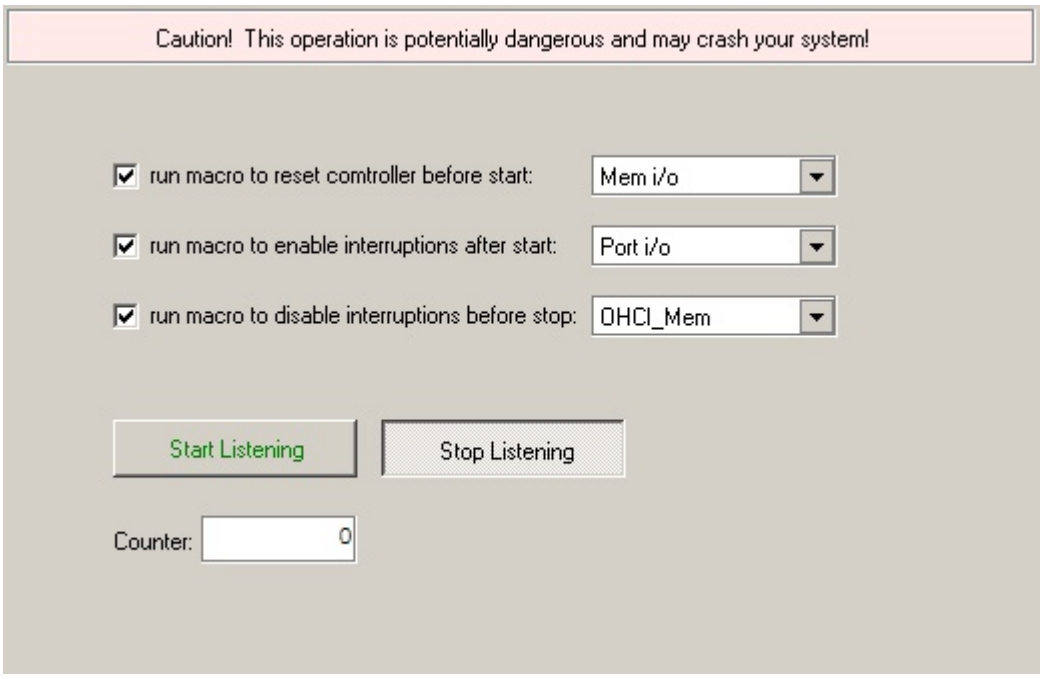
At the ***ISA Registers*** tab page you can do following:

- Read or write any hardware port address.
- Read or write selected memory range(s) .



- Listen for the hardware interruptions.

This operation is potentially dangerous, so to be on the safe side, save all unsaved data before you click the "Start Listening" button. Optionally you may provide three macros to reset controller *before you start* listening, enable the interruptions *after you start,* and disable the interruptions after you click the "Stop Listening" button.



Also you may test your device with the help of built-in RapidDebugger or run any ISA test example **(Menu | Examples | Run Example**)

3.6 PCI Device Project

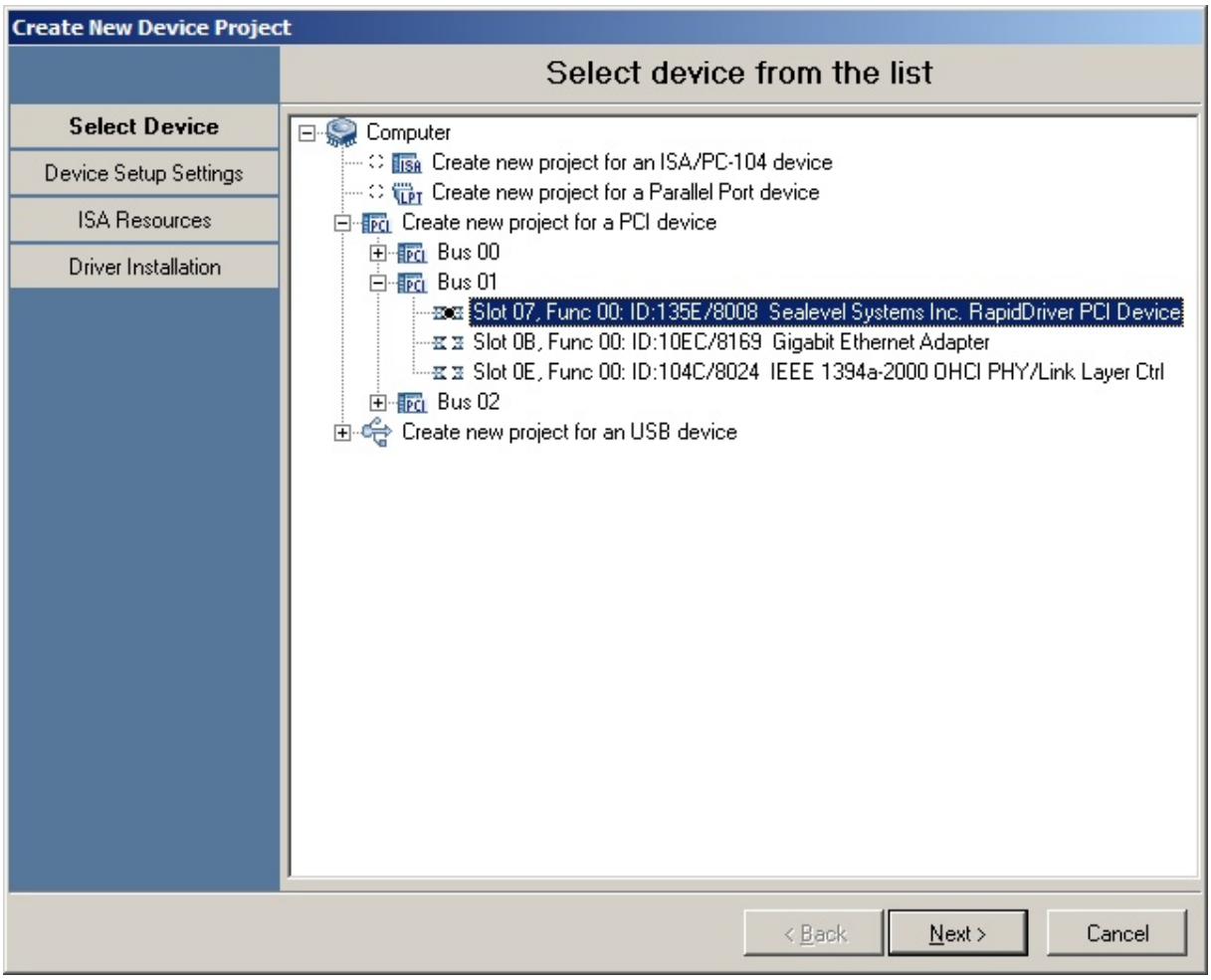
- New PCI Device Project
- Test PCI Device

Also you may test your device with the help of built-in RapidDebugger or run any PCI test example (**Menu | Examples | Run Example**).

3.6.1 New PCI Device

To create a new PCI RapidDriver device project:

1. Select **Device | Create New Device**. The New Device Project dialog box is displayed:



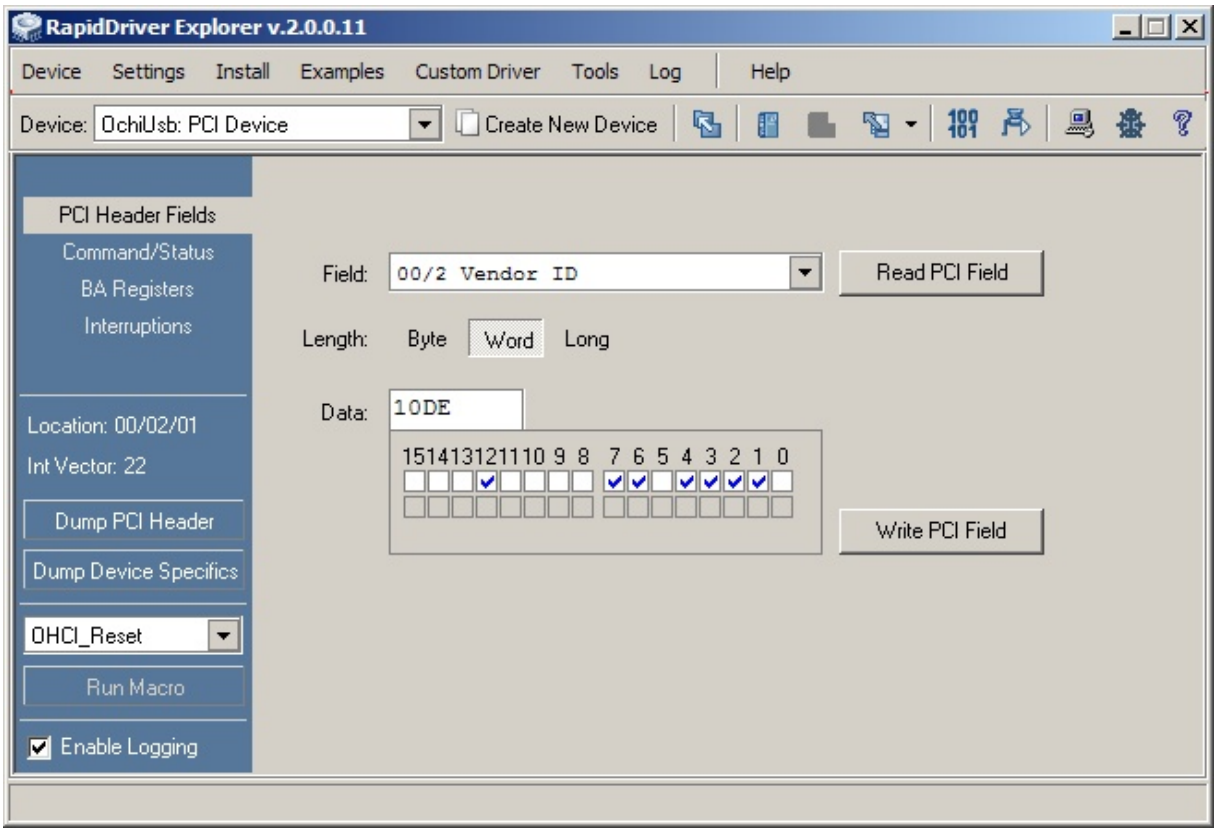
2. Select your PCI device from the list of auto-detected PCI devices.

3. Finish all steps mentioned above

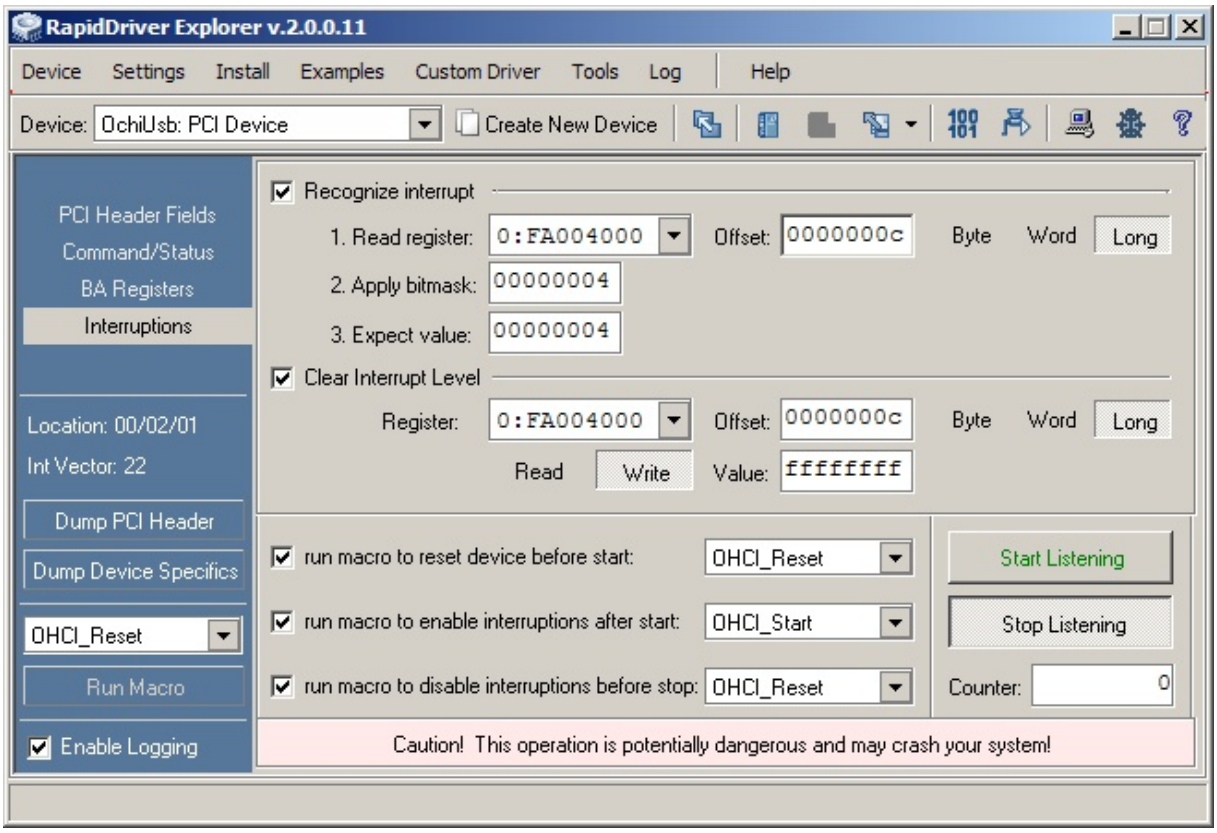
You are now ready to test your PCI device.

3.6.2 Test PCI Device

Attention! You should install your device before you can test it. You can do it through the main menu (**Install | Install Device Using RapidDrv.sys**) or by clicking on the corresponding toolbar button . The picture below shows the PCI device project window:



As you can see, first three tab pages are absolutely the same as it was described above for the "Entire PC: Direct Hardware Acees" device project, and one ***"Interruptions"*** tab page is added:



All PCI interrupts are "level-sensitive" and can be shared by many devices. That means when a device gives an interrupt, it asserts the interrupt signal and it stays asserted until we tell the device to clear the interrupt. Thus we are responsible to:

- recognize if our device caused the interrupt
- if so, then acknowledge (clear) our device's interrupt level by writing to or reading from some register.

At this tab page you can describe both of these matters. Optionally you may provide three macros to reset controller *before you start* listening, enable the interruptions *after you start,* and disable the interruptions after you click the "Stop Listening" button.

Also you may test your device with the help of built-in RapidDebugger or run any PCI test example (**Menu | Examples | Run Example**).

3.7 Parallel Port Device Project

- New Parallel Port Device Project
- Test Parallel Port Device

Also you may test your device with the help of built-in RapidDebugger or run any Parallel Port test example (**Menu | Examples | Run Example**).

3.7.1 New Parallel Port Device

You not need create a new Parallel Port Device project, this project is "pre-defined" in RapidDriver 2.0. Simply select the "Parallel Port Device" item from the drop-down list as it shown in picture below:

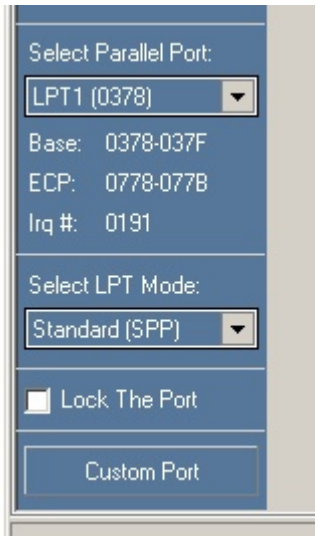
You are now ready to test your Parallel Port device.

3.7.2 Test Parallel Port Device

Attention! You should install your device before you can test it. You can do it through the main menu (**Install | Install Device Using RapidDrv.sys**) or by clicking on the corresponding toolbar button .

- Parallel Port Settings
- Custom Parallel Port
- LPT Pins
- Ports And Bits
- Interrupts

Parallel Port Settings

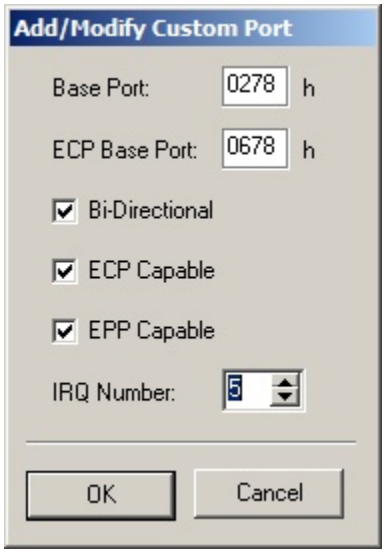


You must select the port from the list before you perform any parallel port operation. Do not forget to lock selected port to prevent possible clashes between programs wanting to use the same port (you

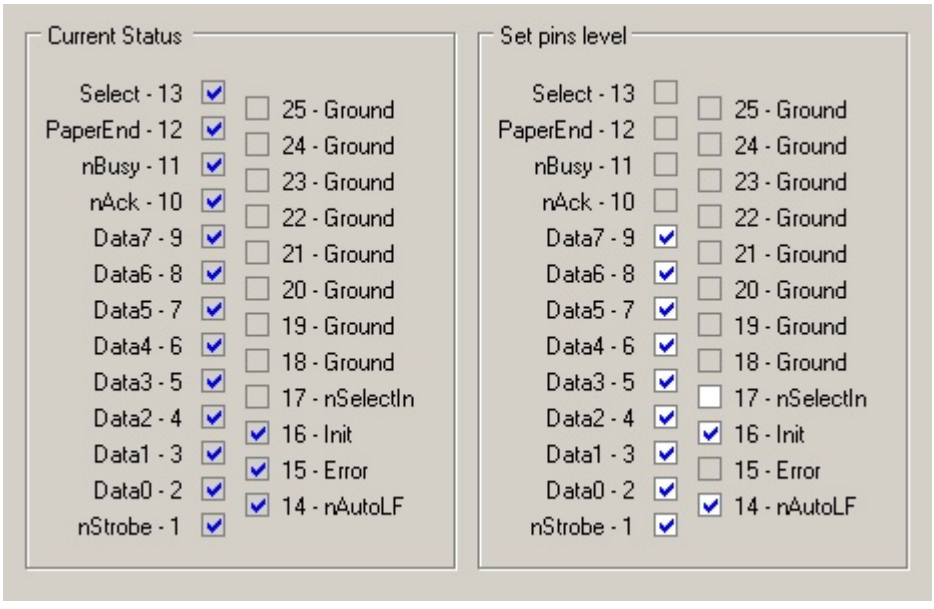
can not lock the "custom" parallel port).

Custom Parallel Port

The RapidDriver auto-detects all installed parallel ports in operating system. If some port does not appear in the list, that means this port was not properly installed. Anyway, you still have a possibility to add one port to the RapidDriver "by hand" as a custom parallel port:

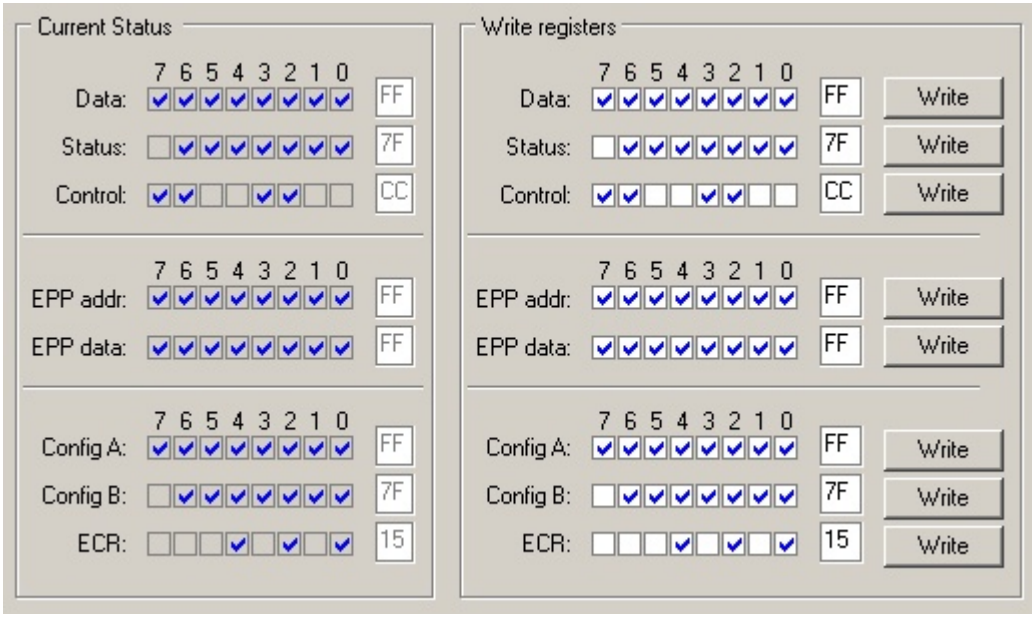


LPT Pins



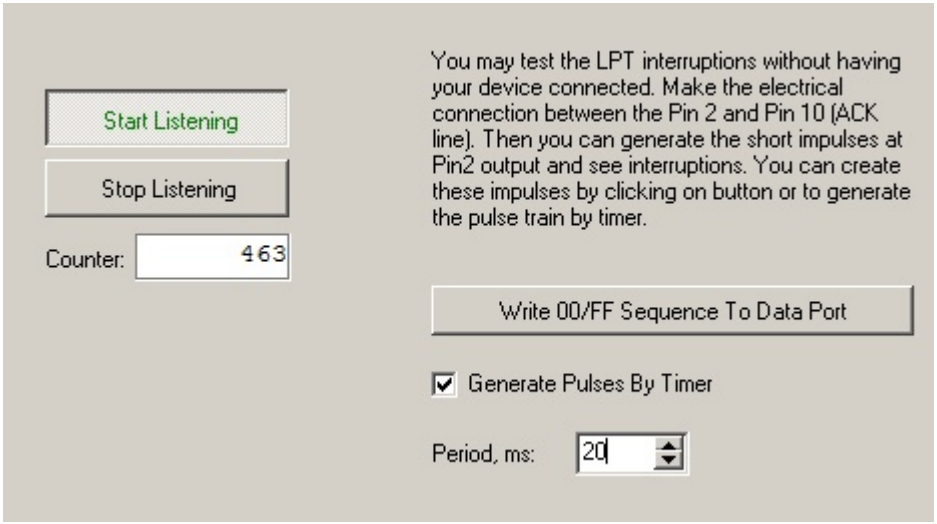
This is the pins state monitor, which also allows write an electrical level to the selected pin of selected parallel port.
Not all pins are accessible for read/write operations, that is why some pins are "disabled".

Ports And Bits



At this page you can see the current state of all parallel port registers and change a value of the separate bits or a whole register.

Interrupts



First of all, you must enable the interruptions for the selected parallel port in BIOS and in Control Panel (Port Settings->Use any IRQ assigned to the port).
Also you must enable this capability by setting to "1" of bit 4 in the parallel port Control Register. Then IRQ pass from ACK line (Pin10) to systems interrupt controller and you can listen for interruptions.

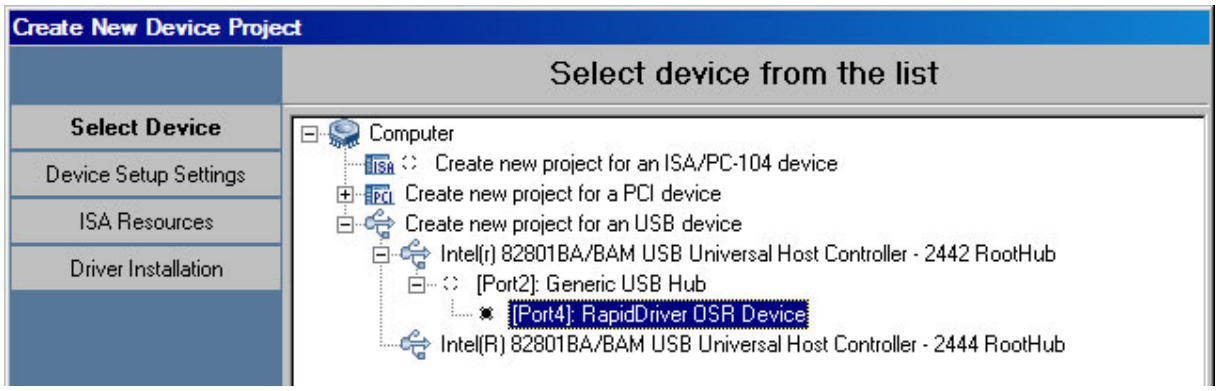
3.8 USB Device Project

- New USB Project
- Test USB Device

3.8.1 New USB Device

To create a new USB device project:

1. Select **Device | Create New Device...** The New Device Project dialog box is displayed:



2. Select your USB device from the list of auto-detected USB devices.

3. Finish all steps mentioned above

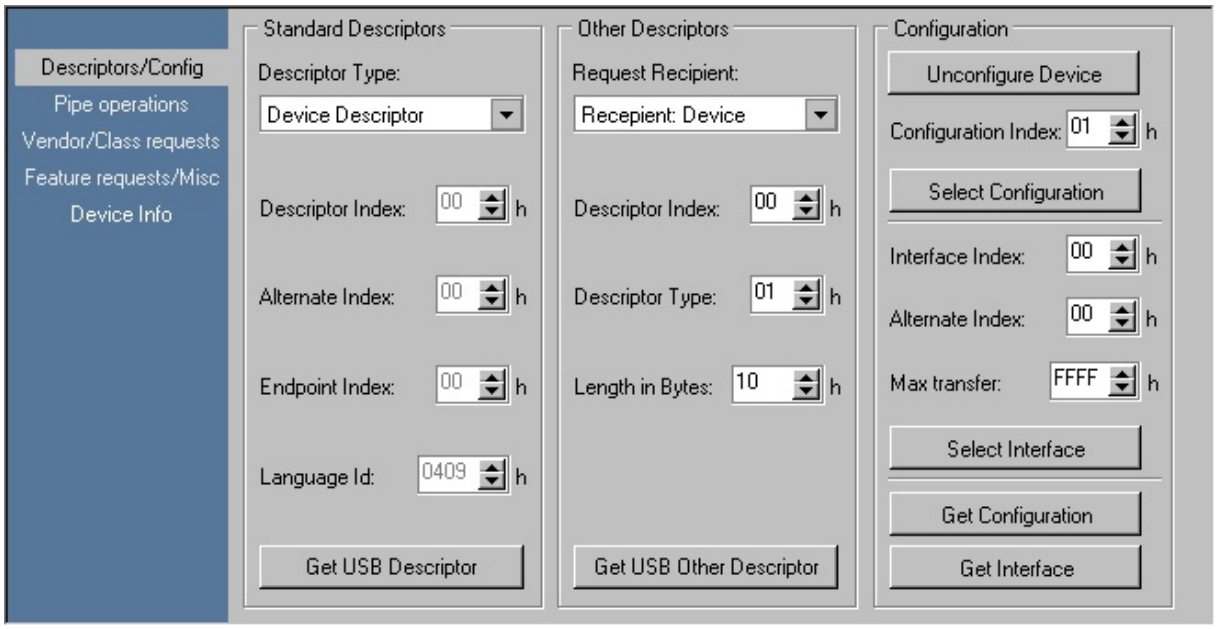
4. Test your USB device

3.8.2 Test USB Device

This section describes RapidDriver USB testing part. More information is available in following topics:

- Descriptors and Configuration
- Vendor/Class Requests
- Pipe Operations
- Feature Requests

3.8.2.1 Descriptors and Configuration



Descriptor requests

To get one of standard USB descriptors specify all necessary information in "Standard Descriptors" box. Language ID filed is intended only for string descriptor requests. To get endpoint descriptor specify device-defined index of endpoint in "Endpoint Index" filed, index of Interface in "Descriptor Index" filed and index of alternate interface in "Alternate Index" filed. To get interface descriptor specify interface and alternate indexes in corresponding fields.

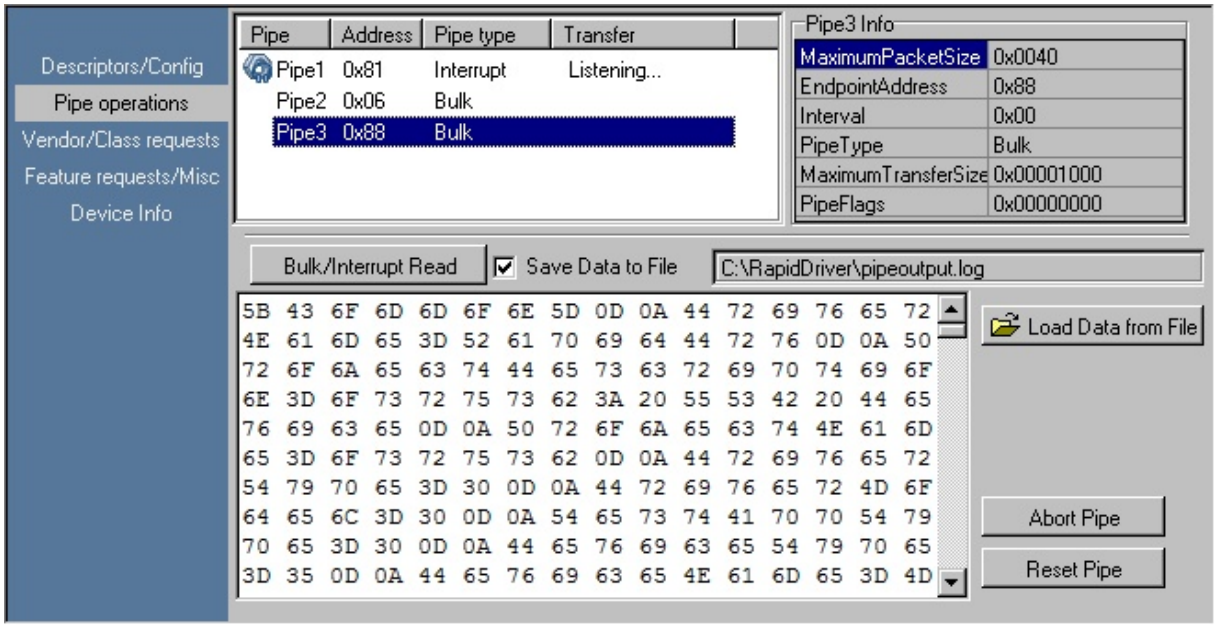
If required descriptor cannot be found in "Descriptor Type" list, use "Other Descriptor" box to get descriptor. Class or vendor-specific descriptors cannot be obtained using this method. To issue class or vendor descriptor request refer to Vendor/Class requests tab.

All received data is displayed in Log window.

Configuring device

Device can be unconfigured, configuration can be changed and alternate interface can be selected from "Configuration" box. Interface selection also can be used for purpose of changing maximum transfer size of selected interface. While selecting configuration "Configuration Index" must be specified, data from other fields is ignored. If "Configuration Index" is equal to zero then "Select Configuration" has the same effect as "Unconfigure Device" (device is placed in unconfigured state). Specify the interface index, interface alternate index and maximum transfer in bytes before pressing "Select Interface" button.

3.8.2.2 Pipe Operations



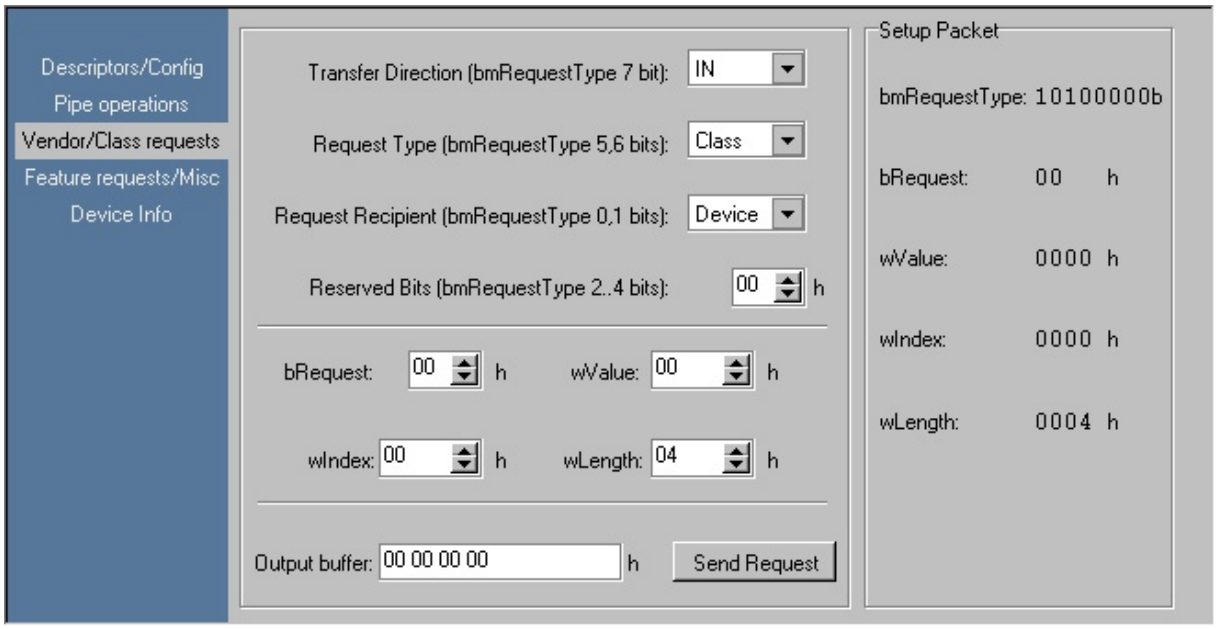
Read operation

To start read operation from interrupt or bulk pipe select pipe and press "Bulk/Interrupt Read" button. After this RapidDriver opens popup window to which all data received from pipe is dumped. Listening process is indicated by  icon. To redirect data received during listening process to file on hard disk click "Save Data to File" check box and select existing or enter new file name in dialog.

Write operation

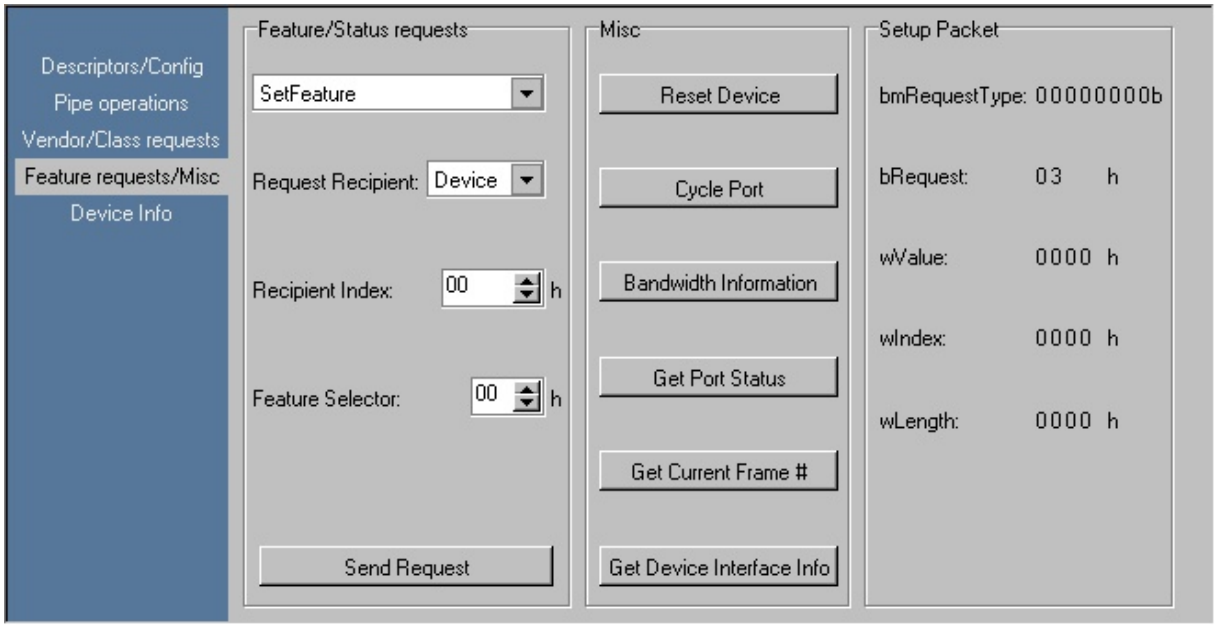
During write operation RapidDriver uses one of two sources as output buffer which is send to specified pipe. First source is "Output Data" memo. Data can be loaded to this memo from file using "Load Data from File" button or entered by hand in hexadecimal format. Other source of data send to pipe can be selected by clicking "Send File to Pipe" check box. File selected in appeared dialog is send to pipe after pressing "Bulk/Interrupt Write".

3.8.2.3 Vendor/Class Requests



This tab is used to issue vendor or class specific requests to USB device. "Setup Packet" box reflects contents of setup packet to be send during vendor/class request.
All information received from device is dumped to Log window.

3.8.2.4 Feature Requests



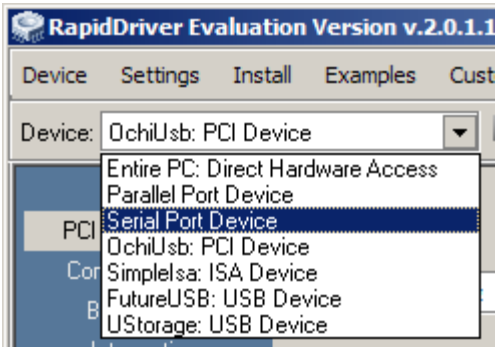
This tab is used to send Set Feature, Clear Feature and Get Status requests to USB device."Setup Packet" box reflects contents of setup packet to be send during feature or status request.
USB operations not included to other categories can be performed from this tab as well.
"Cycle Port" operation simulates a device unplug and re-plug. "Bandwidth Information" - reports total

bandwidth available on the bus and mean bandwidth already in use.

3.9 Serial Port Support

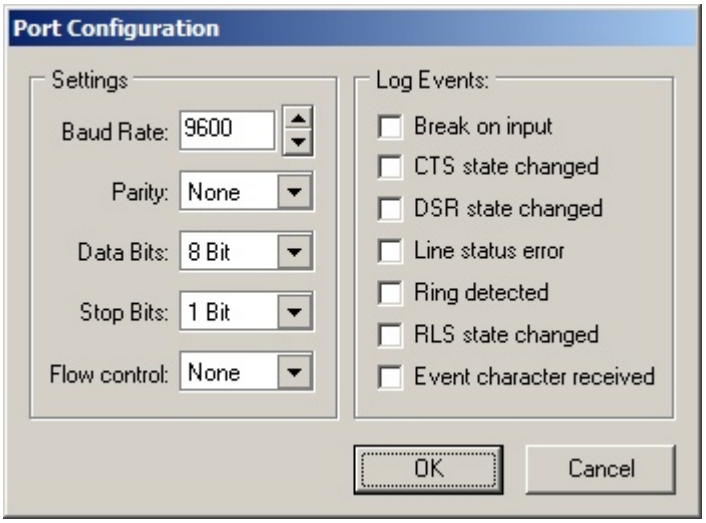
RapidDriver 2.0 has a limited serial port support.

- Select the "Serial Port Device" item form the drop-down list:



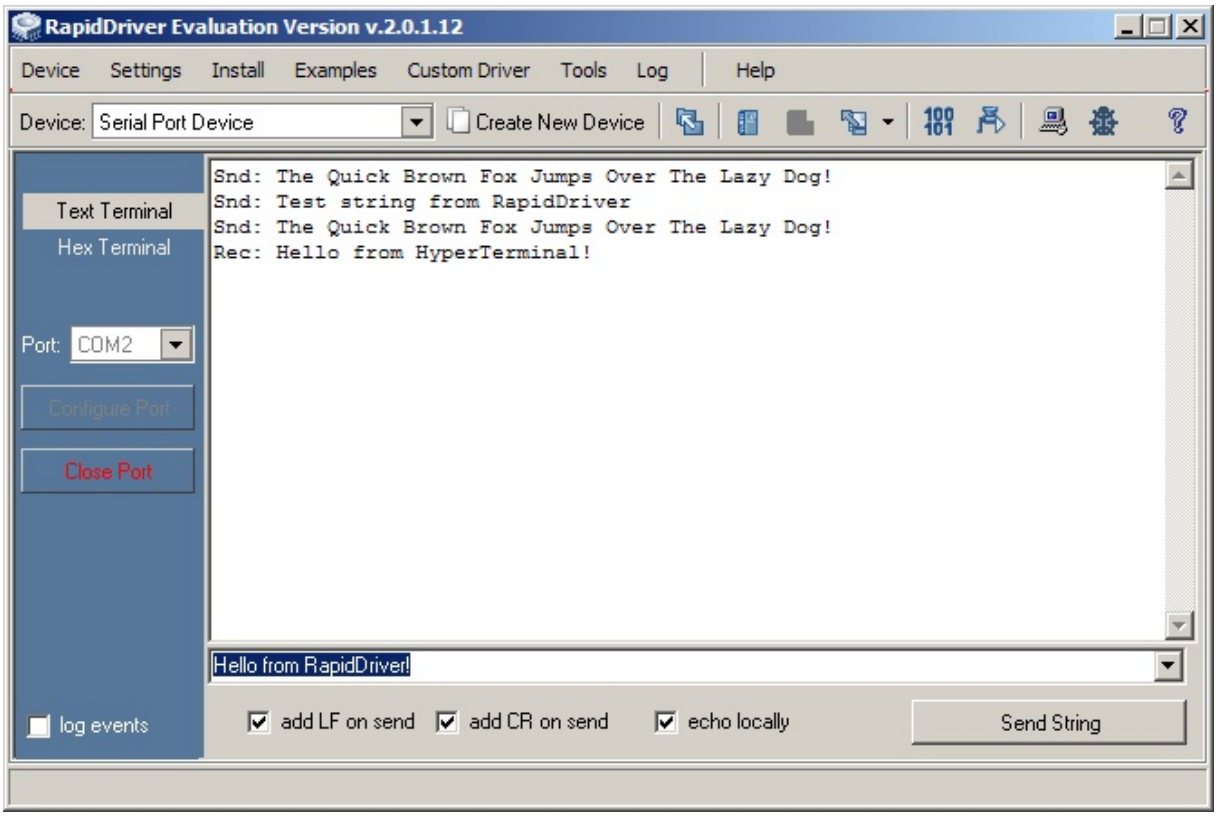
- Configure Serial Port
- Use Text Terminal or Hex Terminal to communicate with the serial port device

3.9.1 Serial Port Configuration



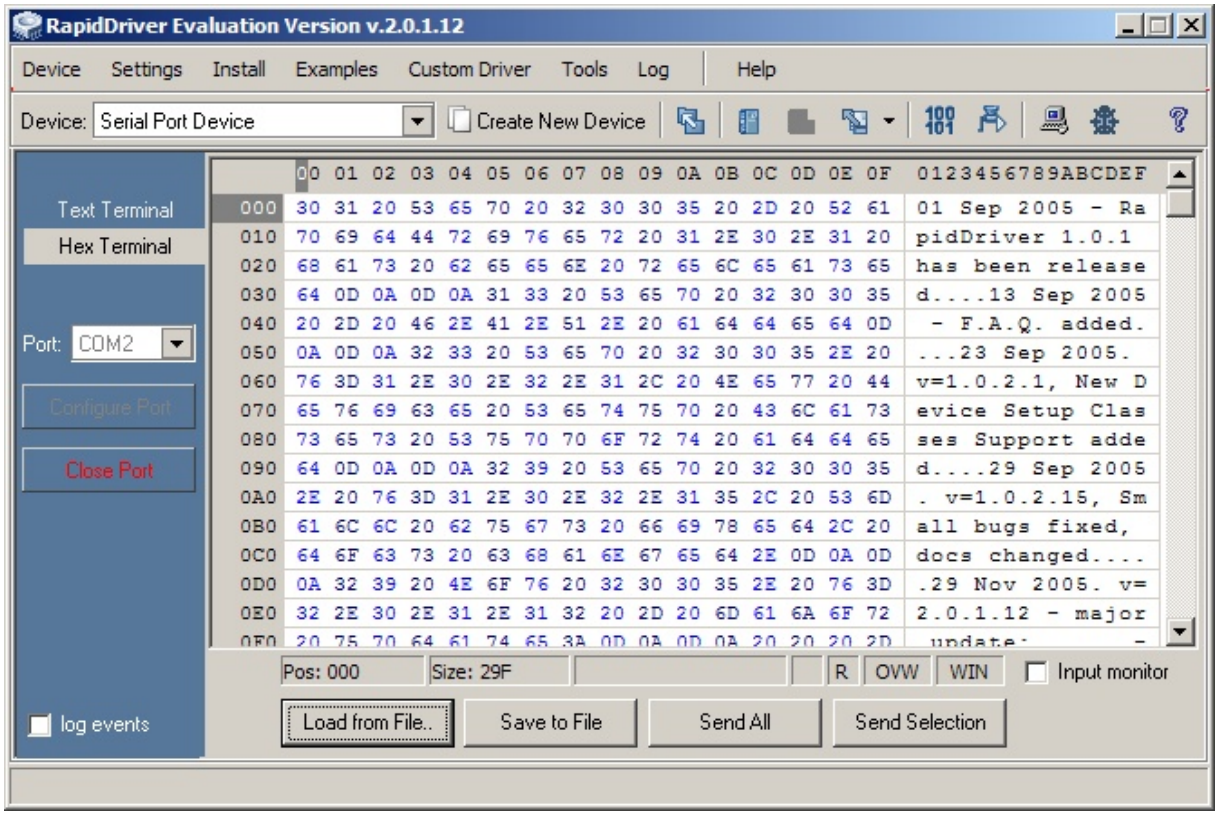
In this dialog window you may set most important serial port parameters as well as enable/disable logging of some serial port events.

3.9.2 Text Terminal



With the help of text terminal you may communicate with another PC or some remote device through the null-modem cable.

3.9.3 Hex Terminal



With the help of RapidDriver Hex Terminal you may load the file, edit it, then send it to another PC, or, for example, upload it as a firmware to some hardware device (if applicable). You may send the whole file or a selected part of it.

.

4 RapidDebugger

RapidDebugger

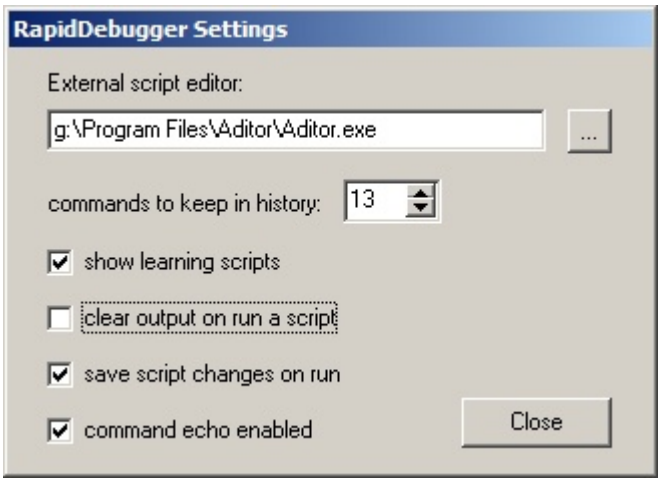
http://www.RapidDriver.com

tools@entechtaiwan.com

4.1 Introduction

RapidDebugger is simple and convenient built-in hardware (not software!) debugger to perform a more intimate analysis of your device.

With RapidDebugger you can create a program written in C-, Pascal-, or Basic-like scripting language. Many example scripts are provided to show how to call most procedures and functions for various kind of devices. You can modify these programs to suit your own needs. To see all learning scripts go to **Options | Settings...** then check the "show learning scripts" option:



RapidDebugger includes also a command line debugger, that allows you to control your device "on-the-fly" (read/write hardware ports, working with the memory-mapped registers, PCI devices and more).

You can work with a *"native"* or a *"foreign"* device in RapidDebugger. A *"native"* device must meet two conditions:

- the device must be linked with the ***current*** RapidDriver project
- the device must be successfully installed with RapidXXX.sys driver and appear in Device Manager under the "RapidDriver Devices" class branch.

Any other device is *"foreign"* as far as RapidDebugger is concerned.

The Script Debugger allows you to work with both *"native"* and *"foreign"* devices, but please note the following difference in treatment:

"Foreign" devices: When working with *"foreign"* devices you need a PCI or USB device handle, which you obtain by calling OpenPciDevice() or OpenUsbDevice().

"Native" devices: When working with *"native"* devices, you should ***not*** call the OpenXxxDevice() function. Use 0 (zero) in place of the device handle.

NB: All devices are *"foreign"* devices as far as the Command-line Debugger is concerned.

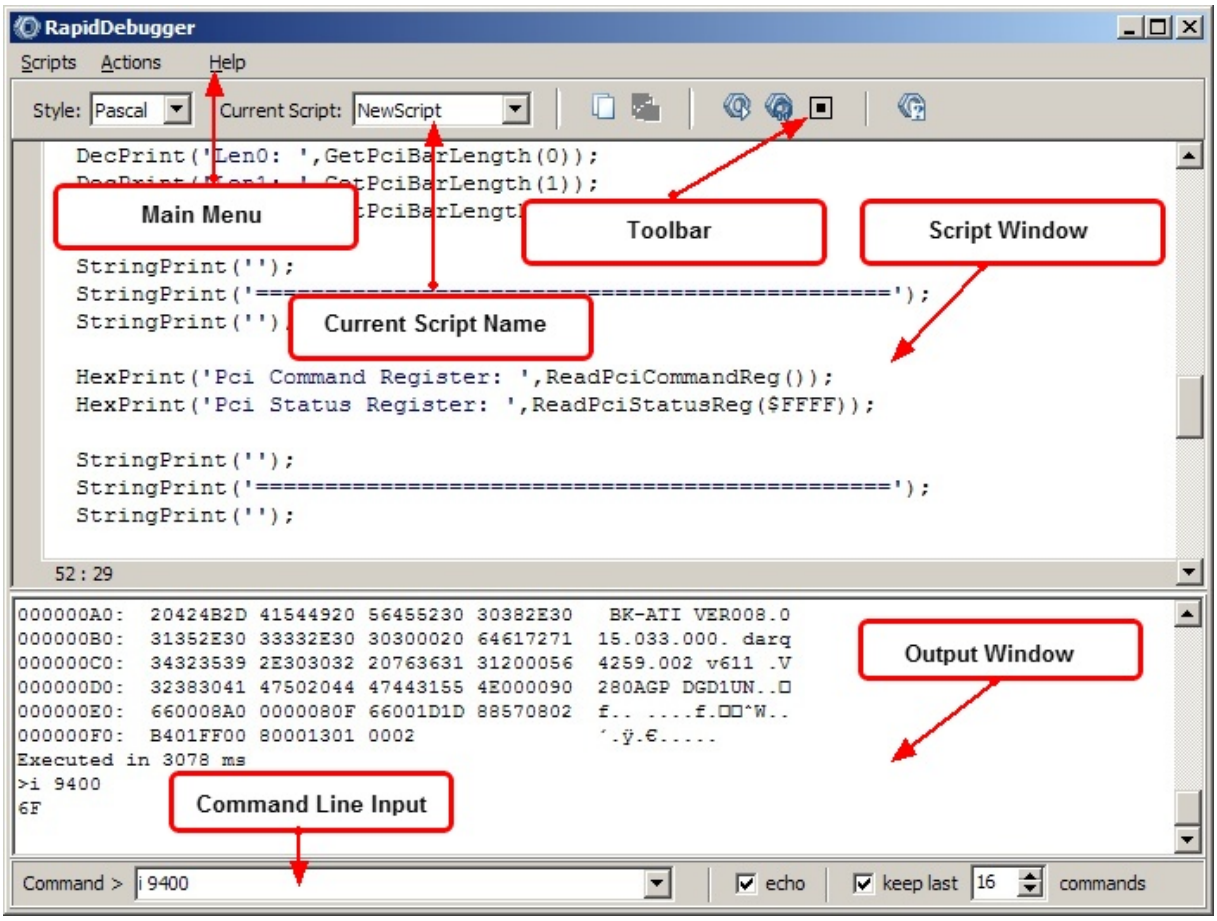
RapidDebugger User Interface

Command-line Debugging

Scripting Debugging

4.2 RapidDebugger User Interface

RapidDebugger provides a consistent interface for debugging hardware. The debugger tool has three working areas, main menu, and a toolbar.



It provides a few windows to let you view and control various aspects of your debugging session.

- Script Window

The built-in simple text editor. If you have selected a script from the list of existing scripts, or created new one, the text editor displays it's source code and enables you to edit it.

- Output Window

Displays output information from the running script and command line debuggers.

- Command Line

The small bottom pane labeled "Command" in the title bar is used to enter text commands. Commands entered on the command line are saved in a list box (ring buffer) whose length can be changed. The RapidDebugger Commands reference sections describes all the commands and their restrictions.

4.3 Command-line Debugging

Notes:

1. All values are hexadecimal

2. All commands are case-insensitive

3. [size]=**'B'** or empty for a byte value, **'W'** for a word, **'D'** for a double word.

HELP, H, or ? - this help file.

Common hardware operations

Command	Description
I*[size] port*	Input a value from an I/O port. Example: ***i 378*** - read LPT data port
O*[size] port value*	Output a ***value*** to an I/O port. Example: ***ow 300 1045*** - write 1045h value to the port
PEEK*[size] address*	Read from physical memory. Example: ***peekd E000040*** - read double word from the memory
POKE*[size] address value*	Write to physical memory. Example: ***pokew 408 278*** - write 278h word value to the memory
DUMP address length	Print a formatted physical memory dump of a given region. Example: ***dump c0040 100*** - print 256 bytes starting with c0040h address.

PCI operations

Note: *You must enumerate all PCI devices, then open selected device before you can issue any other PCI command.*

Command	Description
PCI *[ENUM]*	Enumerate all PCI devices. Example: pci enum
PCI OPEN *bus/dev/func*	Open a PCI device by the location. Example: ***pci open 01/0f/00***
PCI OPEN *number*	Open a PCI device by the number in list printed by PCI ENUM command. Example: ***pci open 5***
PCI HEADER	Print a formatted header of a PCI device (first 64 bytes). Example: ***pci header***
PCI HEADER DUMP	Print a formatted dump of PCI header (all 256 bytes). Example: ***pci header dump***
PCI HEADER DS	Print a formatted dump of device-specific part of PCI header (last 192 bytes). Example: ***pci header ds***
PCI COMMAND [value]	Read or write PCI command register. Write if value is set. Example: ***pci command 03*** - write 03h to the command register
PCI COMMAND BIT *nbit value*	Set or clear selected bit in PCI command register. Example: ***pci command 2 0*** - clear bit 2 in command register
PCI STATUS *[mask]*	Read PCI status register then apply the *mask* value if set. Example: ***pci status f7*** - read PCI status register then apply f7h mask.
I*[size]* ***BAR<number>*** *offset*	Read a value from a register addressing by a PCI BAR number and offset. Example: ***id bar0 8*** - read double word from the register addressing by BAR0 + 8.
O*[size]* ***BAR<number>*** *offset value*	Write a value to a register addressing by a PCI BAR number and offset. Example: ***o bar0 8*** 4A - write 4Ah byte value to the register addressing by BAR0 + 8.

USB operations

Note: *You must enumerate all USB devices, then open selected device before you can issue any other USB command.*

Command	Description
USB *[ENUM]*	Enumerate all USB devices. Device number that is printed as a result of enumeration is used by USB OPEN command and in debugger scripts as parameter of *OpenUsbDevice* function. Example: **usb enum**
USB OPEN *number*	Open a USB device by the number in list printed by USB ENUM command. You can open only RapidUSB device. Example: ***usb open 5***
USB CONINFO	Print the connection info structure assosiated with previously opened USB device. Example: ***usb coninfo***

4.4 Scripting Debugging

Besides the command line debugging, the RapidDebugger provides an alternative way to debug a hardware device based on well known multi-language FastScript scripting engine created by Fast Reports Inc. Thus you can write debugging scripts using your favourite programming language (C/C++, Pascal, Basic). The Fast Reports Inc. has kindly allowed us to use extracts from their FastScript documentation in this help file as well as their learning script examples.

Here are described the functons/procedures that was added in the course of support the hardware debugging process. Along with the RapidDriver toolkit, you will find a lot of examples of using these functions in your own scripts.

4.4.1 C/C++ Declarations

Port I/O group

```
int  GetPortByte(int nPort);
int  GetPortWord(int nPort);
int  GetPortLong(int nPort);

void SetPortByte(int nPort, int bValue);
void SetPortWord(int nPort, int wValue);
void SetPortLong(int nPort, int dValue);
```

Memory group

```
int  MapPhysToLinear(int PhysicalAddress, int PhSize);

int  GetMemByte(int MappedAddr, int Offset);
int  GetMemWord(int MappedAddr, int Offset);
int  GetMemLong(int MappedAddr, int Offset);

void SetMemByte(int MappedAddr, int Offset, int bValue);
void SetMemWord(int MappedAddr, int Offset, int wValue);
void SetMemLong(int MappedAddr, int Offset, int dValue);
```

PCI group

```
int  OpenPciDevice(int Bus, int Dev, int Funct);
void GetPciLocation(int PciDevice, int * Bus, int * Dev, int * Funct);

int  ReadFromPci(int PciDevice, int Offset, int Length);
void WriteToPci (int PciDevice, int Offset, int Length, int Data);

int GetPciBarPhysicalAddress(int PciDevice, int BarNumber);
```

```
int ReadPciBarByte (int PciDevice, int BarNumber, int BarOffset);
int ReadPciBarWord (int PciDevice, int BarNumber, int BarOffset);
int ReadPciBarLong (int PciDevice, int BarNumber, int BarOffset);

int WritePciBarByte(int PciDevice, int BarNumber, int BarOffset, int
bValue);
int WritePciBarWord(int PciDevice, int BarNumber, int BarOffset, int
wValue);
int WritePciBarLong(int PciDevice, int BarNumber, int BarOffset; int
dwValue);

int  ReadPciCommandReg(int PciDevice);
void WritePciCommandReg(int PciDevice, int CommandRegValue);
void ControlPciCommandRegBits(int PciDevice, int CommandRegBit, int
SetClearFlag);
int  ReadPciStatusReg(int PciDevice, int ClearBitMask);
```

USB group

```
function OpenUsbDevice(DevNumber : int) : int;
function CloseUsbDevice(UsbDevice : int) : bool;

function UsbTransfer(UsbDevice : int;
                     PipeNumber : int;
                     var Buffer : array of Byte;
                     var intsReturned : int) : Integer;

function UsbVendorRequest(UsbDevice : int;
                     DirectionIn : bool;
                     RequestRecipient : int;
                     ReservedBits : int;
                     Request : int;
                     Value : int;
                     Index : int;
                     var Buffer: array of Byte;
                     var NumberOfBytesTransferred : int) : bool;

function UsbClassRequest(UsbDevice : int;
                     DirectionIn : bool;
                     RequestRecipient : int;
                     ReservedBits : int;
                     Request : int;
                     Value : int;
                     Index : int;
                     var Buffer: array of Byte;
                     var NumberOfBytesTransferred : int) : bool;

function UsbSetFeature(UsbDevice : int;
                     RequestRecipient : int;
                     FeatureSelector  : int;
                     Index            : int) : bool;

function UsbClearFeature(UsbDevice : int;
                     RequestRecipient : int;
                     FeatureSelector  : int;
                     Index            : int) : bool;
```

```
function UsbGetStatus(UsbDevice : int;
                      RequestRecipient : int;
                      Index            : int;
                      var Status       : int) : bool;
```

Service Group

```
string HexByte(int b);
string HexWord(int w);
string HexLong(int d);

void HexPrint(string sComment, int Val);
void DecPrint(string sComment, int Val);
void StringPrint(string s);
void HexDump(char * dump, int NumOfBytes);

void Delay(int us);
int  GetCurrentProjectType(); // see constants
```

4.4.2 Pascal Declarations

Port I/O group

```
function  GetPortByte(nPort: Word): Byte;
function  GetPortWord(nPort: Word): Word;
function  GetPortLong(nPort: Word): Cardinal;

procedure SetPortByte(nPort: Word; bValue: Byte);
procedure SetPortWord(nPort: Word; wValue: Word);
procedure SetPortLong(nPort: Word; dValue: Cardinal);
```

Memory group

```
function  MapPhysToLinear(PhAddr: Cardinal; PhSize: Cardinal): Cardinal;

function  GetMemByte(MappedAddr: Cardinal; Offset: Cardinal): Byte;
function  GetMemWord(MappedAddr: Cardinal; Offset: Cardinal): Word;
function  GetMemLong(MappedAddr: Cardinal; Offset: Cardinal): Cardinal;

procedure SetMemByte(MappedAddr: Cardinal; Offset: Cardinal; bValue: Byte);
procedure SetMemWord(MappedAddr: Cardinal; Offset: Cardinal; wValue: Word);
procedure SetMemLong(MappedAddr: Cardinal; Offset: Cardinal; dValue:
Cardinal);
```

PCI group

```
function  OpenPciDevice(Bus, Dev, Funct: Cardinal): Integer;
procedure GetPciLocation(PciDevice: Cardinal; var Bus: Byte; var Dev: Byte;
var Funct: Byte);

function  ReadFromPci(PciDevice: Cardinal; Offset: Cardinal; Length:
Cardinal): Cardinal;
procedure WriteToPci (PciDevice: Cardinal; Offset: Cardinal; Length:
Cardinal; Data: Cardinal);
```

```
function GetPciBarPhysicalAddress(PciDevice: Cardinal; BarNumber: Word):
Cardinal;

function ReadPciBarByte (PciDevice: Cardinal; BarNumber: Word; BarOffset:
Cardinal): Byte;
function ReadPciBarWord (PciDevice: Cardinal; BarNumber: Word; BarOffset:
Cardinal): Word;
function ReadPciBarLong (PciDevice: Cardinal; BarNumber: Word; BarOffset:
Cardinal): Cardinal;

procedure WritePciBarByte(PciDevice: Cardinal; BarNumber: Word; BarOffset:
Cardinal; bValue: Byte);
procedure WritePciBarWord(PciDevice: Cardinal; BarNumber: Word; BarOffset:
Cardinal; wValue: Word);
procedure WritePciBarLong(PciDevice: Cardinal; BarNumber: Word; BarOffset:
Cardinal; dwValue: Cardinal);

function  ReadPciCommandReg(PciDevice: Cardinal):Word;
procedure WritePciCommandReg(PciDevice: Cardinal; CommandRegValue: Word);
procedure ControlPciCommandRegBits(PciDevice: Cardinal; CommandRegBit:
Word; SetClearFlag: Word);
function  ReadPciStatusReg(PciDevice: Cardinal; ClearBitMask: Word): Word;
```

USB group

```
function OpenUsbDevice(DevNumber : Byte) : Cardinal;
function CloseUsbDevice(UsbDevice : Cardinal) : Boolean;

function UsbTransfer(UsbDevice : Cardinal;
                     PipeNumber : Byte;
                     var Buffer : array of Byte;
                     var BytesReturned : Cardinal) : Boolean;

function UsbVendorRequest(UsbDevice : Cardinal;
                     DirectionIn : Boolean;
                     RequestRecipient : Byte;
                     ReservedBits : Byte;
                     Request : Byte;
                     Value : Word;
                     Index : Word;
                     var Buffer: array of Byte;
                     var NumberOfBytesTransferred : Cardinal) : Boolean;

function UsbClassRequest(UsbDevice : Cardinal;
                     DirectionIn : Boolean;
                     RequestRecipient : Byte;
                     ReservedBits : Byte;
                     Request : Byte;
                     Value : Word;
                     Index : Word;
                     var Buffer: array of Byte;
                     var NumberOfBytesTransferred : Cardinal) : Boolean;

function UsbSetFeature(UsbDevice : Cardinal;
                     RequestRecipient : Byte;
                     FeatureSelector  : Word;
                     Index            : Word) : Boolean;
```

```
function UsbClearFeature(UsbDevice : Cardinal;
                  RequestRecipient : Byte;
                  FeatureSelector  : Word;
                  Index            : Word) : Boolean;

function UsbGetStatus(UsbDevice : Cardinal;
                  RequestRecipient : Byte;
                  Index            : Word;
                  var Status       : Word) : Boolean;
```

Service Group

```
function HexByte(b: Byte): String;
function HexWord(w: Word): String;
function HexLong(d: Cardinal): String;

procedure HexPrint(sComment:String; Val: Cardinal);
procedure DecPrint(sComment:String; Val: Cardinal);
procedure StringPrint(s:String);
procedure HexDump(dump: Variant; NumOfBytes: Cardinal);

procedure Delay(us: Longword);
function  GetCurrentProjectType(): Integer; // see constants
```

4.4.3 Basic Declarations

Port I/O group

```
function GetPortByte(ByVal nPort As Integer) As Byte
function GetPortInteger(ByVal nPort As Integer) As Integer
function GetPortLong(ByVal nPort As Integer) As Long

sub SetPortByte(ByVal nPort As Integer, ByVal bValue As Byte)
sub SetPortInteger(ByVal nPort As Integer, ByVal wValue As Integer)
sub SetPortLong(ByVal nPort As Integer, ByVal dValue As Long)
```

Memory group

```
function MapPhysToLinear(ByVal PhAddr As Long, ByVal PhSize As Long) As
Long

function GetMemByte(ByVal MappedAddr As Long, ByVal Offset As Long) As Byte
function GetMemInteger(ByVal MappedAddr As Long, ByVal Offset As Long) As
Integer
function GetMemLong(ByVal MappedAddr As Long, ByVal Offset As Long) As Long

sub SetMemByte(ByVal MappedAddr As Long, ByVal Offset As Long, ByVal bValue
As Byte)
sub SetMemInteger(ByVal MappedAddr As Long, ByVal Offset As Long, ByVal
wValue As Integer)
sub SetMemLong(ByVal MappedAddr As Long, ByVal Offset As Long, ByVal dValue
As Long)
```

PCI group

```
function OpenPciDevice(ByRef Bus As Long, ByRef Dev As Long, ByRef Funct As
```

```
Long) As Integer
sub GetPciLocation(ByRef PciDevice As Long, ByRef Bus As Byte, ByRef Dev As
Byte, ByRef Funct As Byte)

function ReadFromPci(ByVal PciDevice As Long, ByVal Offset As Long, ByVal
Length As Long) As Long
sub WriteToPci (ByVal PciDevice As Long, ByVal Offset As Long, ByVal Length
As Long, ByVal Data As Long)

function GetPciBarPhysicalAddress(ByVal PciDevice As Long, ByVal BarNumber
As Integer) As Long

function ReadPciBarByte (ByVal PciDevice As Long, ByVal BarNumber As
Integer, ByVal BarOffset As Long) As Byte
function ReadPciBarWord (ByVal PciDevice As Long, ByVal BarNumber As
Integer, ByVal BarOffset As Long) As Integer
function ReadPciBarLong (ByVal PciDevice As Long, ByVal BarNumber As
Integer, ByVal BarOffset As Long) As Long

sub WritePciBarByte(ByVal PciDevice As Long, ByVal BarNumber As Integer,
ByVal BarOffset As Long, ByVal bValue As Byte)
sub WritePciBarWord(ByVal PciDevice As Long, ByVal BarNumber As Integer,
ByVal BarOffset As Long, ByVal wValue As Integer)
sub WritePciBarLong(ByVal PciDevice As Long, ByVal BarNumber As Integer,
ByVal BarOffset As Long, ByVal
dwValue As Long)

function ReadPciCommandReg(ByVal PciDevice As Long) AsInteger
sub WritePciCommandReg(ByVal PciDevice As Long, ByVal CommandRegValue As
Integer)
sub ControlPciCommandRegBits(ByVal PciDevice As Long, ByVal CommandRegBit
As Integer, ByVal SetClearFlag As Integer)
function ReadPciStatusReg(ByVal PciDevice As Long, ByVal ClearBitMask As
Integer) As Integer
```

USB group

```
function OpenUsbDevice(ByVal DevNumber As Byte) As Long;
function CloseUsbDevice(ByVal UsbDevice As Long) As Boolean;

function UsbTransfer(ByVal UsbDevice As Long;
                     ByVal PipeNumber As Byte;
                     ByVal ByRef Buffer As array of Byte;
                     ByRef BytesReturned As Long) As Boolean;

function UsbVendorRequest(ByVal UsbDevice As Long;
                     ByVal DirectionIn As Boolean;
                     ByVal RequestRecipient As Byte;
                     ByVal ReservedBits As Byte;
                     ByVal Request As Byte;
                     ByVal Value As Integer;
                     ByVal Index As Integer;
                     ByRef BufferAs array of Byte;
                      ByRef NumberOfBytesTransferred As Long) As Boolean;

function UsbClassRequest(ByVal UsbDevice As Long;
                     ByVal DirectionIn As Boolean;
                     ByVal RequestRecipient As Byte;
```

```
                    ByVal ReservedBits As Byte;
                    ByVal Request As Byte;
                    ByVal Value As Integer;
                    ByVal Index As Integer;
                    ByRef BufferAs array of Byte;
                    ByRef NumberOfBytesTransferred As Long) As Boolean;

function UsbSetFeature(ByVal UsbDevice As Long;
                    ByVal RequestRecipient As Byte;
                    ByVal FeatureSelector  As Integer;
                    ByVal Index            As Integer) As Boolean;

function UsbClearFeature(ByVal UsbDevice As Long;
                    ByVal RequestRecipient As Byte;
                    ByVal FeatureSelector  As Integer;
                    ByVal Index            As Integer) As Boolean;

function UsbGetStatus(ByVal UsbDevice As Long;
                    ByVal RequestRecipient As Byte;
                    ByVal Index            As Integer;
                    ByVal ByRef Status     As Integer) As Boolean;
```

Service Group

```
function HexByte(ByVal b As Byte) As String
function HexInteger(ByVal w As Integer) As String
function HexLong(ByVal d As Long) As String

sub HexPrint(ByVal sComment AsString, ByVal Val As Long)
sub DecPrint(ByVal sComment AsString, ByVal Val As Long)
sub StringPrint(ByVal s AsString)
sub HexDump(ByVal dump As Variant, ByVal NumOfBytes As Long)

sub Delay(ByVal us As LongInteger)
function GetCurrentProjectType() As Integer ' see constants
```

4.4.4 Constants

```
TYPE_HARDWARE_NONE   = 0
TYPE_HARDWARE_ISA    = 1
TYPE_HARDWARE_ISAPNP = 2
TYPE_HARDWARE_PCI    = 3
TYPE_HARDWARE_PCMCIA = 4
TYPE_HARDWARE_USB    = 5
TYPE_HARDWARE_LPT    = 6
TYPE_HARDWARE_1394   = 7
```

USB Request recipients

```
USB_REQUEST_RECIPIENT_DEVICE    = 0
USB_REQUEST_RECIPIENT_INTERFACE = 1
USB_REQUEST_RECIPIENT_ENDPOINT  = 2
USB_REQUEST_RECIPIENT_OTHER     = 3
```

5 RapidDriver Developer

RapidDriver Developer

Ver. 2.0

http://www.RapidDriver.com

tools@entechtaiwan.com

5.1 Programming For ISA Hardware

5.1.1 Overview

First of all, you must describe your device with ***RapidDriver Explorer*** then install a device using ISA bus specific driver (RapidIsa.sys). Then you may create and run your own application to control a ISA device for which the driver has been installed. All ISA functions/procedures are placed in RapidIsa.dll interface library.

You can install the RapidIsa.sys driver for many ISA devices, so every ISA device will have it's own *DeviceInstance* parameter value in OpenRapidIsa function.

Note: *In some cases we will use the "**RapidIsa**" term assuming the RapidDriver Developer edition when we are working with a ISA device through the RapidIsa.sys and RapidIsa.dll.*

Here is what you can do with RapidIsa:

- access to any I/O ports (8-, 16- and 32-bit), including reading/writing data arrays
- obtaining pointers to access specified physical memory addresses
- hardware interrupt handling

To work with the driver itself the choice of programming language is not vital. At present RapidDriver package includes support of the following programming systems:

- Microsoft Visual C/C++
- Borland Delphi
- Borland C++ Builder
- Microsoft Visual Basic 6.0
- Microsoft Visual Basic .Net
- Microsoft Visual C#

5.1.2 Programmers Guide

5.1.2.1 Scenario

Before using any of the functions of RapidIsa, the driver must be initialized with the help of the function OpenRapidIsa . It must be done once after starting the application. Also resources used by the driver should be cleaned up(driver closed) before the application is closed.

Schematic session of using RapidIsa:

```
// initialize(open) the driver

hIsa =  OpenRapidIsa( 0 );

// Check if the driver initialized successfully

if ( IsRapidIsaOpened(hIsa)) {
        ...
    ////////////////////////////////////
    ////////  RapidIsa session /////////
    ////////  ............... /////////
    ////////////////////////////////////

    // Free resources used by the driver (close the driver)

    hIsa = CloseRapidIsa(hIsa);

}
else
{
    // An error happened while opening the driver
    MessageBox("Cannot open the driver","Attention!", MB_OK |
    MB_ICONWARNING);
}
```

Driver can be initialized by many applications simultaneously, once for each ISA board. The parameter of OpenRapidIsa() can be used to select
a device supported by RapidIsa driver.

5.1.2.2 I/O Ports Control

I/O ports control function set includes two major groups of functions: - single input/output by byte (word, double word) and data array input/output.

- Single read/write operations
- Data array read/write operations

5.1.2.2.1 Single Read/Write Operations

The following set of functions is used to read/write single values through port. Their assignment is evident from names and requires no comments.

GetPortByte - read a byte from a port

GetPortWord - read a word from a port

GetPortLong - read a double word from a port

SetPortByte - write a byte to a port

SetPortWord - write a word to a port

SetPortLong - write a double word to a port

5.1.2.2.2 Data Array Read/Write operations

RapidIsa also contains two functions for read/write data arrays from/to port. These functions are based on ***insb/insw/insd*** assembly input commands and ***outsb/outsw/outsd*** output commands.

ReadPortBuffer - reads a number of **bytes**, **words**, or **double words** from the specified port address into a buffer.

WritePortBuffer - write an array of bytes

Remember, these functions don't provide another exchange algorithm ("handshaking"). They are used only for devices with hardware support of group input/output operations.

5.1.2.3 Accessing Physical Memory Addresses

Memory Mapping

Additional functions

5.1.2.3.1 Memory Mapping

To access specified physical memory address you have to map this physical address to a linear memory of the current process' address space. RapidIsa includes the function MapPhysToLinear, which performs the above task and returns the pointer to the above linear memory defined by it's physical address and size. When the pointer is not needed by application any more, this physical memory area can and must be unmapped with the function UnmapMemory.

The following is an example how to get a pointer to the ROM BIOS area:

```
char *pBios;
HANDLE hIsa = NULL;
hIsa = OpenRapidIsa( 0 );
if (IsRapidIsaOpened(hIsa))
{

  pBios = (char*)MapPhysToLinear(hIsa,0xF8000,256); //256 bytes, starting
from 0xF8000

  //...pBIOS session...

  UnmapMemory(hIsa, 0xF8000,256); // undo mapping
  hIsa = CloseRapidIsa(hIsa);     //  close the driver
}
else
  ... // failed
```

5.1.2.3.2 Additional Functions

As it was mentioned, the **MapPhysToLinear** function returns the pointer to the linear memory defined by it's physical address and size. You can work with a returned pointer directly or by means of additional functions set which provides physical memory access by bytes, by words or by double words. It's especially useful for Visual Basic programmers as Visual basic has some limitations on working with a pointers directly.

Below is the list of the above mentioned functions:

```
UCHAR  GetMem  (HANDLE hIsa, ULONG LinAddr, ULONG Offset);
USHORT GetMemW (HANDLE hIsa, ULONG LinAddr, ULONG Offset);
ULONG  GetMemL (HANDLE hIsa, ULONG LinAddr, ULONG Offset);
void   SetMem  (HANDLE hIsa, ULONG LinAddr, ULONG Offset, UCHAR bValue);
void   SetMemW (HANDLE hIsa, ULONG LinAddr, ULONG Offset, USHORT wValue);
void   SetMemL (HANDLE hIsa, ULONG LinAddr, ULONG Offset, ULONG dValue);
```

5.1.2.4 Hardware Interrupts Handling

Function UnmaskIsaIrq installs a hardware interrupt handler. It works as follows:

```
ULONG IrqCounter = 0;

// Interrupt handler
void __stdcall MyInterruptHandler(ULONG TimeStampLoPart, ULONG
TimeStampHiPart)
{
    ++ IrqCounter;
}

// Main program
hIsa = OpenRapidIsa( 0 );
if (IsRapidIsaOpened(hIsa)) {
    ...
    set ClearRec and ShareRec structures, see below
    ...
    UnmaskIsaIrq( hIsa, MyInterruptHandler );
}
else
  ... // failed
```

Calling the function MaskIsaIrq stops hardware interrupt processing:

```
MaskIsaIrq ( hIsa );
```

Counter value for the number of handled interrupts is returned by function GetInterruptCounter.

5.1.3 Common Functions

Enter topic text here.

5.1.3.1 OpenRapidIsa

Opens the RapidIsa.sys kernel mode WDM device driver under Windows 2000/XP/2003, providing direct access to the ISA hardware.

Language	Description
C/C++	HANDLE OpenRapidIsa (ULONG DeviceInstance)
Delphi	function OpenRapidIsa (DeviceInstance : Longword) : THandle; stdcall
VB	Function OpenRapidIsa (ByVal DeviceInstance As Long) As Long

Parameters:
DeviceInstance - ISA device instance.

Return Value:
The RapidIsa handle value or NULL if fails. Keep this handle opened all time when you are working with RapidIsa functions and close by CloseRapidIsa() at end of application.

Comments:

If the driver was successfully opened, the IsRapidIsaOpened() returns True; if the function fails, the IsRapidIsaOpened() returns False.
With RapidIsa you can support as many ISA devices as you want. To select the device you need change DriverInstance value from 0 up to maximal number of ISA device installed with RapidIsa driver. For the first application use

```
hIsa = OpenRapidIsa(0);
```

For second:

```
hIsa = OpenRapidIsa(1);
```

and so on.

See also: CloseRapidIsa IsRapidIsaOpened

5.1.3.2 CloseRapidIsa

Closes the kernel-mode driver and releases memory allocated to it. If a hardware interruptions was "unmasked", the "mask" is restored.

Language	Description
C/C++	HANDLE CloseRapidIsa (HANDLE hIsa)
Delphi	function CloseRapidIsa (hIsa: THandle): THandle; stdcall
VB	Function CloseRapidIsa (ByVal hIsa As Long) As Long

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa

Return Value :
NULL.

See also: OpenRapidIsa IsRapidIsaOpened

5.1.3.3 IsRapidIsaOpened

This boolean function specifies whether the driver is open.

Language	Description
C/C++	BOOL IsRapidIsaOpened (HANDLE hIsa)
Delphi	function IsRapidIsaOpened (hIsa : THandle) : BOOL; stdcall
VB	Function IsRapidIsaOpened (ByVal hIsa As Long) As Bool

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa

Return Value :
TRUE if the driver is already open by OpenRapidIsa(), or FALSE if it is not.

Comments:
Use immediately after OpenRapidIsa() call to check if the driver was successfully opened or not.

See also: OpenRapidIsa CloseRapidIsa

5.1.3.4 GetHardwareConfiguration

Reads the hardware configuration (resources) of the ISA device.

Language	Description
C/C++	void GetHardwareConfuguration (HANDLE hIsa, HW_DEV_CONFIG * Cfg);
Delphi	procedure GetHardwareConfuguration (hIsa: THandle, var Cfg: HW_DEV_CONFIG); stdcall
VB	Sub GetHardwareConfuguration (ByVal hIsa As Long, ByRef Cfg As HW_DEV_CONFIG)

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa
Cfg - hardware resources of the ISA device

Return Value :
None

5.1.4 Direct Port I/O

5.1.4.1 GetPortByte

Reads a byte from the specified port address.

Language	Description
C/C++	UCHAR GetPortByte (HANDLE hIsa, ULONG PortAddress);
Delphi	function GetPortByte (hIsa: THandle; PortAddress: Longword): Byte; stdcall;
VB	Function GetPortByte (ByVal hIsa As Long, ByVal PortAddress As Long) As Byte

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa ;
PortAddress - the port address in I/O space

Return Value :
The byte read from the specified port address.

See also: GetPortWord GetPortLong SetPortByte SetPortWord SetPortLong

5.1.4.2 GetPortWord

Reads a WORD (two bytes) from the specified port address.

Language	Description
C/C++	USHORT GetPortWord (HANDLE hIsa, ULONG PortAddress);
Delphi	function GetPortWord (hIsa: THandle; PortAddress: Longword): Word; stdcall;
VB	Function GetPortWord (ByVal hIsa As Long, ByVal PortAddress As Long) As Integer

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa ;
PortAddress - the port address in I/O space

Return Value :
The WORD value read from the specified port address

See also: GetPortByte GetPortLong SetPortByte SetPortWord SetPortLong

5.1.4.3 GetPortLong

Reads a double word (four bytes) from the specified port address.

Language	Description
C/C++	ULONG GetPortLong (HANDLE hIsa, ULONG PortAddress);
Delphi	function GetPortLong (hIsa: THandle; PortAddress: Longword): Longword; stdcall;
VB	Function GetPortLong (ByVal hIsa As Long, ByVal PortAddress As Long) As Long

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa ;
PortAddress - the port address in I/O space

Return Value :
The double word value read from the specified port address.

See also: GetPortByte GetPortWord SetPortByte SetPortWord SetPortLong

5.1.4.4 SetPortByte

Writes a byte to the specified port address.

Language	Description
C/C++	void SetPortByte (HANDLE hIsa, ULONG PortAddress, UCHAR bValue);
Delphi	procedure SetPortByte (hIsa: THandle; PortAddress: Longword; bValue: Byte); stdcall;
VB	Sub SetPortByte (ByVal hIsa As Long, ByVal PortAddress As Long, ByVal bValue As Byte)

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa ;
PortAddress - the port address in I/O space
b*Value* - specifies a byte to be written to the port.

Return Value : None.

See also: GetPortByte GetPortWord GetPortLong SetPortWord SetPortLong

5.1.4.5 SetPortWord

Writes a word (two bytes) value to the specified port address.

Language	Description
C/C++	void SetPortWord (HANDLE hIsa, ULONG PortAddress, USHORT wValue);
Delphi	procedure SetPortWord (hIsa: THandle; PortAddress: Longword; wValue: Word); stdcall;
VB	Sub SetPortWord (ByVal hIsa As Long, ByVal PortAddress As Long, ByVal wValue As Integer)

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa ;
PortAddress - the port address in I/O space
w*Value* - specifies a word to be written to the port.

Return Value : None.

See also: GetPortByte GetPortWord GetPortLong SetPortByte SetPortLong

5.1.4.6 SetPortLong

Writes a double word (four bytes) value to the specified port address.

Language	Description
C/C++	void SetPortLong (HANDLE hIsa, ULONG PortAddress, ULONG dwValue);
Delphi	procedure SetPortLong (hIsa: THandle; PortAddress: Longword; dwValue: Longword); stdcall;
VB	Sub SetPortLong (ByVal hIsa As Long, ByVal PortAddress As Long, ByVal dwValue As Long)

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa ;
PortAddress - the port address in I/O space
dw*Value* - specifies a double word to be written to the port.

Return Value : None.

See also :GetPortByte GetPortWord GetPortLong SetPortByte SetPortWord

5.1.4.7 ReadPortBuffer

Reads a number of **bytes**, **words**, or **double words** from the specified port address into a buffer.

Language	Description
C/C++	void ReadPortBuffer(HANDLE hIsa, ULONG PortAddr, USHORT PortWidth, ULONG NumValues, void * buffer);
Delphi	procedure ReadPortBuffer (hIsa: THandle; PortAddr: LongWord; PortWidth: Word; NumValues: Longword; buffer: Pointer); stdcall;
VB	Sub ReadPortBuffer (ByVal hIsa As Long, ByVal PortAddr As Long, ByRef PortWidth As Integer; ByVal PortWidth As Long, ByRef buffer As Any)

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa ;
PortAddr - port address;
PortWidth - size of port in bytes (1, 2, or 4);
NumValue - number of values to read
buffer - a buffer into which an array of values is read.

Return Value :None.

See also : WritePortBuffer

5.1.4.8 WritePortBuffer

Writes a number of **bytes**, **words**, or **double words** from a buffer to the specified port.

Language	Description
C/C++	void WritePortBuffer(HANDLE hIsa, ULONG PortAddr, USHORT PortWidth, ULONG NumValues, void * buffer);
Delphi	procedure WritePortBuffer (hIsa: THandle; PortAddr: LongWord; PortWidth: Word; NumValues: Longword; buffer: Pointer); stdcall;
VB	Sub WritePortBuffer (ByVal hIsa As Long, ByVal PortAddr As Long, ByRef PortWidth As Integer; ByVal PortWidth As Long, ByRef buffer As Any)

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa ;
PortAddr - port address;
PortWidth - size of port in bytes (1, 2, or 4);
NumValue - number of values to read
buffer - a buffer from which an array of values is to be written.

Return Value :None.

See also: ReadPortBuffer

5.1.5 Memory Access

5.1.5.1 MapPhysToLinear

Maps a specific physical address to a pointer in linear memory for the current process' address space. This function is intended to be used to access
device-specific physical memory (for the memory-mapped devices).

Language	Description
C/C++	void * MapPhysToLinear (HANDLE hIsa, ULONG PhysAddr, ULONG MappedSize);
Delphi	function MapPhysToLinear (hIsa: THandle; PhysAddr:Longword; MappedSize: Longword): Pointer; stdcall;
VB	Function MapPhysToLinear (ByVal hIsa As Long, ByVal PhysAddr As Long, ByVal MappedSize As Long) As Long

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa
PhysAddr - 32-bit physical address of the start of the region to examine
MappedSize - length in bytes of the physical region. The function uses this parameter to verify that the entire range is addressable.

Return Value :
The linear address of the first byte in the specified range of physical addresses.

Comments:
1. Do not forget to unmap this physical memory area by UnmapMemory when your application not need it anymore.
2. No information can be concluded about the linear addresses for physical addresses if they lie outside the range requested.

See also : UnmapMemory

5.1.5.2 UnmapMemory

Unmaps a specific physical address previously mapped with MapPhysToLinear.

Language	Description
C/C++	void UnmapMemory (HANDLE hIsa, ULONG PhysAddr, ULONG MappedSize);
Delphi	procedure UnmapMemory (hIsa: THandle; PhysAddr: Longword; MappedSize: Longword); stdcall
VB	Sub UnmapMemory (ByVal hIsa As Long, ByVal PhysAddr As Long, ByVal MappedSize As Long)

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa
PhysAddr - 32-bit physical address of the start of the region previously mapped
MappedSize - length in bytes of the mapped region

Return Value :
None.

See also : MapPhysToLinear

5.1.5.3 GetMem

Reads **one byte** from the memory region. Mainly is intended for using in the Visual Basic applications where the direct manipulations with the pointers is difficult.

Language	Description
C/C++	UCHAR GetMem (HANDLE hIsa, ULONG MappedAddress, ULONG MemOffset);
Delphi	function GetMem (hIsa: THandle; MappedAddress: Longword; MemOffset: Longword): Byte; stdcall;
VB	Function GetMem (ByVal hIsa As Long, ByVal MappedAddress As Long, ByVal MemOffset As Long) As Byte

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa ;
MappedAddress - the linear address returned by MapPhysToLinear
MemOffset **-** the offset value, should begin from the "0" value.

Return Value :
One byte value from the region.

Comments:

In general, "MappedAddress" can be any valid linear pointer to the memory. But if this pointer was received from the MapPhysToLinear function then you can access physical memory also.

See also: MapPhysToLinear UnmapMemory GetMemW GetMemL SetMem SetMemW SetMemL

5.1.5.4 GetMemW

Reads **two bytes (word)** from the memory region. Mainly is intended for using in the Visual Basic applications where the direct manipulations with the pointers is difficult.

Language	Description
C/C++	USHORT GetMemW (HANDLE hIsa, ULONG MappedAddress, ULONG MemOffset);
Delphi	function GetMemW (hIsa: THandle; MappedAddress: Longword; MemOffset: Longword): Word; stdcall;
VB	Function GetMemW (ByVal hIsa As Long, ByVal MappedAddress As Long, ByVal MemOffset As Long) As Integer

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa ;
MappedAddress - the linear address returned by MapPhysToLinear
MemOffset **-** the offset value, should begin from the "0" value.

Return Value :
16-bit value from the region.

Comments:

In general, "MappedAddress" can be any valid linear pointer to the memory. But if this pointer was received from the MapPhysToLinear function then you can access physical memory also.

See also: MapPhysToLinear UnmapMemory GetMem GetMemL SetMem SetMemW SetMemL

5.1.5.5 GetMemL

Reads **four bytes (dword)** from the memory region. Mainly is intended for using in the Visual Basic applications where the direct manipulations with the pointers is difficult.

Language	Description
C/C++	ULONG GetMemL (HANDLE hIsa, ULONG MappedAddress, ULONG MemOffset);
Delphi	function GetMemL (hIsa: THandle; MappedAddress: Longword; MemOffset: Longword): Longword; stdcall;
VB	Function GetMemL (ByVal hIsa As Long, ByVal MappedAddress As Long, ByVal MemOffset As Long) As Long

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa ;
MappedAddress - the linear address returned by MapPhysToLinear
MemOffset **-** the offset value, should begin from the "0" value.

Return Value :
32-bit value from the region.

Comments:

In general, "MappedAddress" can be any valid linear pointer to the memory. But if this pointer was received from the MapPhysToLinear function then you can access physical memory also.

See also: MapPhysToLinear UnmapMemory GetMem GetMemW SetMem SetMemW SetMemL

5.1.5.6 SetMem

Write **one byte** to the memory region. Mainly is intended for using in the Visual Basic applications where the direct manipulations with the pointers is difficult.

Language	Description
C/C++	void SetMem (HANDLE hIsa, ULONG MappedAddress, ULONG MemOffset , UCHAR bValue);
Delphi	procedure SetMem (hIsa: THandle; MappedAddress: Longword; MemOffset: Longword; bValue: Byte); stdcall;
VB	Sub SetMem (ByVal hIsa As Long, ByVal MappedAddress As Long, ByVal MemOffset As Long, bValue As Byte)

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa ;
MappedAddress - the linear address returned by MapPhysToLinear
MemOffset **-** the offset value, should begin from the "0" value.
bValue - value to write

Return Value : None

Comments:

In general, "MappedAddress" can be any valid linear pointer to the memory. But if this pointer was received from the MapPhysToLinear function then you can access physical memory also.

See also: MapPhysToLinear UnmapMemory GetMem GetMemW GetMemL SetMemW SetMemL

5.1.5.7 SetMemW

Write **two bytes (word)** to the memory region. Mainly is intended for using in the Visual Basic applications where the direct manipulations with the pointers is difficult.

Language	Description
C/C++	void SetMemW (HANDLE hIsa, ULONG MappedAddress, ULONG MemOffset , USHORT wValue);
Delphi	procedure SetMemW (hIsa: THandle; MappedAddress: Longword; MemOffset: Longword; wValue: Word); stdcall;
VB	Sub SetMemW (ByVal hIsa As Long, ByVal MappedAddress As Long, ByVal MemOffset As Long, wValue As Integer)

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa ;
MappedAddress - the linear address returned by MapPhysToLinear
MemOffset **-** the offset value, should begin from the "0" value.
wValue - value to write

Return Value : None

Comments:

In general, "MappedAddress" can be any valid linear pointer to the memory. But if this pointer was received from the MapPhysToLinear function then you can access physical memory also.

See also: MapPhysToLinear UnmapMemory GetMem GetMemW GetMemL SetMem SetMemL

5.1.5.8 SetMemL

Write **four bytes (double word)** to the memory region. Mainly is intended for using in the Visual Basic applications where the direct manipulations with the pointers is difficult.

Language	Description
C/C++	void SetMemL (HANDLE hIsa, ULONG MappedAddress, ULONG MemOffset , ULONG dwValue);
Delphi	procedure SetMemL (hIsa: THandle; MappedAddress: Longword; MemOffset: Longword; dValue: Longword); stdcall;
VB	Sub SetMemL (ByVal hIsa As Long, ByVal MappedAddress As Long, ByVal MemOffset As Long, dwValue As Long)

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa ;
MappedAddress - the linear address returned by MapPhysToLinear
MemOffset **-** the offset value, should begin from the "0" value.
d*wValue* - value to write

Return Value : None

Comments:

In general, "MappedAddress" can be any valid linear pointer to the memory. But if this pointer was received from the MapPhysToLinear function then you can access physical memory also.

See also: MapPhysToLinear UnmapMemory GetMem GetMemW GetMemL SetMem SetMemW

5.1.6 Hardware Interrupts

5.1.6.1 UnmaskIsaIrq

Installs a hardware interrupt handler for the ISA device and unmasks the Irq at hardware level. UnmaskIsaIrq allows to handle the
"shared" and "level sensitive" interrupts as well as the "non-shared" and "edge triggered".

Language	Description
C/C++	void UnmaskIsaIrq(HANDLE hIsa, TOnHwInterrupt InterruptHandler);
Delphi	procedure UnmaskIsaIrq (hIsa: THandle; InterruptHandler: TOnHwInterruptHandler); stdcall;
VB	Sub UnmaskIsaIrq (ByVal hIsa As Long, ByVal HWHandler As Long)

Parameters:

hIsa - the handle returned by a successful call to OpenRapidIsa ;
InterruptHandler - address of the callback procedure that handles hardware interrupts for this Irq (use AddressOf specification in VB)

Return Value : None.

Comments:

The interrupt handler function must be declared as follows:

C/C++: typedef void (__stdcall * TOnHwInterrupt)(ULONG TimeStampLoPart, ULONG TimeStampHiPart);

Delphi: procedure OnHwInterrupt(TimeStampLoPart: Longword; TimeStampHiPart: Longword); stdcall;

VB: Sub OnHwInterrupt(ByVal TimeStampLoPart As Long, ByVal TimeStampHiPart As Long)

The *TimeStampLoPart/TimeStampHiPart* values are parts of value returned by the kernel equivalent of QueryPerformaceCounter() API function.

See also: MaskIsaIrq GetInterruptCounter

5.1.6.2 MaskIsaIrq

Stops the interrupt processing started by UnmaskIsaIrq by masking the ISA Irq at hardware level.

Language	Description
C/C++	void MaskIsaIrq(HANDLE hIsa);
Delphi	procedure MaskIsaIrq (hIsa: THandle); stdcall;
VB	Sub MaskIsaIrq (ByVal hIsa As Long)

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa ;

Return Value : None.

See also: UnmaskIsaIrq GetInterruptCounter

5.1.6.3 GetInterruptCounter

Returns how many interruptions was processed inside of the driver.

Language	Description
C/C++	ULONG GetInterruptCounter(HANDLE hIsa);
Delphi	function GetInterruptCounter (hIsa: THandle): Longword; stdcall;
VB	Function GetInterruptCounter (ByVal hIsa As Long) As Long

Parameters:
hIsa - the handle returned by a successful call to OpenRapidIsa ;

Return Value :
Number of the interruptions processed by the drive.

See also: UnmaskIsaIrq MaskIsaIrq

5.2 Programming For PCI Device

5.2.1 Overview

First of all, you must describe your device with ***RapidDriver Explorer*** then install a device using PCI bus specific driver (RapidPci.sys). Then you may create and run your own application to control a PCI device for which the driver has been installed. All PCI functions/procedures are placed in RapidPci.dll interface library.

You can install the RapidPci.sys driver for many PCI devices, so every PCI device will have it's own *DeviceInstance* parameter value in OpenRapidPci function.

Note: *In some cases we will use the "**RapidPci**" term assuming the RapidDriver Developer edition when we are working with a PCI device through the RapidPci.sys and RapidPci.dll.*

Here is what you can do with RapidPci:

- access to PCI device configuration space, including reading/writing separate registers
- working with the base address registers (BARs)
- access to any I/O ports (8-, 16- and 32-bit), including reading/writing data arrays
- obtaining pointers to access specified physical memory addresses
- hardware interrupt handling

To work with the driver itself the choice of programming language is not vital. At present RapidDriver package includes support of the following programming systems:

- Microsoft Visual C/C++
- Borland Delphi
- Borland C++ Builder
- Microsoft Visual Basic 6.0
- Microsoft Visual Basic .Net
- Microsoft Visual C#

5.2.2 Programmers Guide

5.2.2.1 Scenario

Before using any of the functions of RapidPci, the driver must be initialized with the help of the function OpenRapidPci . It must be done once after starting the application. Also resources used by the driver should be cleaned up(driver closed) before the application is closed.

Schematic session of using RapidPci:

```
// initialize(open) the driver

hPci =  OpenRapidPci( 0 );

// Check if the driver initialized successfully

if ( IsRapidPciOpened(hPci)) {
        ...
   ////////////////////////////////////
   /////////  RapidPci session /////////
   /////////  ............... /////////
   ////////////////////////////////////

   // Free resources used by the driver (close the driver)

   hPci = CloseRapidPci(hPci);

}
else
{
   // An error happened while opening the driver
   MessageBox("Cannot open the driver","Attention!", MB_OK |
   MB_ICONWARNING);
}
```

Driver can be initialized by many applications simultaneously, once for each PCI board. The parameter of OpenRapidPci() can be used to select
a device supported by RapidPci driver.

5.2.2.2 Access to PCI devices information

5.2.2.2.1 How to work with PCI header

With RapidPci you can access PCI header of the device's configuration space (see PCI_COMMON_CONFIG structure).

GetPciHeader - reads data from the header of the device's configuration space.

SetPciHeader - you can change any field of PCI configuration space then write it back.

ReadFromPci - reads byte, word, or double word from the PCI configuration space

WriteToPci - writes byte, word, or double word to the PCI configuration space

5.2.2.2.2 Accessing PCI registers

You can access PCI registers (Command, Status, Interrupt - see PCI_COMMON_CONFIG) with the following set of functions:

ReadPciCommandReg - read the PCI command register
WritePciCommandReg - write the PCI command register
ControlPciCommandRegBits - set or clear a bit in the PCI command register
ReadPciStatusReg - read the PCI status register

5.2.2.2.3 Accessing BAR0-BAR5 areas

The PCI configuration space contains an array of base address registers (BARs). Normal PCI devices have 6 base address registers, while PCI-PCI bridges have 2 and PCI-CardBus bridges have 1 (see PCI_COMMON_CONFIG structure). These registers are used to determine and allocate the type, amount and location of PCI I/O and PCI memory space that the device can use.

RapidPci provides a set of functions to access these spaces:

ReadPciBarByte - read a byte from the specified BAR + offset
ReadPciBarWord- read a 16-bit word from the specified BAR + offset
ReadPciBarLong - read a 32-bit double word from the specified BAR + offset

WritePciBarByte - write a byte to the specified BAR + offset
WritePciBarWord - write a 16-bit word to the specified BAR + offset
WritePciBarLong - write a 32-bit double word to the specified BAR + offset

Important note: These functions serves all requests by regardless if the base address is memory address or port I/O address.

5.2.2.3 I/O Ports control

I/O ports control function set includes two major groups of functions: - single input/output by byte (word, double word) and data array input/output.

- Single read/write operations
- Data array read/write operations

5.2.2.3.1 Single read/write operations

The following set of functions is used to read/write single values through port. Their assignment is evident from names and requires no comments.

GetPortByte - read a byte from a port
GetPortWord - read a word from a port
GetPortLong - read a double word from a port
SetPortByte - write a byte to a port
SetPortWord - write a word to a port
SetPortLong - write a double word to a port

5.2.2.3.2 Data array read/write operations

RapidPci also contains two functions for read/write data arrays from/to port. These functions are based on ***insb/insw/insd*** assembly input commands and ***outsb/outsw/outsd*** output commands.

ReadPortBuffer - reads a number of **bytes**, **words**, or **double words** from the specified port address into a buffer.
WritePortBuffer - write an array of bytes

Remember, these functions don't provide another exchange algorithm ("handshaking"). They are used only for devices with hardware support of group input/output operations.

5.2.2.4 Accessing Physical Memory Addresses

Memory Mapping
Additional functions

5.2.2.4.1 Memory Mapping

To access specified physical memory address you have to map this physical address to a linear memory of the current process' address space. RapidPci includes the function MapPhysToLinear, which performs the above task and returns the pointer to the above linear memory defined by it's physical address and size. When the pointer is not needed by application any more, this physical memory area can and must be unmapped with the function UnmapMemory.
The following is an example how to get a pointer to the ROM BIOS area:

```
char *pBios;
HANDLE hPci = NULL;
hPci = OpenRapidPci( 0 );
if (IsRapidPciOpened(hPci)) {

  pBios = (char*)MapPhysToLinear(hPci,0xF8000,256); //256 bytes, starting
from 0xF8000

  //...pBIOS session...

  UnmapMemory(hPci, 0xF8000,256); // undo mapping
  hPci = CloseRapidPci(hPci);     //  close the driver
}
else
  ... // failed
```

5.2.2.4.2 Additional Functions

As it was mentioned, the **MapPhysToLinear** function returns the pointer to the linear memory defined by it's physical address and size. You can work with a returned pointer directly or by means of additional functions set which provides physical memory access by bytes, by words or by double words. It's especially useful for Visual Basic programmers as Visual basic has some limitations on working with a pointers directly.

Below is the list of the above mentioned functions:

```
UCHAR  GetMem  (HANDLE hPci, ULONG LinAddr, ULONG Offset);
USHORT GetMemW (HANDLE hPci, ULONG LinAddr, ULONG Offset);
ULONG  GetMemL (HANDLE hPci, ULONG LinAddr, ULONG Offset);
void   SetMem  (HANDLE hPci, ULONG LinAddr, ULONG Offset, UCHAR bValue);
void   SetMemW (HANDLE hPci, ULONG LinAddr, ULONG Offset, USHORT wValue);
void   SetMemL (HANDLE hPci, ULONG LinAddr, ULONG Offset, ULONG dValue);
```

5.2.2.5 Hardware interrupts handling at user level

Common issues
Options of hardware interrupt handling

5.2.2.5.1 Common Issues

Function UnmaskPciIrq installs a hardware interrupt handler. It works as follows:

```
ULONG IrqCounter = 0;
PIRQ_SHARE_REC ShareRec;
PIRQ_CLEAR_REC ClearRec;

// Interrupt handler
void __stdcall MyInterruptHandler(ULONG TimeStampLoPart, ULONG
TimeStampHiPart)
{
    ++ IrqCounter;
}

// Main program
hPci = OpenRapidPci( 0 );
if (IsRapidPciOpened(hPci)) {
    ...
    set ClearRec and ShareRec structures, see below
    ...
    UnmaskPciIrq( hPci, &ShareRec, &ClearRec, MyInterruptHandler );
}
else
  ... // failed
```

Calling the function MaskPciIrq stops hardware interrupt processing:

```
MaskPciIrq ( hPci );
```

Counter value for the number of handled interrupts is returned by function GetInterruptCounter.

.

5.2.2.5.2 Extended options of hardware interrupt handling

As it was mentioned earlier, there are two main problems when we are trying to handle PCI interruption.
First, all PCI interruptions are shared. It means that we must provide information for driver to determine if it is our interruption or not.
Second, it's very important for level sensitive interrupts to inactivate bus interrupt level just after it's activation (clear interruption or acknowledge interruption, in other words).

Let's assume that we have to write 0x0F0F to some register for interrupt acknowledge. Register is a word displacement 0x320 in the memory area with the physical address 0x3C000:

```
  IRQ_CLEAR_REC ClearRec;  // Clear control structure

  IRQ_SHARE_REC ShareRec;  // sharing control structure

  memset(&ClearRec,0,sizeof(ClearRec)); // clean structures
  memset(&ShareRec,0,sizeof(ShareRec));

  ClearRec.ClearIrq = 1; // Interrupt acknowledge needed
  ClearRec.TypeOfRegister = 1; // Register type - memory-mapped
  ClearRec.WideOfRegister = 2; // Register size - 2 bytes (a word)
  ClearRec.ReadOrWrite = 1;    // acknowledge means write to a register
  ClearRec.RegBaseAddress = 0x3C000;  // base physical memory address
```

```
ClearRec.RegOffset = 0x320; // register first byte displacement in memory

ClearRec.ValueToWrite = 0x0F0F;    // A value to write to a register for
interrupt acknowledge
```

Also assume we should see 1 in bit 5 of memory mapped register if it is our interruption and the driver must handle it.

```
ShareRec.ShareIrq = 1; // Irq can be shared
ShareRec.TypeOfRegister = 0; // memory-mapped register
ShareRec.WideOfRegister = 1; // 1 - Byte
SharRec.RegBaseAddress = PhysAddr; // Memory register base address
ShareRec.RegOffset = 0x40; // Register offset
ShareRec.MaskValue = 0x20; // Bitmap mask to make AND operation
ShareRec.ResultMask = 0x20; // What should be in result if it is our
interrupt

// Set an interrupt handler and unmask Irq
UnmaskPciIrq(hPci,&ShareRec, &ClearRec,MyInterruptHandler);
```

5.2.3 PCI Support Routines And Structures

5.2.3.1 Common Procedures

5.2.3.1.1 OpenRapidPci

Opens the RapidPci.sys kernel mode WDM device driver under Windows 2000/XP/2003, providing direct access to the PCI hardware.

Language	Description
C/C++	HANDLE OpenRapidPci (ULONG DeviceInstance)
Delphi	function OpenRapidPci (DeviceInstance : Longword) : THandle; stdcall
VB	Function OpenRapidPci (ByVal DeviceInstance As Long) As Long

Parameters:
DeviceInstance - PCI device instance.

Return Value:
The RapidPci handle value or NULL if fails. Keep this handle opened all time when you are working with RapidPci functions and close by CloseRapidPci() at end of application.

Comments:

If the driver was successfully opened, the IsRapidPciOpened() returns True; if the function fails, the IsRapidPciOpened() returns False.
With RapidPci you can support as many PCI devices as you want. To select the device you need change DriverInstance value from 0 up to maximal number of PCI device installed with RapidPci driver. For the first application use

```
hPci = OpenRapidPci(0);
```

For second:

```
hPci = OpenRapidPci(1);
```

and so on.

See also: CloseRapidPci IsRapidPciOpened

5.2.3.1.2 IsRapidPciOpened

This boolean function specifies whether the mode driver is open.

Language	Description
C/C++	BOOL IsRapidPciOpened (HANDLE hPci)
Delphi	function IsRapidPciOpened (hPci : THandle) : BOOL; stdcall
VB	Function IsRapidPciOpened (ByVal hPci As Long) As Bool

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci

Return Value :
TRUE if the driver is already open by OpenRapidPci(), or FALSE if it is not.

Comments:
Use immediately after OpenRapidPci() call to check if the driver was successfully opened or not.

See also: OpenRapidPci CloseRapidPci

5.2.3.1.3 CloseRapidPci

Closes the kernel-mode driver and releases memory allocated to it. If a hardware interruptions was "unmasked", the "mask" is restored.

Language	Description
C/C++	HANDLE CloseRapidPci (HANDLE hPci)
Delphi	function CloseRapidPci (hPci: THandle): THandle; stdcall
VB	Function CloseRapidPci (ByVal hPci As Long) As Long

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci

Return Value :
NULL.

See also: OpenRapidPci IsRapidPciOpened

5.2.3.1.4 GetHardwareConfiguration

Reads the hardware configuration (resources) of the PCI device.

Language	Description
C/C++	void GetHardwareConfuguration (HANDLE hPci, HW_DEV_CONFIG * Cfg);
Delphi	procedure GetHardwareConfuguration (hPci: THandle, var Cfg: HW_DEV_CONFIG); stdcall
VB	Sub GetHardwareConfuguration (ByVal hPci As Long, ByRef Cfg As HW_DEV_CONFIG)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
Cfg - hardware resources of the PCI device

Return Value :
None

Comments:
The hardware configuration is not the same as PCI header.

5.2.3.1.5 GetPciLocation

Reads the location of the device at PCI bus (bus, device, function).

Language	Description
C/C++	void GetPciLocation (HANDLE hPci, PCI_LOCATION * Location);
Delphi	procedure GetPciLocation (hPci: THandle, var Location: PCI_LOCATION); stdcall
VB	Sub GetPciLocation (ByVal hPci As Long, ByRef Location As PCI_LOCATION)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
Location - hardware resources of the PCI device

Return Value :
None

5.2.3.2 PCI specific functions

5.2.3.2.1 PCI Header

5.2.3.2.1.1 GetPciHeader

Reads data from the header of the device's PCI configuration space into the supplied PciConfig structure.

Language	Description
C/C++	BOOL GetPciHeader(HANDLE hPci, PCI_COMMON_CONFIG * PciConfig, USHORT wOffset, USHORT wBytes);
Delphi	function GetPciHeader (hPci: THandle; var PciConfig: PCI_COMMON_CONFIG; wOffset: Word; wBytes: Word): BOOL; stdcall;
VB	Function GetPciHeader (ByVal hPci As Long, ByRef PciConfig as PCI_COMMON_CONFIG, ByVal wOffset As Integer, ByVal wBytes As Integer) As Bool

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
PciConfig - points to the requested information, see PCI_COMMON_CONFIG structure
wOffset - offset in PCI header, in bytes.
wBytes - number of bytes to read.

Return Value :
TRUE, if the device exists or FALSE if not.

See also: SetPciHeader

5.2.3.2.1.2 SetPciHeader

Writes data to the header of the device's PCI configuration space from the supplied buffer.

Language	Description
C/C++	BOOL SetPciHeader(HANDLE hPci, PCI_COMMON_CONFIG * PciConfig, USHORT wOffset, USHORT wBytes);
Delphi	function SetPciHeader (hPci: THandle; var PciConfig: PCI_COMMON_CONFIG; wOffset: Word; wBytes: Word): BOOL; stdcall;
VB	Function SetPciHeader (ByVal hPci As Long, ByRef PciConfig as PCI_COMMON_CONFIG, ByVal wOffset As Integer, ByVal wBytes As Integer) As Bool

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
PciConfig - points to the requested information, see PCI_COMMON_CONFIG structure
wOffset - offset in PCI header, in bytes.
wBytes - number of bytes to write.

Return Value :
TRUE, if the device exists or FALSE if not.

See also: GetPciHeader

5.2.3.2.1.3 ReadFromPci

Reads byte, word, or double word value data from the header of the device's PCI configuration space.

Language	Description
C/C++	ULONG ReadFromPci(HANDLE hPci, USHORT wOffset, USHORT wBytes);
Delphi	function ReadFromPci (hPci: THandle; wOffset: Word; wBytes: Word): Longword; stdcall;
VB	Function ReadFromPci (ByVal hPci As Long, ByVal wOffset As Integer, ByVal wBytes As Integer) As Long

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
PciConfig - points to the requested information, see PCI_COMMON_CONFIG structure
wOffset - offset in PCI header, in bytes.
wBytes - number of bytes to read (1, 2, or 4).

Return Value :
The value read from the specified offset of PCI header.

See also: WriteToPci

5.2.3.2.1.4 WriteToPci

Reads byte, word, or double word value data from the header of the device's PCI configuration space.

Language	Description
C/C++	void WriteToPci (HANDLE hPci, USHORT wOffset, USHORT wBytes, ULONG wValue);
Delphi	procedure WriteToPci (hPci: THandle; wOffset: Word; wBytes: Word; wValue: Longword); stdcall;
VB	Sub WriteToPci (ByVal hPci As Long, ByRef PciConfig as PCI_COMMON_CONFIG, ByVal wOffset As Integer, ByVal wBytes As Integer, ByVal wValue As Long) As Bool

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
PciConfig - points to the requested information, see PCI_COMMON_CONFIG structure
wOffset - offset in PCI header, in bytes.
wBytes - number of bytes to write (1, 2, or 4).
wValue - write value.

Return Value :
None

See also: ReadFromPci

5.2.3.2.2 PCI BAR Areas

5.2.3.2.2.1 GetNumOfPciBars

This function returns a value that specifies the number of PCI base address areas (BARs).

Language	Description
C/C++	ULONG GetNumOfPciBars (HANDLE hPci);
Delphi	function GetNumOfPciBars (hPci: THandle): Longword; stdcall;
VB	Function GetNumOfPciBars(ByVal hPci As Long) As Long

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci

Return Value :
Specifies the number of PCI base address areas (BARs).

5.2.3.2.2.2 GetPciBarLength

This function returns a length of present BAR area

Language	Description
C/C++	ULONG GetPciBarLength (HANDLE hPci, USHORT BarNumber);
Delphi	function GetPciBarLength (hPci: THandle; BarNumber: Word): Longword; stdcall;
VB	Function GetPciBarLength (ByVal hPci As Long, ByVal BarNumber As Integer) As Long

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
BarNumber - the BaseAddress index (0..5)

Return Value :
Length in bytes of present BAR area.

Comment:
This function serves all requests regardless of whether the BarNumber is memory address or port I/O address

5.2.3.2.2.3 GetPciBarPhysicalAddress

This function returns a starting physical base address of present BAR area

Language	Description
C/C++	ULONG GetPciBarPhysicalAddress (HANDLE hPci, USHORT BarNumber);
Delphi	function GetPciBarPhysicalAddress (hPci: THandle; BarNumber: Word): Longword; stdcall;
VB	Function GetPciBarPhysicalAddress (ByVal hPci As Long, ByVal BarNumber As Integer) As Long

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
BarNumber - the BaseAddress index (0..5)

Return Value :
Base physical address of present BAR area.

Comment:
This function serves all requests regardless of whether the BarNumber is memory address or port I/O address

5.2.3.2.2.4 GetPciBarLinearAddress

This function returns a starting linear base address of present BAR area

Language	Description
C/C++	ULONG GetPciBarLinearAddress (HANDLE hPci, USHORT BarNumber);
Delphi	function GetPciBarLinearAddress (hPci: THandle; BarNumber: Word): Longword; stdcall;
VB	Function GetPciBarLinearAddress (ByVal hPci As Long, ByVal BarNumber As Integer) As Long

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
BarNumber - the BaseAddress index (0..5)

Return Value :
Base linear address of present BAR area in current process's address space.

Comment:
This function serves all requests regardless of whether the BarNumber is memory address or port I/O address

5.2.3.2.2.5 ReadPciBarByte

This function reads a byte, starting at the specified offset, from the specified BAR address.

Language	Description
C/C++	UCHAR ReadPciBarByte (HANDLE hPci, USHORT BarNumber, USHORT BarOffset);
Delphi	function ReadPciBarByte (hPci: THandle; BarNumber: Word; BarOffset: Word): Byte; stdcall;
VB	Function ReadPciBarByte (ByVal hPci As Long, ByVal BarNumber As Integer, ByVal BarOffset As Integer) As Byte

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
BarNumber - the BaseAddress index (0..5) which must be read
BarOffset - specifies the byte offset in the BAR area, in bytes

Return Value :
The byte value, starting at the specified offset, from the specified BAR address.

Comment:
This function serves all requests regardless of whether the BarNumber is memory address or port I/O address

5.2.3.2.2.6 ReadPciBarWord

This function reads a 16-bit word, starting at the specified offset, from the specified BAR address.

Language	Description
C/C++	USHORT ReadPciBarWord (HANDLE hPci, USHORT BarNumber, USHORT BarOffset);
Delphi	function ReadPciBarWord (hPci: THandle; BarNumber: Word; BarOffset: Word): Word; stdcall;
VB	Function ReadPciBarWord (ByVal hPci As Long, ByVal BarNumber As Integer, ByVal BarOffset As Integer) As Integer

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
BarNumber - the BaseAddress index (0..5) which must be read
BarOffset - specifies the word offset in the BAR area, in bytes

Return Value :
The 16-bit word value, starting at the specified offset, from the specified BAR address.

Comment:
This function serves all requests regardless of whether the BarNumber is memory address or port I/O address

5.2.3.2.2.7 ReadPciBarLong

This function reads a 32-bit double word, starting at the specified offset, from the specified BAR address.

Language	Description
C/C++	ULONG ReadPciBarLong (HANDLE hPci, USHORT BarNumber, USHORT BarOffset);
Delphi	function ReadPciBarLong (hPci: THandle; BarNumber: Word; BarOffset: Word): Longword; stdcall;
VB	Function ReadPciBarLong (ByVal hPci As Long, ByVal BarNumber As Integer, ByVal BarOffset As Integer) As Long

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
BarNumber - the BaseAddress index (0..5) which must be read
BarOffset - specifies the double word offset in the BAR area, in bytes

Return Value :
The 32-bit double word value, starting at the specified offset, from the specified BAR address.

Comment:
This function serves all requests regardless of whether the BarNumber is memory address or port I/O address

5.2.3.2.2.8 WritePciBarByte

This function writes a byte, starting at the specified offset, to the specified BAR address .

Language	Description
C/C++	void WritePciBarByte (HANDLE hPci, USHORT BarNumber, USHORT BarOffset, UCHAR bValue);
Delphi	procedure WritePciBarByte (hPci: THandle; BarNumber: Word; BarOffset: Word; bValue: Byte); stdcall;
VB	Sub WritePciBarByte (ByVal hPci As Long, ByVal BarNumber As Integer, ByVal BarOffset As Integer, ByVal bValue As Byte)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
BarNumber - the BaseAddress index (0..5) which must be read
BarOffset - specifies the byte offset in the BAR area, in bytes
bValue - specifies a byte to be written

Return Value : None

Comment:
This function serves all requests regardless of whether the BarNumber is memory address or port I/O address

5.2.3.2.2.9 WritePciBarWord

This function writes a 16-bit word, starting at the specified offset, to the specified BAR address .

Language	Description
C/C++	void WritePciBarWord (HANDLE hPci, USHORT BarNumber, USHORT BarOffset, USHORT wValue);
Delphi	procedure WritePciBarWord (hPci: THandle; BarNumber: Word; BarOffset: Word; wValue: Word); stdcall;
VB	Sub WritePciBarWord (ByVal hPci As Long, ByVal BarNumber As Integer, ByVal BarOffset As Integer, ByVal wValue As Integer)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
BarNumber - the BaseAddress index (0..5) which must be read
BarOffset - specifies the byte offset in the BAR area, in bytes
wValue - specifies a word to be written

Return Value : None

Comment:
This function serves all requests regardless of whether the BarNumber is memory address or port I/O address

5.2.3.2.2.10 WritePciBarLong

This function writes a 32-bit double word, starting at the specified offset, to the specified BAR address .

Language	Description
C/C++	void WritePciBarLong (HANDLE hPci, USHORT BarNumber, USHORT BarOffset, ULONG dwValue);
Delphi	procedure WritePciBarLong (hPci: THandle; BarNumber: Word; BarOffset: Word; dwValue: Word); stdcall;
VB	Sub WritePciBarLong (ByVal hPci As Long, ByVal BarNumber As Integer, ByVal BarOffset As Integer, ByVal Value As Long)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
BarNumber - the BaseAddress index (0..5) which must be read
BarOffset - specifies the byte offset in the BAR area, in bytes
d*wValue* - specifies a doubleword to be written

Return Value : None

Comment:
This function serves all requests regardless of whether the BarNumber is memory address or port I/O address

5.2.3.2.3 PCI Registers

5.2.3.2.3.1 ReadPciCommandReg

Read the PCI command register.

Language	Description
C/C++	USHORT ReadPciCommandReg (HANDLE hPci);
Delphi	function ReadPciCommandReg (hPci: THandle): Word; stdcall;
VB	Function ReadPciCommandReg (ByVal hPci As Long) As Integer

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
ClearMask - specifies which bits to clear after reading (0-9).

Return Value :
The 16-bit word read from the PCI command register.

5.2.3.2.3.2 WritePciCommandReg

Write the PCI command register.

Language	Description
C/C++	void WritePciCommandReg (HANDLE hPci, USHORT CommandRegValue);
Delphi	procedure WritePciCommandReg (hPci: THandle, CommandRegValue : Word); stdcall;
VB	Sub WritePciCommandReg (ByVal hPci As Long, ByVal CommandRegValue As Integer)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
CommandRegValue - new value of the command register to write

Comments:
Use WritePciCommandRegBits function to set or clear individual bits. Refer to the PCI specification for more details.

5.2.3.2.3.3 ControlPciCommandRegBits

Set or clear a bit in the PCI command register

Language	Description
C/C++	void ControlPciCommandRegBits (HANDLE hPci, USHORT ControlBitNumber, USHORT BitEnable);
Delphi	procedure ControlPciCommandRegBits (hPci: THandle, ControlBitNumber : Word; BitEnable: Word); stdcall;
VB	Sub ControlPciCommandRegBits (ByVal hPci As Long, ByVal ControlBitNumber As Integer, ByVal BitEnable As Integer)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
ControlBitNumber - identifies the bit of the command register to modify,
BitEnable - "1" to set the bit, "0" to clear it

Comments:
The possible values for parameter ControlBitNumber are 0-9. Refer to the PCI specification for more details.

5.2.3.2.3.4 ReadPciStatusReg

Read the PCI status registers, and optionally clear bits **after** reading.

Language	Description
C/C++	USHORT ReadPciStatusReg (HANDLE hPci, USHORT ClearMask);
Delphi	function ReadPciStatusReg (hPci: THandle, ClearMask : Word): Word; stdcall;
VB	Function ReadPciStatusReg (ByVal hPci As Long, ByVal ClearMask As Integer) As Integer

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;

Return Value :
The current value of the status register before clearing any bits.

Comments:
The status register bits can be cleared but not set. A value of 0x0000 clears any bits that were set. A value of 0xFFFF does not clear any. Refer to the PCI specification for more details.

5.2.3.3 Memory Access

5.2.3.3.1 MapPhysToLinear

Maps a specific physical address to a pointer in linear memory for the current process' address space. This function is intended to be used to access
device-specific physical memory (for the memory-mapped devices).

Language	Description
C/C++	void * MapPhysToLinear (HANDLE hPci, ULONG PhysAddr, ULONG MappedSize);
Delphi	function MapPhysToLinear (hPci: THandle; PhysAddr:Longword; MappedSize: Longword): Pointer; stdcall;
VB	Function MapPhysToLinear (ByVal hPci As Long, ByVal PhysAddr As Long, ByVal MappedSize As Long) As Long

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
PhysAddr - 32-bit physical address of the start of the region to examine
MappedSize - length in bytes of the physical region. The function uses this parameter to verify that the entire range is addressable.

Return Value :
The linear address of the first byte in the specified range of physical addresses.

Comments:
1. Do not forget to unmap this physical memory area by UnmapMemory when your application not need it anymore.
2. No information can be concluded about the linear addresses for physical addresses if they lie outside the range requested.

See also : UnmapMemory

5.2.3.3.2 UnmapMemory

Unmaps a specific physical address previously mapped with MapPhysToLinear.

Language	Description
C/C++	void UnmapMemory (HANDLE hPci, ULONG PhysAddr, ULONG MappedSize);
Delphi	procedure UnmapMemory (hPci: THandle; PhysAddr: Longword; MappedSize: Longword); stdcall
VB	Sub UnmapMemory (ByVal hPci As Long, ByVal PhysAddr As Long, ByVal MappedSize As Long)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci
PhysAddr - 32-bit physical address of the start of the region previously mapped
MappedSize - length in bytes of the mapped region

Return Value :
None.

See also : MapPhysToLinear

5.2.3.3.3 GetMem

Reads **one byte** from the memory region. Mainly is intended for using in the Visual Basic applications where the direct manipulations with the pointers is difficult.

Language	Description
C/C++	UCHAR GetMem (HANDLE hPci, ULONG MappedAddress, ULONG MemOffset);
Delphi	function GetMem (hPci: THandle; MappedAddress: Longword; MemOffset: Longword): Byte; stdcall;
VB	Function GetMem (ByVal hPci As Long, ByVal MappedAddress As Long, ByVal MemOffset As Long) As Byte

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;
MappedAddress - the linear address returned by MapPhysToLinear
MemOffset **-** the offset value, should begin from the "0" value.

Return Value :
One byte value from the region.

Comments:
1. In general, "MappedAddress" can be any valid linear pointer to the memory. But if this pointer was received from the MapPhysToLinear function then you can access physical memory also.
2. If you want access the BAR0-BAR5 PCI areas then use a special functions set instead.

See also: MapPhysToLinear UnmapMemory GetMemW GetMemL SetMem SetMemW SetMemL

5.2.3.3.4 GetMemW

Reads **two bytes (word)** from the memory region. Mainly is intended for using in the Visual Basic applications where the direct manipulations with the pointers is difficult.

Language	Description
C/C++	USHORT GetMemW (HANDLE hPci, ULONG MappedAddress, ULONG MemOffset);
Delphi	function GetMemW (hPci: THandle; MappedAddress: Longword; MemOffset: Longword): Word; stdcall;
VB	Function GetMemW (ByVal hPci As Long, ByVal MappedAddress As Long, ByVal MemOffset As Long) As Integer

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;
MappedAddress - the linear address returned by MapPhysToLinear
MemOffset **-** the offset value, should begin from the "0" value.

Return Value :
16-bit value from the region.

Comments:
1. In general, "MappedAddress" can be any valid linear pointer to the memory. But if this pointer was received from the MapPhysToLinear function then you can access physical memory also.
2. If you want access the BAR0-BAR5 PCI areas then use a special functions set instead.

See also: MapPhysToLinear UnmapMemory GetMem GetMemL SetMem SetMemW SetMemL

5.2.3.3.5 GetMemL

Reads **four bytes (dword)** from the memory region. Mainly is intended for using in the Visual Basic applications where the direct manipulations with the pointers is difficult.

Language	Description
C/C++	ULONG GetMemL (HANDLE hPci, ULONG MappedAddress, ULONG MemOffset);
Delphi	function GetMemL (hPci: THandle; MappedAddress: Longword; MemOffset: Longword): Longword; stdcall;
VB	Function GetMemL (ByVal hPci As Long, ByVal MappedAddress As Long, ByVal MemOffset As Long) As Long

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;
MappedAddress - the linear address returned by MapPhysToLinear
MemOffset **-** the offset value, should begin from the "0" value.

Return Value :
32-bit value from the region.

Comments:
1. In general, "MappedAddress" can be any valid linear pointer to the memory. But if this pointer was received from the MapPhysToLinear function then you can access physical memory also.
2. If you want access the BAR0-BAR5 PCI areas then use a special functions set instead.

See also: MapPhysToLinear UnmapMemory GetMem GetMemW SetMem SetMemW SetMemL

5.2.3.3.6 SetMem

Write **one byte** to the memory region. Mainly is intended for using in the Visual Basic applications where the direct manipulations with the pointers is difficult.

Language	Description
C/C++	void SetMem (HANDLE hPci, ULONG MappedAddress, ULONG MemOffset , UCHAR bValue);
Delphi	procedure SetMem (hPci: THandle; MappedAddress: Longword; MemOffset: Longword; bValue: Byte); stdcall;
VB	Sub SetMem (ByVal hPci As Long, ByVal MappedAddress As Long, ByVal MemOffset As Long, bValue As Byte)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;
MappedAddress - the linear address returned by MapPhysToLinear
MemOffset **-** the offset value, should begin from the "0" value.
bValue - value to write

Return Value : None

Comments:
1. In general, "MappedAddress" can be any valid linear pointer to the memory. But if this pointer was received from the MapPhysToLinear function then you can access physical memory also.
2. If you want access the BAR0-BAR5 PCI areas then use a special functions set instead.

See also: MapPhysToLinear UnmapMemory GetMem GetMemW GetMemL SetMemW SetMemL

5.2.3.3.7 SetMemW

Write **two bytes (word)** to the memory region. Mainly is intended for using in the Visual Basic applications where the direct manipulations with the pointers is difficult.

Language	Description
C/C++	void SetMemW (HANDLE hPci, ULONG MappedAddress, ULONG MemOffset , USHORT wValue);
Delphi	procedure SetMemW (hPci: THandle; MappedAddress: Longword; MemOffset: Longword; wValue: Word); stdcall;
VB	Sub SetMemW (ByVal hPci As Long, ByVal MappedAddress As Long, ByVal MemOffset As Long, wValue As Integer)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;
MappedAddress - the linear address returned by MapPhysToLinear
MemOffset **-** the offset value, should begin from the "0" value.
wValue - value to write

Return Value : None

Comments:
1. In general, "MappedAddress" can be any valid linear pointer to the memory. But if this pointer was received from the MapPhysToLinear function then you can access physical memory also.
2. If you want access the BAR0-BAR5 PCI areas then use a special functions set instead.

See also: MapPhysToLinear UnmapMemory GetMem GetMemW GetMemL SetMem SetMemL

5.2.3.3.8 SetMemL

Write **four bytes (double word)** to the memory region. Mainly is intended for using in the Visual Basic applications where the direct manipulations with the pointers is difficult.

Language	Description
C/C++	void SetMemL (HANDLE hPci, ULONG MappedAddress, ULONG MemOffset , ULONG dwValue);
Delphi	procedure SetMemL (hPci: THandle; MappedAddress: Longword; MemOffset: Longword; dValue: Longword); stdcall;
VB	Sub SetMemL (ByVal hPci As Long, ByVal MappedAddress As Long, ByVal MemOffset As Long, dwValue As Long)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;
MappedAddress - the linear address returned by MapPhysToLinear
MemOffset **-** the offset value, should begin from the "0" value.
d*wValue* - value to write

Return Value : None

Comments:
1. In general, "MappedAddress" can be any valid linear pointer to the memory. But if this pointer was received from the MapPhysToLinear function then you can access physical memory also.
2. If you want access the BAR0-BAR5 PCI areas then use a special functions set instead.

See also: MapPhysToLinear UnmapMemory GetMem GetMemW GetMemL SetMem SetMemW

5.2.3.4 Direct Port I/O

5.2.3.4.1 GetPortByte

Reads a byte from the specified port address.

Language	Description
C/C++	UCHAR GetPortByte (HANDLE hPci, ULONG PortAddress);
Delphi	function GetPortByte (hPci: THandle; PortAddress: Longword): Byte; stdcall;
VB	Function GetPortByte (ByVal hPci As Long, ByVal PortAddress As Long) As Byte

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;
PortAddress - the port address in I/O space

Return Value :
The byte read from the specified port address.

See also: GetPortWord GetPortLong SetPortByte SetPortWord SetPortLong

5.2.3.4.2 GetPortWord

Reads a WORD (two bytes) from the specified port address.

Language	Description
C/C++	USHORT GetPortWord (HANDLE hPci, ULONG PortAddress);
Delphi	function GetPortWord (hPci: THandle; PortAddress: Longword): Word; stdcall;
VB	Function GetPortWord (ByVal hPci As Long, ByVal PortAddress As Long) As Integer

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;
PortAddress - the port address in I/O space

Return Value :
The WORD value read from the specified port address

See also: GetPortByte GetPortLong SetPortByte SetPortWord SetPortLong

5.2.3.4.3 GetPortLong

Reads a double word (four bytes) from the specified port address.

Language	Description
C/C++	ULONG GetPortLong (HANDLE hPci, ULONG PortAddress);
Delphi	function GetPortLong (hPci: THandle; PortAddress: Longword): Longword; stdcall;
VB	Function GetPortLong (ByVal hPci As Long, ByVal PortAddress As Long) As Long

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;
PortAddress - the port address in I/O space

Return Value :
The double word value read from the specified port address.

See also: GetPortByte GetPortWord SetPortByte SetPortWord SetPortLong

5.2.3.4.4 SetPortByte

Writes a byte to the specified port address.

Language	Description
C/C++	void SetPortByte (HANDLE hPci, ULONG PortAddress, UCHAR bValue);
Delphi	procedure SetPortByte (hPci: THandle; PortAddress: Longword; bValue: Byte); stdcall;
VB	Sub SetPortByte (ByVal hPci As Long, ByVal PortAddress As Long, ByVal bValue As Byte)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;
PortAddress - the port address in I/O space
b*Value* - specifies a byte to be written to the port.

Return Value : None.

See also: GetPortByte GetPortWord GetPortLong SetPortWord SetPortLong

5.2.3.4.5 SetPortWord

Writes a word (two bytes) value to the specified port address.

Language	Description
C/C++	void SetPortWord (HANDLE hPci, ULONG PortAddress, USHORT wValue);
Delphi	procedure SetPortWord (hPci: THandle; PortAddress: Longword; wValue: Word); stdcall;
VB	Sub SetPortWord (ByVal hPci As Long, ByVal PortAddress As Long, ByVal wValue As Integer)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;
PortAddress - the port address in I/O space
w*Value* - specifies a word to be written to the port.

Return Value : None.

See also: GetPortByte GetPortWord GetPortLong SetPortByte SetPortLong

5.2.3.4.6 SetPortLong

Writes a double word (four bytes) value to the specified port address.

Language	Description
C/C++	void SetPortLong (HANDLE hPci, ULONG PortAddress, ULONG dwValue);
Delphi	procedure SetPortLong (hPci: THandle; PortAddress: Longword; dwValue: Longword); stdcall;
VB	Sub SetPortLong (ByVal hPci As Long, ByVal PortAddress As Long, ByVal dwValue As Long)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;
PortAddress - the port address in I/O space
dw*Value* - specifies a double word to be written to the port.

Return Value : None.

See also :GetPortByte GetPortWord GetPortLong SetPortByte SetPortWord

5.2.3.4.7 ReadPortBuffer

Reads a number of **bytes**, **words**, or **double words** from the specified port address into a buffer.

Language	Description
C/C++	void ReadPortBuffer(HANDLE hPci, ULONG PortAddr, USHORT PortWidth, ULONG NumValues, void * buffer);
Delphi	procedure ReadPortBuffer (hPci: THandle; PortAddr: LongWord; PortWidth: Word; NumValues: Longword; buffer: Pointer); stdcall;
VB	Sub ReadPortBuffer (ByVal hPci As Long, ByVal PortAddr As Long, ByRef PortWidth As Integer; ByVal PortWidth As Long, ByRef buffer As Any)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;
PortAddr - port address;
PortWidth - size of port in bytes (1, 2, or 4);
NumValue - number of values to read
buffer - a buffer into which an array of values is read.

Return Value :None.

See also : WritePortBuffer

5.2.3.4.8 WritePortBuffer

Writes a number of **bytes**, **words**, or **double words** from a buffer to the specified port.

Language	Description
C/C++	void WritePortBuffer(HANDLE hPci, ULONG PortAddr, USHORT PortWidth, ULONG NumValues, void * buffer);
Delphi	procedure WritePortBuffer (hPci: THandle; PortAddr: LongWord; PortWidth: Word; NumValues: Longword; buffer: Pointer); stdcall;
VB	Sub WritePortBuffer (ByVal hPci As Long, ByVal PortAddr As Long, ByRef PortWidth As Integer; ByVal PortWidth As Long, ByRef buffer As Any)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;
PortAddr - port address;
PortWidth - size of port in bytes (1, 2, or 4);
NumValue - number of values to read
buffer - a buffer from which an array of values is to be written.

Return Value :None.

See also: ReadPortBuffer

5.2.3.5 Hardware Interrupts

5.2.3.5.1 UnmaskPciIrq

Installs a hardware interrupt handler for the PCI device and unmasks the Irq at hardware level. UnmaskPciIrq allows to handle the
"shared" and "level sensitive" interrupts as well as the "non-shared" and "edge triggered".

Language	Description
C/C++	void UnmaskPciIrq(HANDLE hPci, IRQ_SHARE_REC * ShareRec, IRQ_CLEAR_REC * ClearRec, TOnHwInterrupt InterruptHandler);
Delphi	procedure UnmaskPciIrq (hPci: THandle; var ShareRec: IRQ_SHARE_REC; var ClearRec: IRQ_CLEAR_REC; InterruptHandler: TOnHwInterruptHandler); stdcall;
VB	Sub UnmaskPciIrq (ByVal hPci As Long, ByRef ShareRec As IRQ_SHARE_REC, ByRef ClearRec As IRQ_CLEAR_REC, ByVal HWHandler As Long)

Parameters:

hPci - the handle returned by a successful call to OpenRapidPci ;
ShareRec - points to the structure of type IRQ_SHARE_REC, which describes how to determine if it is our interruption or not;
ClearRec - points to the structure of type IRQ_CLEAR_REC, which describes how to acknowledge an interruption inside of the driver.
InterruptHandler - address of the callback procedure that handles hardware interrupts for this Irq (use AddressOf specification in VB)

Return Value : None.

Comments:

1. The interrupt handler function must be declared as follows:

C/C++: typedef void (__stdcall * TOnHwInterrupt)(ULONG TimeStampLoPart, ULONG TimeStampHiPart);
Delphi: procedure OnHwInterrupt(TimeStampLoPart: Longword; TimeStampHiPart: Longword); stdcall;
VB: Sub OnHwInterrupt(ByVal TimeStampLoPart As Long, ByVal TimeStampHiPart As Long)

The *TimeStampLoPart/TimeStampHiPart* values are parts of value returned by the kernel equivalent of QueryPerformaceCounter() API function.

2. This function helps to handle the "level sensitive" interrupts rather than "edge triggered" ones. If the interrupt level at the bus is not dropped at the end of an interrupt handling, the system calls the interrupt handler again and again and this will cause the system to hang. By calling the UnmaskPciIrq function you can provide the information for how the RapidPci driver should clear an interruption inside of the primary interrupt handler. Note: You do not need to clear the interrupt level in your user-level interrupt handler since this was already done by the RapidPci driver!

See also: MaskPciIrq GetInterruptCounter

5.2.3.5.2 MaskPciIrq

Stops the interrupt processing started by UnmaskPciIrq by masking the PCI Irq at hardware level.

Language	Description
C/C++	void MaskPciIrq(HANDLE hPci);
Delphi	procedure MaskPciIrq (hPci: THandle); stdcall;
VB	Sub MaskPciIrq (ByVal hPci As Long)

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;

Return Value : None.

See also: UnmaskPciIrq GetInterruptCounter

5.2.3.5.3 GetInterruptCounter

Returns how many interruptions was processed inside of the driver.

Language	Description
C/C++	ULONG GetInterruptCounter(HANDLE hPci);
Delphi	function GetInterruptCounter (hPci: THandle): Longword; stdcall;
VB	Function GetInterruptCounter (ByVal hPci As Long) As Long

Parameters:
hPci - the handle returned by a successful call to OpenRapidPci ;

Return Value :
Number of the interruptions processed by the drive.

See also: UnmaskPciIrq MaskPciIrq

5.3 Programming for Parallel Port Device

Overview and getting started

5.3.1 Overview

First of all, you must describe your device with ***RapidDriver Explorer*** then install a device using LPT specific driver (RapidLpt.sys). Then you may create and run your own application to control an LPT device for which the driver has been installed. All LPT functions/procedures are placed in RapidLpt.dll interface library.

You can install the RapidLpt.sys driver for many LPT devices, so every LPT device will have it's own *DeviceInstance* parameter value in OpenRapidLpt function.

Note: *In some cases we will use the "**RapidLpt**" term assuming the RapidDriver Developer edition when we are working with a LPT device through the RapidLpt.sys and RapidLpt.dll.*

Here is what you can do with RapidLpt:

- access to data, status and control ports
- access to all the information of existing port - base and ECP addresses and ranges, interrupt features
- hardware interrupt handling
- access to pins
- access to bits

To work with the driver itself the choice of programming language is not vital. At present RapidDriver package includes support of the following programming systems:

- Microsoft Visual C/C++
- Borland Delphi
- Borland C++ Builder
- Microsoft Visual Basic 6.0
- Microsoft Visual Basic .Net
- Microsoft Visual C#

5.3.2 Programmers Guide

5.3.2.1 Scenario

Before using any of the functions of RapidLpt, the driver must be initialized with the help of the function OpenRapidLpt . It must be done once after starting the application. Also resources used by the driver should be cleaned up(driver closed) before the application is closed.

Schematic session of using RapidLpt:

```
// initialize(open) the driver

hLpt =  OpenRapidlpt( 0 );

// Check if the driver initialized successfully

if ( IsRapidLptOpened(hLpt)) {
        ...
```

```
        ////////////////////////////////////
        ////////  RapidLpt session //////////
        ////////  ................ //////////
        ////////////////////////////////////

        // Free resources used by the driver (close the driver)

        hLpt = CloseRapidLpt(hLpt);

    }
    else
    {
        // An error happened while opening the driver
        MessageBox("Cannot open the driver","Attention!", MB_OK |
        MB_ICONWARNING);
    }
```

Driver can be initialized by many applications simultaneously, once for each LPT device. The parameter of OpenRapidLpt() can be used to select a device supported by RapidLpt driver.

5.3.2.2 I/O Ports control

I/O ports control function set includes two major groups of functions: - single input/output by byte (word, double word) and data array input/output.

- Single read/write operations
- Data array read/write operations

5.3.2.2.1 Single Read/Write Operations

The following set of functions is used to read/write single values through port. Their assignment is evident from names and requires no comments.

GetPortByte - read a byte from a port
GetPortWord - read a word from a port
GetPortLong - read a double word from a port
SetPortByte - write a byte to a port
SetPortWord - write a word to a port
SetPortLong - write a double word to a port

5.3.2.2.2 Data Array Read/Write operations

RapidLpt aslo contains two functions for read/write data arrays from/to port. These functions are based on ***insb/insw/insd*** assembly input commands and ***outsb/outsw/outsd*** output commands.

ReadPortBuffer - reads a number of **bytes**, **words**, or **double words** from the specified port address into a buffer.
WritePortBuffer - write an array of bytes

Remember, these functions don't provide another exchange algorithm ("handshaking"). They are used only for devices with hardware support of group input/output operations.

5.3.2.3 Hardware Interrupts handling

Common issues

5.3.2.3.1 Common Issues

Function UnmaskLptIrq installs a hardware interrupt handler at "user level". It works as follows:

```
ULONG IrqCounter = 0;
ULONG LowTimeStamp = 0;
ULONG HighTimeStamp = 0;
PIRQ_SHARE_REC ShareRec;
PIRQ_CLEAR_REC ClearRec;

// Interrupt handler
void __stdcall OnHardwareInterrupt(ULONG LowPart, ULONG HighPart)
{
    ++ IrqCounter;
      ULONG LowTimeStamp = LowPart;
      ULONG HighTimeStamp = HighPart;
}

// Main program
hLpt = OpenRapidLpt( 0 );
if (IsRapidLptOpened(hLpt)) {
    ...
    set ClearRec and ShareRec structures, see below
    ...
    UnmaskLptIrq( hLpt, MyInterruptHandler );
}
else
  ... // failed
```

Calling the function MaskLptIrq stops hardware interrupt processing:

```
MaskLptIrq ( hLpt );
```

Counter value for the number of handled interrupts is returned by function GetInterruptCounter.

5.3.3 LPT Support Routines

5.3.3.1 Common Procedures

5.3.3.1.1 OpenRapidLpt

Opens the RapidLpt.sys kernel mode WDM device driver under Windows 2000/XP/2003, providing direct access to the LPT hardware.

Language	Description
C/C++	HANDLE OpenRapidLpt (ULONG DeviceInstance)
Delphi	function OpenRapidLpt (DeviceInstance : Longword) : THandle; stdcall
VB	Function OpenRapidLpt (ByVal DeviceInstance As Long) As Long

Parameters:
DeviceInstance - LPT device instance.

Return Value:
The RapidLpt handle value or NULL if fails. Keep this handle opened all time when you are working with RapidLpt functions and close by CloseRapidLpt() at end of application.

Comments:

If the driver was successfully opened, the IsRapidLptOpened() returns True; if the function fails, the IsRapidLptOpened() returns False.
With RapidLpt you can support as many LPT devices as you want. To select the device you need change DriverInstance value from 0 up to maximal number of LPT device installed with RapidLpt driver. For the first application use

```
hLpt = OpenRapidLpt(0);
```

For second:

```
hLpt = OpenRapidLpt(1);
```

and so on.

See also: CloseRapidLpt IsRapidLptOpened

5.3.3.1.2 IsRapidLptOpened

This boolean function specifies whether the mode driver is open.

Language	Description
C/C++	BOOL IsRapidLptOpened (HANDLE hLpt)
Delphi	function IsRapidLptOpened (hLpt : THandle) : BOOL; stdcall
VB	Function IsRapidLptOpened (ByVal hLpt As Long) As Bool

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
TRUE if the driver is already open by OpenRapidlpt(), or FALSE if it is not.

Comments:
Use immediately after OpenRapidLpt() call to check if the driver was successfully opened or not.

See also: OpenRapidLpt CloseRapidlpt

5.3.3.1.3 CloseRapidLpt

Closes the kernel-mode driver and releases memory allocated to it. If a hardware interruptions was "unmasked", the "mask" is restored.

Language	Description
C/C++	HANDLE CloseRapidLpt (HANDLE hLpt)
Delphi	function CloseRapidLpt (hLpt: THandle): THandle; stdcall
VB	Function CloseRapidLpt (ByVal hLpt As Long) As Long

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
NULL.

See also: OpenRapidLpt IsRapidLptOpened

5.3.3.1.4 OpenPort

Opens the selected parallel port if it is installed on PC.

Language	Description
C/C++	BOOL OpenPort (HANDLE hLpt, UCHAR PortNumber)
Delphi	function OpenPort (hLpt: THANDLE; PortNumber: Byte) : Boolean; stdcall
VB	Function OpenPort (ByVal hLpt As Long, ByVal PortNumber As Byte) As Boolean

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
PortNumber - number of the port to be opened.

Return Value :
TRUE, if the call to port was successful, FALSE otherwise.

Comments:

Use this function before doing i/o and interrupt operations with the port. If OpenPort() succeeds, you can get access to current
Parallel port info.
For example, LPT1 is opened in such a way:

```
OpenPort(hLpt, 1);
```

See also:
ClosePort

5.3.3.1.5 ClosePort

Closes the parallel port opened with OpenPort.

Language	Description
C/C++	BOOL ClosePort (HANDLE hLpt)
Delphi	function ClosePort (hLpt: THANDLE) : Boolean; stdcall
VB	Function ClosePort (ByVal hLpt As Long) As Boolean

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
TRUE, if the operation succeeds, FALSE otherwise.

Comments:

Use this function after all operations with the port are done.

See also:
OpenPort

5.3.3.1.6 GetNumLPTs

Returns the number of LPT ports installed on the PC.

Language	Description
C/C++	USHORT GetNumLPTs (HANDLE hLpt)
Delphi	function GetNumLpts (hLpt: THANDLE) : Word; stdcall
VB	Function GetNumLpts (ByVal hLpt As Long) As Integer

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
- number of parallel ports found on PC.

5.3.3.1.7 GetPortNumber

Returns the current LPT port number.

Language	Description
C/C++	ULONG GetPortNumber (HANDLE hLpt, ULONG Index)
Delphi	function GetPortNumber (hLpt : THANDLE; Index : Integer) : Integer; stdcall
VB	Function GetPortNumber (ByVal hLpt As Long, ByVal PortIndex As Long) As Long

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
Index - zero-based index which indicates the number of port detected by RapidLpt. For exmaple,

Index = 0 may indicate LPT1,
Index = 2 - LPT3 an so on.

Return Value :
- Current LPT port number (1 - LPT1, 2 - LPT2 and so on).

Comments:
Can be used only after the call of GetNumLPTs function.

Example (VC++): enumerating LPT ports installed on the PC and putting them into combo box.

```
int n = GetNumLPTs(hLpt);
for(int i=0; i<n; i++)
{
        sprintf(buf, "LPT %d", GetPortNumber(hLpt,i));
        m_lptlist.AddString(buf);
}
```

Example (VC++): getting the port number from the combo box (zero-based index) and opening the port:

```
CurrPortNumber = GetPortNumber ( hLpt, m_lptlist.GetCurSel() );
OpenPort(hLpt,CurrPortNumber);
```

5.3.3.1.8 GetReadMode

Acquires direction ("read" or "write" mode) from LPT Data Register.

Language	Description
C/C++	UCHAR GetReadMode (HANDLE hLpt)
Delphi	function GetReadMode (hLpt : THANDLE) : Byte ; stdcall
VB	Function GetReadMode (ByVal hLpt As Long) As Byte

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
current mode
0 - "write only" mode (default).
1 - "read only" mode (not supported by SPP!).

See also: SetReadMode

5.3.3.1.9 SetReadMode

Allows to switch the data register of current LPT to "read" or "write" mode.

Language	Description
C/C++	void SetReadMode (HANDLE hLpt , UCHAR nNewValue)
Delphi	procedure SetReadMode (hLpt : THANDLE ; nNewValue : Byte) ; stdcall
VB	Sub SetReadMode (ByVal hLpt As Long , ByVal nNewValue As Byte)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
nNewValue - requested mode
0 - sets the data register to "write only" mode (default).
1 - sets the data register to "read only" mode (not supported by SPP!).

Return Value :
None

See also: GetReadMode

5.3.3.2 LPT Specific Functions

5.3.3.2.1 Parallel Port Info

With RapidLpt you can access parallel port configurations. These functions specify information about the resources assigned to a parallel port, the capabilities of the parallel port. This information is unique for each port, opened with OpenPort.

Note: you must call OpenPort function before running this function.

GetBasePortAddress

GetBaseSpan

GetEcpAddress

GetEcpSpan

GetInterruptVector

GetInterruptLevel

GetInterruptAffinity

GetInterruptMode

5.3.3.2.1.1 GetBasePortAddress

Specifies base address of parallel port (system-mapped base I/O location of the parallel port registers).

Language	Description
C/C++	USHORT GetBasePortAddress (HANDLE hLpt)
Delphi	function GetBasePortAddress (hLpt: THandle) : Word; stdcall;
VB	Function GetBasePortAddress (ByVal hLpt As Long) As Integer

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
- base address of parallel port

Note: you must call OpenPort function before running this function.

See also: GetBaseSpan

5.3.3.2.1.2 GetBaseSpan

Specifies the size of the I/O space, allocated to the parallel port.

Language	Description
C/C++	ULONG GetBaseSpan (HANDLE hLpt);
Delphi	function GetBaseSpan (hLpt: THandle) : Longword; stdcall;
VB	Function GetBaseSpan (ByVal hLpt As Long) As Long

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
Size of I/O space, in bytes.

Note: you must call OpenPort function before running this function.

See also: GetBasePortAddress

5.3.3.2.1.3 GetEcpAddress

Specifies I/O port resource (address) that is used to control the port in ECP mode.

Language	Description
C/C++	ULONG GetEcpAddress (HANDLE hLpt)
Delphi	function GetEcpAddress (hLpt: THandle) : Longword ; stdcall
VB	Function GetEcpAddress (ByVal hLpt As Long) As Long

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
Address that is used to control the port in ECP mode.

Note: you must call OpenPort function before running this function.

See also: GetEcpSpan

5.3.3.2.1.4 GetEcpSpan

Specifies the size of the I/O port resource that is used to control the port in ECP mode.

Language	Description
C/C++	ULONG GetEcpSpan (HANDLE hLpt)
Delphi	function GetEcpSpan (hLpt : THandle) : Longword; stdcall
VB	Function GetEcpSpan (ByVal hLpt As Long) As Long

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
Size of I/O space, in bytes.

Note: you must call OpenPort function before running this function.

See also: GetEcpAddress

5.3.3.2.1.5 GetInterruptVector

Specifies the interrupt vector for the parallel port.

Language	Description
C/C++	ULONG GetInterruptVector (HANDLE hLpt)
Delphi	function GetInterruptVector (hLpt : THandle) : Longword; stdcall
VB	Function GetInterruptVector (ByVal hLpt As Long) As Long

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
Interrupt vector

Note: you must call OpenPort function before running this function.

See also:
GetInterruptLevel
GetInterruptAffinity
GetInterruptMode

5.3.3.2.1.6 GetInterruptLevel

Specifies the interrupt level for the parallel port.

Language	Description
C/C++	ULONG GetInterruptLevel (HANDLE hLpt)
Delphi	function GetInterruptLevel (hLpt : THandle) : Longword; stdcall
VB	Function GetInterruptLevel (ByVal hLpt As Long) As Long

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
Interrupt level.

Note: you must call OpenPort function before running this function.

See also:
GetInterruptAffinity
GetInterruptMode
GetInterruptVector

5.3.3.2.1.7 GetInterruptAffinity

Specifies the affinity (set of processors on which a particular interrupt is enabled in a given machine).

Language	Description
C/C++	ULONG GetInterruptAffinity (HANDLE hLpt)
Delphi	function GetInterruptAffinity (hLpt : THandle) : Longword; stdcall
VB	Function GetInterruptAffinity (ByVal hLpt As Long) As Long

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
Interrupt affinity.

Note: you must call OpenPort function before running this function.

See also:
GetInterruptLevel
GetInterruptVector
GetInterruptMode

5.3.3.2.1.8 GetInterruptMode

Specifies the interrupt mode.

Language	Description
C/C++	ULONG GetInterruptMode (HANDLE hLpt)
Delphi	function GetInterruptMode (hLpt : THandle) : Longword;stdcall
VB	Function GetInterruptMode (ByVal hLpt As Long) As Long

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
Interrupt mode.

Note: you must call OpenPort function before running this function.

See also:
GetInterruptLevel
GetInterruptAffinity
GetInterruptVector

5.3.3.2.2 Parallel Port Registers

5.3.3.2.2.1 GetDataPort

Allows to read one byte from Data Register of current LPT.

Language	Description
C/C++	USHORT GetDataPort (HANDLE hLpt)
Delphi	function GetDataPort (hLpt : THANDLE) : Byte; stdcall
VB	Function GetDataPort (ByVal hLpt As Long) As Byte

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
The byte read from the LPT Data Register

See also: SetDataPort

5.3.3.2.2.2 SetDataPort

Allows to write one byte to the Data Register of current LPT.

Language	Description
C/C++	void SetDataPort (HANDLE hLpt , UCHAR nNewValue)
Delphi	procedure SetDataPort (hLpt : THANDLE ; bNewValue : Byte) ; stdcall
VB	Sub SetDataPort (ByVal hLpt As Long , ByVal bNewValue As Byte)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

nNewValue - one byte value to write to Data Register.

Return Value :
None

See also: GetDataPort

5.3.3.2.2.3 GetStatusPort

Allows to read one byte from Status Register of current LPT.

Language	Description
C/C++	USHORT GetStatusPort (HANDLE hLpt)
Delphi	function GetStatusPort (hLpt : THANDLE) : Byte; stdcall
VB	Function GetStatusPort (ByVal hLpt As Long) As Byte

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
The byte read from the LPT Status Register

See also: SetDataPort

5.3.3.2.2.4 SetStatusPort

Allows to write one byte to the Status Register of current LPT.

Language	Description
C/C++	void SetStatusPort (HANDLE hLpt , UCHAR nNewValue)
Delphi	procedure SetStatusPort (hLpt : THANDLE ; bNewValue : Byte) ; stdcall
VB	Sub SetPort (ByVal hLpt As Long , ByVal bNewValue As Byte)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
nNewValue - one byte value to write to Status Register.

Return Value :
None

See also: GetStatusPort

5.3.3.2.2.5 GetControlPort

Allows to read one byte from Control Register of current LPT.

Language	Description
C/C++	USHORT GetControlPort (HANDLE hLpt)
Delphi	function GetControlPort (hLpt : THANDLE) : Byte; stdcall
VB	Function GetControlPort (ByVal hLpt As Long) As Byte

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
The byte read from the LPT Control Register

See also: SetControlPort

5.3.3.2.2.6 SetControlPort

Allows to write one byte to the Control Register of current LPT.

Language	Description
C/C++	void SetControlPort (HANDLE hLpt , UCHAR nNewValue)
Delphi	procedure SetControlPort (hLpt : THANDLE ; bNewValue : Byte) ; stdcall
VB	Sub SetControlPort (ByVal hLpt As Long , ByVal bNewValue As Byte)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
nNewValue - one byte value to write to Control Register.

Return Value :
None

See also: GetControlPort

5.3.3.2.2.7 GetEcrPort

Allows to read one byte from ECR Register of current LPT.

Language	Description
C/C++	USHORT GetEcrPort (HANDLE hLpt)
Delphi	function GetEcrPort (hLpt : THANDLE) : Byte; stdcall
VB	Function GetEcrPort (ByVal hLpt As Long) As Byte

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
The byte read from the LPT ECR Register

See also: SetEcrPort

5.3.3.2.2.8 SetEcrPort

Allows to write one byte to the ECR Register of current LPT.

Language	Description
C/C++	void SetEcrPort (HANDLE hLpt , UCHAR nNewValue)
Delphi	procedure SetEcrPort (hLpt : THANDLE ; bNewValue : Byte) ; stdcall
VB	Sub SetEcrPort (ByVal hLpt As Long , ByVal bNewValue As Byte)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
nNewValue - one byte value to write to ECR Register.

Return Value :
None

See also: GetEcrPort

5.3.3.2.2.9 GetEPPAddressPort

Allows to read one byte from EPP Address Register of current LPT.

Language	Description
C/C++	USHORT GetEPPAddressPort (HANDLE hLpt)
Delphi	function GetEPPAddressPort (hLpt : THANDLE) : Byte; stdcall
VB	Function GetEPPAddressPort (ByVal hLpt As Long) As Byte

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
The byte read from the EPP Address Register

See also: SetEPPAddressPort

5.3.3.2.2.10 SetEPPAddressPort

Allows to write one byte to the EPP Address Register of current LPT.

Language	Description
C/C++	void SetEPPAddressPort (HANDLE hLpt , UCHAR nNewValue)
Delphi	procedure SetEPPAddressPort (hLpt : THANDLE ; bNewValue : Byte) ; stdcall
VB	Sub SetEPPAddressPort (ByVal hLpt As Long , ByVal bNewValue As Byte)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
nNewValue - one byte value to write to EPP Address Register.

Return Value :
None

See also: GetEPPAddressPort

5.3.3.2.2.11 GetEPPDataPort

Allows to read one byte from EPP Data Register of current LPT.

Language	Description
C/C++	USHORT GetEPPDataPort (HANDLE hLpt)
Delphi	function GetEPPDataPort (hLpt : THANDLE) : Byte; stdcall
VB	Function GetEPPDataPort (ByVal hLpt As Long) As Byte

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
The byte read from the EPP Data Register

See also: SetEPPDataPort

5.3.3.2.2.12 SetEPPDataPort

Allows to write one byte to the EPP Data Register of current LPT.

Language	Description
C/C++	void SetEPPDataPort (HANDLE hLpt , UCHAR nNewValue)
Delphi	procedure SetEPPDataPort (hLpt : THANDLE ; bNewValue : Byte) ; stdcall
VB	Sub SetEPPDataPort (ByVal hLpt As Long , ByVal bNewValue As Byte)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
nNewValue - one byte value to write to EPP Data Register.

Return Value :
None

See also: GetEPPDataPort

5.3.3.2.3 Centronix Signals

5.3.3.2.3.1 GetLptBusy

Returns the BUSY state of current LPT port.

Language	Description
C/C++	BOOL GetLptBusy (HANDLE hLpt)
Delphi	function GetLptBusy (hLpt : THANDLE) : Boolean ; stdcall
VB	Function GetLptBusy (ByVal hLpt As Long) As Boolean

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
TRUE - if ACTIVE or FALSE - if not

5.3.3.2.3.2 GetLptPaperEnd

Returns the Paper End state of current LPT port.

Language	Description
C/C++	BOOL GetLptPaperEnd (HANDLE hLpt)
Delphi	function GetLptPaperEnd (hLpt : THANDLE) : Boolean ; stdcall
VB	Function GetLptPaperEnd (ByVal hLpt As Long) As Boolean

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
TRUE - if ACTIVE or FALSE - if not

5.3.3.2.3.3 GetLptSlct

Returns the Select state of current LPT port.

Language	Description
C/C++	BOOL GetLptSlct (HANDLE hLpt)
Delphi	function GetLptSlct (hLpt : THANDLE) : Boolean ; stdcall
VB	Function GetLptSlct (ByVal hLpt As Long) As Boolean

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
TRUE - if ACTIVE or FALSE - if not

5.3.3.2.3.4 GetLptAutofd

Returns current AUTOFD state of current LPT port.

Language	Description
C/C++	BOOL GetLptAutofd (HANDLE hLpt)
Delphi	function GetLptAutofd (hLpt : THANDLE) : Boolean ; stdcall
VB	Function GetLptAutofd (ByVal hLpt As Long) As Boolean

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
TRUE - if ACTIVE or FALSE - if not

See also: SetLptAutofd

5.3.3.2.3.5 SetLptAutofd

Sets current AUTOFD state of current LPT port.

Language	Description
C/C++	void SetLptAutofd (HANDLE hLpt, BOOL Flag)
Delphi	procedure SetLptAutofd (hLpt : THANDLE, Flag : Boolean) ; stdcall
VB	Sub SetLptAutofd (ByVal hLpt As Long, ByVal Flag As Boolean)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
Flag - TRUE - if ACTIVE or FALSE - if not

Return Value :

None

See also: GetLptAutofd

5.3.3.2.3.6 GetLptError

Returns the ERROR state of current LPT port.

Language	Description
C/C++	BOOL GetLptError (HANDLE hLpt)
Delphi	function GetLptError (hLpt : THANDLE) : Boolean ; stdcall
VB	Function GetLptError (ByVal hLpt As Long) As Boolean

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
TRUE - if ACTIVE or FALSE - if not

5.3.3.2.3.7 LptInit

Initializes the printer.

Language	Description
C/C++	void LptInit (HANDLE hLpt)
Delphi	procedure LptInit (hLpt : THANDLE) ; stdcall
VB	Sub LptInit (ByVal hLpt As Long)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
None

Comments:
Pulses low to reset the printer and clear its buffer.

5.3.3.2.3.8 LptSelectIn

Enables or disables the printer data inputs.

Language	Description
C/C++	void LptSelectIn (HANDLE hLpt, BOOL Flag)
Delphi	procedure LptSelectIn (hLpt : THANDLE ; Flag : Boolean) ; stdcall
VB	Sub LptSelectIn (ByVal hLpt As Long, ByVal Flag As Boolean)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
Flag - 1 to enable printer data inputs, 0 to disable.

Return Value :
None

5.3.3.2.3.9 GetPrinterReady

Returns the ready state of the printer (printer has no error and is not busy).

Language	Description
C/C++	BOOL GetPrinterReady (HANDLE hLpt)
Delphi	function GetPrinterReady (hLpt : THANDLE) : Boolean ; stdcall
VB	Function GetPrinterReady (ByVal hLpt As Long) As Boolean

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
TRUE - if ready or FALSE - if not ready

5.3.3.2.4 Parallel Port Modes

5.3.3.2.4.1 GetCurrentLptMode

Returns the current mode of current LPT.

Language	Description
C/C++	USHORT GetCurrentLptMode (HANDLE hLpt)
Delphi	function GetCurrentLptMode (hLpt : THANDLE) : Byte; stdcall
VB	Function GetCurrentLptMode (ByVal hLpt As Long) As Byte

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
Current LPT mode from the list:

LPT_SPP_MODE
LPT_PS2_MODE
LPT_EPP_MODE
LPT_ECP_MODE

See also: SetCurrentLptMode

5.3.3.2.4.2 SetCurrentLptMode

Sets the current mode of current LPT.

Language	Description
C/C++	void SetCurrentLptMode (HANDLE hLpt , UCHAR nNewValue)
Delphi	procedure SetCurrentLptMode(hLpt: THANDLE; nNewValue: Byte); stdcall
VB	Sub SetCurrentLptMode (ByVal hLpt As Long , ByVal nNewValue As Byte)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
nNewValue - one from the list:

LPT_SPP_MODE
LPT_PS2_MODE
LPT_EPP_MODE
LPT_ECP_MODE

Return Value :
None

Comments:
You must read the value back immediately after you set the new LPT mode to make sure the selected mode is
supported by current LPT.

See also: GetCurrentLptMode

5.3.3.2.4.3 GetIsPresent

Indicates whether SPP mode is supported.

Language	Description
C/C++	USHORT GetIsPresent (HANDLE hLpt)
Delphi	function GetIsPresent (hLpt : THandle) : Word ; stdcall
VB	Function GetIsPresent (ByVal hLpt As Long) As Integer

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
Non-zero if supported, NULL otherwise.

Comments:
This information is unique for each port.
Note: you must call OpenPort function before running this function.

5.3.3.2.4.4 GetIsPS2Present

Indicates whether PS2 mode is supported.

Language	Description
C/C++	USHORT GetIsPS2Present (HANDLE hLpt)
Delphi	function GetIsPS2Present (hLpt : THandle) : Word ; stdcall
VB	Function GetIsPS2Present (ByVal hLpt As Long) As Integer

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
Non-zero if supported, NULL otherwise.

Comments:
This information is unique for each port.
Note: you must call OpenPort function before running this function.

5.3.3.2.4.5 GetIsEcpPresent

Indicates whether ECP mode is supported.

Language	Description
C/C++	USHORT GetIsEcpPresent (HANDLE hLpt)
Delphi	function GetIsEcpPresent (hLpt : THandle) : Word ; stdcall
VB	Function GetIsEcpPresent (ByVal hLpt As Long) As Integer

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

Return Value :
Non-zero if supported, NULL otherwise.

Comments:
This information is unique for each port.
Note: you must call OpenPort function before running this function.

5.3.3.2.5 Work with Pins

5.3.3.2.5.1 GetPin

Returns an electrical level from the select pin of curren LPT's D25 connector.

Language	Description
C/C++	UCHAR GetPin (HANDLE hLpt , UCHAR nPin);
Delphi	function GetPin (hLpt : THANDLE, nPin: Byte) : Byte ; stdcall
VB	Function GetPin (ByVal hLpt As Long , ByVal nPin As Byte) As Byte

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
nPin - requested pin number (1..25)

Return Value :
Zero if an electrical level is low, non-zero otherwise.

Comments:
The current LPT must be in read mode:
SetReadMode(1)

5.3.3.2.5.2 SetPin

Sets an electrical level from the select pin of curren LPT's D25 connector.

Language	Description
C/C++	void SetPin(HANDLE hLpt, UCHAR nPin, UCHAR nNewValue);
Delphi	procedure SetPin (hLpt : THANDLE, nPin : Byte, nNewValue: Byte) ; stdcall
VB	Sub SetPin (ByVal hLpt As Long, ByVal nPin As Byte, ByVal nNewValue As Byte)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
nPin - requested pin number (1..25)
nNewValue - set it to zero if desired electrical level on pin is LOW, non-zero if HIGH.

Return Value :
None.

Comments:
The current LPT must be in write mode:
SetReadMode(0)

Not all pins are accessible for this operation. Read LPT documentation for more info.

5.3.3.2.6 Work with Bits

5.3.3.2.6.1 GetDataPortBit

Returns the value (0 or 1) of the requested bit in a LPT's Data Register.

Language	Description
C/C++	UCHAR GetDataPortBit (HANDLE hLpt , UCHAR BitNumber)
Delphi	function GetDataPortBit (hLpt : THANDLE ; BitNumber : Byte) : Byte; stdcall
VB	Function GetDataPortBit (ByVal hLpt As Long , ByVal BitNumber As Byte) As Byte

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
BitNumber - requested bit number (0..7)

Return Value :
The value (0 or 1) of the requested bit

See also: SetDataPortBit

5.3.3.2.6.2 SetDataPortBit

Sets the value (0 or 1) of the requested bit in a LPT's Data Register.

Language	Description
C/C++	void SetDataPortBit (HANDLE hLpt , UCHAR BitNumber, UCHAR nNewValue)
Delphi	procedure SetDataPortBit (hLpt : THANDLE ; BitNumber : Byte ; nNewValue : Byte) ; stdcall
VB	Sub SetDataPortBit (ByVal hLpt As Long , ByVal BitNumber As Byte , ByVal nNewValue As Byte)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
BitNumber - requested bit number (0..7)
nNewValue - 1 to set or 0 to reset bit

Return Value :
None

See also: GetDataPortBit

5.3.3.2.6.3 GetStatusPortBit

Returns the value (0 or 1) of the requested bit in a LPT's Status Register.

Language	Description
C/C++	UCHAR GetStatusPortBit (HANDLE hLpt , UCHAR BitNumber)
Delphi	function GetStatusPortBit (hLpt : THANDLE ; BitNumber : Byte) : Byte; stdcall
VB	Function GetStatusPortBit (ByVal hLpt As Long , ByVal BitNumber As Byte) As Byte

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
BitNumber - requested bit number (0..7)

Return Value :
The value (0 or 1) of the requested bit

See also: SetStatusPortBit

5.3.3.2.6.4 SetStatusPortBit

Sets the value (0 or 1) of the requested bit in a LPT's Status Register.

Language	Description
C/C++	void SetStatusPortBit (HANDLE hLpt , UCHAR BitNumber, UCHAR nNewValue)
Delphi	procedure SetStatusPortBit (hLpt : THANDLE ; BitNumber : Byte ; nNewValue : Byte) ; stdcall
VB	Sub SetStatusPortBit (ByVal hLpt As Long , ByVal BitNumber As Byte , ByVal nNewValue As Byte)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
BitNumber - requested bit number (0..7)
nNewValue - 1 to set or 0 to reset bit

Return Value :
None

See also: GetStatusPortBit

5.3.3.2.6.5 GetControlPortBit

Returns the value (0 or 1) of the requested bit in a LPT's Control Register.

Language	Description
C/C++	UCHAR GetControlPortBit (HANDLE hLpt , UCHAR BitNumber)
Delphi	function GetControlPortBit (hLpt : THANDLE ; BitNumber : Byte) : Byte; stdcall
VB	Function GetControlPortBit (ByVal hLpt As Long , ByVal BitNumber As Byte) As Byte

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
BitNumber - requested bit number (0..7)

Return Value :
The value (0 or 1) of the requested bit

See also: SetControlPortBit

5.3.3.2.6.6 SetControlPortBit

Sets the value (0 or 1) of the requested bit in a LPT's Control Register.

Language	Description
C/C++	void SetControlPortBit (HANDLE hLpt , UCHAR BitNumber, UCHAR nNewValue)
Delphi	procedure SetControlPortBit (hLpt : THANDLE ; BitNumber : Byte ; nNewValue : Byte) ; stdcall
VB	Sub SetControlPortBit (ByVal hLpt As Long , ByVal BitNumber As Byte , ByVal nNewValue As Byte)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
BitNumber - requested bit number (0..7)
nNewValue - 1 to set or 0 to reset bit

Return Value :
None

See also: GetControlPortBit

5.3.3.2.6.7 GetCfgaPortBit

Returns the value (0 or 1) of the requested bit in a LPT's CFGA Register.

Language	Description
C/C++	UCHAR GetCfgaPortBit (HANDLE hLpt , UCHAR BitNumber)
Delphi	function GetCfgaPortBit (hLpt : THANDLE ; BitNumber : Byte) : Byte; stdcall
VB	Function GetCfgaPortBit (ByVal hLpt As Long , ByVal BitNumber As Byte) As Byte

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
BitNumber - requested bit number (0..7)

Return Value :
The value (0 or 1) of the requested bit

See also: SetCfgaPortBit

5.3.3.2.6.8 SetCfgaPortBit

Sets the value (0 or 1) of the requested bit in a LPT's CFGA Register.

Language	Description
C/C++	void SetCfgaPortBit (HANDLE hLpt , UCHAR BitNumber, UCHAR nNewValue)
Delphi	procedure SetCfgaPortBit (hLpt : THANDLE ; BitNumber : Byte ; nNewValue : Byte) ; stdcall
VB	Sub SetCfgaPortBit (ByVal hLpt As Long , ByVal BitNumber As Byte , ByVal nNewValue As Byte)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
BitNumber - requested bit number (0..7)
nNewValue - 1 to set or 0 to reset bit

Return Value :
None

See also: GetCfgaPortBit

5.3.3.2.6.9 GetCfgbPortBit

Returns the value (0 or 1) of the requested bit in a LPT's CFGB Register.

Language	Description
C/C++	UCHAR GetCfgbPortBit (HANDLE hLpt , UCHAR BitNumber)
Delphi	function GetCfgbPortBit (hLpt : THANDLE ; BitNumber : Byte) : Byte; stdcall
VB	Function GetCfgbPortBit (ByVal hLpt As Long , ByVal BitNumber As Byte) As Byte

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
BitNumber - requested bit number (0..7)

Return Value :
The value (0 or 1) of the requested bit

See also: SetCfgbPortBit

5.3.3.2.6.10 SetCfgbPortBit

Sets the value (0 or 1) of the requested bit in a LPT's CFGB Register.

Language	Description
C/C++	void SetCfgbPortBit (HANDLE hLpt , UCHAR BitNumber, UCHAR nNewValue)
Delphi	procedure SetCfgbPortBit (hLpt : THANDLE ; BitNumber : Byte ; nNewValue : Byte) ; stdcall
VB	Sub SetCfgbPortBit (ByVal hLpt As Long , ByVal BitNumber As Byte , ByVal nNewValue As Byte)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
BitNumber - requested bit number (0..7)
nNewValue - 1 to set or 0 to reset bit

Return Value :
None

See also: GetCfgbPortBit

5.3.3.2.6.11 GetEcrPortBit

Returns the value (0 or 1) of the requested bit in a LPT's ECR Register.

Language	Description
C/C++	UCHAR GetEcrPortBit (HANDLE hLpt , UCHAR BitNumber)
Delphi	function GetEcrPortBit (hLpt : THANDLE ; BitNumber : Byte) : Byte; stdcall
VB	Function GetEcrPortBit (ByVal hLpt As Long , ByVal BitNumber As Byte) As Byte

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
BitNumber - requested bit number (0..7)

Return Value :
The value (0 or 1) of the requested bit

See also: SetEcrPortBit

5.3.3.2.6.12 SetEcrPortBit

Sets the value (0 or 1) of the requested bit in a LPT's ECR Register.

Language	Description
C/C++	void SetEcrPortBit (HANDLE hLpt , UCHAR BitNumber, UCHAR nNewValue)
Delphi	procedure SetEcrPortBit (hLpt : THANDLE ; BitNumber : Byte ; nNewValue : Byte) ; stdcall
VB	Sub SetEcrPortBit (ByVal hLpt As Long , ByVal BitNumber As Byte , ByVal nNewValue As Byte)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
BitNumber - requested bit number (0..7)
nNewValue - 1 to set or 0 to reset bit

Return Value :
None

See also: SetEcrPortBit

5.3.3.3 Direct Port I/O

5.3.3.3.1 GetPortByte

Reads a byte from the specified port address.

Language	Description
C/C++	UCHAR GetPortByte (HANDLE hLpt, USHORT PortAddress);
Delphi	function GetPortByte (hLpt: THandle; PortAddress: Longword): Byte; stdcall;
VB	Function GetPortByte (ByVal hLpt As Long, ByVal PortAddress As Long) As Byte

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
PortAddress - the port address in I/O space

Return Value :
The byte read from the specified port address.

See also: GetPortWord GetPortLong SetPortByte SetPortWord SetPortLong

5.3.3.3.2 GetPortWord

Reads a WORD (two bytes) from the specified port address.

Language	Description
C/C++	USHORT GetPortWord (HANDLE hLpt, ULONG PortAddress);
Delphi	function GetPortWord (hLpt: THandle; PortAddress: Longword): Word; stdcall;
VB	Function GetPortWord (ByVal hLpt As Long, ByVal PortAddress As Long) As Integer

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
PortAddress - the port address in I/O space

Return Value :
The WORD value read from the specified port address

See also: GetPortByte GetPortLong SetPortByte SetPortWord SetPortLong

5.3.3.3.3 GetPortLong

Reads a double word (four bytes) from the specified port address.

Language	Description
C/C++	ULONG GetPortLong (HANDLE hLpt, ULONG PortAddress);
Delphi	function GetPortLong (hLpt: THandle; PortAddress: Longword): Longword; stdcall;
VB	Function GetPortLong (ByVal hLpt As Long, ByVal PortAddress As Long) As Long

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
PortAddress - the port address in I/O space

Return Value :
The double word value read from the specified port address.

See also: GetPortByte GetPortWord SetPortByte SetPortWord SetPortLong

5.3.3.3.4 SetPortByte

Writes a byte to the specified port address.

Language	Description
C/C++	void SetPortByte (HANDLE hLpt, ULONG PortAddress, UCHAR bValue);
Delphi	procedure SetPortByte (hLpt: THandle; PortAddress: Longword; bValue: Byte); stdcall;
VB	Sub SetPortByte (ByVal hLpt As Long, ByVal PortAddress As Long, ByVal bValue As Byte)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
PortAddress - the port address in I/O space
b*Value* - specifies a byte to be written to the port.

Return Value : None.

See also: GetPortByte GetPortWord GetPortLong SetPortWord SetPortLong

5.3.3.3.5 SetPortWord

Writes a word (two bytes) value to the specified port address.

Language	Description
C/C++	void SetPortWord (HANDLE hLpt, ULONG PortAddress, USHORT wValue);
Delphi	procedure SetPortWord (hLpt: THandle; PortAddress: Longword; wValue: Word); stdcall;
VB	Sub SetPortWord (ByVal hLpt As Long, ByVal PortAddress As Long, ByVal wValue As Integer)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
PortAddress - the port address in I/O space
w*Value* - specifies a word to be written to the port.

Return Value : None.

See also: GetPortByte GetPortWord GetPortLong SetPortByte SetPortLong

5.3.3.3.6 SetPortLong

Writes a double word (four bytes) value to the specified port address.

Language	Description
C/C++	void SetPortLong (HANDLE hLpt, ULONG PortAddress, ULONG dwValue);
Delphi	procedure SetPortLong (hLpt: THandle; PortAddress: Longword; dwValue: Longword); stdcall;
VB	Sub SetPortLong (ByVal hLpt As Long, ByVal PortAddress As Long, ByVal dwValue As Long)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
PortAddress - the port address in I/O space
dw*Value* - specifies a double word to be written to the port.

Return Value : None.

See also :GetPortByte GetPortWord GetPortLong SetPortByte SetPortWord

5.3.3.3.7 ReadPortBuffer

Reads a number of **bytes**, **words**, or **double words** from the specified port address into a buffer.

Language	Description
C/C++	void ReadPortBuffer(HANDLE hLpt, ULONG PortAddr, USHORT PortWidth, ULONG NumValues, void * buffer);
Delphi	procedure ReadPortBuffer (hLpt: THandle; PortAddr: LongWord; PortWidth: Word; NumValues: Longword; buffer: Pointer); stdcall;
VB	Sub ReadPortBuffer (ByVal hLpt As Long, ByVal PortAddr As Long, ByRef PortWidth As Integer; ByVal PortWidth As Long, ByRef buffer As Any)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt

PortAddr - port address;
PortWidth - size of port in bytes (1, 2, or 4);
NumValue - number of values to read
buffer - a buffer into which an array of values is read.

Return Value :None.

See also : WritePortBuffer

5.3.3.3.8 WritePortBuffer

Writes a number of **bytes**, **words**, or **double words** from a buffer to the specified port.

Language	Description
C/C++	void WritePortBuffer(HANDLE hLpt, ULONG PortAddr, USHORT PortWidth, ULONG NumValues, void * buffer);
Delphi	procedure WritePortBuffer (hLpt: THandle; PortAddr: LongWord; PortWidth: Word; NumValues: Longword; buffer: Pointer); stdcall;
VB	Sub WritePortBuffer (ByVal hLpt As Long, ByVal PortAddr As Long, ByRef PortWidth As Integer; ByVal PortWidth As Long, ByRef buffer As Any)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt
PortAddr - port address;
PortWidth - size of port in bytes (1, 2, or 4);
NumValue - number of values to read
buffer - a buffer from which an array of values is to be written.

Return Value :None.

See also: ReadPortBuffer

5.3.3.4 Hardware Interrupts

5.3.3.4.1 UnmaskLptIrq

Installs a hardware interrupt handler for the LPT device and unmasks the Irq at hardware level. UnmaskLptIrq allows to handle the "edge triggered" interrupts.

Language	Description
C/C++	void UnmaskLptIrq (HANDLE hLpt, TOnHwInterruptHandler InterruptHandler)
Delphi	procedure UnmaskLptIrq (hLpt: THandle; InterruptHandler TOnHwInterruptHandler); stdcall;
VB	Sub UnmaskLptIrq (ByVal hlpt As Long, ByVal HWHandler As Long)

Parameters:

hLpt - the handle returned by a successful call to OpenRapidLpt ;
InterruptHandler - address of the callback procedure that handles hardware interrupts for this Irq (use AddressOf specification in VB)

Return Value : None.

Comments:

1. The interrupt handler function must be declared as:

typedef void (__stdcall * TOnHwInterruptHandler)(ULONG TimeStampLowPart, ULONG TimeStampHighPart);

2. This function helps to handle "edge triggered" interrupts.

See also: MaskLptIrq GetInterruptCounter

5.3.3.4.2 MaskLptIrq

Stops the interrupt processing started by UnmaskLptIrq by masking the LPT Irq at hardware level.

Language	Description
C/C++	void MaskLptIrq(HANDLE hLpt);
Delphi	procedure MaskLptIrq (hLpt: THandle); stdcall;
VB	Sub MaskLptIrq (ByVal hLpt As Long)

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt ;

Return Value : None.

See also: UnmaskLptIrq GetInterruptCounter

5.3.3.4.3 GetInterruptCounter

Returns how many interruptions was processed inside of the driver.

Language	Description
C/C++	ULONG GetInterruptCounter(HANDLE hLpt);
Delphi	function GetInterruptCounter (hLpt: THandle): Longword; stdcall;
VB	Function GetInterruptCounter (ByVal hLpt As Long) As Long

Parameters:
hLpt - the handle returned by a successful call to OpenRapidLpt ;

Return Value :
Number of the interruptions processed by the drive.

See also: UnmaskLptIrq MaskLptIrq

5.4 Programming For USB Device

5.4.1 Overview

First of all, you must describe your device with ***RapidDriver Explorer*** then install a device using USB bus specific driver (RapidUSB.sys). Then you may create and run your own application to control a USB device for which the driver has been installed. All USB functions/procedures are placed in RapidUSB.dll interface library.

You can install the RapidUSB.sys driver for many USB devices, so every USB device will have it's own *DeviceInstance* parameter value in OpenRapidUSB function.

Note: *In some cases we will use the "**RapidUSB**" term assuming the RapidDriver Developer edition when we are working with a USB device through the RapidUSB.sys and RapidUSB.dll.*

Here is what you can do with RapidUSB:

- access USB device descriptors
- thread safe access to USB pipes
- support of Interrupt and Bulk pipes
- support devices with multiple interfaces
- support multiple devices simultaneously

To work with the driver itself the choice of programming language is not vital. At present RapidDriver package includes support of the following programming systems:

- Microsoft Visual C/C++
- Borland Delphi
- Borland C++ Builder
- Microsoft Visual Basic 6.0
- Microsoft Visual Basic .Net
- Microsoft Visual C#

5.4.2 Programmers Guide

5.4.2.1 Scenario

Before using any of the functions of RapidUSB interface DLL must be intinalized with call to UsbEnumerateDevices. After this friver can be requested to open handle of one of the devices enumerated during call to UsbEnumerateDevices. This is done with the help of the function OpenRapidUSB. It must be done once after starting the application. Also resources used by the driver should be cleaned up (CloseRapidUsb) before the application is closed.

Schematic session of using RapidUSB:

```
// find RapidUSB devices
nDevices = UsbEnumerateDevices();

if ( nDevices < 1 )
{
   // no RapidUSB devices present in system
   MessageBox("Cannot find any RapidUSB device!","Attention!", MB_OK |
   MB_ICONWARNING);
   return;
}

// initialize(open) the driver
hUSB =  OpenRapidUSB( 0 );

// Check if the driver initialized successfully

if ( IsRapidUSBOpened(hUSB)) {
        ...
   ////////////////////////////////////
   /////////  RapidUSB session //////////
   /////////  ............... //////////
   ////////////////////////////////////

   // Free resources used by the driver (close the driver)

   hUSB = CloseRapidUSB(hUSB);

}
else
{
   // An error happened while opening the driver
   MessageBox("Cannot open the driver","Attention!", MB_OK |
   MB_ICONWARNING);
}
```

Driver can be initialized by many applications simultaneously. The parameter of OpenRapidUSB() can be used to select a device supported by RapidUSB driver.

5.4.2.2 Data transfer

RapidUSB supports synchronous (blocking) and asynchronous types of transfer.

5.4.2.2.1 Synchronous Transfer

Synchronous Transfer
Transfer data in synchronous mode using function UsbTransfer is rather simple but it blocks calling

thread execution until transfer completes. The following is an example of blocking transfer assuming that all handles are opened. Error checks are omitted. For extended information about synchronous transfer refer to 'Pipes' example.

```
      USB_DEVICE_HANDLE hUsb;
      USB_PIPE_HANDLE hPipe;
      PVOID pBuffer;
      ULONG BufferLength, BytesReturned;

      // Open first RapidUSB device
      hUsb = OpenRapidUsb(0);

      // Open first RapidUSB device pipe
      hPipe = UsbPipeOpen(hUsb, 0);

      // Allocate transfer buffer
      pBuffer = malloc(BufferLength);

      // Start transfer...
      BOOL success = UsbTransfer(hPipe, pBuffer, BufferLength,
&BytesReturned);

      // thread waits transfer operation to finish

      if ( success)
      {
            // Transfer successful...
      }
      else
      {
            // Transfer failed....
      }

      free(pBuffer);

      UsbPipeClose(hPipe);
      CloseRapidUsb(hUsb);
```

5.4.2.2.2 Asynchronous Transfer

Asynchronous Transfer
Data transfer in asynchronous mode is done using functions UsbTransferAsync, UsbCancelTransfer and TRANSFER_COMPLETION_ROUTINE. This type of transfer is more complicated than synchronous but is allows an application to continue execution during transfer is pending or in process. Interface DLL calls user specified callback to signal that transfer is finished or canceled. For more information about asynchronous transfer refer to 'PipesAsync' example.

5.4.2.3 Reading Descriptors

Use one of functions described in Descriptors section to retrieve needed descriptor. This functions issue GetDescriptor USB request. For more infromation about standard USB descriptors refer to 9.6 section of USB 1.1 specification.
Use UsbGetDescriptor function to get descriptor of user-specified type and request recipinent. In some cases when descriptor is class (vendor) -specific use UsbClassRequestIn (UsbVendorRequestIn) to

retrieve it.
Functions UsbGetConfigDescriptor and UsbGetInterfaceDescriptor which retrieve descriptors of variable length allocate memory buffer to hold result. So after descriptor is no more needed call respective **UsbFreeXXXXXDescriptor** function to free allocated memory.

Note: For more information regarding reading descriptors refer to "Descriptors" example.

The following is an example of how to get a configuration descriptor:

```
// open first RapidUSB device
USB_HANDLE hUsb = OpenRapidUsb(0);

PUSB_CONFIGURATION_DESCRIPTOR pConfDesc;
UsbGetConfigDescriptor(hUsb, 0, pConfDesc);

if (pConfDesc != NULL)
{
      // configuration retrieved successfully
      // work with descriptor
      // ....................

      // Free descriptor when it is not needed
      UsbFreeConfigDescriptor(pConfDesc);
}
else
{
      // Error! Check if hUsb is valid
}


CloseRapidUsb(hUsb);
```

5.4.3 USB Support Routines And Structures

5.4.3.1 Common Procedures

5.4.3.1.1 UsbEnumerateDevices

Call this function before opening any device with OpenRapidUsb.

Language	Description
C/C++	UCHAR UsbEnumerateDevices ()
Delphi	function UsbEnumerateDevices () : Byte; stdcall
VB	Function UsbEnumerateDevices () As Byte

Return Value:
Return Value of this function is number of RapidUSB devices present in system.

Comments:
In most cases this function is called on application startup. It makes all arrangements for accessing other RapidUSB.dll functions. So call to OpenRapidUsb without preliminary call to UsbEnumerateDevices is not allowed.

5.4.3.1.2 OpenRapidUsb

Opens the RapidUSB.sys kernel mode WDM device driver under Windows 2000/XP/2003, providing direct access to the USB hardware.

Language	Description
C/C++	HANDLE OpenRapidUsb (ULONG DeviceInstance)
Delphi	function OpenRapidUsb (DeviceInstance : Longword) : THandle; stdcall
VB	Function OpenRapidUsb (ByVal DeviceInstance As Long) As Long

Parameters:
DeviceInstance - USB device instance.

Return Value:
The RapidUSB handle value or NULL if fails. Keep this handle opened all time when you are working with RapidUSB functions and close by CloseRapidUSB() at end of application.

Comments:

If the driver was successfully opened, the IsRapidUSBOpened() returns True; if the function fails, the IsRapidUSBOpened() returns False.
With RapidUSB you can support as many USB devices as you want. To select the device you need to change DeviceInstance value from 0 up to maximum number of USB device installed with RapidUSB driver. For the first application use

```
hUSB = OpenRapidUsb(0);
```

For second:

```
hUSB = OpenRapidUsb(1);
```

and so on.

See also: CloseRapidUSB IsRapidUSBOpened

5.4.3.1.3 IsRapidUsbOpened

This boolean function specifies whether the driver is opened.

Language	Description
C/C++	BOOL IsRapidUsbOpened (HANDLE hUSB)
Delphi	function IsRapidUsbOpened (hUSB : THandle) : BOOL; stdcall
VB	Function IsRapidUsbOpened (ByVal hUSB As Long) As Bool

Parameters:
hUSB - the handle returned by a successful call to OpenRapidUsb

Return Value :
TRUE if the driver was already opened by OpenRapidUsb(), or FALSE if it was not.

Comments:
Use immediately after OpenRapidUsb() call to check if the driver was successfully opened or not.

See also: OpenRapidUsb CloseRapidUSB

5.4.3.1.4 CloseRapidUsb

Closes the kernel-mode driver and releases memory allocated for it.

Language	Description
C/C++	HANDLE CloseRapidUsb (HANDLE hUSB)
Delphi	function CloseRapidUsb (hUSB: THandle): THandle; stdcall
VB	Function CloseRapidUsb (ByVal hUSB As Long) As Long

Parameters:
hUSB - the handle returned by a successful call to OpenRapidUsb

Return Value :
If function succeeds return value is TRUE.

See also: OpenRapidUsb IsRapidUsbOpened

5.4.3.2 Descriptors

Functions described in this section are used for retrieving descriptors from USB device.
UsbGetDescriptor
UsbGetDeviceDescriptor
UsbGetStringDescriptor
UsbGetConfigDescriptor
UsbFreeConfigDescriptor
UsbGetInterfaceDescriptor
UsbFreeInterfaceDescriptor
UsbGetEndpointDescriptor

5.4.3.2.1 UsbGetDescriptor

Language	Description

C/C++	`BOOL UsbGetDescriptor(IN USB_DEVICE_HANDLE hUsb,`
	`IN RDUSB_REQUEST_RECIPIENT RequestRecipient,`
	`IN UCHAR Type,`
	`IN UCHAR Index,`
	`OUT PUSB_COMMON_DESCRIPTOR pDescr,`
	`IN ULONG DescriptorLength);`

Delphi	`function UsbGetDescriptor(hUsb : USB_DEVICE_HANDLE;`
	`RequestRecipient : RDUSB_REQUEST_RECIPIENT;`
	`DescrType : UCHAR;`
	`Index : UCHAR;`
	`var Descr : USB_COMMON_DESCRIPTOR;`
	`Length : ULONG) : BOOL; stdcall;`

VB	`Function UsbGetDescriptor(ByVal hUsb As Long, _`
	`ByVal RequestRecipient As Byte, _`
	`ByVal bType As Byte, _`
	`ByVal Index As Byte, _`
	`ByRef pDescr As USB_COMMON_DESCRIPTOR, _`
	`ByVal DescrLength As Long) As Boolean`

Parameters

hUsb
Specifies handle to USB device.

RequestRecipient
Specifies recipient of get descriptor request.

Type
Indicates what type of descriptor is being retrieved.

Index
Descriptor index.

pDescr
Pointer to user allocated memory block. Retrieved descriptor is saved to this buffer on function successful return.

DescriptorLength
Specifies length of pDescr buffer.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Use this function to retrieve descriptor of any type. This function can be used only after getting valid handle from OpenRapidUsb function. User must allocate memory buffer which points to pDescr before handling pointer parameter to function.

Note

To get class specific descriptor (e.g. Hub descriptor class) use function UsbClassRequestIn.

5.4.3.2.2 UsbGetDeviceDescriptor

Language	Description
C/C++	`BOOL UsbGetDeviceDescriptor(IN  USB_DEVICE_HANDLE hUsb,` `OUT PUSB_DEVICE_DESCRIPTOR pDevDescr);`

Delphi	`function UsbGetDeviceDescriptor(hUsb : USB_DEVICE_HANDLE;`
	`var DevDescr :`
	`USB_DEVICE_DESCRIPTOR) : BOOL; stdcall;`

VB	`Function UsbGetDeviceDescriptor (ByVal hUsb As Long, _`
	`ByRef DevDescr As`
	`USB_DEVICE_DESCRIPTOR) As Boolean`

Parameters

hUsb
Specifies handle to USB device.

pDevDescr
Pointer to device descriptor.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Use this function to retrieve device descriptor. This function can be used only after getting valid handle from OpenRapidUsb function.

Example

```
if (UsbEnumerateDevices())
{
      // open first RapidUSB device
      USB_HANDLE hUsb = OpenRapidUsb(0);

      USB_DEVICE_DESCRIPTOR DeviceDescriptor;

      if (UsbGetDeviceDescriptor(hUsb, &DeviceDescriptor))
            // Descriptor retrieved successfully
      else
            // Error! Check if hUsb is valid
      CloseRapidUsb(hUsb);
}
```

5.4.3.2.3 UsbGetStringDescriptor

Language	Description

C/C++	`BOOL UsbGetStringDescriptor(IN USB_DEVICE_HANDLE hUsb,`
	`IN UCHAR Index,`
	`IN LANGID LanguageId,`
	`OUT PUSB_STRING_DESCRIPTOR pStrDescr,`
	`IN OUT PULONG pStrLen);`

Delphi	`function UsbGetStringDescriptor(hUsd : USB_DEVICE_HANDLE;`
	`Index : UCHAR;`
	`LanguaugeId : LANGID;`
	`var StrDescr : USB_STRING_DESCRIPTOR;`
	`var StrLen : Longint) : BOOL; stdcall;`

VB	`Function UsbGetStringDescriptor (ByVal hUsb As Long, _`
	`ByVal StrIndex As Byte, _`
	`ByVal LanguageId As Integer, _`
	`ByRef pDesc As USB_STRING_DESCRIPTOR, _`
	`ByRef Length As Long) As Boolean`

Parameters

hUsb
Specifies Handle to USB device.

Index
Specifies index of string descriptor.

LanguageId
Language ID of string descriptor.

pStrDescr
Points to user allocated memory. If function succeeds this buffer is filled with data from requested descriptor up to *pStrLen* length.

pStrLen
Length of *pStrDescr* buffer as input parameter. On successful return *pStrLen* equals to *pStrDescr->bLength.*

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Use this function to retrieve string contained in USB string descriptor. This function can be used only after getting valid handle from OpenRapidUsb function.

Example

```
// open first RapidUSB device
USB_HANDLE hUsb = OpenRapidUsb(0);

USB_STRING_DESCRIPTOR StrDescr;
PUSB_STRING_DESCRIPTOR pStrDescr;
ULONG nBytes;
```

```
        BOOL success;

        nBytes = sizeof(USB_STRING_DESCRIPTOR);

        // get length of string descriptor
        if (UsbGetStringDescriptor(hUsb, 1, 0x409, &StrDescr, &nBytes))
        {
              pStrDescr = (PUSB_STRING_DESCRIPTOR) calloc(1, nBytes + 2);

              if (UsbGetStringDescriptor(hUsb,StrIndex,LanguageId, pStrDescr,
&nBytes))
              {
                    // String retrieved successfully
              }

              free(pStrDescr);
        }

        CloseRapidUsb(hUsb);
```

5.4.3.2.4 UsbGetConfigDescriptor

Language	Description

C/C++	`BOOL UsbGetConfigDescriptor(IN USB_DEVICE_HANDLE hUsb,`
	`IN UCHAR Index,`
	`OUT PUSB_CONFIGURATION_DESCRIPTOR*`
	`pConfDescr);`

Delphi	`function UsbGetConfigDescriptor(hUsb : USB_DEVICE_HANDLE;`
	`Index : UCHAR;`
	`var pConfigDescr : PUSB_CONFIGURATION_DESCRIPTOR) : BOOL; stdcall;`

VB	`Function UsbGetConfigDescriptor (ByVal hUsb As Long, _`
	`ByVal ConfIndex As Byte, _`
	`ByRef pDesc As Long) As Boolean`

Parameters

hUsb
Specifies handle to USB device

Index
Device-defined index of configuration descriptor.

pConfDescr
Pointer to retrieved configuration descriptor.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Use this function to retrieve configuration descriptor. This function can be used only after getting valid handle from OpenRapidUsb function. Do not forget to free descriptor after use with UsbFreeConfigDescriptor.

Example

```
// open first RapidUSB device
USB_HANDLE hUsb = OpenRapidUsb(0);

PUSB_CONFIGURATION_DESCRIPTOR pConfDesc;
UsbGetConfigDescriptor(hUsb, 0, pConfDesc);

if (pConfDesc != NULL)
{
      // configuration retrieved successfully
      // work with descriptor
      // ....................

      // Free memory allocated for descriptor
      UsbFreeConfigDescriptor(pConfDesc);
}
else
{
      // Error! Check if hUsb is valid
}

CloseRapidUsb(hUsb);
```

5.4.3.2.5 UsbFreeConfigDescriptor

Language	Description
C/C++	`BOOL UsbFreeConfigDescriptor(IN PUSB_CONFIGURATION_DESCRIPTOR pConfDescr);`
Delphi	`function UsbFreeConfigDescriptor(pConfDescr : PUSB_CONFIGURATION_DESCRIPTOR) : BOOL; stdcall;`
VB	`Function UsbFreeConfigDescriptor (ByVal pConfigurationDescriptor As Long) As Boolean`

Parameters

pConfDescr

Specifies pointer to configuration descriptor obtained with UsbGetConfigDescriptor.

Description

Call this function to free memory allocated in UsbGetConfigDescriptor call.

5.4.3.2.6 UsbGetInterfaceDescriptor

Language	Description

C/C++	`BOOL UsbGetInterfaceDescriptor(IN USB_DEVICE_HANDLE hUsb,`
	`IN UCHAR Index,`
	`IN UCHAR AltIndex,`
	`OUT PUSB_INTERFACE_DESCRIPTOR*`
	`pIntDescr);`

Delphi	`function UsbGetInterfaceDescriptor(hUsb : USB_DEVICE_HANDLE;`
	`Index : UCHAR;`
	`AltIndex : UCHAR;`
	`var pIntDescr : PUSB_INTERFACE_DESCRIPTOR) : BOOL; stdcall;`

VB	`Function UsbGetInterfaceDescriptor (ByVal hUsb As Long, _`
	`ByVal MainIndex As Byte, _`
	`ByVal AlternateIndex As Byte, _`
	`ByRef pDesc As Long) As Boolean`

Parameters

hUsb
Specifies handle to USB device.

Index
Index of interface.

AltIndex
Number of alternate setting.

pIntDescr
Pointer to retrieved interface descriptor.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Use this function to retrieve interface descriptor. This function can be used only after getting valid handle from OpenRapidUsb function. Do not forget to free descriptor after use with UsbFreeInterfaceDescriptor.

Example

```
// open first RapidUSB device
USB_HANDLE hUsb = OpenRapidUsb(0);

PUSB_INTERFACE_DESCRIPTOR pIntDesc;

UsbGetInterfaceDescriptor(hUsb, 0, 0, pIntDesc);

if (pIntDesc != NULL)
{
```

```
        // interface retrieved successfully

        // Free descriptor when it is not needed
        UsbFreeInterfaceDescriptor(pIntDesc);
}
else
{
        // Error! Check if hUsb is valid
}
CloseRapidUsb(hUsb);
```

5.4.3.2.7 UsbFreeInterfaceDescriptor

Language	Description
C/C++	`BOOL UsbFreeInterfaceDescriptor(IN PUSB_INTERFACE_DESCRIPTOR* pIntDescr);`
Delphi	`function UsbFreeInterfaceDescriptor(pIntDescr : PUSB_INTERFACE_DESCRIPTOR) : BOOL; stdcall;`
VB	`Function UsbFreeInterfaceDescriptor (ByVal pInterfaceDescriptor As Long) As Boolean`

Parameters

pIntDescr

Specifies pointer to interface descriptor obtained with UsbGetInterfaceDescriptor.

Description

Call this function to free memory allocated in UsbGetInterfaceDescriptor call.

5.4.3.2.8 UsbGetEndpointDescriptor

Language	Description

C/C++	`BOOL UsbGetEndpointDescriptor(IN USB_DEVICE_HANDLE hUsb,`
	`IN UCHAR Index,`
	`IN UCHAR InterfaceIndex,`
	`IN UCHAR AlternateIndex,`
	`OUT PUSB_ENDPOINT_DESCRIPTOR`
	`pEndDescr);`

Delphi	`function UsbGetEndpointDescriptor(hUsb : USB_DEVICE_HANDLE;`
	`Index : UCHAR;`
	`InterfaceIndex : UCHAR;`
	`AlternateIndex :UCHAR;`
	`var EndDescr : USB_ENDPOINT_DESCRIPTOR) : BOOL; stdcall;`

VB	`Function UsbGetEndpointDescriptor (ByVal hUsb As Long, _`
	`ByVal EndpointIndex As Byte, _`
	`ByVal MainIndex As Byte, _`
	`ByVal AlternateIndex As Byte, _`
	`ByRef pDesc As USB_ENDPOINT_DESCRIPTOR) As Boolean`

Parameters

hUsb
Specifies handle to USB device.

Index
Endpoint descriptor index.

IntefaceIndex
Index of interface containing endpoint descriptor.

AlternateIndex
Index of alternate settings.

pEndDescr
Speicifies pointer to endpoint descriptor.

Description

Use this function to retrieve endpoint descriptor. This function can be used only after getting valid handle from OpenRapidUsb function.

Example

```
// open first RapidUSB device
USB_HANDLE hUsb = OpenRapidUsb(0);

USB_ENDPOINT_DESCRIPTOR EndDesc;

// getting second endpoint descriptor from first interface
if (UsbGetEndpointDescriptor(hUsb, 0, 0, 1, &EndDesc))
      // Descriptor retrieved successfully
else
```

```
        // Error! Check if hUsb and Index are valid
    CloseRapidUsb(hUsb);
```

5.4.3.3 Configure Device

Use functions described in this section to configure/unconfigure/reset device, select interface and get information about current configuration and interface.

UsbGetDeviceInfo
UsbGetPipeCount
UsbUnconfigureDevice
UsbSelectConfig
UsbGetConfig
UsbSelectInterface
UsbGetInterface
UsbResetDevice
UsbGetBandwidthInfo
UsbCyclePort

5.4.3.3.1 UsbGetDeviceInfo

Language	Description

C/C++	`BOOL UsbGetDeviceInfo(IN USB_DEVICE_HANDLE hUsb,`
	`IN OUT PRDUSB_DEVICE_INFORMATION pDevInfo,`
	`IN ULONG DevInfoLen);`

Delphi	`function UsbGetDeviceInfo(hUsb : USB_DEVICE_HANDLE;`
	`var DeviceInfo : RAPIDUSB_DEVICE_INFORMATION;`
	`DeviceInfoLen : ULONG) : BOOL stdcall;`

VB	`Function UsbGetDeviceInfo (ByVal hUsb As Long, _`
	`ByRef DevInfo As RDUSB_DEVICE_INFORMATION, _`
	`ByVal DevInfoLen As Long) As Boolean`

Parameters

hUsb
Specifies device handle. This handle can be obtained with OpenRapidUsb.

pDevInfo
Pointer to user allocated buffer.

DevInfoLen
Length of *pDevInfo* buffer.

Return Value

If device is in unconfigured state function returns FALSE and data containing in *pDevInfo* buffer is invalid .

Description

Call this function to get currently active settings of USB device.

5.4.3.3.2 UsbUnconfigureDevice

Language	Description
C/C++	`BOOL UsbUnconfigureDevice(IN USB_DEVICE_HANDLE hUsb);`
Delphi	`function UsbUnconfigureDevice(hUsb : USB_DEVICE_HANDLE) : BOOL; stdcall;`
VB	`Function UsbUnconfigureDevice (ByVal hUsb As Long) As Boolean`

Parameters

hUsb

Specifies device handle. This handle can be obtained with OpenRapidUsb.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Use this function to place device to unconfigured state. Before calling this function stop all transfers and close all opened pipes with UsbPipeClose function. All opened pipe handles became invalid.

5.4.3.3.3 UsbSelectConfig

Language	Description
C/C++	`BOOL UsbSelectConfig(IN USB_DEVICE_HANDLE hUsb,` `  IN UCHAR ConfigIndex);`
Delphi	`function UsbSelectConfig(hUsb : USB_DEVICE_HANDLE;` `  ConfigIndex : UCHAR) : BOOL; stdcall;`
VB	`Function UsbSelectConfig (ByVal hUsb As Long, _` `  ByVal ConfIndex As Byte) As Boolean`

Parameters

hUsb

Specifies device handle. This handle can be obtained with OpenRapidUsb.

ConfigIndex

index of configuration descriptor

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Use this function to select configuration if device have more than one configuration or if device is in unconfigured state. To place device to this state use UsbUnconfigureDevice. Before calling this function stop all transfers and close all opened pipes with UsbPipeClose function. All opened pipe handles became invalid after configuration selection.

5.4.3.3.4 UsbGetConfig

Language	Description
C/C++	`BOOL UsbGetConfig(IN  USB_DEVICE_HANDLE hUsb,` `OUT PUCHAR ConfigurationNum);`

Delphi	`function UsbGetConfig(hUsb : USB_DEVICE_HANDLE;`
	`var ConfigurationNum : UCHAR) : BOOL;`
	`stdcall;`

VB	`Function UsbGetConfig (ByVal hUsb As Long,` `ByRef ConfigNum As Byte) As Boolean`

Parameters

hUsb

Specifies handle to USB device

ConfigurationNum

Value returned from device.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Use this function to retrieve number of currently active configuration. See USB specification for more information

5.4.3.3.5 UsbSelectInterface

Language	Description

C/C++	`BOOL UsbSelectInterface(IN USB_DEVICE_HANDLE hUsb,`
	`IN UCHAR InterfaceIndex,`
	`IN UCHAR AlternateIndex,`
	`IN ULONG MaximumTransfer = 0);`

Delphi	`function UsbSelectInterface(hUsb : USB_DEVICE_HANDLE;`
	`InterfaceIndex : UCHAR;`
	`AlternateIndex : UCHAR;`
	`MaximumTransfer : ULONG {= 0}) : BOOL;`
	`stdcall;`

VB	`Function UsbSelectInterface (ByVal hUsb As Long, _`
	`ByVal InterfaceIndex As Byte, _`
	`ByVal AlternateIndex As Byte, _`
	`ByVal MaximumTransfer As Long) As`
	`Boolean`

Parameters

hUsb
Specifies device handle. This handle can be obtained with OpenRapidUsb.

InterfaceIndex
Device defined index of interface.

AlternateIndex
Device defined alternate interface index.

MaximumTransfer
Optional parameter. Maximum pipe transfer request size in bytes for newly selected interface.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Use this function to select interface if device have more than one interface or interface alternate setting. Before calling this function stop all transfers and close all opened pipes with UsbPipeClose function. All opened pipe handles became invalid after interface selection.

5.4.3.3.6 UsbGetInterface

Language	Description

C/C++	`BOOL UsbGetInterface(IN USB_DEVICE_HANDLE hUsb,`
	`IN UCHAR InterfaceIndex,`
	`IN PUCHAR InterfaceNum);`

Delphi	`function UsbGetInterface(hUsb : USB_DEVICE_HANDLE;`
	`InterfaceIndex : UCHAR;`
	`var InterfaceNum : UCHAR) : BOOL;`
	`stdcall;`

VB	`Function UsbGetInterface (ByVal hUsb As Long, _`
	`ByVal InterfaceIndex As Byte, _`
	`ByRef InterfaceNumber As Byte) As`
	`Boolean`

Parameters

hUsb
Specifies handle to USB device

InterfaceIndex
Device defined index of interface

InterfaceNum
Value returned from device

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Function return in InterfaceNum currently active alternate setting for a particular interface.

Note

This function is not supported in WindowsXP.

5.4.3.3.7 UsbResetDevice

Language	Description
C/C++	`BOOL UsbResetDevice(IN USB_DEVICE_HANDLE hUsb);`
Delphi	`function UsbResetDevice(hUsb : USB_DEVICE_HANDLE) : BOOL; stdcall;`
VB	`Function UsbResetDevice (ByVal hUsb As Long) As Boolean`

Parameters

hUsb
Specifies device handle. This handle can be obtained with OpenRapidUsb.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

This function resets USB device. Before calling this function stop all transfers and close all opened pipes with UsbPipeClose function. All opened pipe handles became invalid.

5.4.3.3.8 UsbGetBandwidthInfo

Language	Description

C/C++	`BOOL UsbGetBandwidthInfo(IN USB_DEVICE_HANDLE hUsb`
	`OUT PRDUSB_BANDWIDTH_INFORMATION`
	`BandwidthInfo);`

Delphi	`function UsbGetBandwidthInfo(hUsb : USB_DEVICE_HANDLE;`
	`var BandwidthInfo :`
	`RDUSB_BANDWIDTH_INFORMATION) : BOOL; stdcall;`

VB	`Function UsbGetBandwidthInfo (ByVal hUsb As Long, _`
	`ByRef BandwidthInfo As`
	`RDUSB_BANDWIDTH_INFORMATION) As Boolean`

Parameters

hUsb
Specifies device handle. This handle can be obtained with OpenRapidUsb.

BandwidthInfo
Pointer to RDUSB_BANDWIDTH_INFORMATION structure.

Return value

If function succeeds return value is TRUE.

Description

Use this function to retrieve information about total and consumed bandwidth on USB bus.

5.4.3.3.9 UsbCyclePort

Language	Description
C/C++	`BOOL UsbCyclePort(IN USB_DEVICE_HANDLE hUsb);`
Delphi	`function UsbCyclePort(hUsb : USB_DEVICE_HANDLE) : BOOL; stdcall;`
VB	`Function UsbCyclePort (hUsb As Long)`

Parameters

hUsb
Specifies device handle. This handle can be obtained with OpenRapidUsb.

Return value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Function simulates a device unplug and replug on the connected hub port. Before calling this function stop all transfers and close all opened pipes with UsbPipeClose function. All opened pipe handles became invalid after this function call.

5.4.3.4 Features and Status

This section describes functions for set/clear features and retrieve status from USB device, interface or endpoint.

UsbSetFeature
UsbClearFeature
UsbGetStatus

5.4.3.4.1 UsbSetFeature

Language	Description

C/C++	`BOOL UsbSetFeature(IN USB_DEVICE_HANDLE hUsb,`
	`IN RDUSB_REQUEST_RECIPIENT RequestRecipient,`
	`IN USHORT FeatureSelector,`
	`IN USHORT Index);`

Delphi	`function UsbSetFeature(hUsb : USB_DEVICE_HANDLE;`
	`RequestRecipient : RDUSB_Request_Recipient;`
	`FeatureSelector : USHORT;`
	`Index : USHORT) : BOOL;`
	`stdcall;`

VB	`Function UsbSetFeature (ByVal hUsb As Long, _`
	`ByVal Recepient As Byte, _`
	`ByVal FeatSelector As Byte, _`
	`ByVal FeatIndex As Byte) As Boolean`

Parameters

hUsb
Specifies device handle. This handle can be obtained with OpenRapidUsb.

RequestRecipient
Specifies recipient of feature request.

FeatureSelector
Specifies the USB-defined feature code that should be set on the target specified by *RequestRecipient*.

Index
For a feature request for an endpoint or interface, specifies the index of the endpoint or interface within the configuration descriptor. For the device, this must be zero.

Return Value

If function fails it return FALSE. To detect error call GetLastError function.

Description

UsbSetFeature sends request to set specified feature of USB device.

5.4.3.4.2 UsbClearFeature

Language	Description

C/C++	`BOOL UsbClearFeature(IN USB_DEVICE_HANDLE hUsb,`
	`IN RDUSB_REQUEST_RECIPIENT RequestRecipient,`
	`IN USHORT FeatureSelector,`
	`IN USHORT Index);`

Delphi	`function UsbClearFeature(hUsb : USB_DEVICE_HANDLE;`
	`RequestRecipient : RDUSB_Request_Recipient;`
	`FeatureSelector : USHORT;`
	`Index : USHORT) : BOOL; stdcall;`

VB	`Function UsbClearFeature (ByVal hUsb As Long, _`
	`ByVal Recepient As Byte, _`
	`ByVal FeatSelector As Byte, _`
	`ByVal FeatIndex As Byte) As Boolean`

Parameters

hUsb
Specifies device handle. This handle can be obtained with OpenRapidUsb.

RequestRecipient
Specifies recipient of feature request.

FeatureSelector
Specifies the USB-defined feature code that should be cleared on the target specified by *RequestRecipient*.

Index
For a feature request for an endpoint or interface, specifies the index of the endpoint or interface within the configuration descriptor. For the device, this must be zero.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

UsbClearFeature sends request to clear specified feature of USB device.

5.4.3.4.3 UsbGetStatus

Language	Description

C/C++	`BOOL UsbGetStatus(IN USB_DEVICE_HANDLE hUsb,`
	`IN RDUSB_REQUEST_RECIPIENT RequestRecipient,`
	`IN USHORT Index,`
	`OUT PUSHORT Status);`

Delphi	`function UsbGetStatus(hUsb : USB_DEVICE_HANDLE;`
	`RequestRecipient : RDUSB_Request_Recipient;`
	`Index : USHORT;`
	`var Status : USHORT): BOOL; stdcall;`

VB	`Function UsbGetStatus (ByVal hUsb As Long, _`
	`ByVal Recepient As Byte, _`
	`ByVal FeatIndex As Byte, _`
	`ByRef Status As Integer) As Boolean`

Parameters

hUsb
Device handle. This handle can be obtained with OpenRapidUsb.

RequestRecipient
Specifies recipient of feature request.

FeatureSelector
Specifies the USB-defined feature code that should be cleared on the target specified by *RequestRecipient*.

Index
Specifies the device-defined index, returned by a successful configuration request, if the request is for an endpoint or interface. Otherwise, Index must be zero.

Status
Output parameter. If function succeeds, Status contains *RequestRecipient* status data.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

`UsbGetStatus` is used to obtain state of *RequestRecipient* features.

5.4.3.5 Vendor/Class Requests

This section describes functions for issuing Vendor/Class requests.
UsbVendorRequestIn
UsbVendorRequestOut
UsbClassRequestIn
UsbClassRequestOut

5.4.3.5.1 UsbVendorRequestIn

Language	Description
C/C++	BOOL UsbVendorRequestIn(IN USB_DEVICE_HANDLE hUsb, IN RDUSB_REQUEST_RECIPIENT RequestRecipient, IN UCHAR ReservedBits, IN UCHAR Request, IN USHORT Value, IN USHORT Index, OUT LPVOID lpBuffer, IN DWORD nNumberOfBytesToTransfer, OUT LPDWORD lpNumberOfBytesTransfered);
Delphi	function UsbVendorRequestIn(hUsb : USB_DEVICE_HANDLE; RequestRecipient : RDUSB_REQUEST_RECIPIENT; ReservedBits : UCHAR; Request : UCHAR; Value : USHORT; Index : USHORT; var Buffer; nNumberOfBytesToTransfer : DWORD; var lpNumberOfBytesTransfered : DWORD): BOOL; stdcall;
VB	Function UsbVendorRequestIn (ByVal hUsb As Long, _ ByVal RequestRecipient As Byte, _ ByVal ReservedBits As Byte, _ ByVal Request As Byte, _ ByVal Value As Integer, _ ByVal Index As Integer, _ ByRef Buffer As Any, _ ByVal NumberOfBytesToTransfer As Long, _ ByRef NumberOfBytesTransfered As Long) As Boolean

Parameters

hUsb
Specifies device handle. This handle can be obtained with OpenRapidUsb.

RequestRecipient
Specifies recipient of vendor request.

ReservedBits
Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by vendor.

Request
Specifies vendor-defined request code for the *RequestRecipient*.

Value

Specifies a value, specific to *Request*, that becomes part of the USB-defined setup packet for the *RequestRecipient*. This value is defined by the creator of the code used in *Request*.

Index

Specifies the device-defined index, returned by a successful configuration request, if the request is for an endpoint or interface. Otherwise, *Index* must be zero.

lpBuffer

Output parameter. Pointer to buffer for the transfer. This buffer will contain data read from the device.

nNumberOfBytesToTransfer

Specifies the length, in bytes, of the buffer specified in *lpBuffer*.

lpNumberOfBytesTransfered

UsbVendorRequestIn returns the number of bytes read from device in this parameter.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

UsbVendorRequestIn is called to issue a vendor-specific command to a device, interface, endpoint or other device-defined target. Transfer direction is from device to host (IN).

5.4.3.5.2 UsbVendorRequestOut

Language	Description
C/C++	BOOL UsbVendorRequestOut(IN USB_DEVICE_HANDLE hUsb, IN RDUSB_REQUEST_RECIPIENT RequestRecipient, IN UCHAR ReservedBits, IN UCHAR Request, IN USHORT Value, IN USHORT Index, IN const LPVOID lpBuffer, IN DWORD nNumberOfBytesToTransfer, OUT LPDWORD lpNumberOfBytesTransfered);
Delphi	function UsbVendorRequestOut(hUsb : USB_DEVICE_HANDLE; RequestRecipient : RDUSB_REQUEST_RECIPIENT; ReservedBits : UCHAR; Request : UCHAR; Value : USHORT; Index : USHORT; const Buffer : LPVOID; nNumberOfBytesToTransfer : DWORD; var lpNumberOfBytesTransfered : DWORD): BOOL; stdcall;
VB	Function UsbVendorRequestOut (ByVal hUsb As Long, _ ByVal RequestRecipient As Byte, _ ByVal ReservedBits As Byte, _ ByVal Request As Byte, _ ByVal Value As Integer, _ ByVal Index As Integer, _ ByRef Buffer As Any, _ ByVal NumberOfBytesToTransfer As Long, _ ByRef NumberOfBytesTransfered As Long) As Boolean

Parameters

hUsb

Device handle. This handle can be obtained with OpenRapidUsb.

RequestRecipient

Specifies recipient of vendor request.

ReservedBits

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by vendor.

Request

Specifies vendor-defined request code for the *RequestRecipient*.

Value

Specifies a value, specific to Request, that becomes part of the USB-defined setup packet for the *RequestRecipient*. This value is defined by the creator of the code used in *Request*.

Index

Specifies the device-defined index, returned by a successful configuration request, if the request is for an endpoint or interface. Otherwise, Index must be zero.

lpBuffer

Pointer to buffer for the transfer. This buffer contains user-supplied data for transfer to the device.

nNumberOfBytesToTransfer

Specifies the length, in bytes, of the buffer specified in *lpBuffer*.

lpNumberOfBytesTransfered

UsbVendorRequestOut returns the number of bytes sent to *RequestRecipient* in this parameter.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

UsbVendorRequestOut is called to issue a vendor-specific command to a device, interface, endpoint or other device-defined target. Transfer direction is from host to device (OUT).

5.4.3.5.3 UsbClassRequestIn

Language	Description
C/C++	BOOL UsbClassRequestIn(IN USB_DEVICE_HANDLE hUsb, IN RDUSB_REQUEST_RECIPIENT RequestRecipient, IN UCHAR ReservedBits, IN UCHAR Request, IN USHORT Value, IN USHORT Index, IN const LPVOID lpBuffer, IN DWORD nNumberOfBytesToTransfer, OUT LPDWORD lpNumberOfBytesTransfered);
Delphi	function UsbClassRequestIn(hUsb : USB_DEVICE_HANDLE; RequestRecipient : RDUSB_REQUEST_RECIPIENT; ReservedBits : UCHAR; Request : UCHAR; Value : USHORT; Index : USHORT; var Buffer; nNumberOfBytesToTransfer : DWORD; var lpNumberOfBytesTransfered : DWORD): BOOL; stdcall;
VB	Function UsbClassRequestIn (ByVal hUsb As Long, _ ByVal RequestRecipient As Byte, _ ByVal ReservedBits As Byte, _ ByVal Request As Byte, _ ByVal Value As Integer, _ ByVal Index As Integer, _ ByRef Buffer As Any, _ ByVal NumberOfBytesToTransfer As Long, _ ByRef NumberOfBytesTransfered As Long) As Boolean

Parameters

hUsb
Device handle. This handle can be obtained with OpenRapidUsb.

RequestRecipient
Specifies recipient of feature request.

ReservedBits
Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request.

Request
Specifies USB-defined request code for the *RequestRecipient*.

Value

Specifies a value, specific to *Request*, that becomes part of the USB-defined setup packet for the *RequestRecipient*. This value is defined by the creator of the code used in *Request*.

Index

Specifies the device-defined index, returned by a successful configuration request, if the request is for an endpoint or interface. Otherwise, Index must be zero.

lpBuffer

Output parameter. Pointer to buffer for the transfer. This buffer will contain data read from the device.

nNumberOfBytesToTransfer

Specifies the length, in bytes, of the buffer specified in *lpBuffer*.

lpNumberOfBytesTransfered

UsbClassRequestIn returns the number of bytes read from device in this parameter.

Return Value

If function fails it return FALSE. To detect error call GetLastError function.

Description

UsbClassRequestIn is called to issue a class-specific command to a device, interface, endpoint or other device-defined target. Transfer direction is from device to host (IN).

5.4.3.5.4 UsbClassRequestOut

Language	Description
C/C++	BOOL UsbClassRequestOut(IN USB_DEVICE_HANDLE hUsb, IN RDUSB_REQUEST_RECIPIENT RequestRecipient, IN UCHAR ReservedBits, IN UCHAR Request, IN USHORT Value, IN USHORT Index, OUT LPVOID lpBuffer, IN DWORD nNumberOfBytesToTransfer, OUT LPDWORD lpNumberOfBytesTransfered);
Delphi	function UsbClassRequestOut(hUsb : USB_DEVICE_HANDLE; RequestRecipient : RDUSB_REQUEST_RECIPIENT; ReservedBits : UCHAR; Request : UCHAR; Value : USHORT; Index : USHORT; const Buffer : LPVOID; nNumberOfBytesToTransfer : DWORD; var lpNumberOfBytesTransfered : DWORD): BOOL; stdcall;
VB	Function UsbClassRequestOut (ByVal hUsb As Long, _ ByVal RequestRecipient As Byte, _ ByVal ReservedBits As Byte, _ ByVal Request As Byte, _ ByVal Value As Integer, _ ByVal Index As Integer, _ ByRef Buffer As Any, _ ByVal NumberOfBytesToTransfer As Long, _ ByRef NumberOfBytesTransfered As Long) As Boolean

Parameters

hUsb
Specifies Device handle. This handle can be obtained with OpenRapidUsb.

RequestRecipient
Specifies recipient of feature request.

ReservedBits
Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request..

Request

Specifies USB-defined request code for the *RequestRecipient*.

Value
Specifies a value, specific to *Request*, that becomes part of the USB-defined setup packet for the *RequestRecipient*. This value is defined by the creator of the code used in *Request*.

Index
Specifies the device-defined index, returned by a successful configuration request, if the request is for an endpoint or interface. Otherwise, Index must be zero.

lpBuffer
Pointer to buffer for the transfer. This buffer contains user-supplied data for transfer to the device.

nNumberOfBytesToTransfer
Specifies the length, in bytes, of the buffer specified in *lpBuffer*.

lpNumberOfBytesTransfered
UsbClassRequestOut returns the number of bytes sent to device in this parameter.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

UsbClassRequestOut is called to issue a class-specific command to a device, interface, endpoint or other device-defined target. Transfer direction is from host to device (OUT).

5.4.3.6 Pipes

This section describes functions working with pipes.

5.4.3.6.1 IsRapidUsbPipeOpened

Language	Description
C/C++	`BOOL IsRapidUsbPipeOpened(IN USB_PIPE_HANDLE hPipe);`
Delphi	`function IsRapidUsbPipeOpened(hPipe : USB_PIPE_HANDLE) : BOOL; stdcall;`
VB	`Function IsRapidUsbPipeOpened (ByVal hPipe As Long) As Boolean`

Parameters

hPipe
Specifies pipe handle. This handle can be obtained with OpenRapidUsb.

Return Value

Function returns TRUE if pipe is opened.

Description

Call this function to check if *hPipe* is handle of opened USB device pipe. You cannot perform any transfer or other pipe operation if this function returns FALSE. To make pipe active call UsbPipeOpen function.

5.4.3.6.2 UsbPipeOpen

Language	Description
C/C++	`USB_PIPE_HANDLE UsbPipeOpen(IN USB_DEVICE_HANDLE hUsb,` `IN UCHAR PipeNum);`
Delphi	`function UsbPipeOpen(hUsb : USB_DEVICE_HANDLE;` `PipeNum : UCHAR) : USB_PIPE_HANDLE; stdcall;`
VB	`Function UsbPipeOpen (ByVal hUsb As Long, _` `ByVal PipeNumber As Byte) As Long`

Parameters

hUsb
Specifies device handle. This handle can be obtained with OpenRapidUsb.

PipeNum
Number of pipe to make active.

Return Value

Function returns valid pipe handle if succeed. If function fails return value is INVALID_HANDLE_VALUE.

Description

Function returns handle to USB pipe of specified number. Pipe is taken from active interface. All input/output operations can be performed only after successful call to this function (return value is not INVALID_HANDLE_VALUE). When all operations with pipe are finished call UsbPipeClose.

5.4.3.6.3 UsbPipeClose

Language	Description
C/C++	`BOOL UsbPipeClose(IN USB_PIPE_HANDLE hPipe);`
Delphi	`function UsbPipeClose(hPipe : USB_PIPE_HANDLE) : BOOL; stdcall;`
VB	`Function UsbPipeClose (ByVal hPipe As Long) as Boolean`

Parameters

hPipe

Specifies pipe handle. This handle can be obtained with UsbPipeOpen.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Function closes handle obtained with UsbPipeOpen. Call **UsbPipeClose** when all work with pipe is finished.

5.4.3.6.4 UsbGetPipeCount

Language	Description
C/C++	`BOOL UsbGetPipeCount(IN USB_DEVICE_HANDLE hUsb,` `OUT PUCHAR pPipeCount);`
Delphi	`function UsbGetPipeCount(hUsb : USB_DEVICE_HANDLE;` `var PipeCount : UCHAR) : BOOL; stdcall;`
VB	`Function UsbGetPipeCount (ByVal hUsb As Long, _` `ByRef pipeCount As Byte) As Boolean`

Parameters

hUsb

device handle. This handle can be obtained with OpenRapidUsb.

pPipeCount

If function succeeds this parameter points to number of pipes in active interface

Return Value

If function fails it return FALSE and value of PipeCount is undefined. To detect error call **GetLastError** function.

Description

Call this function to retrieve number of pipes. If device is in unconfigured state p*PipeCount* is zero

5.4.3.6.5 UsbPipeGetInfo

Language	Description
C/C++	`BOOL UsbPipeGetInfo(IN  USB_PIPE_HANDLE hPipe,` `OUT PRDUSB_PIPE_INFORMATION  pPipeInfo);`

Delphi	`function UsbPipeGetInfo(hPipe : USB_PIPE_HANDLE;`
	`var PipeInfo : RAPIDUSB_PIPE_INFORMATION)`
	`: BOOL; stdcall;`

VB	`Function UsbPipeGetInfo (ByVal hPipe As Long, _`
	`ByRef PipeInfo As RDUSB_PIPE_INFORMATION)`
	`As Boolean`

Parameters

hPipe

Specifies pipe handle. This handle can be obtained with UsbPipeOpen.

pPipeInfo

Pointer to RDUSB_PIPE_INFORMATION structure.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Call this function to define currently active settings of USB pipe.

5.4.3.6.6 UsbPipeDirectionIn

Language	Description
C/C++	`BOOL UsbPipeDirectionIn(IN  USB_PIPE_HANDLE hPipe,` `OUT PBOOL pDirectionIn);`

Delphi	`function UsbPipeDirectionIn(hPipe : USB_PIPE_HANDLE;`
	`var DirectionIn : BOOL) : BOOL;`
	`stdcall;`

VB	`Function UsbPipeDirectionIn (ByVal hPipe As Long, _`
	`ByRef pDirectionIn As Integer) As`
	`Boolean`

Parameters

hPipe

Specifies pipe handle. This handle can be obtained with UsbPipeOpen.

pDirectionIn

Pointer to boolean variable storing TRUE if pipe handles data from device to host (direction IN) or FALSE if pipe sends data from host to device (OUT).

Return Value

If function fails or pipe type is control the return value is FALSE. To detect error call **GetLastError** function.

Description

Call this function to get pipe direction.

5.4.3.6.7 UsbPipeGetType

Language	Description
C/C++	`BOOL UsbPipeGetType(IN  USB_PIPE_HANDLE hPipe,` `OUT RDUSB_PIPE_TYPE* pPipeType);`

Delphi	`function UsbPipeGetType(hPipe : USB_PIPE_HANDLE;`
	`var PipeType : RAPIDUSB_PIPE_TYPE) : BOOL;`
	`stdcall;`

VB	`Function UsbPipeGetType (ByVal hPipe As Long, _` `ByRef PipeType As Long) As Boolean`

Parameters

hPipe

Specifies pipe handle. This handle can be obtained with UsbPipeOpen.

pPipeType

Output parameter. Pointer to RDUSB_PIPE_TYPE.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Function retrieves pipe type.

5.4.3.6.8 UsbTransfer

Language	Description

C/C++	`BOOL UsbTransfer(IN  USB_PIPE_HANDLE hPipe,`
	`IN  OUT LPVOID pBuffer,`
	`IN  DWORD nNumberOfBytesToTransfer,`
	`OUT LPDWORD pNumberOfBytesTransfered);`

Delphi	`function UsbWrite(hPipe : USB_PIPE_HANDLE;`
	`const Buffer : LPVOID;`
	`nNumberOfBytesToTransfer : DWORD;`
	`var lpNumberOfBytesTransfered : DWORD) : BOOL;`
	`stdcall;`
	
	`function UsbRead(hPipe : USB_PIPE_HANDLE;`
	`var Buffer;`
	`nNumberOfBytesToTransfer : DWORD;`
	`var lpNumberOfBytesTransfered : DWORD) : BOOL;`
	`stdcall;`

VB	`Function UsbTransfer (ByVal hPipe As Long, _`
	`ByRef Buffer As Any, _`
	`ByVal NumberOfBytesToTransfer As Long, _`
	`ByRef NumberOfBytesTransferred As Long) As Boolean`

Parameters

hPipe

Pipe handle. This handle can be obtained with UsbPipeOpen.

pBuffer

Pointer to a resident buffer for the transfer. The contents of this buffer depend on pipe direction. If pipe direction is IN (from device to host) this buffer will contain data read from the device. Otherwise, this buffer contains user-supplied data for transfer to the device.

nNumberOfBytesToTransfer

Specifies the length, in bytes, of the buffer specified in *lpBuffer*.

pNumberOfBytesTransfered

Function returns the number of bytes sent to or read from the pipe in this parameter.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Use this function to transfer data to/from USB device pipe. Function performs transfer with bulk, interrupt and isochronous pipes. Transfer is made in blocking mode. Thread calling this function is blocked until transfer completes.

5.4.3.6.9 UsbTransferAsync

Language	Description

C/C++	`BOOL UsbTransferAsync(IN USB_PIPE_HANDLE hPipe,`
	`IN OUT LPVOID pBuffer,`
	`IN DWORD nNumberOfBytesToTransfer,`
	`IN PTRANSFER_COMPLETION_ROUTINE`
	`pCompletionRoutine);`

Delphi	`function UsbWriteAsync(hPipe : USB_PIPE_HANDLE;`
	`const Buffer : LPVOID;`
	`nNumberOfBytesToTransfer : DWORD;`
	`CompletionRoutine :`
	`TTRANSFER_COMPLETION_ROUTINE) : BOOL; stdcall;`
	
	`function UsbReadAsync(hPipe : USB_PIPE_HANDLE;`
	`var Buffer;`
	`nNumberOfBytesToTransfer : DWORD;`
	`CompletionRoutine :`
	`TTRANSFER_COMPLETION_ROUTINE) : BOOL; stdcall;`

VB	`Function UsbTransferAsync (ByVal hPipe As Long, _`
	`ByRef lpBuffer As Any, _`
	`ByVal nNumberOfBytesToTransfer As Long, _`
	`ByVal pCompletionRoutine As Long) As Boolean`

Parameters

hPipe

Pipe handle. This handle can be obtained with UsbPipeOpen.

pBuffer

Pointer to a resident buffer for the transfer. The contents of this buffer depend on pipe direction. If pipe direction is IN (from device to host) this buffer will contain data read from the device. Otherwise, this buffer contains user-supplied data for transfer to the device.

nNumberOfBytesToTransfer

Specifies the length, in bytes, of the buffer specified in *lpBuffer*.

pCompletionRoutine

Pointer to the completion routine to be called when the transfer operation is complete. For more information about transfer completion routine, see TRANSFER_COMPLETION_ROUTINE.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Use this function to transfer data to/from USB device pipe. Function performs transfer with bulk, interrupt and isochronous pipes. The **UsbTransferAsync** transfers data asynchronously. Function lets an application perform other processing during a transfer operation. When an operation completes *pCompletionRoutine* routine is called.

5.4.3.6.10 TRANSFER_COMPLETION_ROUTINE

Language	Description

C/C++	`typedef void (__stdcall *PTRANSFER_COMPLETION_ROUTINE)(`
	`ULONG dwErrorCode,`
	`ULONG dwNumberOfBytesTransfered);`

Delphi	`TTRANSFER_COMPLETION_ROUTINE = procedure (ErrorCode : Longint;` `NumberOfBytesTransfered : Longint); stdcall;`
VB	`Sub CompletionRoutine(ByVal ErrorCode As Long, ByVal NumberOfBytesTransfered As Long)`

Parameters

dwErrorCode
If transfer was successful error code is zero.

dwNumberOfBytesTransfered
Number of bytes that were transferred during operation.

Description

This function is callback that is used with UsbTransferAsync function. It is called when the asynchronous input and output (I/O) operation is completed or canceled.

Note

In Visual Basic use AddressOf operator to handle callback to UsbTransferAsync.

5.4.3.6.11 UsbCancelTransfer

Language	Description
C/C++	`BOOL UsbCancelTransfer(IN USB_PIPE_HANDLE hPipe);`
Delphi	`function UsbCancelTransfer(hPipe : USB_PIPE_HANDLE) : BOOL; stdcall;`
VB	`Function UsbCancelTransfer (ByVal hPipe As Long) As Boolean`

Parameters

hPipe
Pipe handle. This handle can be obtained with UsbPipeOpen.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Function cancels all pending transfer operations that were issued by UsbTransferAsync for the

specified pipe handle.

5.4.3.6.12 UsbPipeReset

Language	Description
C/C++	`BOOL UsbPipeReset(IN USB_PIPE_HANDLE hPipe);`
Delphi	`function UsbPipeReset(hPipe : USB_PIPE_HANDLE) : BOOL; stdcall;`
VB	`Function UsbPipeReset (ByVal hPipe As Long) As Boolean`

Parameters

hPipe

Pipe handle. This handle can be obtained with UsbPipeOpen.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Function clears both the halt condition on the host side of the pipe and the stall condition on the endpoint.

5.4.3.6.13 UsbPipeAbort

Language	Description
C/C++	`BOOL UsbPipeAbort(IN USB_PIPE_HANDLE hPipe);`
Delphi	`function UsbPipeAbort(hPipe : USB_PIPE_HANDLE) : BOOL; stdcall;`
VB	`Function UsbPipeAbort (ByVal hPipe As Long) As Boolean`

Parameters

hPipe

Specifies pipe handle. This handle can be obtained with UsbPipeOpen.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Call this function to indicate that all outstanding requests for a pipe should be canceled.

5.4.3.7 Other

UsbGetPortStatus
UsbGetCurrentFrameNumber

5.4.3.7.1 UsbGetPortStatus

Language	Description
C/C++	`BOOL UsbGetPortStatus(IN  USB_DEVICE_HANDLE hUsb,` `OUT PULONG PortStatus);`
Delphi	`function UsbGetPortStatus(hUsb : USB_DEVICE_HANDLE;` `var PortStatus : ULONG) : BOOL stdcall;`
VB	`Function UsbGetPortStatus (ByVal hUsb As Long, _` `ByRef PortStatus As Long) As Boolean`

Parameters

hUsb

Specifies device handle. This handle can be obtained with OpenRapidUsb.

PortStatus

Output parameter. Return Value contains device state flags. The flags can be one or both of USBD_PORT_ENABLED, USBD_PORT_CONNECTED.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

5.4.3.7.2 UsbGetCurrentFrameNumber

Language	Description
C/C++	`BOOL UsbGetCurrentFrameNumber(IN  USB_DEVICE_HANDLE hUsb,` `OUT PULONG FrameNumber );`

Delphi	`function UsbGetCurrentFrameNumber(hUsb : USB_DEVICE_HANDLE;`
	`var CurrentFrameNumber : ULONG)`
	`: BOOL; stdcall;`

VB	`Function UsbGetCurrentFrameNumber (ByVal hUsb As Long, _`
	`ByRef CurrentFrameNumber As`
	`Long) As Boolean`

Parameters

hUsb

Specifies device handle. This handle can be obtained with OpenRapidUsb.

FrameNumber

Output parameter. Is equal to current frame number.

Return Value

If function fails it return FALSE. To detect error call **GetLastError** function.

Description

Use this function to retrieve current frame number.

5.4.3.8 USB Structures and Types

5.4.3.8.1 RDUSB_DEVICE_INFORMATION

```
typedef struct _RDUSB_DEVICE_INFORMATION {
    USHORT Length;
    UCHAR InterfaceNumber;
    UCHAR AlternateSetting;
    UCHAR Class;
    UCHAR SubClass;
    UCHAR Protocol;
    UCHAR Reserved;
    PVOID Reserved1;
    ULONG NumberOfPipes;
    RDUSB_PIPE_INFORMATION Pipes[1];
} RDUSB_DEVICE_INFORMATION, *PRDUSB_DEVICE_INFORMATION;
```

Members

Length
Length of this structure, including all pipe information structures that follow.

InterfaceNumber
Currently selected interface number.

AlternateSetting
Currently selected alternate interface number.

Class
Contains a USB-assigned identifier that specifies a USB-defined class that this interface conforms to.

SubClass
Contains a USB-assigned identifier that specifies a USB-defined subclass that this interface conforms to. This code is specific to the code in Class.

Protocol
Contains a USB-assigned identifier that specifies a USB-defined protocol that this interface conforms to. This code is specific to the codes in Class and SubClass.

NumberOfPipes
Specifies the number of pipes (endpoints) in this interface.

Pipes[1]
Specifies a variable length array of RDUSB_PIPE_INFORMATION structures to describe each pipe in the interface.

5.4.3.8.2 RDUSB_PIPE_INFORMATION

```
typedef struct _RDUSB_PIPE_INFORMATION {
    USHORT MaximumPacketSize;
    UCHAR EndpointAddress;
    UCHAR Interval;
    RDUSB_PIPE_TYPE PipeType;
    PVOID Reserved;
    ULONG MaximumTransferSize;
    ULONG PipeFlags;
} RDUSB_PIPE_INFORMATION, *PRDUSB_PIPE_INFORMATION;
```

Members

MaximumPacketSize
Specifies the maximum packet size, in bytes, that this pipe handles.

EndpointAddress
Specifies the bus address for this pipe.

Interval
Specifies a polling period for this pipe in milliseconds. This value is only valid if **PipeType** is set to UsbPipeTypeInterrupt.

PipeType
Identifies type of transfer valid for this pipe. See RDUSB_PIPE_TYPE.

MaximumTransferSize
Maximum size for a single request in bytes.

5.4.3.8.3 RDUSB_PIPE_TYPE

```
typedef enum _RDUSB_PIPE_TYPE {
    UsbPipeTypeControl,
    UsbPipeTypeIsochronous,
    UsbPipeTypeBulk,
    UsbPipeTypeInterrupt
} RDUSB_PIPE_TYPE;
```

UsbPipeTypeControl
Specifies that this pipe is a control pipe.
UsbPipeTypeIsochronous
Specifies that this pipe uses isochronous transfers.
UsbPipeTypeBulk
Specifies that this pipe uses bulk transfers.
UsbPipeTypeInterrupt
Specifies that this pipe uses interrupt transfers. A value will be set in Interval member of RDUSB_PIPE_INFORMATION to indicate how often this pipe is polled for new data.

5.4.3.8.4 RDUSB_BANDWIDTH_INFORMATION

```
typedef struct _RDUSB_BANDWIDTH_INFORMATION {
    ULONG TotalBandwidth;
    ULONG ConsumedBandwidth;
} RDUSB_BANDWIDTH_INFORMATION, *PRDUSB_BANDWIDTH_INFORMATION;
```

Members

TotalBandwidth
Specifies the total bandwidth, in bits per second, available on the bus.

ConsumedBandwidth
Specifies the mean bandwidth that is already in use, in bits per second.

Description
This structure is used in **UsbGetBandwidthInfo** to retrieve bandwidth information from USB controller.

5.5 RapidDriver Structures

5.5.1 HW_DEV_CONFIG

```
typedef struct _HW_DEV_CONFIG {
   ULONG HwType;        // TYPE_HARDWARE_ISA, .._ISAPNP, .._PCI
   ULONG IrqN;          // IRQ #
   ULONG DmaN;          // DMA Channel
   ULONG nBars;         // Number of ranges
   ULONG BarA[6];       // Base Physical Addresses of Ranges
   ULONG BarL[6];       // Length of Ranges in Bytes
   ULONG BarT[6];       // Type Of  Ranges (CmResourceTypePort,
   CmResourceTypeMemory)
   ULONG BarM[6];       // Mapped Ranges (to user space)
   CHAR  HardwareId[256];
} HW_DEV_CONFIG, * PHW_DEV_CONFIG;
```

5.5.2 IRQ_CLEAR_REC

```
typedef struct _IRQ_CLEAR_REC
{
  UCHAR ClearIrq;       // 1 - Irq must be cleared, 0 - not
  UCHAR TypeOfRegister; // 0 - memory-mapped register, 1 - port
  UCHAR WideOfRegister; // wide of register: 1-Byte,2-Word,4-Double Word
  UCHAR ReadOrWrite;    // 0 - read register to clear Irq, 1 - write
  ULONG RegBaseAddress; // register base address
  ULONG RegOffset;      // register offset (in bytes)
  ULONG ValueToWrite;   // value to write (if ReadOrWrite=1)

} IRQ_CLEAR_REC, * PIRQ_CLEAR_REC;
```

5.5.3 IRQ_SHARE_REC

```
typedef struct _IRQ_SHARE_REC {

  UCHAR ShareIrq;       // 1 - Irq must be shared, 0 - not
  UCHAR TypeOfRegister; // 0 - memory-mapped register, 1 - port
  UCHAR WideOfRegister; // wide of register: 1-Byte,2-Word,4-Double Word
  UCHAR Reserved;       // reserved
  ULONG RegBaseAddress; // register base address
  ULONG RegOffset;      // register offset
  ULONG MaskValue;      // bitmap mask to make AND operation
  ULONG ResultMask;     // what should be in result if it is our
interruption

} IRQ_SHARE_REC, * PIRQ_SHARE_REC;
```

5.5.4 PCI_LOCATION

```
typedef struct _PCI_LOCATION {
   ULONG IsaBus;
   ULONG IsaDevice;
   ULONG IsaFunction;
}PCI_LOCATION, *PPCI_LOCATION;
```

5.5.5 PCI_COMMON_CONFIG

This structure describes the PCI configuration space for PCI devices. All PCI devices have a common set of registers that include VendorID, DeviceID, and so on. This structure differs for PCI devices of header type 0 for devices, of header type 1 for PCI-PCI bridges, and of header type 2 for PCI-CardBus bridges. For more information, see the PCI Local Bus Specification, revision 2.1 or 2.2.

```
typedef struct _PCI_COMMON_CONFIG {
    USHORT  VendorID;
    USHORT  DeviceID;
    USHORT  Command;
    USHORT  Status;
    UCHAR   RevisionID;
    UCHAR   ProgIf;
    UCHAR   SubClass;
    UCHAR   BaseClass;
    UCHAR   CacheLineSize;
    UCHAR   LatencyTimer;
    UCHAR   HeaderType;
    UCHAR   BIST;

    union {
        struct _PCI_HEADER_TYPE_0 {
            ULONG   BaseAddresses[PCI_TYPE0_ADDRESSES];
            ULONG   CIS;
            USHORT  SubVendorID;
            USHORT  SubSystemID;
            ULONG   ROMBaseAddress;
            ULONG   Reserved2[2];

            UCHAR   InterruptLine;
            UCHAR   InterruptPin;
            UCHAR   MinimumGrant;
            UCHAR   MaximumLatency;
        } type0;

        struct _PCI_HEADER_TYPE_1 {
            ULONG   BaseAddresses[PCI_TYPE1_ADDRESSES];
            UCHAR   PrimaryBusNumber;
            UCHAR   SecondaryBusNumber;
            UCHAR   SubordinateBusNumber;
            UCHAR   SecondaryLatencyTimer;
            UCHAR   IOBase;
            UCHAR   IOLimit;
            USHORT  SecondaryStatus;
            USHORT  MemoryBase;
            USHORT  MemoryLimit;
            USHORT  PrefetchableMemoryBase;
            USHORT  PrefetchableMemoryLimit;
            ULONG   PrefetchableMemoryBaseUpper32;
            ULONG   PrefetchableMemoryLimitUpper32;
            USHORT  IOBaseUpper;
            USHORT  IOLimitUpper;
            ULONG   Reserved2;
            ULONG   ExpansionROMBase;
            UCHAR   InterruptLine;
            UCHAR   InterruptPin;
            USHORT  BridgeControl;
```

```
        } type1;

        struct _PCI_HEADER_TYPE_2 {
            ULONG   BaseAddress;
            UCHAR   CapabilitiesPtr;
            UCHAR   Reserved2;
            USHORT  SecondaryStatus;
            UCHAR   PrimaryBusNumber;
            UCHAR   CardbusBusNumber;
            UCHAR   SubordinateBusNumber;
            UCHAR   CardbusLatencyTimer;
            ULONG   MemoryBase0;
            ULONG   MemoryLimit0;
            ULONG   MemoryBase1;
            ULONG   MemoryLimit1;
            USHORT  IOBase0_LO;
            USHORT  IOBase0_HI;
            USHORT  IOLimit0_LO;
            USHORT  IOLimit0_HI;
            USHORT  IOBase1_LO;
            USHORT  IOBase1_HI;
            USHORT  IOLimit1_LO;
            USHORT  IOLimit1_HI;
            UCHAR   InterruptLine;
            UCHAR   InterruptPin;
            USHORT  BridgeControl;
            USHORT  SubVendorID;
            USHORT  SubSystemID;
            ULONG   LegacyBaseAddress;
            UCHAR   Reserved3[56];
            ULONG   SystemControl;
            UCHAR   MultiMediaControl;
            UCHAR   GeneralStatus;
            UCHAR   Reserved4[2];
            UCHAR   GPIO0Control;
            UCHAR   GPIO1Control;
            UCHAR   GPIO2Control;
            UCHAR   GPIO3Control;
            ULONG   IRQMuxRouting;
            UCHAR   RetryStatus;
            UCHAR   CardControl;
            UCHAR   DeviceControl;
            UCHAR   Diagnostic;
        } type2;

    } u;

    UCHAR   DeviceSpecific[108];

} PCI_COMMON_CONFIG , *PPCI_COMMON_CONFIG;
```

Members

VendorID
PCI vendor identifier register.

DeviceID
PCI device identifier register.

Command
PCI command register.
Status
PCI status register.
RevisionID
PCI revision identifier register.
ProgIf
PCI programming interface register.
SubClass
PCI device sub-class register.
BaseClass
PCI device base-class register.
CacheLineSize
PCI cache line size register.
LatencyTimer
PCI latency timer register.
HeaderType
PCI header type register. 0 indicates a normal PCI device, 1 indicates a PCI-PCI bridge, and 2 indicates a PCI-CardBus bridge.
BIST
PCI built-in self test register.
BaseAddresses
Array of PCI base address registers. Normal PCI devices have 6 base address registers, while PCI-PCI bridges have 2 and PCI-CardBus bridges have 1.
CIS
CardBus CIS pointer register.
SubVendorID
PCI subsystem vendor identifier register.
SubSystemID
PCI subsystem identifier register.
ROMBaseAddress
PCI ROM base address register.
Reserved2, **Reserved3**, **Reserved4**
Reserved.
InterruptLine
PCI interrupt line register.
InterruptPin
PCI interrupt pin register.
MinimumGrant
PCI minimum grant register.
MaximumLatency
PCI maximum latency register.
BaseAddress
PCI base address register.
PrimaryBusNumber
PCI primary bus number register.
SecondaryBusNumber
PCI secondary bus number register.
SubordinateBusNumber
PCI subordinate bus number.
SecondaryLatencyTimer
PCI secondary latency timer.
IOBase
Lower 8 bits of PCI I/O base address register.
IOLimit
Lower 8 bits of PCI I/O limit address register.

SecondaryStatus
PCI secondary status register.
MemoryBase
PCI memory base address register.
MemoryLimit
PCI memory limit address register
PrefetchableMemoryBase
Lower 16 bits of PCI prefetchable memory base address register.
PrefetchableMemoryLimit
Lower 16 bits of PCI prefetchable memory limit address register.
PrefetchableMemoryBaseUpper32
Upper 32 bits of PCI prefetchable memory base address register.
PrefetchableMemoryLimitUpper32
Upper 32 bits of PCI prefetchable memory limit address register.
IOBaseUpper
Upper 16 bits of PCI I/O base address register.
IOLimitUpper
Upper 16 bits of PCI I/O limit address register.
ExpansionROMBase
PCI expansion ROM base address register.
BridgeControl
PCI bridge control register.
CapabilitiesPtr
PCI capabilities pointer register.
SecondaryStatus
PCI secondary status register.
CardbusBusNumber
CardBus bus number register.
CardbusLatencyTimer
CardBus latency timer register.
MemoryBase0
CardBus memory base address register 0.
MemoryLimit0
CardBus memory limit address register 0.
MemoryBase1
CardBus memory base address register 1.
MemoryLimit1
CardBus memory limit address register 1.
IOBase0_LO
Lower 16 bits of CardBus I/O base address register 0.
IOBase0_HI
Upper 16 bits of CardBus I/O base address register 0.
IOLimit0_LO
Lower 16 bits of CardBus I/O limit address register 0.
IOLimit0_HI
Upper 16 bits of CardBus I/O limit address register 0.
IOBase1_LO
Lower 16 bits of CardBus I/O base address register 1.
IOBase1_HI
Upper 16 bits of CardBus I/O base address register 1.
IOLimit1_LO
Lower 16 bits of CardBus I/O limit address register 1.
IOLimit1_HI
Upper 16 bits of CardBus I/O limit address register 1.
LegacyBaseAddress
CardBus legacy base address register.

SystemControl
CardBus system control register.
MultiMediaControl
CardBus multimedia control register.
GeneralStatus
CardBus general status register.
GPIO0Control, **GPIO1Control**, **GPIO2Control**, **GPIO3Control**
CardBus GPIO control registers
IRQMuxRouting
CardBus IRQ multiplexer routing register.
RetryStatus
CardBus retry status register.
CardControl
CardBus card control register.
DeviceControl
CardBus device control register.
Diagnostic
CardBus diagnostic register.
DeviceSpecific
Device specific registers in the PCI configuration space, varies by device.

5.6 Deployment

There are three steps in the installation procedure:

- creating the deployment package
- installation of the driver on the target machine
- installation of your own RapidDriver-based application

5.6.1 Distribution Package

Your distribution package should include the following files in the same directory:

- RapidXXX.sys
- RapidXXX.dll
- RapidInstaller32.dll
- <project name>.inf file from RapidDriver\Kernel directory (or RapidLPT.inf file for a parallel port device)
- RdInstall.exe
- your RapidDriver-based application

where XXX is one of following: USB, ISA, PCI, or LPT.

5.6.2 Installation

Note: you should close all RapidDriver-based applications before you install. In other case you may be asked to reboot after the installation is complete.

There are three steps should be performed in your installation procedure:

1. Copy your own application to the installation directory, for example c:\Program Files\MyProgram
2. Copy RapidXXX.dll to any accessible directory, say <windows>\system32
3. Install the driver with the help of RdInstall utility as follows:

   RdInstall.exe <full path to the inf-file>

   Note: Full path required!

You may be asked to reboot in case of ISA device.

Advise: Run Device Manager and search for your device to make sure that the driver installation was successful.

6 RapidDriver Source Builder

RapidDriver Source Builder

Ver. 2.0

http://www.RapidDriver.com

tools@entechtaiwan.com

6.1 Introduction

Caution:

We have decided to discontinue the Source Builder Edition of RapidDriver in favor of the option to purchase the drivers and DLLs source code. The source code can be directly edited and compiled with MS Visual C/C++ or with the DDK "build" utility.

Visit our web site for details.

7 RapidDriver Frequently Asked Questions (F.A.Q.)

RapidDriver

Frequently Asked Questions (F.A.Q.)

http://www.RapidDriver.com

tools@entechtaiwan.com

7.1 What is the difference between RapidDriver editions?

Shortly:

- RapidDriver Explorer is designed for use ***by hardware developers*** who wants to discover and debug their USB, PCI, parallel port, or ISA/PC-104 device under Windows operating system.
- RapidDriver Developer includes all the features of the Explorer edition, and additionally allows you to create and distribute your own applications with integrated ISA/PCI/USB/LPT drivers to control your hardware. ***No Microsoft DDK knowledge is required***.
- The Source Builder Edition of RapidDriver includes all the features of both the Explorer and the Developer Editions, and adds the ***ability to generate source code to a fully-featured device driver*** that conforms to the Windows Driver Model (WDM).

	Explorer	**Developer**	**Source Builder**	**Demo** (*)
ISA, PCI, USB, and LPT support through RapidDriver.exe	**+**	**+**	+	+
Built-in Hardware Debugger	**+**	**+**	+	+
Software development based on re-distributable bus-specific drivers and DLL	**-**	**+**	+	+ (**)
Custom drivers source code generation	**-**	-	+	-

(*) 30-day evaluation period

(**) Based on "universal" non-redistributable RapidDrv.sys driver. RapidDriver.exe must be running in background.

7.2 What is the difference between the various licenses?

Personal License:

Once registered, you are granted a non-exclusive license to use RapidDriver on one computer at a time, for any legal purpose. For RapidDriver Developer and Source Builder editions you are granted the license to distribute ISA, PCI, USB, and LPT drivers and DLL's as part of your software without having to pay royalties.

Group License:

For teams of developers. This license grants use of RapidDriver by up to 5 persons.

Visit our web site for the pricing and ordering information.

7.3 Can not install RapidDriver package!

There are two possible reasons for this:

1) The installation package is damaged, try re-download and re-install.

2) You must have administrative privileges to install and uninstall RapidDriver in Windows 2000, Windows XP, and Windows 2003 Server.

7.4 Some RapidDriver functionality does not work!

Yes, some controls on the dialogs can be disabled or inoperative. It is possible in two cases as follows:

- This functionality is only available in the higher editions of RapidDriver.
- This functionality is not yet realized, but we want to highlight our next steps.

7.5 I can not open the driver in my application...

The correct answer might depend upon several things according to the RapidDriver Edition.

RapidDriver Demo Version

- The evaluation period for this installation of RapidDriver has expired.
- You must keep RapidDriver application running in background to evaluate the RapidDriver Developer functionality.
- The driver for your device was not installed. Make sure that your device appears under the RapidDriver Devices class (*) branch of Device Manager and has no problems.
- Possible collision with the device project in RapidDriver - select "Entire PC: Direct Hardware Access" then try to run an example application again.

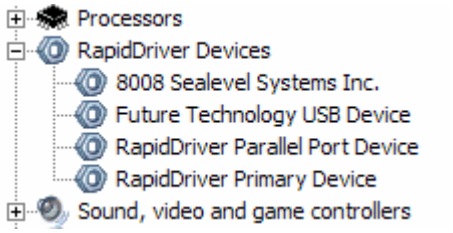
RapidDriver Explorer

You can not create any application with RapidDriver Explorer. You should upgrade your RapidDriver license to the either of Developer or Source Builder editions.

RapidDriver Developer or RapidDriver Source Builder

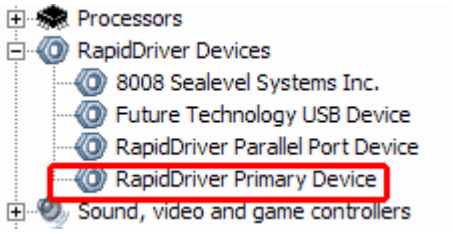
- You should install the "bus-specific" driver (RapidIsa.sys, RapidPci.sys, RapidUsb.sys, or RapidLpt.sys) instead of "universal" RapidDriver.sys.
- Make sure that appropriate DLL exist in <Windows>\System directory and appropriate driver exist in <Windows>\System32\Driver directory.
- The driver was not installed. Make sure that your device appears under the RapidDriver Devices (*) class branch of Device Manager and has no problems.
- Wrong "instance number" in OpenRapidXXX() function.

(*) RapidDriver Devices are marked by the RapidDriver icon in Device Manager, see picture:



7.6 Error message 'RapidDrv.sys driver cannot be opened.'

This message can appear in any edition of RapidDriver. It means that device named "RapidDriver Primary Device" (see picture below) is no more available.



This device is created during the RapidDriver installation and is used for internal purposes. RapidDriver cannot start if this device has been deleted from the system.

You can fix this problem in two ways:

1. Start Command Prompt (Start > Run > cmd), then type following on the command line:

```
cd <RapidDriver Path>
RapidDriver.exe -i
```

2.Reinstall RapidDriver.

7.7 How do I run RapidDriver under Windows 95 or Windows NT 4.0?

No way. RapidDriver can not do work in (and for) Windows 95 or Windows NT 4.0 operating systems.

Supported operating systems are Windows 98, Windows ME, Windows 2000, Windows XP, and Windows 20003 Server.

7.8 What programming languages does RapidDriver support?

We have tested RapidDriver Developer with the following programming systems:

- Microsoft Visual C++ 6.0, 7.0
- Microsoft Visual Basic 6.0
- Borland Delphi versions 2 to 7

- Borland C++ Builder versions 1, 3..6
- Microsoft Visual C#
- Microsoft Visual Basic .Net

In general, you can use RapidDriver with any programming language if it allows calls to DLL functions.

7.9 How do I create a custom class and icon to be placed in the Device Manager?

You are actually asking two questions in one:

Q1. How do I create a custom installation class?
A: Simply click the button labeled "Add New Setup Class.." in "Device Settings" dialog window then enter all reqired fields. That's all.

Q2. Ho do I create a custom class icon
A. The answer is much more complicated than the question. The simlest way is to see how this is done in the actual INF-file that was created by the RapidDriver (assuming you have RapidDriver Developer or higher edition). Go to the directory where RapidDriver holds all INF-files (say, <RapidDriver>\Kernel) then open any INF-file in your preferred text editor. This file contains two sections as follows:

```
; Windows 2000/XP/2003 style

[...]

[ClassInstall32AddReg]
HKR,,,,%RapidClassName%
HKR,,Installer32,,"RapidInstaller32.dll,RapidClassInstaller"
HKR,,Icon,,100

; Windows 98/ME style  - NOT SUPPORTTED!

[...]

[ClassInstallAddReg]
HKR,,,,%RapidClassName%
HKR,,EnumPropPages,,RdInst16.dll
HKR,,Icon,,102
```

As you can see, you must provide two DLLs: 32-bit one for Windows 2000/XP/2003, and 16-bit second for Windows 98/ME (***not supported anymore!***). These DLLs must contain an icon in it's resources (16x16 and 32x32). Use icon's resource ID in [AddReg] sections. You may write these DLLs yourself, or simly add your own icon with the help of tools like Borland Resource Workshop.

The source code of two DLLs mentioned above are available (by request) for registered users of RapidDriver Developer or Source Builder editions.
Remember that you must have MSVC 1.52 and Windows 98 DDK installed in case you want to create your own 16-bit DLL for Windows 98 (***not supported anymore!***).

7.10 "New Hardware Found" dialog appear every time I install the driver...

This dialog window appear when RapidDriver enumerates all PnP devices. Do nothing, simply close this dialog by clicking the "Cancel" button.

7.11 Where is the Source Builder Edition of RapidDriver?

We have decided to discontinue the Source Builder Edition of RapidDriver in favor of the option to purchase the ***drivers and DLL source code***. The source code can be directly edited and compiled with MS Visual C/C++ or with the DDK "build" utility. Prices are available here.

7.12 What about Windows 98?

Windows 98 and Windows ME are obsolete and no more supported in RapidDriver.

Index

- A -

- C -

- D -

- E -

- F -

- G -

- I -

- L -

- M -

- N -

- O -

- P -

- R -

- S -

- T -

- U -

- W -