KUKA Roboter GmbH

SOFTWARE

VxWin RT 3.5

VxWorks & MS Windows

Product Manual
Edition: 2007-12-26

VxWin RT Documentation

This manual contains both general and technical discussions of VxWin RT, a KUKA Roboter GmbH real-time software product. Although Windows XP and VxWorks are occasionally referenced, no explanations are offered as to how these operating systems function or how to program them. The best sources for that kind of information may be gotten from Microsoft Corporation, Wind River Systems, the Internet and the many excellent books and training courses on those products.

While some exposition in this manual will interest managers, purchasing agents and other decision-makers, the majority of the material presented is highly technical in nature, aimed at the engineers and programmers who are responsible for developing applications for the VxWin RT environment.

While great care was taken to ensure that this document completely describes the product’s features and facilities, it is possible that errors, omissions, or further development could affect the correspondence between the written descriptions and the product's behavior. If you discover such an error, please inform Customer Support. At the time of publication, however, it was believed that the information contained herein was both complete and accurate.

Also at the time of publication, external URLs, referenced in the text, led to valid web-sites. However, if a site does not belong to KUKA Roboter, no assurance can be given that such a link will continue to be valid or deliver the desired information. If you discover an invalid URL, Customer Support would appreciate hearing from you.

Concise Table of Contents

1 Notifications 15

2 Product Overview 16

3 Principles of Operation 24

4 System requirements 34

5 Development Environments 37

6 Using peripheral devices under VxWin 42

7 Interrupts and VxWin 55

8 PCI bus interrupts and interrupt-sharing 63

9 Real-Time Device Manager - Interrupt Configuration Tool 66

10 Real-time Clocks and Time/Calendar functions 66

11 VxWorks Floating-Point 74

12 Blue Screen Error Management 77

13 Multiprocessor/Multicore support 79

14 Forcing a context change — VxWorks to Windows 81

15 Limitations under VxWin 83

16 VxWorks System overload 85

17 Program Memory Layout 87

18 Inter-System Communication 93

19 System configuration file (vxwin.config) 131

20 Interrupt configuration file (interrupt.config) 137

21 Uploader Utility Program 139

22 Preparing for Installation 145

23 Installing VxWin 147

24 Starting VxWin 173

25 Stopping VxWin 180

26 Demonstration Programs 181

27 VxWin – Board Support Package 186

28 System Library Functions (sysLib.c, sysALib.s) 189

Extended Table of Contents

1 Notifications ... 15

2 Product Overview ... 16

2.1 Two operating systems / one computer ... 16

2.2 A world of applications ... 17

2.3 Windows XP / General Operating System ... 18

2.4 VxWorks / Real-time Operating System ... 19

2.5 Two systems in harmony ... 19

2.6 Product Highlights ... 20
- General ... 20
- Technical ... 21

2.7 New Features in version 3.5 ... 22

2.8 Customer Support ... 22
- Product Support ... 23
- Technical Consulting ... 23
- Product Training ... 23

3 Principles of Operation ... 24

3.1 Memory partitioning ... 24

3.2 Shared Mode Operation ... 26

3.3 Exclusive Mode Operation ... 28

3.4 Operating System Device Driver ... 29

3.5 Uploader Utility preview ... 29

3.6 Virtual Network ... 30

3.7 Board Support Package (BSP) ... 30

3.8 Programming Interfaces ... 31

3.8.1 VxWin Interface for Windows programs ... 31
- Uploader Library ... 31
- Virtual Network ... 32

3.8.2 Interface for VxWorks programs 32

3.9 The VxWin configuration file 32

3.10 Prototype Boot Line for VxWorks 33

3.11 A word about compatibility 34

4 System requirements 34

4.1 Minimum hardware requirements 34

4.2 Hardware Abstraction Layer (HAL) 35

4.3 Prerequisite Software 36

4.4 VxWin / Software components 36

5 Development Environments 37

5.1 Windows — Software development environment 37

5.2 VxWorks — Software development environment 37

5.2.1 Debugging -- Using Tornado/Workbench with VxWin 38

5.2.2 VxWorks Debugging Environment — Support for WindView / SystemViewer 39

5.3 Two development arrangements 39

5.3.1 The one-system method 40

5.3.2 The two-system method 41

6 Using peripheral devices under VxWin 42

6.1 Introduction 42

6.2 Sharing Devices 43

6.3 Sharing interrupts 43

6.4 Real-time Devices 44

6.5 Custom real-time devices and VxWorks 44

6.6 Windows .INF and .PNF files 44

6.7 Real-time .INF and .PNF files 45

6.8 Assigning standard Windows devices to VxWorks 45

6.8.1 Assigning an LPT port to VxWorks 46

6.8.2 Assigning a COM port to VxWorks 46

6.8.3 Assigning other standard devices to VxWorks 47

6.9 Assigning PCI Devices to VxWorks 49
6.9.1 Example: Assigning an NE2000 PCI card to VxWorks 50

7 Interrupts and VxWin 55

7.1 Exception-Handling with VxWorks 56

7.2 usrRoot Task Exceptions 57

7.3 Hardware Interrupt Assignment 57

7.4 Interrupt processing in VxWin 58
7.4.1 Interrupt Service Routine 59
7.4.2 More about servicing Interrupts 60

7.5 Interrupt Vectors 60

7.6 Connecting an ISR to an interrupt 62

7.7 Additional interrupt functions – VxWorks 62

8 PCI bus interrupts and interrupt-sharing 63

8.1 Introduction 63

8.2 Investigate the PCI hardware 63

8.3 The problem with shared interrupts 64

9 Real-Time Device Manager - Interrupt Configuration Tool 66

10 Real-time Clocks and Time/Calendar functions 66

10.1 Introduction 66

10.2 VxWin System Clock 67

10.3 VxWin Auxiliary Clock 68

10.4 ANSI Time Function 68

10.5 Time, date & time zone synchronization 68
10.5.1 Controlling time-synchronization 70

10.6 Time zone designations 71

10.7 Time synchronizing functions 72
RtosStartTimeSync 72
RtosStopTimeSync 73
RtosSetClockMaster 73

11 VxWorks Floating-Point 74

11.1 Using the Arithmetic Coprocessor 74

11.2 Floating-Point Exceptions 74

12 Blue Screen Error Management 77

13 Multiprocessor/Multicore support 79

13.1.1 Shared mode 79

13.1.2 Exclusive mode 80

14 Forcing a context change — VxWorks to Windows 81

15 Limitations under VxWin 83

16 VxWorks System overload 85

17 Program Memory Layout 87

17.1 VxWin memory layout 87

17.2 Shared Memory 91

17.3 Specifying the Extent of Shared Memory 92

18 Inter-System Communication 93

18.1 Windows XP / The RTOS Library (Uploader Library) 94

18.2 VxWorks caveats 95

RtosCommIsStarted 95

18.3 Using Shared Memory 95

18.3.1 Shared Memory under Windows 96

RtosShmAddrGet 97

RtosShmAddrGetAt 97

RtosShmTotalSizeGet 98

RtosInterlockedCompareExchange 99

18.3.2 Shared Memory under VxWorks 100

RtosShmAddrGet 100

RtosShmAddrGetAt 101

RtosShmTotalSizeGet 102

RtosInterlockedCompareExchange 102

18.4 Shared Events 103

18.4.1 Restrictions for Shared Events 104

18.4.2 Steps to using Shared Events 105

18.4.3 Shared Events under Windows XP 105

RtosCreateEvent 105
RtosOpenEvent 107
RtosCloseHandle 108
RtosSetEvent 108
RtosWaitForSingleObject 109
18.4.4 Shared Events under VxWorks 110
RtosCreateEvent 110
RtosOpenEvent 111
RtosCloseEvent 112
RtosSetEvent 113
RtosWaitForEvent 113

18.5 Virtual Network Communications 115
18.5.1 The Basics 115
18.5.2 Setting up the Windows side 117
18.5.3 Setting up the VxWorks side 123

18.6 Packet routing: VxWorks Uses Windows network adapter 125
18.6.1 IP forwarding 126
18.6.2 IP routing table 126

18.7 Accessing the PC's hard disk 127

19 System configuration file (vxwin.config) 131

20 Interrupt configuration file (interrupt.config) 137

21 Uploader Utility Program 139

21.1 Introduction to Uploader Utility 139

21.2 Uploader Utility – Command Line Options 139

21.3 Developing your own Uploader Utility 140

21.4 User-Interface to the UploadVxWin DLL 142

22 Preparing for Installation 145

23 Installing VxWin 147

23.1 Automatic installation of VxWin from CD-ROM 147

23.2 Manual product installation 154
23.2.1 Setting up registry entries 155
23.2.2 Setting up environment variable 155
23.2.3 Setting up device drivers 156
Installing the Operating System Device Driver 156
Installing the Real-time Virtual Network Driver 161

Setting MAC Addresses for the Virtual Network Driver 166
23.2.4 Installing and uninstalling the RTOS service 166
23.2.5 Copy the Uploader to the hard disk 167
23.2.6 Setting up Windows Firewall 167
Setting up Memory Reservation 167
Copy the VxWorks image to the hard disk 168
Create shortcuts on your desktop 169

24 Starting VxWin 173

24.1 VxWin tray-icon application (RtosControl.exe) 173

24.2 Loading and starting VxWin / VxWorks 173

24.3 Using Desktop Icons 175

24.4 Starting VxWin – Step by Step 176

24.5 Common start-up problems 178

24.6 Starting Debug Target Server 179

25 Stopping VxWin 180

26 Demonstration Programs 181

26.1 Shared Memory Demo 181

26.2 Shared Events Demo 183

26.3 Blue Screen Demo 185

27 VxWin – Board Support Package 186

27.1 Introduction 186

27.2 Important files in the VxWin BSP 186

27.3 VxWorks / Software Configuration 187

27.4 VxWorks / PC Serial Ports (COM Ports) 188

28 System Library Functions (sysLib.c, sysALib.s) 189
sysToMonitor 189
sysFPExceptHandler 189
sysTaskFPExceptionAdd 190
RtosIdle 191

28.1 Blue Screen Errors Module 192

RtosBsodSetMode 192
RtosBsodExit 193

1 Notifications

This manual contains both general and technical discussions of VxWin RT, a KUKA Roboter GmbH real-time software product. Although Windows XP and VxWorks are occasionally referenced, no explanations are offered as to how these operating systems function or how to program them. The best sources for that kind of information may be gotten from Microsoft Corporation, Wind River Systems, the Internet and the many excellent books and training materials on those products.

While some exposition in this manual will interest managers, purchasing agents and other decision-makers, the majority of the material presented is highly technical in nature, aimed at engineers and programmers, those responsible for developing applications for the VxWin RT environment.

While great care was taken to ensure that this document completely describes the product’s features and facilities, it is possible that errors, omissions, or further development could affect the correspondence between written descriptions and the product's behavior. If you discover an error or omission, please inform Customer Support. At the time of publication, however, it was believed that the information contained herein was both complete and accurate.

Also at the time of publication, external URLs, referenced in the text, led to valid web-sites. However, no assurance can be given that a link or a site not belonging to KUKA Roboter will continue to be valid or deliver the desired information. If you discover an invalid URL, Customer Support would appreciate hearing from you.

Tradenames and trademarks Throughout this document, reference is made to other companies, their products and their legally registered tradenames or trademarks. Among the protected names are: Microsoft®, Microsoft Corporation®, Windows®, MS Windows®, Win32®, Windows XP®, Windows 2000®, MSDN®, VxWorks®, Wind River®, Wind River Systems®, Tornado®, Workbench®, Adobe®, Acrobat Reader®, Adobe Reader®, IBM® and so on. KUKA Roboter GmbH freely acknowledges that all such trademarks and tradenames are the property of the organizations that registered them.

publication, the information contained in this document was carefully verified and found to be correct. Nonetheless, it cannot be ruled out that discrepancies may arise in the course of time. This document will be reviewed at reasonable intervals to assure that subsequent editions reflect the current product status.
Undocumented Features While a technical product such as VxWin RT may contain undocumented features, such features are not considered part of the product and their functionality is therefore not subject to any form of maintenance, support or guarantee.
VxWin User's Manual - Edition: 0529-600-006

2 Product Overview

2.1 Two operating systems / one computer

Scores of industrial applications require both the deterministic characteristics of a real-time operating system and the universal capabilities of a general operating system.
Because real-time applications, in contrast to most conventional programs, generally require only a modest amount of CPU processing time, it is desirable to integrate a real-time operating system and a conventional operating system on a single execution platform.
In such a dual-operating system environment, developers could then dedicate the real-time operating system to the time-critical functions of their end-product while simultaneously using the standard operating system to run conventional application programs as well as console functions.
KUKA Roboter GmbH has implemented just such a software system. VxWin RT makes possible the concurrent use of two dissimilar operating systems on a single (IBM-compatible) personal computer. In particular, it provides for the simultaneous operation of ...

- **Windows XP®**, a general operating system from Microsoft Corporation® (Redmond, Washington)

  -and-

- **VxWorks®**, a highly-acclaimed, real-time operating system from Wind River Systems, Inc. (Alameda, California)

VxWin RT provides users with the full functionality of each operating system and does it without resorting either to invasive techniques or specialized hardware, which strongly contributes to the robustness of the overall system. VxWorks deterministically executes the user's real-time code while Windows concurrently runs standard software applications and/or console programs. Because the real-time operating system, its tasks and interrupts, is assigned the highest priorities in the system, the deterministic characteristics of VxWorks remain uncompromised.
And while Windows® and its applications run at the same priority as the VxWorks® idle-task, it is seldom that a Windows user will notice any diminishment of Windows power, no matter how much real-time processing is performed on the other side of the system.
VxWin has more than a decade of field experience in challenging real-world applications.[1] And since being brought into the general marketplace, VxWin has been chosen again and again as the most desirable execution environment for a broad variety of applications.
KUKA Roboter believes that VxWin is one of the most flexible and powerful computing environments around. It is an intelligent system that offers even the most demanding users a extensive range of technical and economic advantages.

2.2 A world of applications

VxWin's all-around capabilities make it possible to run a wide variety of interesting and challenging real-time applications, especially those requiring simultaneous console and other conventional processing functionality. Whether your application draws, regulates, cuts, mixes, grinds, measures, moves, steers, chops, mills, switches, counts, sorts, Roboter, manipulates or monitors — you will find VxWin to be a most appropriate software execution environment, the right solution for your systems.
Some of the many possible applications for VxWin are:

- Computer-controlled machines, such as: lathes, drills, printing presses, custom-built machines...
- Robotics
- Real-time Data Management: switching, routing, monitoring...

[1] KUKA Roboter GmbH, Augsburg, Germany, one of the world's leading firms in robot technology, has long used VxWin to control their industrial robots.

- Monitoring and running: chemical processes, assembly lines, safety zones, and many other factory processes
- Building-management: lighting, power, climate, pumps, ventilators...
- Safety and security systems
- Process monitoring and control
- Medical monitoring, manipulation and instrumentation
- Testing: measurement, parts handling, analysis, and diagnostic systems
- Controlling remote systems, whether nearby or on the other side of the world
- Distributed industrial systems
- Materials handling activities, such as sorting, steering, lifting, shifting and actuating...
- Fluids monitoring and control
- Traffic monitoring and control
- Quality control and automated sorting
- General-purpose industrial panels
- Real-time simulation for training personnel in the operation of vehicles, airplanes, boats, industrial processes and other complex systems.
- Controlling complex office equipment, such as: high-speed printers, copiers, etc.
- And many other possibilities...

2.3 Windows XP / General Operating System

Microsoft Windows XP, a product of Microsoft Corporation, is without a doubt the best known and most widely used standard operating system and graphical user-interface in the world.
Among its many advantages are the numerous low-priced applications and utility programs that have been created for it, its universal familiarity, data-integrity, and flexible networking capabilities.
But, despite its intrinsic power, it has seldom been used for controlling industrial applications because its task-scheduler was not designed with a deterministic abilities in mind. A task-scheduler without guaranteed time-based responses is unsuitable for real-time applications.
While the Windows XP side of the VxWin system may be used without restriction for software development, it may also be used as an integral part of the end-product. One idea with obvious appeal is to use it as an intelligent, graphics-oriented operating console and computing base that runs in close cooperation with real-time operations on the other side of the system. Windows can also be

used to collect and process data histories, for example, acquire data for deferred transmission, and to dynamically exchange data with other networked elements or systems.

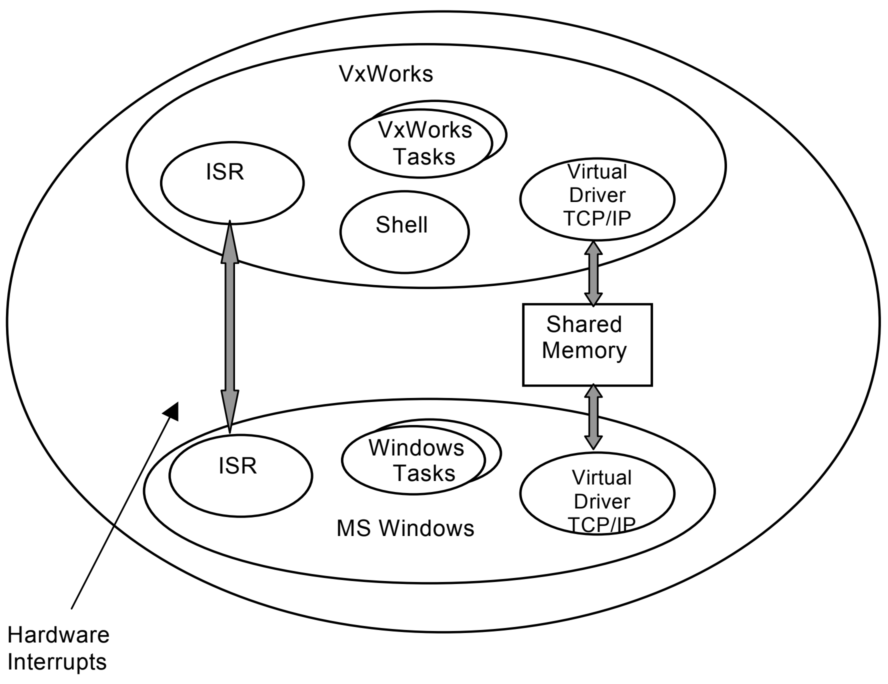
2.4 VxWorks / Real-time Operating System

VxWorks, a product of Wind River Systems Inc., is one of the world's most popular deterministic real-time operating systems for embedded applications. Robust and accurate, it is aimed at controlling embedded and other industrial applications. It has earned its reputation for prompt, predictable, repeatable operation.
For many years, VxWorks has proven its value in thousands of applications ranging from simple embedded systems, to complex industrial, scientific or medical systems, to robotic control. It has even been certified for use in airplanes, rockets, satellites and other space vehicles.

2.5 Two systems in harmony

VxWin unites VxWorks and Windows XP on a single IBM-compatible PC without requiring special versions of either operating system. This dual operating system environment permits the user to take full advantage of the best qualities of both systems.
Whether you are considering VxWin as a platform for educational, scientific or industrial purposes, you will soon discover the strength and flexibility inherent in this software system.
Tasks that run under VxWin always enjoy the highest priorities in the system. Windows, or any of its running applications, are only permitted to run after all real-time events have been completely processed. In this environment, Windows and its tasks run as if they were conceptually part of VxWorks' idle routine. Even though the two operating systems share a single execution platform, real-time system performance is never compromised. And since most real-time operations require little CPU-power, applications running on the Window’s side of the system do not decline in performance even as numerous asynchronous events are simultaneously serviced by the real-time system.
The following diagram illustrates graphically how two operating systems can coexist on the same computer:

PC with MS Windows



2.6 Product Highlights

General

- Since VxWin brings together two standard operating systems on a common execution platform, it relieves systems designers, engineers and programmers of a long learning-cycle. They can concentrate their development time on their own end-systems.
- While Microsoft development tools are used to develop software for the Windows side of the system, Wind Rivers' Tornado/Workbench development system are used to develop real-time software for VxWorks.
- Data may be exchanged with remote systems by means of standard network protocols. By taking advantage of designers' existing knowledge, this facility also contributes to abbreviated development schedules.
- VxWin has a mechanism to ensure that real-time operations will long continue to run even after a Windows XP (blue screen) exception occurs. Alternatively, upon being notified of such an exception, some systems might choose instead to suspend real-time operations in a safe and controlled fashion.

- Neither host- nor target-systems require additional hardware for running or testing the user's application software.
- Computer operations, such as memory or device accesses, undertaken within the Windows XP context are unaffected by VxWorks activities and vice versa.
- Each operating system runs entirely independently of the other, as if it were executing alone on the host computer.
- Despite a strict logical separation between the two operating system contexts, VxWin provides multiple means for inter-system communication (shared memory, events and Virtual Network), so that applications, regardless under which operating system they run, can exchange data and synchronize activities with applications on the other side.
- By taking advantage of Windows full-featured Graphical User Interface (with keyboard and pointer-device support), users can implement a powerful control console at minimal cost.
- Real-time applications are programmed using Wind Rivers' Tornado/Workbench development system in the Windows environment. In addition, a wide variety of additional tools, including third-party tools, are available to enhance software development.
- Because the system is based on widely-used technology, a tremendous base of third-party hardware and software, including device drivers, is readily available.
- Peripheral devices of all kinds may be used by the real-time or Windows systems, including Plug and Play devices. Under VxWin, most standard devices can be controlled by either operating system, using standard device drivers. It is intended that, except under extraordinary circumstances, users should not have to program their own device drivers.

Technical

- Since VxWin can be started with a Windows XP service program, it can be run without requiring the user to first log on to Windows.
- The operating systems do not share either system tables or registers that control virtual memory translations.
- Two real-time high-resolution timers make it convenient to synchronize programmed functions at varying clock rates.
- Since each operating system is physically unable to access the other’s execution context, all programs maintain complete data integrity.
- Three system mechanisms permit programs under one operating system to exchange data and coordinate activities with programs running in the other context. (Shared Memory, Shared Events, and a Virtual Network.)
- On multiprocessor/multicore systems VxWin may either operate in shared mode (Windows XP is running on all processor cores while VxWorks is running on one single processor core together with Windows XP) in or in exclusive mode (Windows XP is running on all but the last processor core while VxWorks is running on the last processor core).
- Shared mode: As long as one real-time task is active, the processor’s execution time is exclusively assigned to real-time processing. Only after all active real-time tasks have gone to

a wait-condition will Windows be given processor time. That said, user’s tasks can nonetheless force control to be returned to the Windows execution context via the RtosIdle function.

- Shared mode: Any real-time interrupt causes control to be immediately switched to the real-time system. Since real-time interrupt latencies are short, critical tasks are executed within a few microseconds after a real-time interrupt occurs.
- Exclusive mode: VxWorks at any time runs on its own processor core, it never returns back to Windows. Interrupt and task latencies are therefore much shorter compared with shared mode operation.

2.7 New Features in version 3.5

- Beginning with this release, you may run VxWorks exclusively on one dedicated processor core. This leads to significantly reduced interrupt and thread latencies. More details how to set up exclusive mode can be found in section 13.1.2.
- In previous versions of VxWin the interrupts which were used by VxWorks had to be configured in the interrupt.config file. In VxWin 3.5 interrupts by default are configured automatically.
  The file interrupt.config can still be used to manually force a specific interrupt configuration (e.g. setting PCI interrupt into edge triggered mode instead of level triggered mode). See section 20 for more information about how to configure interrupts.
- Interrupt conflict detection: In case an interrupt is already used by Windows a message box will be shown which signals such a situation. The function sysIntEnablePIC() will return ERROR in such a case.
- In case of a VxWorks reboot request a message box will inform the user of such a situation. Additional information about the cause of the reboot is stored in the VxWorks exception message area. This message will also be printed.

2.8 Customer Support

Because most customers have critical development schedules with little leeway for unplanned delays, KUKA Roboter has established a comprehensive spectrum of technical support services which can significantly contribute to an efficient use of KUKA Roboter products.
Through its support services, KUKA Roboter can accompany a customer throughout the development process – from before he receives the products (training), through installation (support service) and the various stages of development (support, consulting, and update services), right up to the successful conclusion of his project.

Product Support

A KUKA product support team assists customers by promptly answering their technical questions and helping them solve any problems that may arise during application development.

Technical Consulting

For customers who would like help in realizing their system goals, KUKA Roboter can provide highly-qualified personnel to consult with them on conceptual, practical or managerial levels.
On a case-by-case basis, KUKA Roboter may also be contracted to take on full responsibility for any or all phases of a customer's VxWin-based project development.

Product Training

KUKA Roboter offers a selection of training classes at its own facilities. Alternatively, it can be arranged for classes to be conducted at a customer-designated location.
Taking into consideration participants' knowledge and experience, the content and pace of each class can be adjusted to address the specific needs of each group.
The classes cover:

- Hands-on training
- Instruction on how to use the products to achieve specific objectives
- Tips and techniques to help avoid common mistakes and to accelerate the development process
- Discussions and examples of solutions common to embedded and real-time problems

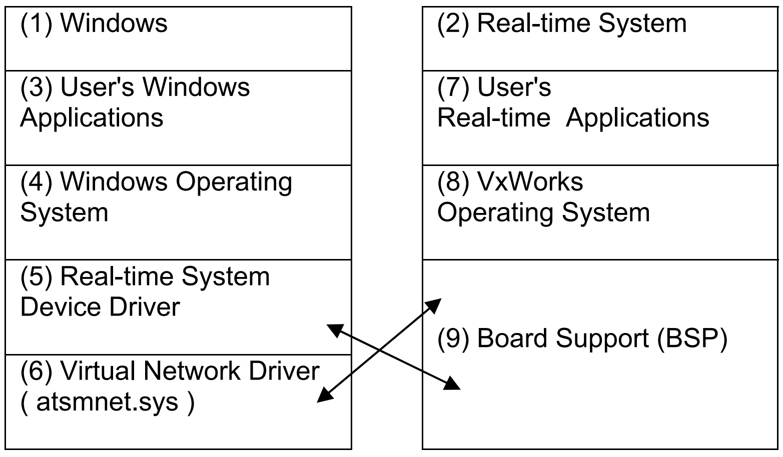
3 Principles of Operation

These Principles of Operation are designed to introduce the reader to the most important aspects of VxWin in the belief that the reader who first attains a general knowledge of VxWin will better be able to make smart choices of system parameters that affect the running of their systems.

3.1 Memory partitioning

A good way to start familiarizing yourself with the VxWin environment, is to have a look at how the system utilizes main memory.
The following diagram presents an elementary sketch of memory usage. Because this diagram is only intended to display functionality, no attempt was made to designate actual memory addresses. For a further discussion of the various memory areas, you may wish to consult Program Memory Layout.

Diagram - two operating systems / one PC


(1) Windows
(3) User's Windows Applications
(4) Windows Operating System
(5) Real-time System Device Driver
(6) Virtual Network Driver (atsmnet.sys)
(2) Real-time System
(7) User's Real-time Applications
(8) VxWorks Operating System
(9) Board Support (BSP)

- MS Windows XP resides mainly in lower RAM(1). Under VxWin, it is unaware that it physically shares memory with another operating system.
- Real-time system: VxWorks, resides entirely in upper RAM(2). Like Windows, it has no programmatic awareness that it shares memory with another operating system.
- The typical user will provide one or more application programs to run under Windows(3). These could be programs that realize console functions but could also be typical Windows applications such as MS Access or MS Word.

- A standard Microsoft Windows XP operating system runs here(4). VxWin provides several means for programs under Windows to communicate with programs under VxWorks, but more about that later.

- Real-time System Driver: (5)this is one of the most important parts of VxWin. Although called a "driver," it is not a driver in the typical sense of the word, for it does not control a computer peripheral device. Instead, it incorporates a number of functions that implement VxWin system- and user-services.

  It is a *driver* because it was written to conform to Windows driver interface specifications. For that reason, it must be installed in Windows' Device Manager, just like an ordinary driver. More information about this special driver may be found in Operating System Device Driver and Installing the Operating System Device Driver .

- Virtual Network Driver: This(6) also interfaces to the VxWin system as a *pseudo* or *virtual* network device. Since it was implemented in accord with NDIS (Network Driver Interface Specification), you can use it with sockets, just like you would any use any conventional network adapter.

  This driver Roboter, not an I/O card, but rather an I/O mechanism. Specific to the VxWin system, it is one of the means applications programs under one operating system have to communicate with applications running under the other operating system.

  The medium of exchange for this communication is not a cable. Instead, the two drivers communicate with each other by exchanging data through a segment of shared memory, reserved for that purpose. Refer to Program Memory Layout and Virtual Network.

  Like any true hardware driver, it too must be installed under the Windows Device Manager. See Installing the Virtual Network under Windows.

- This section of memory(7) is dedicated to the exclusive use of the real-time side of the system. It is here that the user's real-time tasks, i.e., applications will be loaded.

  While no program that runs in this area can directly access the Windows side of the system, such programs can communicate with the Windows side via VxWin functions provided for that purpose. Refer to Inter-System Communication elsewhere in this manual.

- VxWorks runs In this part of memory(8). It has no programmatic awareness of the Windows side of the system.

- The VxWin Board Support Package (BSP) runs in this part of memory(9). It is the means by which VxWorks and the entire real-time system are adapted to the execution environment. For the most part, this BSP consists of Wind River's standard Pentium PC BSP. Only a few functions have been added to it or modified to adapt to the VxWin environment. See VxWin – Board Support Package.

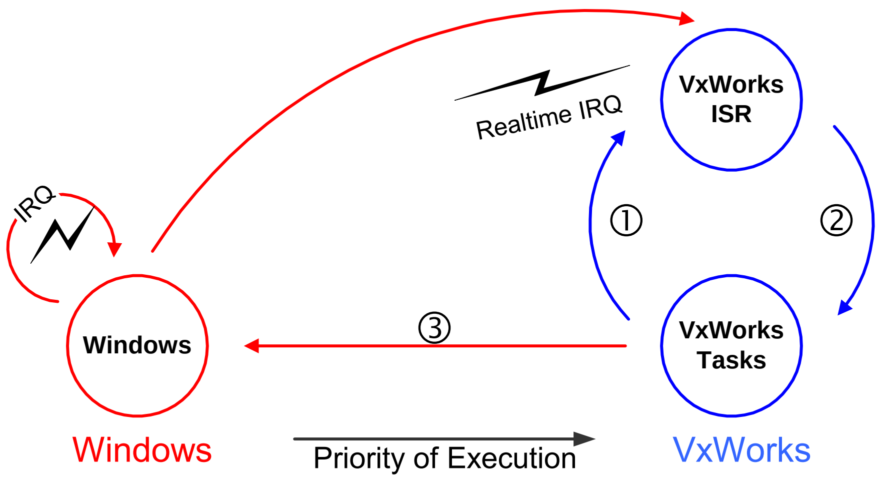
Note: Shared Memory, Shared Events, and the Virtual Network; the means by which programs running under one operating system communicate with programs (or tasks) in the other. You can read more about these mechanisms under Inter-System Communication.

3.2 Shared Mode Operation

When VxWin operates in shared mode (default setting) the CPU core is shared by Windows and VxWorks. Windows only runs after all VxWorks tasks are in a pending or delayed state and VxWorks thus enters its idle state.

Upon the occurrence of a hardware interrupt assigned to VxWorks, VxWin will halt execution of Windows, a Windows application or a lower-priority VxWorks task.
Control is then transferred to whichever VxWorks Interrupt Service Routine (ISR) is attached to the newly activated interrupt. The ISR then runs. If more work than is appropriate for an ISR must be done, the ISR will set a *signal* so that one or more *normal* (non-ISR) tasks can subsequently complete the processing.
When the ISR concludes, the system checks to see if any other VxWorks task is ready to run. Should that be the case, it activates the corresponding task. When all outstanding VxWorks tasks have run to conclusion and there is momentarily nothing more for VxWorks to do, VxWorks enters its idle-task. From the VxWorks idle-task, control reverts to Windows.
Since Windows and its applications are active only when all VxWorks tasks are inactive, one may conceptually think of Windows (and its applications) as a logical part of the VxWorks idle-task.

The following diagram illustrates the flow of control:



Operating states of VxWin in shared mode

① Exception-handling or a higher priority interrupt becomes outstanding.

② Interrupt Service Routine optionally starts a new task and then finishes.

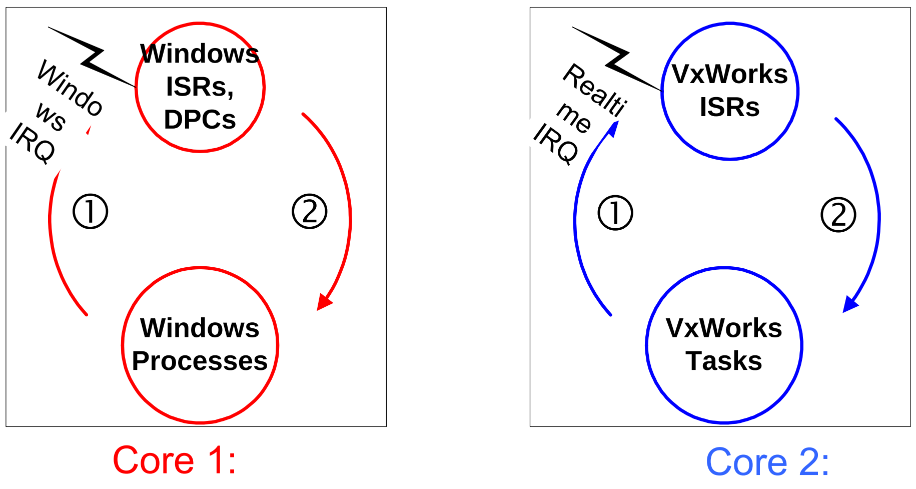
③ From the idle-state, VxWorks transfers control to Windows operating system.

Note: When running VxWin in shared mode on multiprocessor/multicore systems this state diagram is only applicable for one CPU core in the system (by default on the first core). All other CPU cores will run Windows only.

3.3 Exclusive Mode Operation

When VxWin operates in exclusive mode (only available on multiprocessor/multicore systems) the last CPU core in the system is dedicated to VxWorks. All remaining CPUs are used by Windows.

The following diagram, illustrates the flow of control on a dual core system:



<u>Operating states of VxWin in exclusive mode</u>

① Exception-handling or a higher priority interrupt becomes outstanding.

② Interrupt Service Routine optionally starts a new task and then finishes.

Note: When running VxWin in exclusive mode Windows will never be interrupted. Application and interrupt processing run concurrently and independently on both CPU cores. There is no need in VxWorks to enter the idle state.

3.4 Operating System Device Driver

The Operating System Device Driver is one of the most important elements of the VxWin system. It implements those data structures and system functions that make managing two operating systems on the same hardware platform possible.
Important: Although the Operating System Device Driver was programmed to conform to Microsoft's driver specifications, it is not a driver in the usual sense; it does not serve as the interface to a hardware peripheral device. It is a kernel-mode driver that can freely access all hardware features of the PC, including chip-set features. Attaining this level of privilege is important for implementing a system like VxWin that must engage in certain supra-operating system activities.
That it was formally implemented as a driver, implies the following points:

- For programs to access the Operating System Device Driver's functions, the driver must be entered into Windows Device Manager. This may either be done automatically, by running the VxWin set-up, or manually. More detailed information about this may be found elsewhere in this manual. See Installing the Operating System Device Driver.
- VxWin function calls that invoke some service in the Operating System Device Driver do so via the software trap mechanism.

The code that implements numerous VxWin services is located in the System Device Driver. The user accesses these services by way of pre-defined functions, described elsewhere in this manual.
To use these functions the user needs to include the following header in his code: *rtosdrvif.h*. The code that accesses these functions is located, correspondingly, in the Uploader library – *UploadRTOS.lib*.
In particular, some of the services performed by the Operating System Device Driver are:

- Manages interrupt-handling and context-switching for two operating systems
- Monitors and manages Bluescreen exception processing
- Performs system management functions for the Windows side of Shared Events
- Performs system management functions for the Windows side of Shared Memory Functions
- Manages system-timer activities
- Provides system-level services for the Uploader Utility program

3.5 Uploader Utility preview

The Uploader Utility program is a small application program that loads its own DLL in order to perform the following functions:

- Loads a user-specified or a default VxWorks image into RAM.

  Note: The Board Support Package, bound to the VxWorks image, is always loaded with the image.

- Starts or stops the real-time operating system.

Detailed information about this utility program may be found elsewhere in this manual. Refer to Uploader Utility Program.
Note: The actions for *freeze* and *unfreeze* functions are carried out in the Real-time OS System Device Driver because it has kernel-mode access to the entire operating environment.

3.6 Virtual Network

Earlier in the Principals of Operation, the Virtual Network Driver was mentioned as a part of VxWin that runs in the Windows execution context. Refer to the diagram in Memory partitioning.
This driver, conforming to the NDIS standard[2], has a counterpart on the VxWorks side of the system. Although the drivers generally exchange information by writing and reading data-packets to and from shared memory, because they adhere to the NDIS standard, their data packets may optionally be routed outside the PC onto a physical Ethernet network.
For more information about the Virtual Network, see Virtual Network Communications and Packet routing: VxWorks Uses Windows network adapter.

3.7 Board Support Package (BSP)

Real-time tasks use hardware resources and other system services only indirectly by means of standard operating system function calls. In other words, real-time applications must be insulated from the execution environment by a standard Application Program Interface (API).
Accordingly, the VxWin Board Support Package (BSP) provides this necessary layer of abstraction between the operating system and the hardware, while BSP libraries provide drivers for architecture-specific functions such as:
- Hardware initialization
- Interrupt generation and handling

[2] Network Driver Interface Specification.

- Clock and timer management
- Peripheral bus initialization and address mapping

The Wind River Systems document "BSP Developer's Guide" is a good source for learning more about BSP Libraries.

Note 1: The separation between the two operating systems is such that peripheral devices used by VxWorks may not be shared with Windows and vice versa.
Note 2: When you install Windows, standard device drivers for running PC's peripherals are loaded. These drivers can then be used only by Windows and its applications, not by VxWorks and its real-time tasks.
Note 3: To assign a peripheral device the real-time system, a real-time driver must be substituted for the Windows driver. How make such a substitution is described elsewhere in this manual; see Using peripheral devices under VxWin.
Since one of the design goals for VxWin was to take advantage of existing (standard) software modules as much as possible, KUKA Roboter chose Wind River's *pcPentium* BSP to serve as the basis for VxWin. That BSP has been only slightly modified for VxWin, which contributes significantly to the robustness and maintainability of the overall system.
The few modifications made to accommodate VxWin mainly concern context-switching, interrupt-handling and shared memory.
Because Wind River Systems provides BSPs to cover a variety of standard PC environments, it is unlikely that VxWin users will ever need to create their own BSP.
To utilize a custom device or another device not supported by Wind River, the user must provide a real-time driver for it. In any case, those who must develop a custom real-time driver or BSP should consult Wind River System's documentation for helpful information.

3.8 Programming Interfaces

3.8.1VxWin Interface for Windows programs

Uploader Library

The Uploader Library (*UploadRTOS.dll*) contains code for Windows applications to use Shared Events and Shared Memory functions. It also embodies code for starting and stopping the real-time system.

To access these functions, you need to include *rtosdrvif.h* in your program module and link to the library *UploadRTOS.lib*. This library contains program stubs that complete the function calls to executable dll-code.

Virtual Network

To Windows applications programs, the Virtual Network appears in the Device Manager as just another Microsoft network interface device. Thus, after you have installed and configured the Virtual Network, your own Windows programs can easily exchange data (via sockets and TCP/IP protocol) with the real-time system (VxWorks).
Your program will use the Virtual Network just as it would any conventional, hardware-based network interface, accessing it by means of conventional socket calls. See Virtual Network Communications elsewhere in this manual.

3.8.2Interface for VxWorks programs

On the VxWorks side of the system, all VxWin functionality has been implemented in the VxWin Board Support Package BSP.
Header information required to use VxWin functionality, including Shared Events and Shared Memory, are contained in the following header file: *RtosLib.h*. You should include this header file in any VxWorks task that makes use of these features.
Similarly, any VxWorks task that is to access the Virtual Network must include the following header file: *sockLib.h* . The function-prototypes in this header are the same ones that VxWorks programmers generally use to access a standard network.

—

Later in this manual, where the functions are described in programming-level detail, the corresponding library is indicated in the text.

3.9 The VxWin configuration file

The VxWin configuration file provides you with a means for dynamically characterizing your VxWin system. Each time VxWin is started, the system reads and interprets the configuration file. To familiarize yourself with the various possibilities, it is recommended that you review the parameters in this important file.

The VxWin configuration file contains, for example, information such as:

- Parameters for the VxWin Uploader Utility
- Global system parameters
- A prototype boot line for VxWorks

A detailed discussion of the configuration file may be found here: System configuration file (vxwin.config).

Note: In previous VxWin versions you had to populate an Interrupt Configuration file - ***interrupt.config*** – in which setup commands for each interrupt utilized under VxWorks had to be made. For the interrupt configuration file to be processed by the system, it must be referenced via an ***Include*** command in the system configuration file. See also Interrupt configuration file (interrupt.config). These configuration settings are now by default determined automatically.

3.10 Prototype Boot Line for VxWorks

When VxWorks is booted up in an embedded system, among the first things it does is read a line of text from a pre-defined location in RAM memory[3]. This so-called *boot line*[4] contains parameters that VxWorks uses to dynamically characterize itself during its boot-up phase. Boot-line parameters convey information such as: network addresses, boot device designation, password, and start-up script identifier.

When VxWin is started, it retrieves boot-line parameters from the prototype boot line in the VxWin configuration file. These parameters are then used to modify or replace corresponding VxWorks boot-line parameters. By this means, VxWin users can influence, upon each startup, how VxWorks boots, without having to generate a new image.

You can use a simple text editor to set or modify the prototype parameters in the System Configuration file. For details, see System configuration file (vxwin.config), in particular, Prototype Boot Line for VxWorks, Global Parameters in System configuration file (vxwin.config) and Boot line syntax.

[3] The boot line is stored in the variable ***sysBootLine*** as defined in the VxWorks ***sysLib.c*** library. The boot line is a simple ASCII text-string.

[4] "Boot line," a VxWorks term, is described in greater detail in Wind River Systems documentation.

3.11 A word about compatibility

IBM-compatible Personal Computers are developed and manufactured by diverse companies around the globe. Because peripheral controllers and other hardware elements differ to a certain degree, as do system software components and device drivers, it is impossible that every brand of PC behave precisely like every other one. Despite a surprisingly high degree compatibility, it is clear that some differences will always exist among them.
While it is desirable that all explanations, screenshots and procedures provided in this manual repeat exactly as given, because absolute compatibility of the host computers is not attainable, variant behavior has occasionally been observed during non-trivial procedures.
It may therefore become necessary, when assigning peripheral devices to the real-time operating system, for example, to try reasonable variations of some of the procedural steps.
lf while installing or setting up KUKA software products, your PC behaves in an unexpected fashion, please contact KUKA Roboter Customer Service for help.
Important: KUKA provides a utility program, the **Real-Time Device Manager (RTDM)**, that automatically analyzes your host system. It not only displays detailed device information about peripheral devices and interrupt routing, but also automates the process of assigning a device from the Windows context to the real-time system and vice versa. To learn more about the RTDM utility program, locate it and its supplementary documentation in ***...\VxWin\RTDM***.

4 System requirements

4.1 Minimum hardware requirements

- Pentium I / 200 MHz (min.) PC;
  IBM-compatible, Uniprocessor system
- 128 MB RAM (min.)
- 16 MB hard disk space (min.)

4.2 Hardware Abstraction Layer (HAL)

The Windows XP operating system consists of a hardware independent kernel and a library of system functions that are known as the Hardware Abstraction Layer (HAL). Each HAL provides the operating system with a generic software interface to various types of execution platforms.
When a Windows® XP® operating system is first installed, you must decide which HAL library is most compatible with your platform's hardware. Your decision will depend on some of the system properties that are discovered during automatic hardware recognition phase of the boot process. Within limits, some HAL libraries may be used interchangeably. Please refer to Microsoft’s documentation for more information.

Specific considerations for VxWin RT

Because the VxWin RT Operating System Device Driver (see Operating System Device Driver) runs in close association with the installed HAL and can only be used with specific versions of the HAL libraries, the choice of which HAL you use is very important.
As summarized in the following table, VxWin RT supports several types of HAL:

HAL	DLL	VxWin RT support
Standard PC	hal.dll	not supported
ACPI PC	halacpi.dll	not supported
MPS Uniprocessor PC	halapic.dll	not supported
MPS Multiprocessor PC	halmps.dll	not supported
ACPI Uniprocessor PC	halaacpi.dll	supported
ACPI Multiprocessor PC	halmacpi.dll	supported

The following table indicates HALs that are currently not supported:

HAL	DLL	VxWin RT support
Compaq PC	halps.dll	not supported
SGI Multiprocessor	halborg.dll	not supported

During the installation of Windows® XP, a HAL that VxWin does not currently support may automatically be installed. If this happens, you can try to switch to one of the supported HALs.
Practice has shown, however, that it sometimes is not possible to successfully swap HAL libraries in an already installed system. It is often better simply to re-install the operating system. If you decide to do this, you should keep in mind

that reinstalling the operating system jeopardizes the integrity of all data files on the hard disk. Therefore, before doing this, it would be advisable to backup your important files.
More information regarding how to exchange HAL libraries may be found on the Web. Microsoft's on-line Knowledge Base is a good resource for this kind of information.

4.3 Prerequisite Software

To manage the host computer's resources and applications you will need:

- **Windows XP with Service Pack 2**

  -or-

  Windows XP Embedded, Service Pack 2

  -and-

- **Device drivers for Windows XP**

Note: Regardless which operating system you choose, the Professional Edition is required.

To generate new VxWorks images, develop custom drivers or a real-time application for VxWorks, the following software is required:

- **Tornado 2.x**

  -or-

- **Workbench 2.x**

4.4 VxWin / Software components

The following software components are delivered on the VxWin RT release medium:

- VxWin **System Device Driver** – ***rtosdrv.sys*** – for Windows (See Operating System Device Driver)
- VxWin RT Board Support Packages (BSP) for Workbench 2.x and VxWorks 6.x (See VxWin – Board Support Package)
- The Virtual Network Driver (***atsmnet.sys***) for Windows XP (See Virtual Network Communications)

- ***vxwin.config***, the default system configuration file. An ASCII text file containing parameters that characterize the VxWin operating environment. You may have to adjust some of these parameters to tailor the system to suit your needs. (See System configuration file (vxwin.config))

- Uploader Utility program (***UploadRTOS***). A Windows application program for starting and stopping VxWin, and performing other services. (See Uploader Utility Program)

- Uploader Utility source-code. For users who wish to program a custom Uploader

- VxWin Service (***RtosService.exe***). When the PC boots up, the program that Roboter date and time synchronization between Windows XP and VxWorks will automatically be started by means of a Windows XP service (RtosService).

- Example programs. These demonstrate the use of Shared Memory, Shared Events and Blue-screen Error Management.

- Several .INF files. Used for assigning devices to the real-time system

- A Plug-and-Play Dummy Driver (***rtospnp.sys***). Required by the procedure to install a PCI peripheral for the real-time operating system. See The role of the real-time PnP device driver in Assigning PCI devices to VxWorks.

- For libraries and other important files see Files in the VxWin BSP.

5 Development Environments

5.1 Windows — Software development environment

If you are going to develop your own Windows application program or modify (i.e., customize) the VxWin Uploader Utility for your own purposes, **Microsoft Visual Studio 6, Service Pack 5 with C++ (Professional)** is required. Refer to Developing a custom Uploader Utility program elsewhere in this manual.

5.2 VxWorks — Software development environment

Tornado / Workbench, a Wind River Systems Inc.® product, is an Integrated Development Environment (IDE) that provides professional software tools for creating Board Support Packages (BSPs) and real-time applications for VxWorks. Tornado/Workbench, your means for generating new VxWorks images, includes a Project Manager, Source-Code Control System, Compiler and Loader (for building executable modules). It also provides real-time debugger tools for tracing, visualizing, controlling, measuring time and manipulating variables in real-time processes.

5.2.1Debugging -- Using Tornado/Workbench with VxWin

The following debugging methods may be used:

Task-Mode Debugging

Customarily, Tornado/Workbench is used to debug one application task at a time. For this kind of debugging, a debugger must be initially attached to a task. Thereafter, the user can display and modify variables, perform stepped execution, or let the task run to a breakpoint.
In this debugging-mode, when a breakpoint is reached, only the task being tested is stopped, the rest of the system continues to run.
For detailed discussions of how to use task-mode debugging, refer to the Tornado User's Guide (especially Chapter 10), or the Workbench User’s Guide (beginning at Chapter 18).
To debug Interrupt Service Routines (ISR's) or multiple threads, however, system-mode debugging must be used, as described in the next section.

System-Mode Debugging

System-mode debugging (external-mode debugging) is typically used to develop Interrupt Service Routines (ISR), because IRSs run outside of any real-time task context. It is also used whenever multiple threads are to be simultaneously debugged.
Under system-mode debugging, whenever a breakpoint is reached, the entire target system is stopped. For this reason, when using system-mode debugging, the Tornado/Workbench debugger must be hosted on an external system, not on the VxWin system.

For more detailed information about system-mode debugging, please refer to the Tornado User's Guide or the Workbench User's Guide.

—

The following example illustrates how to prepare a host and target system for external debugging. The example assumes that the target and host systems will be connected to each other over COM1 ports. In this case, COM2 on the target will be used by Windows (for a mouse).
Add the following lines to *config.h* and then generate a new VxWorks image:

```
#undef WDB_COMM_TYPE
#define WDB_COMM_TYPE WDB_COMM_SERIAL

#undef WDB_TTY_CHANNEL
#define WDB_TTY_CHANNEL 0

#undef CONSOLE_TTY
#define CONSOLE_TTY 2     /* target resident shell only
                                        via network */
```

—

The following example calls out the target server with the -B and -d options. A serial baud rate of 9600 (default) was assumed:

```
tgtsvr  VxWin -B wdbserial -d COM1 -V –c
            C:\Tornado\target\config\VxWin\vxWorks
```

5.2.2VxWorks Debugging Environment — Support for WindView / SystemViewer

IRQ0 to IRQ15 on the PC motherboard are made to correspond to INT0 to INT15 in WindView/SystemViewer. INT16 is used to implement software-interrupts for the Virtual Network.
Please refer to the *WindView User's Guide*, *WindView User's Reference*, *System Viewer User's Guide* as well as the *System Viewer's Reference Guide* for more information about how to configure support for this tool.

5.3 Two development arrangements

To develop software for VxWin, the user may choose either of two basic host/target configurations:

- **The one-system method**: Develop a VxWorks application under Windows XP and then run it under VxWin on the same computer...

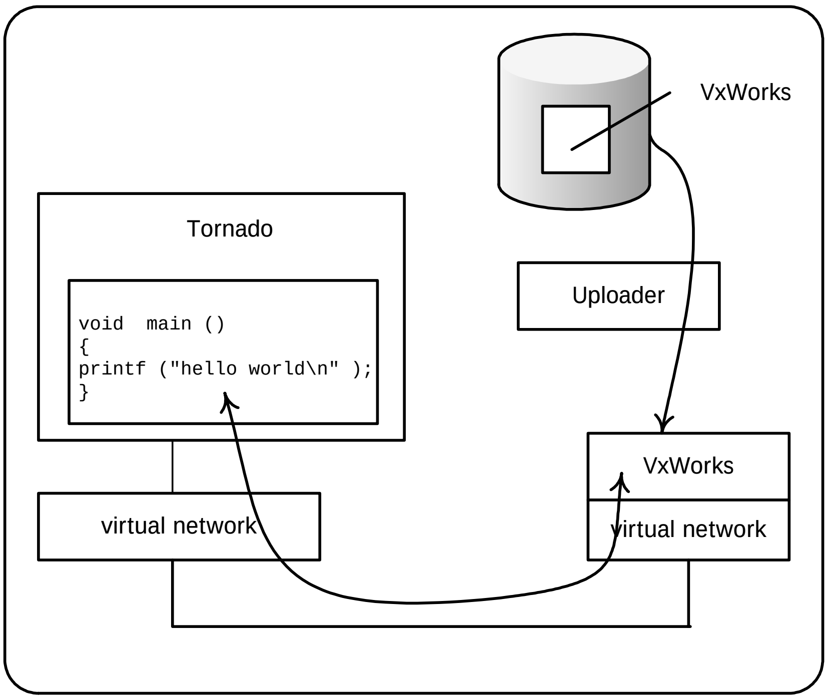
-or-

- **The two-system method**: Develop a VxWorks application on one computer and run it on another computer, on which VxWin is installed. If you choose the two-system method, you can use a single- or a double-network technique.

In the following discussion, reference is made to a "target server" and a "run-time" or "target" agent. These debugging elements are part of Tornado/Workbench. For details on how to configure and use them, please refer to Wind River's *Tornado User's Guide* or *Workbench User's Guide*.

5.3.1The one-system method

In a combined host/target system the connection between Windows and the target is made using the VxWin Virtual Network.



In this configuration, the development software runs under Windows XP and the run-time target's components run under VxWorks, which was installed on the same PC. The development process is as follows:

- Step 1: Build a VxWorks image.
- Step 2: Using the Uploader Utility, transfer the VxWorks image into the program memory.

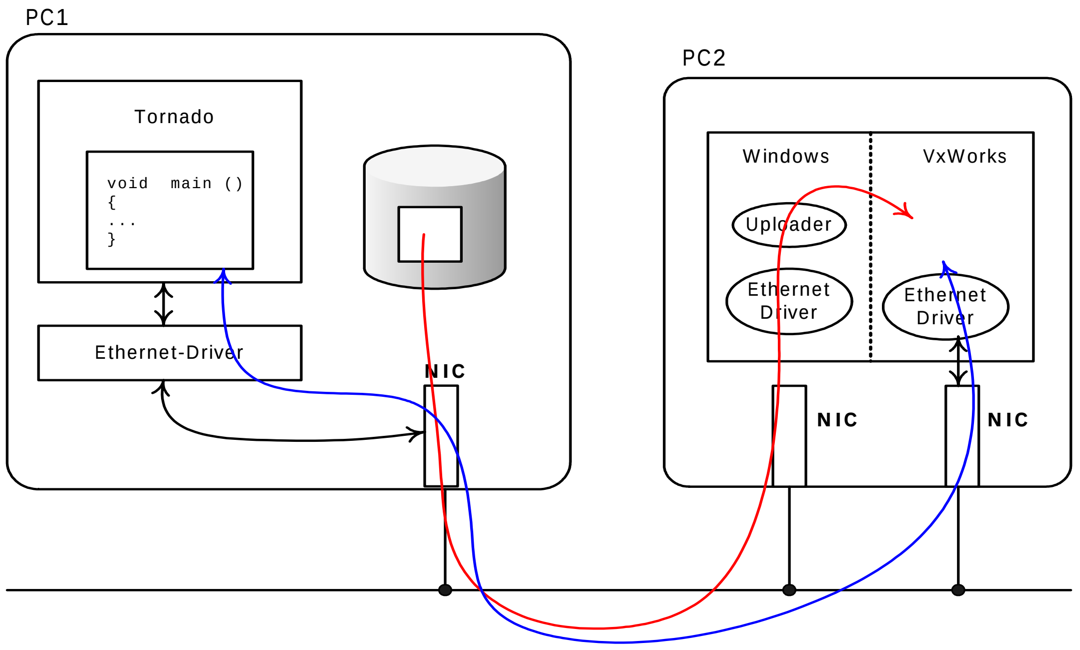
- Step 3: Under Windows XP, create a target-server connection to a VxWorks target agent.
- Step 4: Download the application and start debugging it.

5.3.2The two-system method

When host and target systems are physically separated, the connection between them can be made using either of two techniques.

Two systems – three network adapters

Using this technique (as illustrated below), the target system has two network adapters, one for Windows alone and one for VxWin:

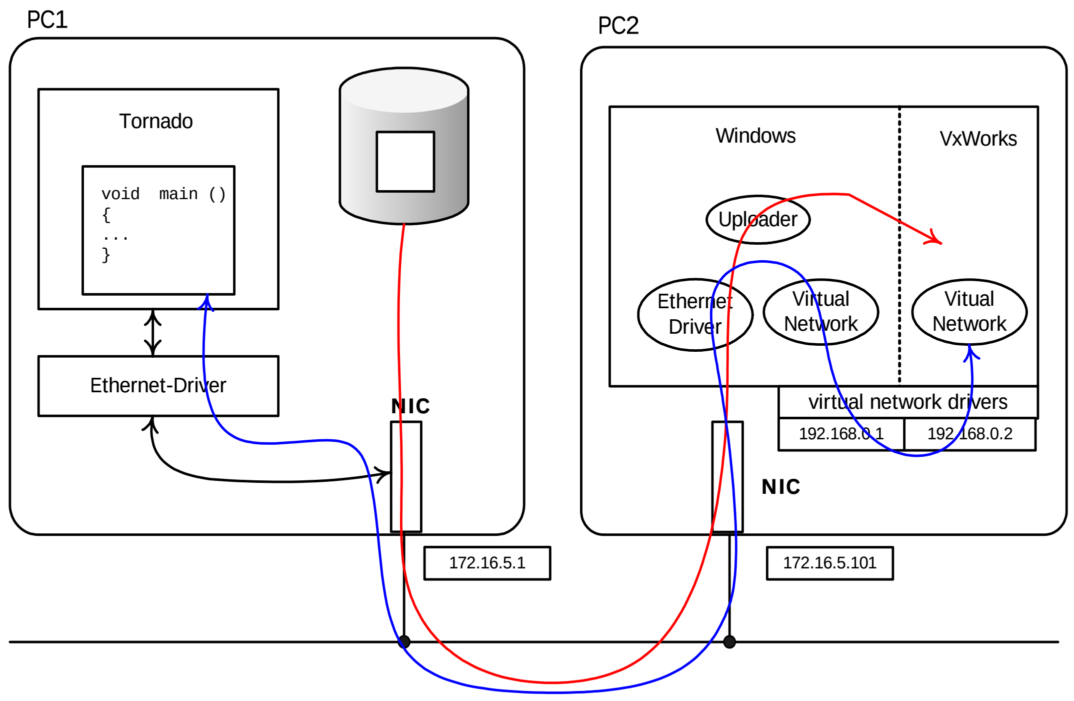


Development components are installed on the host computer – PC1, while run-time components are installed on the target computer – PC2. The development process is as follows:

- Step1: Build a new VxWorks image on PC1.
  Note: As shipped, VxWorks images may not contain support for a network adapter
- Step 2: To make the image on PC1 accessible from PC2, you must manually create a network share on PC1. Then, using the Uploader Utility on PC2, load the VxWorks image.
- Step 3: Create a target-server connection on PC1 to a VxWorks target agent on PC2.
- Step 4: Download the application and start debugging it.

Two systems – two network adapters

Using this technique, the target system has only one network adapter which is used by both Windows and VxWorks. The dual use of one adapter is accomplished by using "packet routing," described elsewhere in this manual. See Packet routing: VxWorks Uses the Windows network adapter. Packet routing ensures that all data packets sent to or originating from VxWorks on PC2 will be sent through PC2's Ethernet adapter.



Here too, development components must be installed on PC1 while run-time components must be installed on PC2.

6 Using peripheral devices under VxWin

6.1 Introduction

In a typical Windows system, each standard peripheral device and PCI-device and is accessed and controlled by a Windows driver. When it is desired to make devices available for the exclusive use of the real-time operating system, the

devices must first be logically removed from Windows control and thenn reassigned to the real-time system.
PCI boards: Because Windows deals with Plug and Play devices differently than it deals with standard devices, the procedure for assigning PCI devices to the real-time context differs from that for standard devices. For more information about assigning PCI devices to VxWorks, see Assigning PCI devices to VxWorks elsewhere in this manual.

6.2 Sharing Devices

No device may be shared between the two operating systems. A device must be assigned either to Windows or to VxWorks.
If an application program in one operating system context needs to use a device assigned to the other system context, VxWin provides several means for communicating data between operating system contexts. The means for doing this are straightforward.
While the transfer of data between systems cannot generally be done in an absolutely deterministic fashion, the real-time programmer can use techniques, such as multiple buffering, to sustain synchronized operations.
For more information about inter-system communication, please read the chapter on Inter-System Communication.

6.3 Sharing interrupts

Just as devices assigned to Windows may share an interrupt in the Windows context, devices assigned to VxWorks may also share an interrupt with other VxWorks devices. **But under no circumstances may a Windows device share an interrupt with a VxWorks device.** If you violate this rule, the resulting system behavior will be unpredictable and undoubtedly incorrect.
Important: To gather information about interrupt routing in your PC and to determine its overall suitability for hosting a VxWin system, users are provided with a utility program called the ***Real-Time Device Manager*** **(RTDM)**. This convenient program develops, displays and identifies potential interrupt conflicts and manipulates IRQ routing for the purpose of eliminating said conflicts; it also assigns devices from Windows to the real-time context (or vice versa). To learn more about this utility program, locate it and its supplementary documentation in ***..\VxWin\RTDM***.

6.4 Real-time Devices

Every hardware device to be used by VxWorks must first be assigned to the real-time operating system via a manual procedure using Windows Device Manager or semi-automatically via KUKA's ***Real-Time Device Manager*** **(RTDM)**. For more information about the RTDM, locate it and its supplementary documentation in **...\VxWin\RTDM**.

Exceptions: The following resources may not be reassigned to the real-time context:

- Floppy disk drive
- Hard disk drive (See note below.)
- On-board clock
- DMA controller
- Programmable Interrupt Controller
- Keyboard controller
- Graphic controller

Information: Although VxWorks tasks cannot directly access these devices, there are nonetheless several methods to help you accomplish this. For that, refer to Inter-System Communication and Accessing the PC's hard disk , elsewhere in this manual.

6.5 Custom real-time devices and VxWorks

If you are a VxWorks user that wishes to use an I/O device not already supported by VxWorks, you will have to write a real-time driver that conforms to Wind River Systems driver specifications. The VxWorks Programmers Guide is a good place to seek information on how to create such a driver.

6.6 Windows .INF and .PNF files

An .INF file is an ASCII text-file (defined by Microsoft) that contains comprehensive installation information for a peripheral device. When Windows

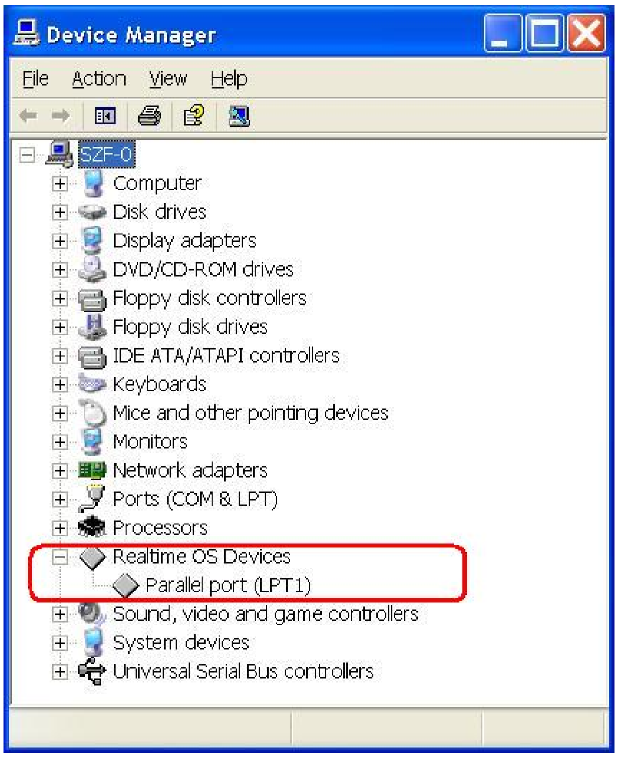
boots, it installs the device driver specified in the .INF file, builds registry entries, and enables any associated interrupts.
A .PNF file is a precompiled .INF file, containing information extracted from its corresponding .INF file.

6.7 Real-time .INF and .PNF files

Each device to be assigned to the real-time system requires a real-time .INF and .PNF file that provide installation information and declare it to be a real-time device.
VxWin provides pre-configured .INF files for assigning the LPT and COM ports to the real-time system. These files may be found on the VxWin software release medium (CD-ROM).
In general, to enable a device to be used by the real-time system, you must first remove the device from Windows control (uninstall it) and then reassign it as a real-time device. For a step-by-step procedure to make such re-assignments, see Assigning standard windows devices to the real-time system .

6.8 Assigning standard Windows devices to VxWorks

With some exceptions, you can reassign standard Windows peripheral devices to the real-time system. (See the exceptions in Real-time Devices .) To make the reassignment, you must either carry out a manual procedure, described below, or use KUKA's ***Real-Time Device Manager*** **(RTDM)**. To learn more about the RTDM utility program, locate it and its supplementary documentation in ***...\VxWin\RTDM***.
Important: There are separate procedures to assign PCI devices to the real-time system. See Assigning PCI devices to VxWorks.
In the following Device Manager screenshot, you can see a parallel port (LPT1) that was assigned to the **Realtime OS Devices** class. Diamond-shaped icons indicate real-time devices.



6.8.1Assigning an LPT port to VxWorks

When Windows boots, it generally acquires printer port devices (LPTx) for itself. You can nonetheless reassign these devices to the real-time operating system context.
By removomg the device from the Windows context and then reassigning it to the real-time system, you will prevent Windows from acquiring the a device for itself during the boot process. A step-by-step procedure to accomplish this is presented in Assigning standard Windows devices to the real-time system. This procedure guides you in replacing the appropriate Windows .INF file for a device with a real-time .INF file.
Depending on which LPT port you wish to acquire, you will use either ***RTOS_Parallel1.inf*** or ***RTOS_Parallel2.inf***.
The path to access either of those files is:
```
  CD:\Windows\Drivers\Rtos-Infs\RTOS_Parallel?.inf
```

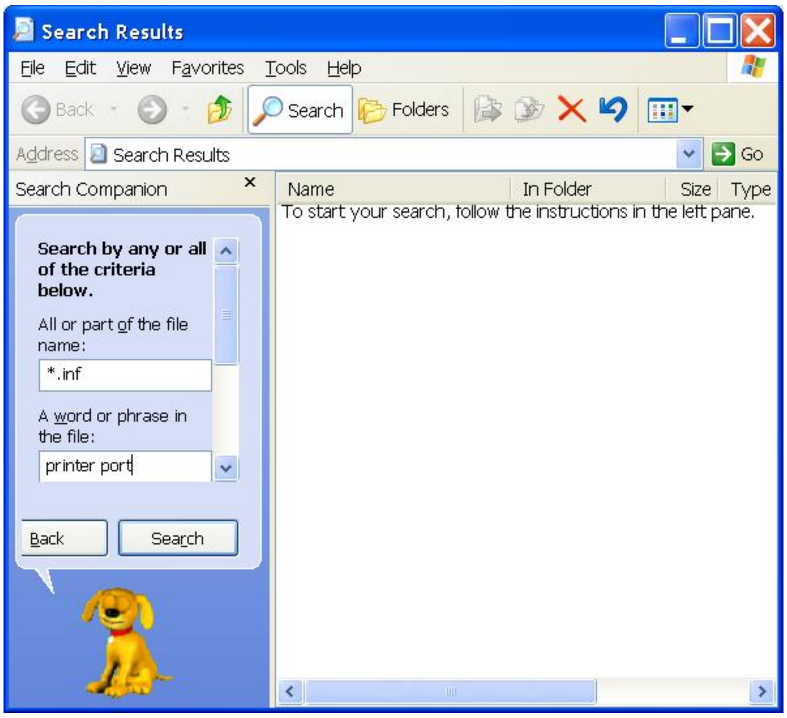
6.8.2Assigning a COM port to VxWorks

When Windows boots, just like LPT ports, COM ports are generally acquired automatically for use by Windows. But you can can also reassign COM ports for the exclusive use by the real-time system.
To prevent Windows from accessing a COM port, it must first be disengaged from service as a Windows device and then reassigned for use by the real-time system. The step-by-step procedure that describes how to accomplish this – Assigning standard Windows devices to the Real-time System – will direct you to replace the Windows .INF file for a particular device with a real-time .INF file from the VxWin RT distribution.
Depending on which COM port is to be acquired, you will use either ***RTOS_Serial1.inf*** or ***RTOS_Serial2.inf***. These files may be found at the following path:

```
  CD:\Windows\Drivers\Rtos-Infs\RTOS_Serial?.inf
```

6.8.3Assigning other standard devices to VxWorks

If you want to use some other standard Windows device under VxWorks, you must, of course, remove it from the Windows context and reassign it to the real-time system. The steps described below summarize a procedure to accomplish this, using Windows Device Manager:
Important: This procedure is not appropriate for PCI devices. To reassign PCI devices, which are Plug and Play devices, you must use a different procedure. See - Assigning PCI devices to VxWorks.
1. Identify the .INF file that is responsible for the device.
In Windows XP this can be done using the Windows Explorer file-search mechanism to find the device. INF files are stored in the Windows INF subdirectory: C:\WINDOWS\INF.
2. The following screenshot illustrates using Windows Explorer to search for the printer port's .INF file. For some other standard device, you must instead search for the correspondingly appropriate text.



To start your search, follow the instructions in the left pane.

Search by any or all of the criteria below.

All or part of the file name: *.inf

A word or phrase in the file: printer port

3. Once the proper .INF file has been identified, you should change its filename from XXX.inf to XXX.inf.org (or to some other name that cannot be recognized by Windows). Then find and rename the corresponding .PNF file from XXX.pnf to XXX.pnf.org (a name not recognizable by Windows).
XXX is the part of the file name that uniquely identifies the specific device.
Note: The .org file ending was chosen arbitrarily for this example. It is only important that the old file be renamed in such a fashion that Windows will no longer recognize it.
4. Next, copy the real-time device .INF file from the VxWin release medium into the Windows .INF subdirectory and...
5. Remove (uninstall) the Windows device entry from the Device Manager.
6. Because Windows can automatically detect hardware devices, you do not have to manually install the real-time version now.
7. Reboot the system
8. In the course of rebooting, Windows will automatically detect the hardware and the new (real-time) .INF file and assign it to the real-time class of devices. The real-time .INF file points to the real-time device driver.
9. If you open the Device Manager again, you will now see that the device has been registered as a real-time device.
Remember: Once a device has been assigned to the real-time system, it can no longer be used by Windows – unless you reverse the above procedure. To restore

a standard device to Windows, go through the procedure once more, this time disabling the real-time .INF and .PNF files by renaming them and restoring the original Windows .INF and .PNF files by giving them back their original names.

6.9 Assigning PCI Devices to VxWorks

Whenever Windows XP boots, its PCI bus ennumerator scans through all possible PCI bus addresses. Whenever it finds a device, it searches for an .INF file that corresponds to that device, creates proper registry entries for that device and loads whichever device driver pointed to within that .INF file. When that process is done, the PCI device will be associated with the class of devices indicated in the .INF file. If you open a Device Manager window, you should see all the devices properly listed under their respective classes.
Since a PCI card that is already assigned to Windows XP is not immediately available for use by VxWorks, you must perform some additional steps to make it so.
1. You must first remove the device from the direct control of Windows XP. To do this, you will make invalid the existing .INF and .PNF files for that device by deleting or renaming them.
2. Then you will prepare a real-time .INF file for that device. This .INF file will point to the realtime operating system PnP driver. (A real, functional driver is not required for this purpose.) Both the .INF file (a ***template.INF*** file) and the driver (***rtospnp.sys***) are delivered with VxWin.
3. You can then either re-install the device using the real-time **.INF** file you have modified or re-boot the system. Thereafter, the PCI device you designated should appear in Windows XP Device Manager display as a member of the ***Realtime OS Devices***.

—

How to do it:
To create the required **.INF** file, you begin with the prototype .INF file (***RTOS_Template.inf***) that is provided for this purpose. You will find it on the CD-ROM at: ***CD:\Windows\Drivers\Rtos-Infs***.
The following illustration shows excerpts from an **.INF** file that has already been modified for the real-time class of devices. Observe that the **VendorID** and **DeviceID** values specific to the PCI card have already been set to real values, corresponding to those on the physical PCI card.

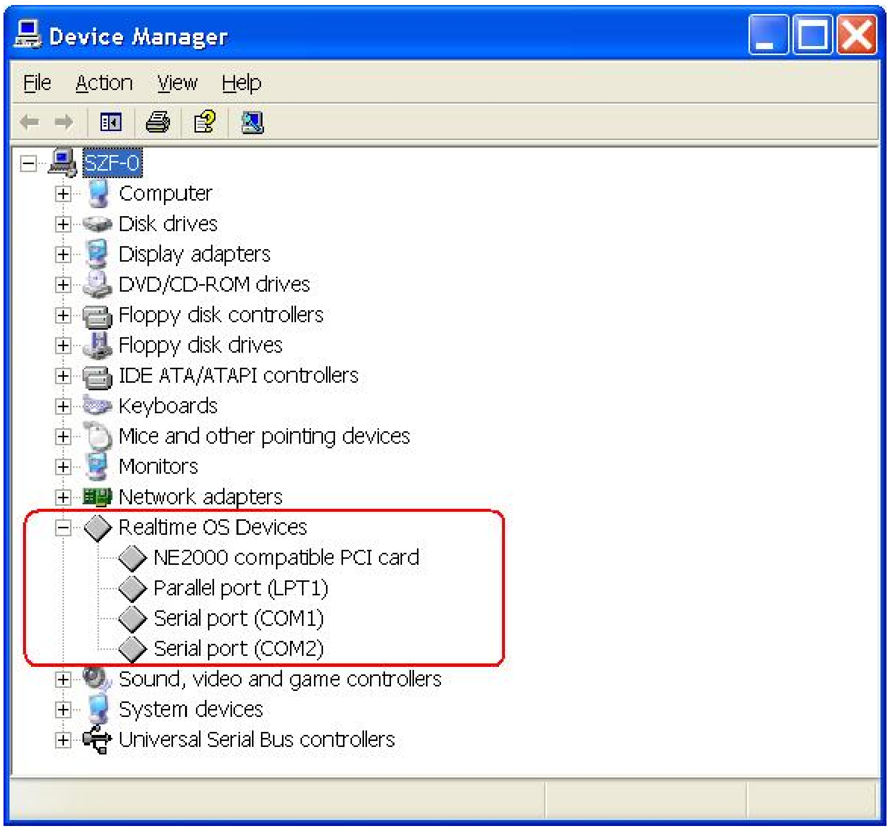
```
[DeviceList]
 %DISPLAYNAME% = DriverInstall,
 PCI\VEN_10EC&DEV_8029
```

And you may also enter meaningful names of your choice for the fields ***DisplayName*** and ***FriendlyName***:

```
        [Strings]
          MFGNAME      = "KUKA Roboter"
          INSTDISK     = "Installation Disc"
          DISPLAYNAME  = "NE2000 compatible PCI card"
          FRIENDLYNAME = "NE2000 compatible PCI card"
          rtospnp.SVCDESC = "RTOS PnP Driver"
```

These INF files then have to be used to install the RtosPnp driver.

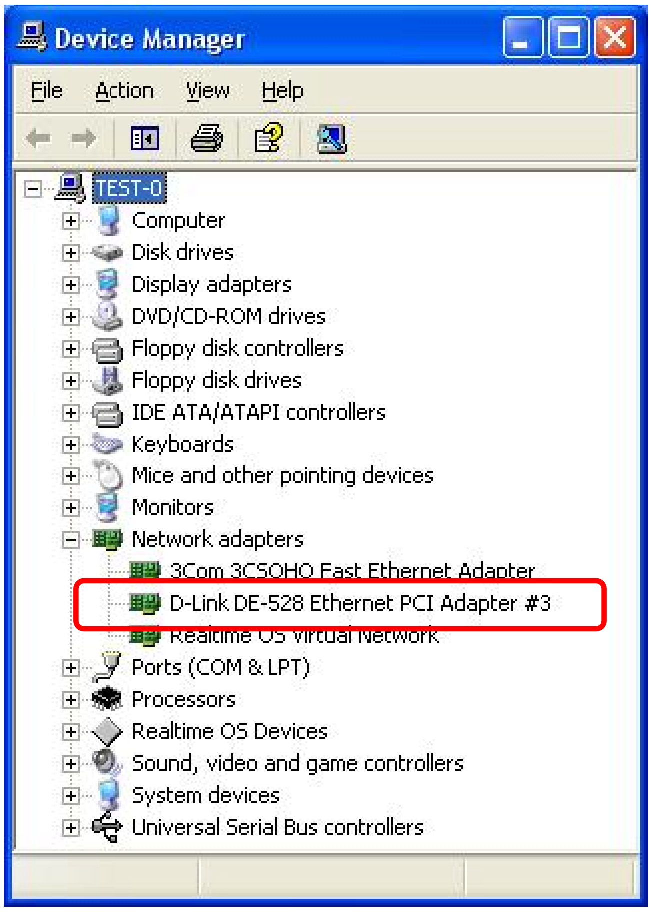
Example: The following screenshot plainly shows that an **NE2000** compatible PCI card (among other devices) has been (re-)assigned to the class of Real-time OS devices. (VxWin uses Gray diamond-shaped icons to signify real-time OS).



```
SZF-0
  Computer
  Disk drives
  Display adapters
  DVD/CD-ROM drives
  Floppy disk controllers
  Floppy disk drives
  IDE ATA/ATAPI controllers
  Keyboards
  Mice and other pointing devices
  Monitors
  Network adapters
  Realtime OS Devices
    NE2000 compatible PCI card
    Parallel port (LPT1)
    Serial port (COM1)
    Serial port (COM2)
  Sound, video and game controllers
  System devices
  Universal Serial Bus controllers
```

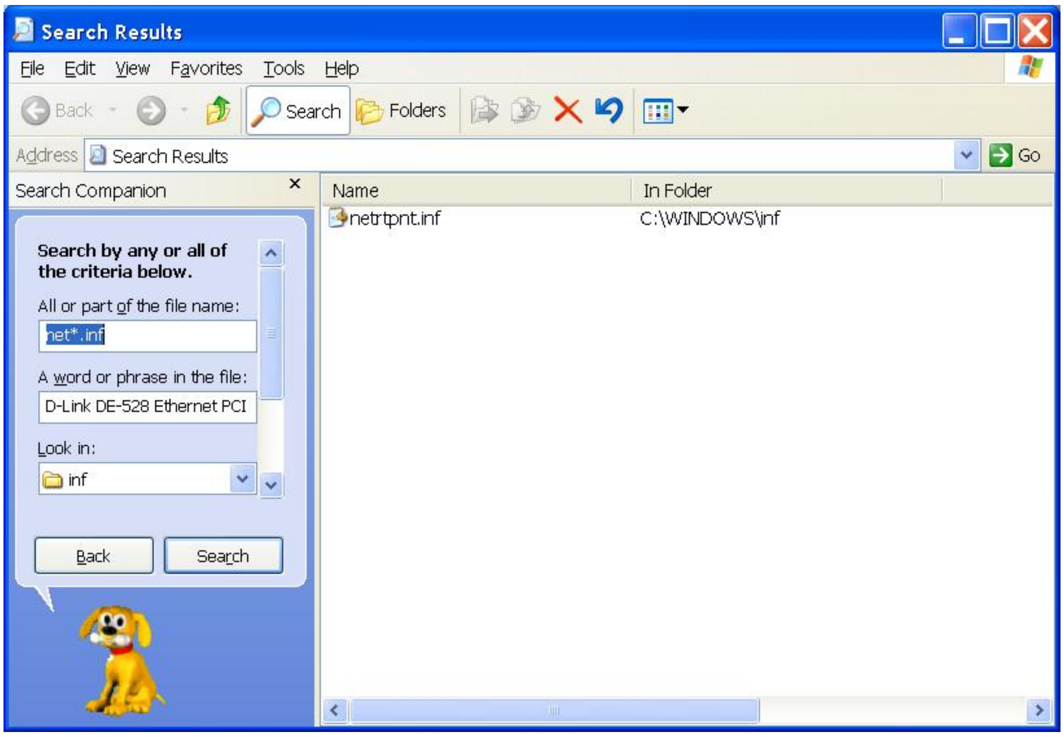
6.9.1 Example: Assigning an NE2000 PCI card to VxWorks

Here we show you how to assign a NE2000 compatible network adapter card to VxWorks.
The following steps must be performed:
1. Identify a slot in your PC that does not share its interrupt with a Windows XP device.
2. Physically install the device in the available slot.
3. Use the Windows XP Device Manager to reassign the device as a Real-time OS
• You will probably find your network adapter card listed among the ***Network adapters*** in the **Windows XP Device Manage**r window. In the screenshot below, you can see a ***D-Link DE-528 Ethernet adapter card*** (using the RTL8029 chip) listed among others.



RtosPnp.sys (stored at C:\Program Files\VxWin\Rtos-Infs) is a driver used to make certain that when Windows XP initially boots, the Plug-and-Play mechanism won't disable the PCI slot where the card is physically located. Without this driver, upon booting Windows XP, that particular I/O slot would be made physically inaccessible.

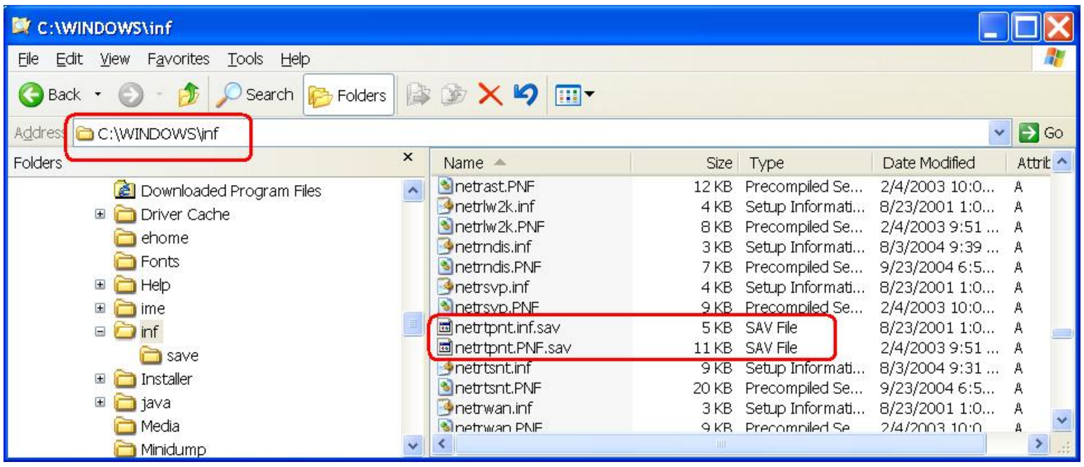
• If the INF-file ***RTOS_NE2000.INF*** (stored at C:\Program Files\VxWin\Rtos-Infs) is not suitable for your particular adapter card – i.e., the Vendor or Device IDs in the file are incorrect – you will have to modify the file to correct them.
• The standard Windows XP ..INF and .PNF files must now be deleted or renamed (with a new suffix, e.g. ".sav") so that when Windows boots, it will not use them. As illustrated in the following screenshot, you must first identify the files by searching for them. Performing a content search on the device-name may be an effective means to accomplish this. Once you have found the .INF file, you may assume that the .PNF file has the same filename but with the .PNF extension.


Search Results
File Edit View Favorites Tools Help
Back Search Folders
Address Search Results Go
Search Companion
Back Search

Search by any or all of the criteria below.
All or part of the file name: net*.inf
A word or phrase in the file: D-Link DE-528 Ethernet PCI
Look in: inf

Name	In Folder
netrtpnt.inf	C:\WINDOWS\inf

• After changing the names of the standard Windows .INF and .PNF files, you can display them to verify your work. In the following screenshot, you can see that the user changed both filenames so that the operating system can't recognize them, but a user can still read and understand them:


C:\WINDOWS\inf

Address: C:\WINDOWS\inf

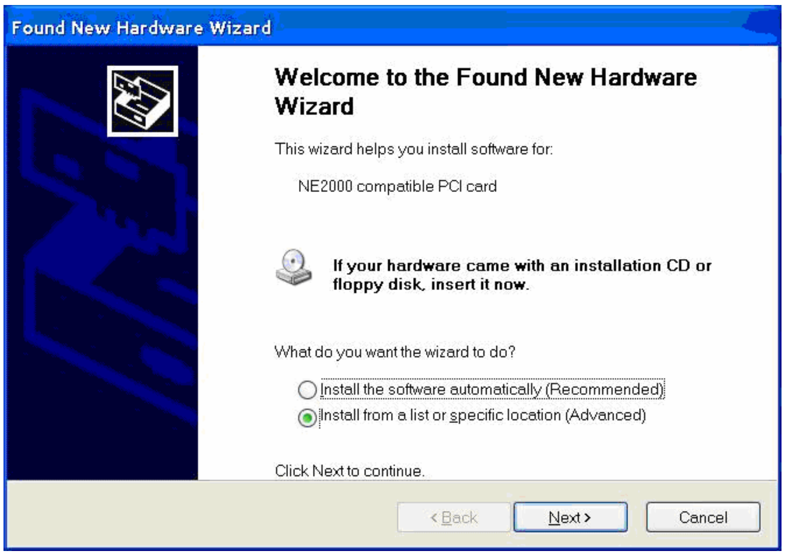
Folders: Downloaded Program Files, Driver Cache, ehome, Fonts, Help, ime, inf, save, Installer, java, Media, Minidump

Name	Size	Type	Date Modified	Attrib
netrast.PNF	12 KB	Precompiled Se...	2/4/2003 10:0...	A
netrlw2k.inf	4 KB	Setup Informati...	8/23/2001 1:0...	A
netrlw2k.PNF	8 KB	Precompiled Se...	2/4/2003 9:51 ...	A
netrndis.inf	3 KB	Setup Informati...	8/3/2004 9:39 ...	A
netrndis.PNF	7 KB	Precompiled Se...	9/23/2004 6:5...	A
netrsvp.inf	4 KB	Setup Informati...	8/23/2001 1:0...	A
netrsvp.PNF	9 KB	Precompiled Se...	2/4/2003 10:0...	A
netrtpnt.inf.sav	5 KB	SAV File	8/23/2001 1:0...	A
netrtpnt.PNF.sav	11 KB	SAV File	2/4/2003 9:51 ...	A
netrtsnt.inf	9 KB	Setup Informati...	8/3/2004 9:31 ...	A
netrtsnt.PNF	20 KB	Precompiled Se...	9/23/2004 6:5...	A
netrwan.inf	3 KB	Setup Informati...	8/23/2001 1:0...	A
netrwan.PNF	9 KB	Precompiled Se...	2/4/2003 10:0...	A

• You are now ready to remove the network card from the control of Windows XP. Use the Windows Device Manager to do this.

• Then reboot the system.

• During booting, Windows Plug and Play mechanism automatically recognizes the physical presence of the network card. At the same time, it realizes that it has not yet got a driver for it. Windows will therefore display a dialog box, similar to the one in the next screenshot. In this dialog box you are offered a choice between an automatic and a manual method for installing the real-time device driver. You should choose the manual installation option.


Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

This wizard helps you install software for:

NE2000 compatible PCI card

If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

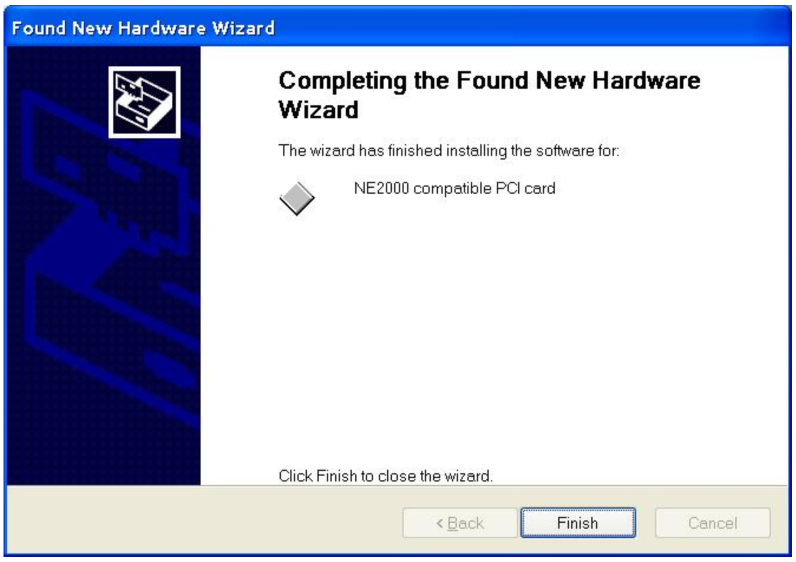
- ○ Install the software automatically (Recommended)
- [✓ Install from a list or specific location (Advanced)]

Click Next to continue.

< Back | Next > | Cancel

• Later on you should select the INF file to be used. The file is stored at C:\Program Files\VxWin\Rtos-Infs.

• The system then acknowledges a successful driver installation, as shown in the following screenshot:



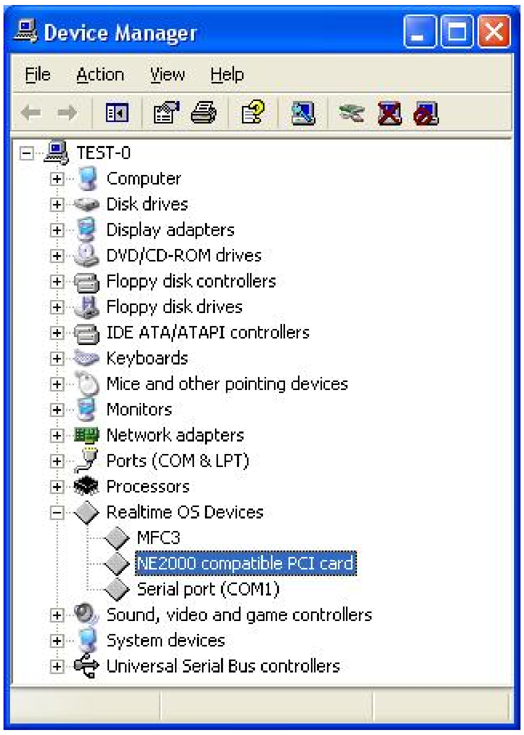
Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

NE2000 compatible PCI card

Click Finish to close the wizard.

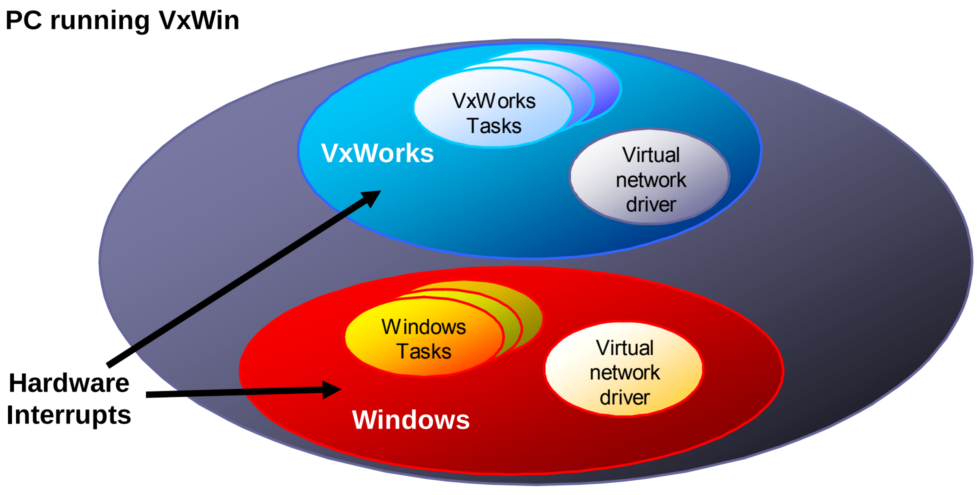
• If you now return to the Device Manager, you should see that the network adapter card has been reassigned as a real-time device. In the following screenshot, real-time devices are indicated symbolically by gray diamond icons.



```
TEST-0
  Computer
  Disk drives
  Display adapters
  DVD/CD-ROM drives
  Floppy disk controllers
  Floppy disk drives
  IDE ATA/ATAPI controllers
  Keyboards
  Mice and other pointing devices
  Monitors
  Network adapters
  Ports (COM & LPT)
  Processors
  Realtime OS Devices
    MFC3
    NE2000 compatible PCI card
    Serial port (COM1)
  Sound, video and game controllers
  System devices
  Universal Serial Bus controllers
```

7 Interrupts and VxWin

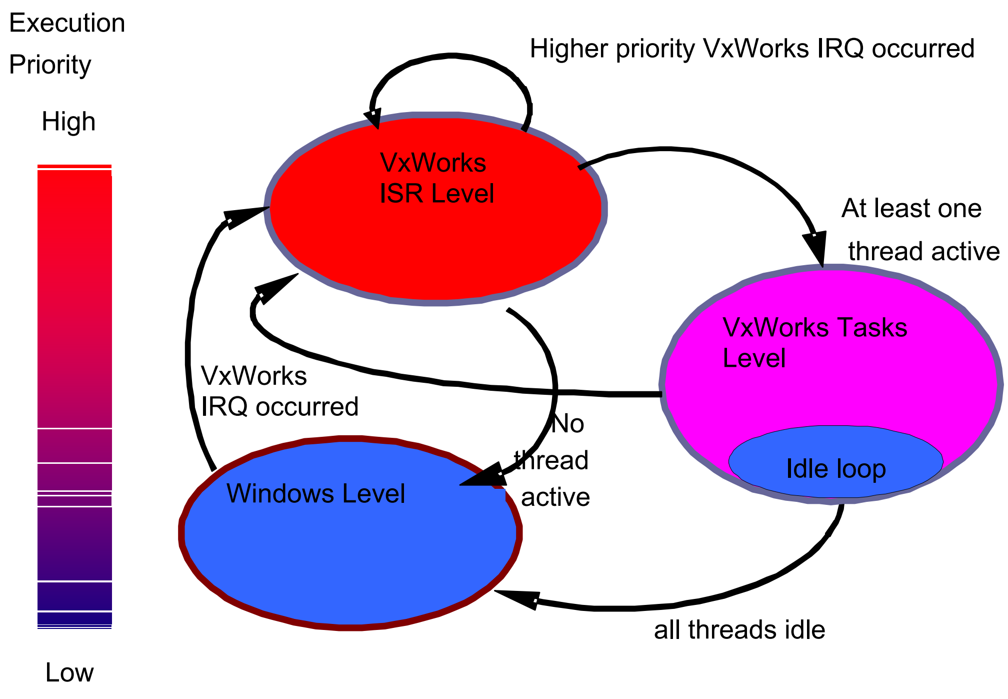
Hardware devices, with the help of their associated controller/adapters, generate interrupts to signal when they require attention from the CPU.
Upon receiving an interrupt signal, the CPU will promptly set aside its current processing to attend instead to the needs of the interrupting device. This rapid response assures that real-time operating systems respond to external events as quickly as possible.
Note: While peripheral devices may share an interrupt under *either* of the two operating systems, any attempt to share an interrupt *between* the two systems under VxWin will result in unpredictable and certainly incorrect system behavior.
The hardware interrupts under VxWorks have the same priority as they have under Windows. The highest priority is IRQ 0, the lowest, IRQ 15.



All interrupts for VxWorks as well as those for Windows are connected to the PC's Programmable Interrupt Controller which, upon the occurrence of a connected event, will interrupt the CPU.
VxWin guarantees that interrupts controlled by VxWorks always have a higher priority than any program or any interrupt under Windows. Within microseconds after an interrupt occurs, whichever Windows program or lower priority VxWorks task was running is stopped and control is transferred to the VxWorks Interrupt Service Routine (ISR) that the user assigned to that interrupt during initialization.
After the ISR finishes executing, control is returned to the VxWorks kernel, which then checks to see if any other real-time threads are ready to run. If that is

the case, the kernel will activate the now-ready-to-run task. VxWorks tasks will continue to run until all are suspended or blocked. After which, the system will enter the VxWorks idle loop, which in turn returns control to Windows.
Since Windows is re-activated only when all VxWorks tasks are idle, one could say that Windows runs as if it were part of VxWorks idle loop.

State diagram of VxWin



7.1 Exception-Handling with VxWorks

The processor hardware is able to detect and respond to certain kinds of serious errors such as faulty addressing (page/segment errors) or floating-point processor errors (such as divide by zero). This class of hardware errors are generically called exceptions. When an exception occurs, it triggers a high-priority interrupt. The correct reaction to the occurrence to such an exception depends on which operating system, Windows or VxWorks, was running at the time. To enable them to respond, each in its own way, VxWin provides two different exception tables (IDT), one for each system context, which it swaps at appropriate times. Whenever a real-time interrupt activates VxWorks, VxWin replaces the Windows exception table with one defined for VxWorks. Correspondingly,

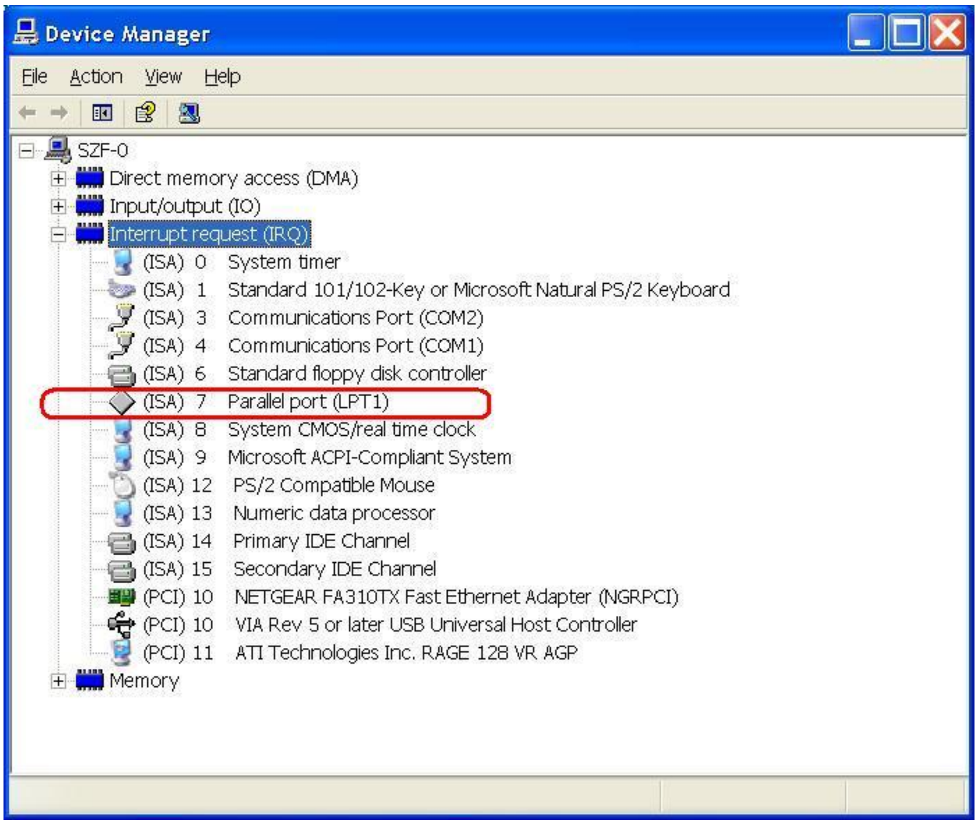
whenever control is returned to Windows, the Windows exception table is likewise restored.
Exception-table-swapping, is only one of the mechanisms VxWin implements to enable the operating systems to share a single execution platform.

7.2 usrRoot Task Exceptions

If the usrRoot task (VxWorks) generates an exception, i.e., a page error, the entire system will crash. While experience has shown that this occurs only infrequently, when it does occur, the most common reason for it, it has been determined, is that VxWorks had been incorrectly configured.

7.3 Hardware Interrupt Assignment

VxWin users must carefully consider which interrupts to assign to which system. More information about interrupt configuration can be found in Interrupt configuration file (interrupt.config).
The Windows Device Manager can be used to display a list of devices showing which interrupt sources are assigned to which Interrupt Request Numbers (IRQs). In the following screenshot, for example, you can see that the parallel port (LPT1) has been assigned as a real-time device to IRQ 7. (The gray diamond-shaped icon indicates that it is a real-time device.) Since Interrupt 7 (LPT1) is assigned to VxWorks, it follows that no Windows device may use IRQ 7.



SZF-0
Direct memory access (DMA)
Input/output (IO)
Interrupt request (IRQ)

```
(ISA) 0    System timer
(ISA) 1    Standard 101/102-Key or Microsoft Natural PS/2 Keyboard
(ISA) 3    Communications Port (COM2)
(ISA) 4    Communications Port (COM1)
(ISA) 6    Standard floppy disk controller
(ISA) 7    Parallel port (LPT1)
(ISA) 8    System CMOS/real time clock
(ISA) 9    Microsoft ACPI-Compliant System
(ISA) 12   PS/2 Compatible Mouse
(ISA) 13   Numeric data processor
(ISA) 14   Primary IDE Channel
(ISA) 15   Secondary IDE Channel
(PCI) 10   NETGEAR FA310TX Fast Ethernet Adapter (NGRPCI)
(PCI) 10   VIA Rev 5 or later USB Universal Host Controller
(PCI) 11   ATI Technologies Inc. RAGE 128 VR AGP
```

Memory

7.4 Interrupt processing in VxWin

A hardware interrupt could occur when the VxWin system is either in Windows mode or VxWorks mode.
Assuming the system is in Windows mode when the interrupt occurs, the Interrupt Vector Table (Windows context) will not send execution directly to VxWorks but rather first to a routine in the Operating System Device Driver. (See Operating System Device Driver).
Knowing that the interrupt is assigned to VxWorks, that routine causes the execution context to be switched to the real-time system and then transfers control to the Interrupt Service Routine that the VxWorks user had attached to that interrupt.
Interrupt processing then proceeds as usual for VxWorks without in any way having to take into account that a context change was undertaken before the interrupt processing began.

If that interrupt or any other real-time interrupt occurs while the real-time execution context is in effect, execution will vector directly to the corresponding real-time Interrupt Service Routine.
If an interrupt assigned to Windows occurs while the real-time context is in effect, it will be forced to wait until the real-time system has entered its idle-state. Later, after the Windows context has been restored, that interrupt will be serviced in the usual fashion for Windows.

7.4.1Interrupt Service Routine

A short piece of code that generally does only a limited amount of processing, an Interrupt Service Routine (ISR) is the first routine to be given control upon the occurrence of an interrupt.
Since Interrupt Service Routines operate in their own context, they are not, in this sense, *normal* routines. For this reason, ISRs may only use a limited number of VxWorks system calls. They aren't permitted, for example, to use the floating-point coprocessor or any VxWorks functions that would cause interrupt blocking. See the *VxWorks Programmer's Guide* for more detail.
And as long as an ISR is running, the real-time operating system is unable to pursue its priority scheduling scheme, which implies that no other routine can run, no matter what its priority, until the ISR has *returned*.
Therefore, if an ISR has more elaborate processing to perform than can be done in just a few instructions, it should be programmed to trigger an event to wake-up another (high-priority) task that has been programmed to wait for that event. After an ISR returns, the real-time operating system checks its priority scheduler and then starts the highest priority task that is waiting to run.
If your ISR does trigger a secondary task, as mentioned above, remember that such secondary tasks, while in every sense normal tasks, should be programmed with priorities higher than all other less urgent tasks. This is to ensure that all work associated with an interrupt will be processed to conclusion before any less urgent task can run.
It is not required to service interrupts in two stages. If no processing beyond the brief action performed by an ISR is required, then it is unnecessary to introduce a secondary task. For example: if the occurrence of an interrupt conveyed the entire information – e.g., the opening or closing of a switch – and all that need be done is to set a variable to reflect the state of the switch, an ISR alone would be sufficient to perform that work.

In any case, to get the best overall response times from your system, you should employ a secondary interrupt task whenever an interrupt requires more than minimal processing.

7.4.2More about servicing Interrupts

If the occurrence of an interrupt requires that more than a minimal amount of work be performed, a good programming strategy is to have the ISR set an event so that the additional required work can be performed by an ordinary but high-priority task. The ISR itself is generally programmed to do little more than to wait for its interrupt to occur, to acknowledge its interrupt, and to signal an event. To make certain that interrupts will be processed to conclusion before any other task in the real-time system can run, programmers must assign a high priority to any secondary interrupt task.
Even then, it is important that such high-priority tasks be used only to carry out critical, time-related functions. Any processing that can be performed at a less urgent pace, should be done by tasks to which you have assigned lower priorities.
The following enumerated list explains the stages of processing an interrupt:

- A real-time interrupt occurs.
- If Windows was executing when the interrupt occurred, VxWin performs a context switch, and program execution proceeds, starting at the Interrupt Service Routine (ISR) specified for that interrupt.
- If the system was running in the VxWorks context when the interrupt occurs, execution is immediately transferred to the corresponding ISR defined for VxWorks.
- Whether interrupted out of Windows or out of a lower priority VxWorks process, the ISR has no awareness of which context had previously been running.
- The Interrupt Service Routine runs. It may acknowledge the interrupt in the corresponding hardware device (if required by the specific device) and then perform some minimal processing.
- If more than minimal processing is required, it will also signal an event so that a secondary high-priority task will complete the required interrupt processing.
- When the IST finishes, it returns to the operating system. Not until the IST has returned can other tasks run.

7.5 Interrupt Vectors

That VxWin maintains two execution environments, one for each operating system, means not only that each operating system has its own program memory but also that each system's peripheral devices are invisible to the other and that the CPU's hardware interrupt mechanism is managed in a special fashion. Among other important structures replicated in the VxWin environment is the Interrupt Vector Table (IVT). The IVT for the Windows side remains as defined for Windows, but the IVT for the real-time side of the system is somewhat different. The following diagram illustrates the IVT for the real-time system context. The entries reserved for the user's own peripheral devices are indicated. The table is 4 bytes wide (address) and 256 entries deep:

Interrupt Vector Table

Available	0xFF 0x41
Virtual Network Driver Interrupt	0x40
Hardware Interrupts	0x3F 0x30
Available	0x2F 0x20
Intel Hardware Exceptions	0x1F 0x00

Regarding the foregoing table:
Vectors are specified as offsets into the CPU's vector table. By default, the vector table begins at offset 0.
When using VxWin, certain conventions concerning interrupt vectors must be adhered to:

- Interrupt numbers in the range (*0x30* to *0x3F*) are reserved for hardware interrupts.

- Certain interrupts are reserved for Intel hardware exceptions (0x00 to 0x1F). While such interrupts are usually attached to VxWorks exception handlers, if you adhere to certain conventions outlined in Wind River's documentation, you may change the ISR assignments.
- Interrupts in the range (0x41 to 0xFF) can be attached by the user with help of the function: ***intHandlerCreate***, as described in Wind River's documentation.

7.6 Connecting an ISR to an interrupt

After VxWorks has started, you can use the VxWorks ***intConnect( )*** function to connect an Interrupt Service Routine (ISR) to a specific interrupt level.
In the following example, the Interrupt Service Routine *proc* is connected to interrupt level 7. Thereafter, interrupt level 7 in the interrupt controller is enabled.

```
intLvl = 7;
intNum = INT_NUM_GET(intLvl);
intVec = INUM_TO_IVEC(intNum);
intConnect(intVec, proc, param);
sysIntEnablePIC(intLvl);
```

Note: *intLvl* is the interrupt level, *intNum* is the interrupt number, *intVec* is the interrupt vector. The macro *INT_NUM_GET* supplies the required interrupt vector from either of two arrays in accord with the interrupt controller type. One can also use predefined macros, such as *INT_NUM_COM1* or *INT_VEC_COM1*, which are defined in the header file *configInum.h* . While *proc* is the C routine to be called in response to the interrupt (the ISR), *param* is a parameter to be passed to the routine each time it is called.
For more information about this topic, see Wind River's documentation, *VxWorks Programmer's Guide*.

7.7 Additional interrupt functions – VxWorks

When your VxWorks application must use other functions that pertain to interrupts, you should use the standard functions as defined by Wind River for VxWorks. Please refer first to the *intLib* and *intArchLib* libraries in the Wind River documentation (*VxWorks OS Libraries API Reference* and *VxWorks Programmer's Guide*).

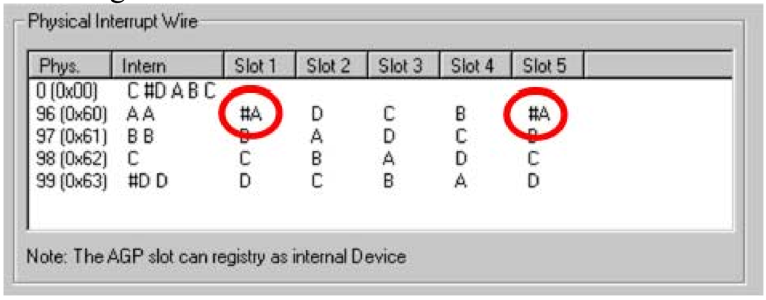
8 PCI bus interrupts and interrupt-sharing

8.1 Introduction

Under VxWin, sharing interrupts between Windows and VxWorks is prohibited. PCI cards assigned to VxWorks may not be plugged into a PCI slot that uses the same hard-wired interrupt line as does any external or internal Windows device. In order to prevent interrupt conflicts between several PCI bus cards, the physical arrangement of these cards must be carefully planned. If it turns out that there is a conflict in the interrupt assignments, the easiest and most common solution is simply to move a PCI card into a different PCI slot and/or choose a different card.

8.2 Investigate the PCI hardware

The PCI bus specification physically allows different cards in the same PC system to share an interrupt. While each PCI board may generate up to four hardware interrupts on four physical interrupt lines (INTA, INTB, INTC and INTD), PCI cards are generally laid out (by their manufacturers) to assert an interrupt on just one line, INTA. Only complex single-function boards or multi-function boards commonly use more than one interrupt line.
In most IBM-compatible PCs, each of the four interrupt lines is hard wired to a different (adjacent) connector position in the neighboring slots. This means, for example, that INTA of slot 1 is hard wired to INTD of slot 2 and to INTC of slot 3 and to INTB of slot 4 and finally also to INTA of slot 5.
This would have the effect of forcing the four A-level interrupt of four adjacent cards to assert physically on INTA, INTB, INTC and INTD, but it would also cause a fifth card to assert the same interrupt as the first card. Refer to the following illustration:



Physical Interrupt Wire

Phys.	Intern	Slot 1	Slot 2	Slot 3	Slot 4	Slot 5
0 (0x00)	C #D A B C					
96 (0x60)	A A	#A	D	C	B	#A
97 (0x61)	B B	B	A	D	C	B
98 (0x62)	C	C	B	A	D	C
99 (0x63)	#D D	D	C	B	A	D

Note: The AGP slot can registry as internal Device

Complicating the situation is the fact that some internal peripheral devices, such as a USB host controller, may also use one of those same physical interrupt lines.

—

Important: To help a VxWin user investigate his/her PC hardware, the standard VxWin product release contains a utility program that helps users by graphically depicting the layout of those interrupt lines, just as discussed above. To learn more about the ***Real-Time Device Manager*** **(RTDM)**, locate it and its supplementary documentation in ***...\VxWin\RTDM***.

—

But even in those cases when PCI devices use different physical interrupt lines, they might still be hard-wired together by means of a PC **Interrupt Router**.
The physical interrupt lines (as in above screenshot) are connected to the inputs of the **Interrupt Router**. And the **Interrupt Router** outputs are in turn connected via the **Interrupt Controller** to the CPU chip.
The **Interrupt Router** maps physical interrupt lines (PCI bus, ISA bus and internal devices) to the input of the Programmable Interrupt Controller. Just how many physical interrupt lines the **Interrupt Router** merges onto one line depends on the **Interrupt Router** itself and the number of interrupts available on the interrupt controller.
If you need to make interrupt lines available to your application, you can do so by using BIOS to disable one or more of the standard devices. For example, you could, using BIOS, turn off the COM ports, USB controller or Audio/Sound controller.
Caution: The number of output lines the **Interrupt Router** provides depends on the PC hardware (chipset). If the ***Interrupt Router*** has a small number of output lines, it is quite likely that several physical interrupts are wired to a common interrupt on the controller.

8.3 The problem with shared interrupts

PCI devices that interrupt along the same interrupt pin route are forced to share an interrupt, and the routing of interrupt pins to an Interrupt Router is system (chipset) dependent.
In most cases, finding and isolating the desired interrupt lines for use under VxWin is not much of a problem, but experience has shown that in some PCs it is not possible to separate the interrupt lines.
While devices used by Windows may share interrupts within the Windows context, devices used by the real-time operating system may only share interrupts

within the real-time context. Under no circumstances, may an interrupt be shared between both systems.
Although one might be tempted to imagine that permitting Windows and VxWorks to share one or more interrupts should be possible, it is in reality a difficult matter. The following discussion reveals more about the nature of the problem:

Assume that Device A (installed under Windows) physically shares an interrupt with device B (installed under VxWorks). If Device A generates an interrupt while VxWorks is running, how could interrupt-handling software process the interrupt without impairing the ability of the real-time system to fulfill its tasks within prescribed times?
One might be tempted to solve this problem in either of the following ways:

- Disable the interrupt in the interrupt controller and re-enable it only when VxWorks returns to its idle state and gives control to Windows.
- Or one could imagine implementing a VxWorks interrupt handler that could prevent Device A from generating an interrupt until after VxWorks returns to Windows, thinking that it wouldn't be so bad if the interrupt intended for Windows would be handled later.

- The first approach would block interrupts generated by device B for some time, which is totally unacceptable for real-time operations.
- While the second method appears promising at first, it would require that a special interrupt handler be written for each VxWorks device that must share an interrupt with a Windows device. And the Windows device driver would probably have to be modified as well. With a lot of thought and work, this approach might solve the problem, but it is a risky approach without guarantee of success and would therefore not be recommended for building a stable commercial product.

Fortunately, most of the time, system designers can eliminate interrupt conflicts. Some custom PCI boards, for example, provide flexibility in combining real-time devices by allowing the user to specify on which of the interrupt pins (INTA, INTB, INTC, or INTD) the board’s interrupts should be asserted. This might very well solve the requirement imposed by fixed interrupt lines.
Or a PC architecture that permits each input of a *programmable interrupt router* to be associated with a specific interrupt pin would also be favorable for real-time applications. Given such an architecture, one could force each interrupt to be associated with a specific hardware level.

Unfortunately, under the current *CompactPCI* bus scheme, if five or more PCI cards are plugged into the same PCI bus subsystem, at least two cards will share an interrupt line.

9 Real-Time Device Manager - Interrupt Configuration Tool

KUKA Roboter provides VxWin users with a versatile utility program called the ***Real-Time Device Manager*** **(RTDM)**. This useful Windows program can help you determine whether a particular PC is suitable for hosting VxWin.
It can display information regarding IRQ routing, expose and help resolve possible interrupt conflicts, and (logically) transfer PCI devices from Windows control to the real-time system and vice versa, automatically installing and uninstalling device drivers as required.
To learn more about this handy tool, see the ***Real-Time Device Manager*** **(RTDM)** and its companion documentation in ***..\VxWin\RTDM***.

10 Real-time Clocks and Time/Calendar functions

10.1 Introduction

In every conventional VxWorks system, the real-time operating system is driven by a steady stream of clock interrupts. These interrupts are used to apportion time for various real-time activities, including task-scheduling and program delays.
If VxWorks were to run alone on a PC platform, it would take the required clock interrupts from the onboard (hardware) Programmable Interval Timer (PIT), a

device that is always present in IBM-compatible PCs. Under VxWin, however, those clock-ticks are used for multiple purposes. Beyond providing ticks to the Windows system clock, they are also used to drive the VxWorks system and auxiliary clocks. The software-based calendar/clocks under both operating systems are also driven by these interrupts. Each VxWorks clock may be set by the user to interrupt at a rate between 10 and 10,000 ticks/second
When VxWin is started, it assumes control of the host computer's PIT timer and administers ticks to each of the various clock functions. While the PIT typically runs at 1 kHz in a non-VxWin system, under VxWin it is driven at 2 kHz or more.
The program functions that control the system and auxiliary clocks are described in the Wind River manual: 'VxWorks OS Libraries API Reference'.
In the course of normal VxWin operations, it is natural and unavoidable that the real-time system runs for brief periods of time with interrupts blocked. This implies that when a VxWin system is run continuously over a long period of time, Windows time tends to slowly drift behind the real-time operating system's time. For some applications, this doesn't matter, but for others it is important and sometimes essential, that Windows and VxWorks times and dates remain locked in step. For this reason, VxWin implements a flexible time-synchronization mechanism that is described in greater detail below: Chapter 10.5 - Time, date & time zones synchronization.

10.2 VxWin System Clock

The System Clock, serving as a time-base for VxWorks programmed delays and time-sliced (non-preemptive) scheduling, is also available for programmers to use in their own applications.
The five functions needed to control the system clock —*sysClkConnect(), sysClkDisable(), sysClkEnable(), sysClkRateGet(), sysClkRateSet()* — are available in a VxWorks OS library. For detailed descriptions of these functions, see Wind River documentation: 'VxWorks OS Libraries API Reference'.
Caution: Calling any of the three routines listed below while the real-time system is running affect the system clock by offsetting the timed interrupts by a single tick:

sysClkEnable(), sysClkDisable, sysClkRateSet()

Beyond that, each time new clock rates are calculated, other interrupts and task switching are briefly blocked.

10.3 VxWin Auxiliary Clock

The auxiliary clock, a secondary time base, may be used just like the system clock to time delays or events. The main advantage to the user is that this clock may be set a rate different from that of the system clock.
Before this feature can be used, you must activate it by defining the C-macro — ***AUXILIARY_CLOCK_EMULATION*** — in the BSP file *config.h* .
The five functions you need to control the auxiliary clock —*sysAuxClkConnect(), sysAuxClkDisable(), sysAuxClkEnable(), sysAuxClkRateGet(), sysAuxClkRateSet()* — are available in a VxWorks OS library. For programming descriptions of these functions, see Wind River documentation: 'VxWorks OS Libraries API Reference'.
Caution: Calling any of the three routines listed below while the real-time system is running influence the system clock by offsetting timed interrupts by a single tick:

sysAuxClkEnable(), sysAuxClkDisable, sysAuxClkRateSet().

Whenever new clock rates are calculated, other interrupts and task switching are briefly blocked.

10.4 ANSI Time Function

Every 10 seconds the VxWin system updates the VxWorks internal time and date with information obtained from the current Windows date and time.
The VxWorks "time" function delivers the current UTC date-time value. (UTC = Universal Time Coordinate).
For detailed information about UTC and VxWorks time/date functions, refer to Wind River manual “VxWorks OS Libraries API Reference”.
Example:

```
Windows-time: 20:30; Windows time zone: (GMT +01:00)
   Daylight Savings Time

               _

VxWorks-time = Windows time – TimeZone –
DaylightSavingsTime

VxWorks-time = 18:30
```

10.5 Time, date & time zone synchronization

Windows and VxWorks each maintain the current date, time of day, and time zone in their own software clocks, driven by a stream of fixed-timer interrupts. If Windows and VxWorks were started at identical times and nothing were done to keep the two synchronized, over a long period of continuous running their values would slowly drift apart.

This occurs because the real-time system's interrupts have the highest priority in the VxWin environment and when a real-time Interrupt Service Routine receives control, all interrupts in the system are momentarily suspended. This causes the Windows clock to occasionally miss a timer interrupt. Even though the physical timer (driving timer) runs at a brisk pace, the slow loss of ticks results in Windows time slowly drifting behind that of VxWorks.

While maintaining the same time in both operating systems is unimportant for some applications, it may be desirable or essential for others. For this reason, VxWin has implemented a mechanism to synchronize the two operating systems' clocks.

When taking advantage of this synchronizing mechanism in your system, you must decide whether Windows XP or VxWorks will serve as the clock-master. You must also consider how to set up the feature using configuration parameters and which VxWorks library functions your application will call.

VxWin's time-synchronization feature provides your application with a great deal of flexibility. You can:

- Run the two systems independently, i.e., without any time synchronization whatsoever.
- Synchronize the two software clocks (time, date and time zone) with Windows XP serving as the master.
- Synchronize the two software clocks (time, date and time zone) with VxWorks serving as the master.
- Start or stop the synchronization mechanism using program functions. See *RtosStartTimeSync* and *RtosStopClockSync*.
- Use a program function to dynamically specify which software clock should act as the master. See *RtosSetClockMaster*.

When you use this VxWin synchronization mechanism, the time and date of the two operating systems will remain locked in step, even over long periods of time.

—

Note: Upon VxWorks start-up, its clock/calendar is set to a date and time that lies decades in the distant past. If nothing is done to initialize this clock/calendar, this clock will begin to mark the passage of time starting from that point.

IMPORTANT: Although VxWin is shipped with the static configuration parameter *ClockSyncEnable* set to enabled (see *ClockSyncEnable*), the synchronization activity can only begin after your real-time application has called *RtosStartTimeSync* (see *RtosStartTimeSync*).

10.5.1 Controlling time-synchronization

To determine the behavior of the time-synchronism mechanism, the user must specify the values of a few parameters in the VxWin configuration file. Refer to the System configuration file (vxwin.config) in this manual.
Although the configuration parameters are summarized here, you should also read the descriptions in the configuration file chapter:

ClockSyncEnable
Enables or disables clock synchronization (default: 1 – enabled).
IMPORTANT: This parameter specifies the setting of a *guardian* bit. If set to *disabled*, the other parameters will not be evaluated.

RtosClockMaster
Determines whether Windows or VxWorks serves as time master (default: 0 - Windows is master).

InitializeRtosClock
Causes or suppresses a one-time initialization of the VxWorks clock (default: 0 - do not initialize the VxWorks clock upon startup).

ClockSyncInterval
Specifies the interval between clock updates
(default: 2 sec).

LegacyClockMasterMode
Specifies that time synchronization should be performed as was done in earlier VxWin versions. (default: 0 - don’t use).

If you specify Windows to be the master (*RtosClockMaster* = 0), the clock value from Windows is transferred into the VxWorks clock once during VxWin startup. Thereafter, the Windows clock values are automatically transferred into VxWorks at the end of each *ClockSyncInterval*.

If you specify VxWorks to be clock-master (*RtosClockMaster* = 1), you will have to consider how to initialize the VxWorks clock/calendar.

- If you set *InitializeRtosClock* to 0 (default), the clock value from Windows will not be transferred into the VxWorks clock during VxWin startup. The user's program is then responsible for setting the initial value of the VxWorks clock.

  If you plan to do this, you should be familiar with VxWorks clock functions: refer to Wind River documentation 'VxWorks OS Libraries API Reference' (clockLib: clock_settime, etc).

- At the end of each ***ClockSyncInterval*** , the VxWorks clock time will automatically be transferred into Windows clock.

- If you set ***InitializeRtosClock*** to 1, however, the Windows clock value will be transferred into the VxWorks clock once, during VxWin initialization.

Controlling Time-synchronization - Dynamically

Those who use time synchronization should also be aware of the following three functions. To be called from the VxWorks context, these functions, enable you to dynamically start or stop the synchronization and, if desired, change which operating system serves as the time master.

RtosStartTimeSync	–	Starts time-synchronization.
RtosStopTimeSync	–	Stops time-synchronization.
RtosSetClockMaster	–	Determines which clock serves as the time master.

For programming details see Time synchronizing functions.

10.6 Time zone designations

Windows designations for time zones are expressed as lists of well-known place-names along with a numeric indication of their temporal displacement, in whole hours, from GMT (Greenwich Mean Time).
Most of the time zones listed internally in Windows, however, are tagged with multiple place-names. This means that a number of different names may be associated with the same time zone. The following names, for example, all express just one time zone, namely: GMT -07:00.

- Arizona

- Chihuahua, La Paz, Mazatlan
- Mountain Time (US & Canada)

While considering the foregoing discussion, you should be aware that if you use VxWorks as the time-master in your system, when a *time zone* change must be made (a rare occurrence), the synchronizing task will choose the first place-name from the Windows list to set into the data structure on the Windows side. In the above example, this means that Arizona would be chosen.
If a Windows user doesn't like the place-name that appears in the time zone designation, he/she can change it programmatically to another name, provided it is in the same time zone. In the above example, the Windows user could change the place-name from Arizona to Chihuahua, La Paz, Mazatlan or Mountain Time (US & Canada).

10.7 Time synchronizing functions

Those who wish to use time synchronization should be aware of the following three functions. These functions enable you to start or stop the synchronizing task or to change which system serves as the time master.
Note: These functions may only be called from the VxWorks context.

RtosStartTimeSync

This function starts the time-synchronization mechanism. It causes a real-time task to supervise the two clocks.

```
void      RtosStartTimeSync
              (
              Int    nPriority
              );
```

PARAMETERS
nPriority
[in] Specifies the priority for this synchronization task.
RETURN VALUE
N/A
REMARK
–
PROGRAM DETAIL
VxWorks header: rtosLib.h

EXAMPLE

```
RtosStartTimeSync( 254 );
```

RtosStopTimeSync

This function stops the time and time zone synchronization task.

```
void    RtosStopTimeSync    (void);
```

RETURN VALUE

N/A

PROGRAM DETAIL

VxWorks header: rtosLib.h

EXAMPLE

```
RtosStopTimeSync();
```

RtosSetClockMaster

This function, which may be invoked any time VxWin is running, changes which of the two clocks serves as the time master.

```
void    RtosSetClockMaster
            (
            BOOL    bRtosClkMaster
            );
```

PARAMETERS

bRtosClkMaster

[in] - If **TRUE,** the real-time operating system is master;
if **FALSE**, Windows provides the master-clock.

RETURN VALUE

N/A

PROGRAM DETAIL

VxWorks header: rtosLib.h

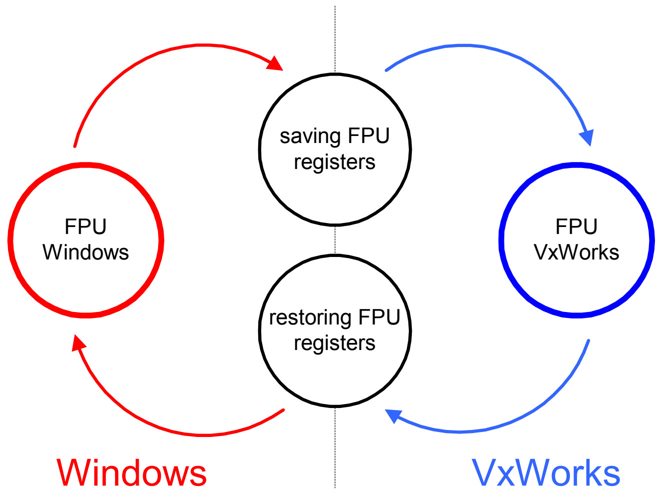
EXAMPLE

```
RtosSetClockMaster( TRUE );
```

11 VxWorks Floating-Point

11.1 Using the Arithmetic Coprocessor

Under VxWin you can use the arithmetic coprocessor without limitation.
To preserve the data-integrity of a Windows program, it would be sufficient to save the internal state of the coprocessor whenever a VxWorks interrupt occurs and restore it when control is returned to Windows.
Under VxWin, however, in order to conserve CPU cycles, the Interrupt Service Routine (ISR) does not save the coprocessor state every time it is invoked. Instead, its state will be saved and restored only as required.
More precisely, the ISR sets a control bit in the coprocessor that causes a Trap Handler to be activated the first time a VxWorks task issues a floating-point command. The Trap Handler saves the coprocessor's internal state. Then, before control is returned to Windows, VxWin restores the previously saved coprocessor state.



11.2 Floating-Point Exceptions

Although VxWorks supports floating-point, the user should not use it without making provision for handling floating-point exceptions.
Without this precaution, a task that makes an erroneous floating-point calculation – dividing by zero, for example – would continue to execute without knowledge

of the error and would inevitably use incorrect data for subsequent processing steps.
In many real-world applications, such as machine control, this could lead to inappropriate, expensive or even dangerous actions.
Under VxWin, provision has been made for the user to trap FP-Exceptions, so that if such an event occurs, the real-time application has an opportunity to take steps to recover.
The following two functions are available in the System Library.

- ***sysFPExceptHandler***
- ***TaskFPExceptionAdd***

For descriptions of these functions, refer to System Library Functions (sysLib.c, sysALib.s) elsewhere in this manual.
If a floating-point exception handler has been implemented, the occurrence of such an exception[5] will immediately stop the offending task.
The user's exception handler then runs (in the offending task's context), thus providing an opportunity to take whatever actions the user deems necessary to manage the event. One could, for example, initiate an alarm, move machinery into a safe position, ask for guidance at the user's console, and so on.
The following code illustrates how support for floating-point exceptions may be implemented:

An Exception Handler that processes floating-point errors might look like this:

```
/***************************************************
* Declare an Exception Handler (will run in the faulting task's context).
***************************************************/
void ExceptHandler  /* Return:  N/A */
    (int    nTid
    ,int    nVecNum
    ,ESF0*  pESF
    )
{
    switch (nVecNum)
    {
        case (IN_DIVIDE_ERROR):
        {
            logMessage ("division by zero\n", 0,0,0,0,0,0);
            break ;
        }
        case (IN_CP_ERROR):
        {
            /* call floating point exception handler */
            sysFPExceptHandler ();
            break ;
        } /* case (IN_CP_ERROR) */
```

[5] The user may select the specific exceptions to which the exception-handler will respond.

```
            default:
            {
                logMessage ("exception #%d occurred\n", nVecNum, 0,0,0,0,0);
                break ;
            }
        }
    }
          __
```

Install the exception handler

```
    /* install the exception hook */
        excHookAdd ((FUNCPTR) ExceptHandler);

          __
    The following code illustrates the enabling of specific exceptions
    and then forces an fp-exception to occur:

    /*********************************************
    * This task will cause an FP exception.
    *********************************************/

    void fppExcTask (void)
    {
    volatile double dblVal0;
    volatile double dblVal1;
    volatile double dblVal2;

          __
    /* enable fp exceptions underflow and zero divide*/

        sysTaskFPExceptionAdd (0, FP_EXC_ZERO_DIVIDE |
    FP_EXC_UNDERFLOW);

          __
    /* force underflow */
        dblVal0 = 5.0e-308;
        dblVal1 = 1.0e100;

    /* this calculation forces an exception */
        dblVal2 = dblVal0/dblVal1;}
          __
```

12 Blue Screen Error Management

A program running under Windows might arrive at a point in its execution where it can neither continue executing nor recover from an erroneous state. If this should happen, it may take advantage of a Windows facility and issue an emergency system call. Such an emergency call causes Windows to terminate the application and display an error message over a blue background on the PC monitor.
Most PC-users have, at one time or another, seen this kind of error message. Because of the background color over which the error message appears, it is often referred to as 'the blue screen of death' or BSOD.
If a PC were not involved in real-time processing when this kind of event occurred, a user would probably just reboot the system.
Under VxWin, though, a real-time application might indeed be running, controlling or monitoring critical processes, when such an error occurs. Depending on what the real-time application was doing at the time, stopping the entire system with a sudden reboot might be inconvenient, dangerous or expensive.
Note: Under VxWin, even if a blue screen error is being displayed on the PC monitor, the real-time system will continue running and can continue (assuming no hardware problems) without limitation.
It is desirable, however, that a user's real-time application be informed if a bluescreen error has occurred. To this end, VxWin provides a facility to do just that.
To use this mechanism, the user initially calls the *bsodSetMode* function, which returns a semaphore. The user then provides a task to wait for the semaphore. Should that semaphore be set, it is then known that a bluescreen has occurred. The task that waited for that semaphore can then undertake appropriate action. Note that Windows operations remain suspended until the task calls the *bsodExit* function, which then allows Windows execution to proceed.
For a detailed description of the two functions that are used to control the blue screen error function, see Blue Screen Errors Module under System Library Functions (sysLib.c, sysALib.s) .

13 Multiprocessor/Multicore support

A multiprocessor/multicore system is a personal computer system with more than one processor core. Currently there are several kinds of multiprocessor-based systems available in the marketplace:

- Systems with more than one physical CPU, such as Intel® XEON™ or AMD® Opteron™;
- Systems that support Hyper-Threading Technology;
- Multi-Core processor-based systems (e.g. Intel Core2 Duo, Core2 Quad, …).

VxWin may operate in two different modes on a multiprocessor system.
In cases where Windows shall get as much CPU power as possible VxWin should be set to operate in shared mode. In this mode VxWorks will share one dedicated CPU core with Windows and Windows will still be able to run on this CPU (whenever VxWorks enters its idle state).
Alternatively, one CPU core can be used exclusively by VxWorks (VxWin running in exclusive mode). When running in exclusive mode the need to switch between Windows and VxWorks on this CPU core is eliminated. This will lead to much shorter interrupt and task latencies.
More information about principles of operation can be found in chapter 3.

13.1.1 Shared mode

When VxWin is operating in shared mode on a multi core system, VxWorks runs on just one processor core while Windows XP runs on all. You can specify which processor core VxWorks should use by setting the *RtosProcessor* parameter in the main VxWin configuration file vxwin.config.

Caution: Just as in a single-core system, Windows XP can only run correctly in such a environment if it receives sufficient processor time on all CPUs. Should your real-time application cause the VxWorks context to monopolize its CPU, Windows XP will probably not run effectively.

13.1.2 Exclusive mode

When VxWin is operating in exclusive mode on a multi core system, VxWorks usually runs on the last processor core while Windows XP runs on all other. You can specify which processor core VxWorks should use by setting the *RtosProcessor* parameter in the main VxWin configuration file vxwin.config.
To block Windows XP from using the same processor core the number of cores which Windows XP will use has to be limited.
This can be achieved by adding the appropriate setting for the /NUMPROC parameter in the Windows XP boot.ini file (usually located in C:\).
Note: after changing the boot.ini file you have to reboot your system so that the changes become effective.

Example boot.ini file when Windows XP is allowed to use all processor cores
(VxWin is running in shared mode):
```
[boot loader]
default=multi(0)disk(0)rdisk(0)partition(0)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(0)\WINDOWS="WinXP" /noexecute=alwaysoff
```
Example boot.ini file when Windows XP is allowed to use one processor core
(VxWin may run in exlusive mode on a dual-core system if RtosProcessor is set to 1):
```
[boot loader]
default=multi(0)disk(0)rdisk(0)partition(0)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(0)\WINDOWS="WinXP" /noexecute=alwaysoff /numproc=1
```
Example boot.ini file when Windows XP is allowed to use three processor cores
(VxWin may run in exlusive mode on a quad-core system if RtosProcessor is set to 3):
```
[boot loader]
default=multi(0)disk(0)rdisk(0)partition(0)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(0)\WINDOWS="WinXP" /noexecute=alwaysoff /numproc=3
```

Example configuration settings on different systems:

Number of CPU cores	boot.ini setting	vxwin.config setting
1	-	"RtosProcessor"=dword:0
2	/NUMPROC=1	"RtosProcessor"=dword:1
4	/NUMPROC=3	"RtosProcessor"=dword:3
8	/NUMPROC=7	"RtosProcessor"=dword:7

14 Forcing a context change — VxWorks to Windows

This section is only relevant if VxWorks is running in shared mode.

During VxWin startup, a VxWorks task is created to serve as the idle task for the real-time side of the system. This task is assigned the lowest priority in the system.

When there are no outstanding real-time interrupts to process, the idle task will run and subsequently invoke the *RtosIdle* function (RtosIdle Function). This function forces the execution context to be switched from VxWorks to Windows. Thereafter, whenever a real-time interrupt occurs, the execution context will be switched back to VxWorks.

If you wish to do so, you may program your own idle task. In this case, you must make certain that you suspend or delete the VxWin-supplied *RtosIdle* task and substitute your own in its place.

Why might you write your own idle task? Because it is not always desirable to call *RtosIdle* from as soon as possible. Only a user is familiar with and can anticipate the specific needs of his/her own application. And for certain applications or situations it may sometimes be to your advantage to call *RtosIdle* only after a programmed delay, thus extending the time spent in the real-time context. Whether you decide to provide your own idle routine depends mainly on your application's timing requirements.

For example, to accommodate a critically timed event, a user might decide to stay in the real-time context until that event has occurred. Or knowing that a series of interrupt driven events will rapidly follow one another, one might decide to extend the stay in the real-time context to prevent thrashing back and forth between the two contexts.

For a detailed description of the function call and to see an example of a custom-written idle task, see the RtosIdle function.

15 Limitations under VxWin

Running two operating systems on one hardware platform inevitably imposes some restrictions. Even though one could theoretically reassign any Windows device to the real-time operating system, it would make little sense to reassign devices that Windows needs for its essential operations.
Take the hard disk, for example. Theoretically one could assign this device to VxWorks, but doing so would make little sense, as Windows depends it for normal operation.
Is there nonetheless a way to access the hard disk from the real-time context?
Yes! Alternative methods by which your real-time application can read and write the hard disk are described elsewhere in this manual. For more information, see: Accessing the PC's hard disk.
But because of the limitation discussed above, certain drivers and features that depend on those devices cannot be supported under VxWin/VxWorks. The following libraries and drivers are not supported:

BootLib	Support for Boot-ROM
RebootLib	Support for system reboot**
SmXxxLib	Shared memory support libraries and drivers (including VxMP)

****Note**: Because the *RebootLib* is not yet supported by VxWin, the *reboot* function currently causes the entire system to reboot. In the future, with help from the VxWin Uploader Utility, the *reboot* function will reboot only the VxWorks image.

16 VxWorks System overload

It is possible to overwhelm a real-time system with so many interrupts that it can not dispose of the resulting workload. This causes work-queues to become full so that they cannot accept additional entries. For most real-time applications this would lead to an overall system malfunction.
Note: This is not a limitation of VxWin. System overload is a condition that can arise in any real-time system.
Therefore, it is important that real-time system designers plan their systems carefully, taking into account the worst-case demands that will be placed on it, so that this kind of error does not occur.
When a native VxWorks detects an internal work-queue overrun, it typically re-boots itself. After which, the BSP routine ***sysToMonitor*** is called. If we were dealing with a VME system, for example, under these circumstances, the ROM-monitor would be called.
But a VxWin system has no ROM-monitor. In this case, a call to ***sysToMonitor*** causes all interrupts to be deactivated and the rom-monitor routine enters an endless loop. From inside that loop, distinctive acoustic signals are generated, repetitive sounds that can be heard at the PC’s own loudspeaker. When you hear such tones, it probably means that the real-time system has become overloaded.
While ***sysToMonitor*** can be used to help developers determine the cause of a system overload, once the system gets into this condition, there is no practical means for returning to normal operation.

17 Program Memory Layout

17.1 VxWin memory layout

The following information should provide you with everything you need to know about memory partitioning. Recall that any change to memory settings becomes effective only after the system has been rebooted.

—

Adding or deleting memory and/or peripheral devices to your system changes the physical memory layout. Whenever you start VxWin, the Uploader Utility checks to see if the physical memory configuration has been changed. If this is the case, the Uploader will inform you of the changes and will also remind you to reboot the entire system. If, under these circumstances, you fail to reboot your system, it may subsequently not work correctly and may not work at all.

Like most real-time systems, VxWorks and its applications use substantially less memory than Windows and its applications. Therefore, when you plan your memory usage, you will typically want to allocate 32 megabytes of RAM for VxWorks, leaving the rest for Windows.

While the minimal memory requirement to run the delivered prebuilt VxWorks image for VxWin is approximately 5 MB, at least 96 MB of RAM must remain available for the successful operation of Windows.

The first memory address available to the real-time operating system is specified by the configuration parameter RtosMemoryStartAddress; its extent is given by RtosTotalMemorySize. Both parameters are located in the System configuration file (vxwin.config). While most systems should run satisfactorily with the default values, the user may nonetheless wish to adjust them.

Cautions: If you adjust the ***RtosMemoryStartAddress***, you can no longer use the BSP as delivered in the product release and will have to rebuild it to make it compatible with your new memory start address. Should you decide to change ***RtosTotalMemorySize***, please keep in mind that the value must be expressed in hexadecimal as a multiple of **4096 (0x1000)**; a value of 0x10000FFF, for example, would not be permitted.

The following table depicts the PC's physical memory layout under VxWin:

		PCI / USB (11)
		BIOS (10)
Windows (1b)		
		Memory used by Windows
VxWorks (2)		Configuration Area (9)
		Internal Shared Memory (8)
		User Shared Memory (7)
		System Memory Pool
		Interrupt Stack
		WDB Memory Pool
	RtosMemoryStartAddress + 0x8000	System Image (6)
	RtosMemoryStartAddress + 0x5000	Initial Stack
	RtosMemoryStartAddress + 0x2000	FD DMA Area
	RtosMemoryStartAddress + 0x1300	Exception Message Area (5)
	RtosMemoryStartAddress + 0x1200	VxWorks / Boot Line (4)
	RtosMemoryStartAddress + 0x1100	
	RtosMemoryStartAddress + 0x800	GDT (Global Descriptor Table

	RtosMemoryStartAddress + 0x0	IDT (Interrupt Vector Table)
Windows (1a)		
		Virtual Network (3)

The enumerated points in the table are explained as follows:

(1a & 1b) Memory used by Windows

(2) Memory used by VxWorks

(3) An area of memory allocated to the NDIS Driver for data exchange. This area will be accessed by two Virtual Network Drivers, one running under Windows and the other under VxWorks.

(4) The boot line stored here is the one that VxWorks physically uses to characterize itself upon initialization. Before this boot line is used, the Loader Utility will (optionally) modify it based on user-specified information in the VxWin configuration file. See Dynamic boot line in System configuration file (vxwin.config) and Prototype Boot Line for VxWorks.

(5) Exception Message Area

(6) The VxWorks OS image, together with the VxWin BSP, is loaded here.

(7) With the help of system functions provided for that purpose, programs under either operating system can access the User Shared Memory space. See Using Shared Memory. You can also specify the extent of the User Shared Memory; see IntlShmSize in System configuration file (vxwin.config)

(8) The Internal Shared Memory is not available for VxWin users. It is used only by internal system functions.

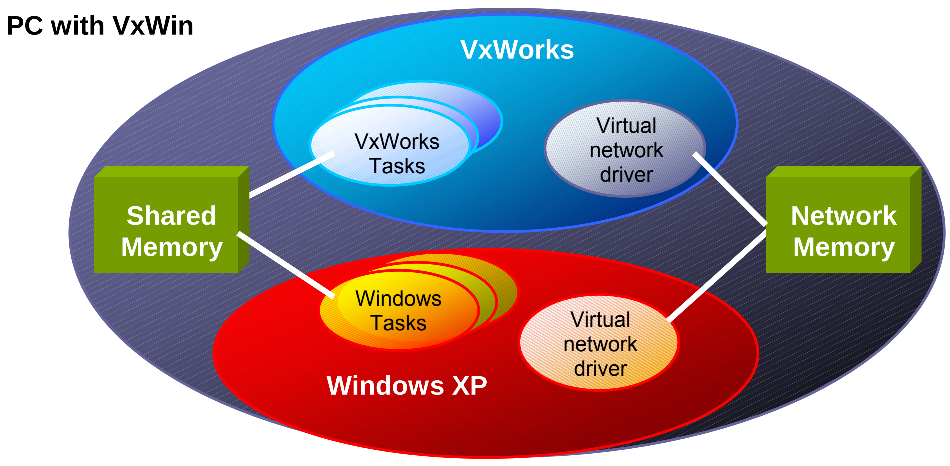
(9) During VxWin start-up, the user's configuration file is transferred (as ASCII text) into the Configuration Area. When VxWin system programs require configuration information, they extract it from this area, not directly from the configuration file. See System configuration file (vxwin.config).

(10) Certain BIOS information resides in this RAM area.

(11) Windows automatically assigns this range of addresses to PCI and USB devices. VxWin-users should not attempt to use it.

17.2 Shared Memory

One of the ways that real-time and background processes can communicate with one another with a minimal expenditure of processing power is by exchanging information via shared memory areas.
Shared memory is nothing more than a block of ordinary RAM that has been designated for use as Shared Memory. The shared memory extent will be set by the user at the time VxWin is installed. See Shared memory size in System configuration file (vxwin.config).
Neither Windows nor VxWorks are aware of the existence of the shared memory; only users' application programs know how to gain access to it.
Gaining access to shared memory is accomplished via VxWin functions that have been provided for that purpose. (Refer to Using Shared Memory.) Using these functions, applications acquire blocks of shared memory (address and extent) which can then be freely used. Your programs will address shared memory via dynamically acquired pointers.
While all tasks running under VxWorks may freely access shared memory, under Windows, only one application at a time can acquire a pointer into the shared memory.
Important: Even though VxWin provides applications with access to the shared memory area, responsibility for synchronizing the use of the data remains with the applications. To ensure data integrity, you may wish to use techniques such as: double-buffering, user-owner flags, and/or shared-events.
The following diagram illustrates the relationship of the two operating systems to shared memory.



17.3 Specifying the Extent of Shared Memory

The user can specify the size of the two shared memory regions by setting corresponding parameters in the VxWin configuration file ***(vxwin.config)***.
See System configuration file (vxwin.config).
Under the [***Upload***] key in the configuration file:

- You can specify the size of the user shared memory region with the parameter ***UserShmSize*** (default 100 kilobytes). See User Shared Memory Size.
- And you can also specify the size of the internal shared memory region with the parameter ***InternalShmSize*** (default 100 kilobytes). See Internal Shared Memory Size.

Note 1: Memory sizes in the preceding points must be entered as multiples of 4096 (0x1000).
Note 2: Any change to memory settings becomes effective only after the system has been rebooted.

18 Inter-System Communication

Under the VxWin environment, each of the supported operating systems are standard, unmodified version as released by their respective manufacturers.
In accord with a fundamental principle of VxWin, each operating system runs without any awareness of or compensation for, the other's existence. This means that applications destined to run under the one or other system, will be programmed in the usual fashion for that system, without regard for the presence of the other system.
While strict separation between the two operating systems and their applications is desirable, in order not to limit the advantage of using a dual operating system environment, application programs under one system must nonetheless be able to communicate with applications under the other system. To clarify this, consider the following example:
A customer wishes to implement console functions under Windows. It is to accept keyboard entries and display data in behalf of the real-time system. To implement this kind of functionality, clearly data must be passed across the logical boundary between the systems.
VxWin has implemented three basic mechanisms for inter-system communications and/or synchronization. The following sections describe these methods.

- Shared Memory
- Shared Events
- Virtual Network

Since the prototypes for user-functions implemented in the ***UploadRTOS.DLL*** library are located in ***rtosdrvif.h***, to use shared memory or shared event functions in a Windows application, the user should include the file ***rtosdrvif.h*** and link to the library ***UploadRTOS.lib***.
Note:While function names for each side of the system are to a degree symmetrical, please note that differences do exist.

Important: Even while the hard disk formally remains under the control of Windows, there are additional methods for VxWorks users to access (read/write) files on the hard disk. Refer to Accessing the PC's hard disk.

18.1 Windows XP / The RTOS Library (Uploader Library)

The VxWin RTOS Library contains the files you need to access the VxWin features that run under Windows.
Important: Only one Windows process may use the RTOS Library. It may not be shared between two or more processes.
You should be aware of the following relevant files which you will need to use the RTOS library:

rtosdrvif.h	Header file – To be included in a user's source files.
UploadRTOS.lib	Import library for the Uploader DLL - To be included in a user's project linker settings.
UploadRTOS.dll	RTOS Uploader library DLL - This DLL is located in the Uploader directory: C:\Program Files\VxWin.
RTOSdrv.dll	An additional DLL needed by the RTOS Uploader library DLL. This is also in the Uploader directory: C:\Program Files\VxWin

18.2 VxWorks caveats

Prior to using the event API (and also the BSOD API) a VxWorks application has to assure the communication subsystem is available. This can be accomplished by calling function RtosCommIsStarted().
Note: this function is new for VxWin version 3.5.
—

RtosCommIsStarted

This function can be used to wait until the communication subsystem is ready..

```
BOOL    RtosCommIsStarted
            (
            VOID
            );
```

PARAMETER

RETURN VALUE

TRUE if the communication subsystem is started, FALSE if not.

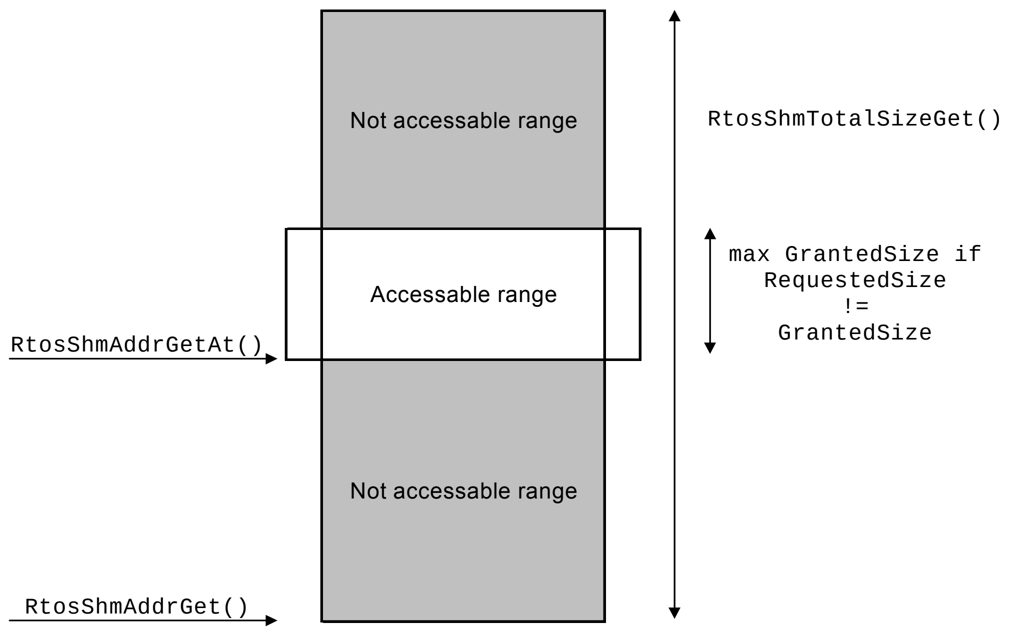
18.3 Using Shared Memory

To use Shared Memory, the following steps must be performed:

- Windows: Using the Uploader library function ***RtosShmAddrGet***, an application acquires a pointer to a block of Shared Memory.

- VxWorks: Using the VxWin library function ***RtosShmAddrGet***, a task acquires a pointer to a block of Shared Memory.

Note: While the size of the Shared Memory is limited only by the physical size of RAM, Windows has internal limitations that restrict the amount of shared memory that can be accessed at one time. Therefore, if you wish to take full advantage of a physically large shared memory, you will have to use windowing techniques to do so:



- Use the ***RtosShmTotalSizeGet*** function to acquire the size of the shared memory range.
- After setting the input parameter ***RequestedSize*** to the value returned by ***RtosShmTotalSizeGet***, call the ***RtosShmAddrGet*** function.
- If the returned parameter ***GrantedSize*** is not equal to ***RequestedSize***, then ***GrantedSize*** represents the largest shared memory window you can access at that time.
- Through successive calls to ***RtosShmAddrGetAt***, however, you can acquire the entire shared memory.

Note 1: Once an ***RtosShmAddrGet*** or ***RtosShmAddrGetAt*** function has been used, subsequent use of the same function will invalidate a previously returned address.
Note 2: VxWorks and Windows applications address *virtual* memory space. While multiple calls to these functions might return numerically identical virtual addresses, such addresses, acquired at different times, will very likely point to different data areas.

18.3.1 Shared Memory under Windows

The following three functions enable a process running under Windows to utilize the Shared Memory….

—

RtosShmAddrGet

This function causes a user-specified number of bytes of shared memory (block 0) to be allocated for use by the calling routine.

```
PBYTE     RtosShmAddrGet
              (
              DWORD      dwRequestedSize,
              DWORD*     pdwGrantedSize
              );
```

PARAMETER

dwRequestedSize

[in] - The size of the requested memory block (in bytes).

pdwGrantedSize

[out] - After the function call, will contain the actual size of the granted memory (in bytes)

RETURN VALUE

If the function has executed successfully, the return value will be the starting-address of the allocated memory-block. If the function fails, NULL will be returned.

PROGRAM DETAIL

Windows header:	rtosdrvif.h
Windows code:	UploadRTOS.dll

EXAMPLE

```
PBYTE pSMAddress;
DWORD dwRequestedSize = 32*1024*1024; //32 MByte
DWORD dwGrantedSize;

pSMAddress = RtosShmAddrGet( dwRequestedSize, &dwGrantedSize );
if (pSMAddress != NULL)
   return OK;
```

RtosShmAddrGetAt

This function is used to acquire a block of shared memory starting at a user-specified offset into the shared memory. It may be called multiple times with different offsets. When an offset of zero (0) is specified, this function yields the same result as the ***RtosShmAddrGet*** function.

```
RtosShmAddrGetAt( 0, dwRequestedSize, &dwGrantedSize )
```

is equivalent to...

```
RtosShmAddrGet( dwRequestedSize, &dwGrantedSize ).
```

```
PBYTE    RtosShmAddrGetAt
           (
           DWORD    dwOffset,
           DWORD    dwRequestedSize,
           DWORD*   pdwGrantedSize
           );
```

PARAMETER

DwOffset

[in] - The absolute offset of the requested block (in bytes).

dwRequestedSize

[in] - The requested memory-block size (in Bytes).

pdwGrantedSize

[out] - Will be set to the actual size of the allocated memory (in bytes).

RETURN VALUE

When the function is successful, the starting address of the requested memory will be returned. If there is an ERROR, the returned address will be NULL.

PROGRAM DETAIL

Windows header: rtosdrvif.h

Windows code: UploadRTOS.dll

EXAMPLE

```
PBYTE pSMAddress;
DWORD dwRequestedSize = 16*1024*1024; //16 MByte
DWORD dwGrantedSize;
DWORD dwOffset = 16*1024*1024; //16 Mbyte - Offset

pSMAddress = RtosShmAddrGetAt( dwOffset, dwRequestedSize, &dwGrantedSize );
if ((pSMAddress != NULL) && (dwGrantedSize == dwRequestedSize))
   return OK;
```

RtosShmTotalSizeGet

This function acquires the total size of the shared memory area.

```
DWORD    RtosShmTotalSizeGet    ( void );
```

RETURN VALUE

If the function was successful, the size of the entire User Shared Memory (in bytes) will be returned. If there was an error, for example, the Shared Memory had not been initialized, 0 (zero) will be returned.

PROGRAM DETAIL

Windows header: rtosdrvif.h

Windows code: UploadRTOS.dll

EXAMPLE

```
DWORD dwSHMSize;

dwSHMSize = RtosShmTotalSizeGet();
```

```
if (dwSHMSize != 0)
   return OK;
```

RtosInterlockedCompareExchange

The ***RtosInterlockedCompareExchange*** function performs an atomic comparison of the specified values and, based on the outcome of the comparison, exchanges the values. The function prevents a conflict from occurring when a real-time system task and a Windows application want to simultaneously access the same variable in shared memory.

```
LPLONG    RtosInterlockedCompareExchange
            (
            LPLONG volatile    plDestination,
            LONG               lExchange,
            LONG               lComperand
            );
```

PARAMETERS

plDestination

[in/out] - Specifies the address of the destination value.

lExchange

[in] - Specifies the exchange value.

lComperand

[in] - Specifies the value to compare to Destination.

RETURN VALUE

The return value is the initial value of the destination. If the *plDestination* address is invalid, however, the function returns 0xABABABAB.

REMARKS

Referenced variables must be aligned on a 32-bit boundary.

After performing an atomic comparison of the Destination value with the Comperand value, the Exchange value will be stored in the address specified by Destination, but only if the Destination value is equal to the Comperand value. Otherwise, no operation is performed.

PROGRAM DETAIL

Windows header:	rtosdrvif.h
Windows code:	UploadRTOS.dll

EXAMPLE

```
LONG  lVirtAddr = 0;
int    timeout = 1000;
BOOL  bTimeOut  = FALSE;

// map physical address range into lVirtAddr
// …

// waiting for counterpart
while( 0 != RtosInterlockedCompareExchange( lDestAddr, 1, 0 ) )
{
```

```
            timeout--;
            if( 0 == timeout )
            {
                bTimeOut = TRUE;
                break;
            }
            Sleep(1);
        }
```

18.3.2 Shared Memory under VxWorks

The following three functions enable a process running under VxWorks to utilize the Shared Memory.

—

RtosShmAddrGet

This function causes a user-specified number of bytes of shared memory (block 0) to be allocated for use by the calling routine.

```
char*   RtosShmAddrGet
          (
          long unsigned int      dwRequestedSize,
          long unsigned int*     pdwGrantedSize
          );
```

PARAMETER

dwRequestedSize

[in - The size of the requested memory block (in bytes).

pdwGrantedSize

[out] - After the function call, will contain the actual size of the granted memory (in bytes)

RETURN VALUE

If the function has executed successfully, the returned value will be the starting-address of the allocated memory-block. If the function fails, NULL will be returned.

PROGRAM DETAIL

VxWorks header: rtosLib.h

EXAMPLE

```
char* pSMAddress;
long unsigned int dwRequestedSize = 32*1024*1024; //32 MByte
long unsigned int dwGrantedSize;
```

```
pSMAddress = RtosShmAddrGet( dwRequestedSize, &dwGrantedSize );
if (pSMAddress != NULL)
   return OK;
```

RtosShmAddrGetAt

This function is used to acquire a block of shared memory starting at a user-specified offset into the shared memory. It may be called multiple times with different offsets. When an offset of zero (0) is specified, this function is exactly the same as the RtosShmAddrGet function.

```
RtosShmAddrGetAt( 0, dwRequestedSize, &dwGrantedSize )
```

is equivalent to...

```
RtosShmAddrGet( dwRequestedSize, &dwGrantedSize ).
```

This function requests a block of shared memory to be allocated. The address of the allocated memory will begin at the user-specified offset.

```
char*    RtosShmAddrGetAt
          (
          long unsigned int     dwOffset,
          long unsigned int     dwRequestedSize,
          long unsigned int*     pdwGrantedSize
          );
```

PARAMETER

dwOffset

[in] - The absolute offset-value of the requested memory-block – in bytes.

dwRequestedSize

[in] - The size of the requested memory-block – in bytes.

pdwGrantedSize

[out] -Contains the actual size of the allocated memory-block – in bytes

RETURN VALUE

When the function executes successfully, the address of the requested memory-block will be returned. In case there is an error, the returned value will be NULL.

PROGRAM DETAIL

VxWorks header:	rtosLib.h

EXAMPLE

```
char* pSMAddress;
long unsigned int dwRequestedSize = 16*1024*1024; //16 MByte
long unsigned int dwGrantedSize;
long unsigned int dwOffset = 16*1024*1024; //16 Mbyte - Offset

pSMAddress = RtosShmAddrGetAt( dwOffset, dwRequestedSize, &dwGrantedSize );
if ((pSMAddress != NULL) && (dwGrantedSize == dwRequestedSize))
   return OK;
```

RtosShmTotalSizeGet

This function acquires the total size of the shared memory area.

```
long unsigned int     RtosShmTotalSizeGet ( void );
```

RETURN VALUE

If the function was successful, the size of the entire User Shared Memory (in bytes) will be returned. If there was an error, for example, the Shared Memory had not been initialized, 0 (zero) will be returned.

PROGRAM DETAIL

VxWorks header:	rtosLib.h

EXAMPLE

```
long unsigned integer dwSHMSize;

dwSHMSize = RtosShmTotalSizeGet();
if (dwSHMSize != 0)
   return OK;
```

RtosInterlockedCompareExchange

The ***RtosInterlockedCompareExchange*** function performs an atomic comparison of the specified values and, based on the outcome of the comparison, exchanges the values. The function prevents a conflict from occurring when a real-time system task and a Windows application both want simultaneously to access the same variable in shared memory.

```
long     RtosInterlockedCompareExchange
           (
           long* volatile     plDestination,
           long               lExchange,
           long               lComperand
           );
```

PARAMETERS

plDestination

[in/out] - Specifies the address of the destination value.

lExchange

[in] - Specifies the exchange value.

lComperand

[in] - Specifies the value to compare to Destination.

RETURN VALUE

The return value is the initial value of the destination. If the *plDestination* address is invalid, however, the function returns 0xABABABAB.

REMARKS

Referenced variables must be aligned on a 32-bit boundary.
After performing an atomic comparison of the Destination value with the Comperand value, the Exchange value will be stored in the address specified by Destination only if the Destination value is equal to the Comperand value. Otherwise, no operation is performed.

PROGRAM DETAIL

VxWorks header: rtosLib.h

EXAMPLE

```
LONG  lVirtAddr = 0;
int    timeout = 1000;
BOOL  bTimeOut  = FALSE;

// map physical address range into lVirtAddr
// …

// waiting for counterpart
while( 0 != RtosInterlockedCompareExchange( lDestAddr, 1, 0 ) )
{
    timeout--;
    if( 0 == timeout )
    {
        bTimeOut = TRUE;
        break;
    }
    Sleep(1);
}
```

18.4 Shared Events

Shared Events is a VxWin mechanism that allows an application (task or process), running under one of the VxWin supported operating systems to asynchronously signal an application running under the other operating system. Bi-directional signaling is supported.
While Shared Events can be used for many purposes, they are typically used to synchronize operations between users' Windows and VxWorks applications.
Example: An application running under VxWorks acquires and writes a set of data to shared memory. If a Windows program is to display that data, using ***RtosWaitForSingleObject***, it could wait to be notified when the data in a mutually agreed upon shared memory area are valid. Similarly, after displaying the data on the PC's monitor (or processing it in some other way), using ***RtosSetEvent***, the Windows program could signal a task under VxWorks when the display was made and that the data area area can then be overwritten.
Example: Another possible use for the shared events mechanism could simply be to inform a process running under the other operating system that some sort of

simple event has occurred — that a mechanical switch was set or that a gate has closed.
Caution: While the Shared Events API for VxWorks is very similar to that for Windows, they are not entirely identical.
In each operating system, an application must create or open the same named event. The functions ***RtosCreateEvent*** and ***RtosOpenEvent*** serve this purpose in the Windows context. Similarly named functions are used in the VxWorks context.
Using ***RtosWaitForSingleObject*** in the Windows context or ***RtosWaitForEvent*** function in the VxWorks context, an application can wait for the event to occur. Meanwhile, on the opposite operating system, an application that wishes to signal the event will eventually call ***RtosSetEvent*** (same function name for both systems). When the signal arrives on the other side, the waiting application will be reactivated and utlimately permitted to run.
When a shared event is no longer needed, each application should close it by calling ***RtosCloseHandle*** (under Windows) and ***RtosCloseEvent*** (under VxWorks).

18.4.1 Restrictions for Shared Events

- Under VxWin, only auto-reset events may be used.
- Prior to using the event API it has to be assured that the communication subsystems is already started. This can be accomplished by calling function ***RtosCommIsStarted*** .
- An application in each operating system must create or open the same named event. The functions ***RtosCreateEvent*** and ***RtosOpenEvent*** serve this purpose under both Windows and VxWorks.
- Shared events only work across the boundary separating the two operating systems. Setting an event under Windows, is supposed to wake a task under VxWorks, and vice versa. If you attempt to wake up a program/task in the same operating system context in which the ***RtosSetEvent*** was issued, the results are unpredictable.
- The initial state of an event must be specified as non-signaled.
- Only one program/task at a time can wait for an impending event.

- Although the Windows and VxWorks functions have similar names, caution is advised: There are differences!

18.4.2 Steps to using Shared Events

To use a Shared Event, a user must perform the following steps:

1. A single program/task in each operating system creates the same named event. Refer to the two ***RtosCreateEvent***. functions.

2. If the waiting application is on the Windows side, it uses the ***RtosWaitForSingleObject*** function to wait for the event to occur. If the waiting task is on the real-time side, it uses ***RtosWaitForEvent***.

3. On the opposite operating system side, an application can trigger the event by calling ***RtosSetEvent*** .

4. After the signal arrives on the waiting side, the waiting application is activated and scheduled to run.

5. When the event is no longer needed, an application on the Windows side can close it with the ***RtosCloseHandle*** function. On the real-time side, the event can be closed with ***RtosCloseEvent***.

18.4.3 Shared Events under Windows XP

RtosCreateEvent

This function creates a named object. Under VxWin, only named objects are supported.

```
HANDLE    RtosCreateEvent
            (
           LPSECURITY_ATTRIBUTES    lpEventAttributes,
           BOOL                     bManualReset,
           BOOL                     bInitialState,
           LPCTSTR                  szName
           );
```

PARAMETERS

lpEventAttributes

[in] - A pointer to a SECURITY_ATTRIBUTES structure. Under VxWin, this value will be ignored.

bManualReset

[in] - Must be given as FALSE. Under VxWin only auto-reset events are supported. After a single waiting thread has been released, the system will automatically reset the state to non-signaled.

bInitialState

[in] - Must be given as FALSE to indicate non-signaled. Under VxWin, the initial state always must be non-signaled.

szName

[in] - A pointer to a null-terminated string specifying the name of the event object. The name can contain any characters except the backslash path-separator character (\). Name comparison is case-sensitive.

If szName matches the name of an existing named event object, this function then requests access to the existing object. In this case, bManualReset and bInitialState will be ignored because they have already been set by the creating process.

If szName matches the name of an existing mutex, semaphore, or other shared memory object, the function will fail and *GetLastError* will return ERROR_INVALID_HANDLE. This occurs because Windows XP event, mutex and semaphore objects all share the same namespace. Additionally, the name of the event must not be NULL.

RETURN VALUE

If the function succeeds, the return value is a handle to the event object. If the function fails, the return value is NULL. To get extended error information, call *GetLastError*. If the named event object existed before the function call, *GetLastError* returns ERROR_ALREADY_EXISTS. Otherwise, *GetLastError* returns zero.

REMARKS

The handle returned by RtosCreateEvent has all accesses to the new event object and can be used in any function that requires a handle to an event object.
Any thread of the calling process can specify the event-object handle in a call to *RtosWaitForSingleObject*. This wait function returns when the state of the specified object is signaled. When it returns, the waiting thread is released to continue its execution.
The initial state of the event object is specified by the bInitialState parameter. Use the RtosSetEvent function to set the state of an event object to *signaled*.
When the state of an auto-reset event object is signaled, it remains signaled until a single waiting thread is released; the system then resets the state to non-signaled. If no threads are waiting, the event object remains signaled.
Use RtosCloseHandle to close the handle. The system closes the handle automatically when the process terminates. The event object is destroyed when its last handle has been closed.

Important: Only auto-reset events are supported; the initial state of a event must be set to non-signaled, otherwise an error will be returned.

PROGRAM DETAIL

Windows header:	rtosdrvif.h
Windows code:	UploadRTOS.dll

EXAMPLE

```
HANDLE hEvent;

hEvent = RtosCreateEvent (NULL, FALSE, FALSE, "MyEvent");
```

RtosOpenEvent

This function returns the handle of an existing named event object.

```
HANDLE    RtosOpenEvent
            (
            DWORD     dwDesiredAccess,
            BOOL      bInheritHandle,
            LPCTSTR   szName
            );
```

PARAMETERS

dwDesiredAccess

[in] - This value will be ignored.

bInheritHandle

[in] - This value will be ignored.

szName

[in] - A pointer to a null-terminated string specifying the name of the event object.

RETURN VALUE

If the function succeeds, the returned value is a handle to the event object. If the function fails, the return value is NULL. To get extended error information, call ***GetLastError***.

REMARKS

RtosOpenEvent enables a process to open handles of a named event object. The function will succeeds only if a process has already created the event by using *RtosCreateEvent*.

The calling process can use the returned handle in any function that requires a handle of an event object, such as a wait function.

Use *RtosCloseHandle* to close the handle. If the process terminates, the operating system will automatically close the handle.

When its last handle has been closed, the event object will be automatically destroyed.

PROGRAM DETAIL

Windows header:	rtosdrvif.h
Windows code:	UploadRTOS.dll

EXAMPLE

```
HANDLE hEvent;

hEvent = RtosOpenEvent(0, FALSE, “MyEvent”);
```

RtosCloseHandle

This function closes a handle to an event object.

```
BOOL    RtosCloseHandle
          (
          HANDLE    hObject
          );
```

PARAMETERS

hObject

[in] - An open object handle.

RETURN VALUE

If the function succeeds, the returned value is TRUE. If the function fails, the returned value is FALSE. To get extended error information, call *GetLastError*.

REMARKS

RtosCloseHandle invalidates the specified object handle, decrements the object's handle count, and performs object retention checks. Once the last handle to an object has been closed, the object will automatically be deleted.

PROGRAM DETAIL

Windows header:	rtosdrvif.h
Windows code:	UploadRTOS.dll

EXAMPLE

```
HANDLE hEvent;
BOOL bSuccess;

hEvent = RtosCreateEvent( NULL, FALSE, FALSE, “MyEvent” );
bSuccess = RtosCloseHandle( hEvent );
```

RtosSetEvent

Sets the state of the specified event object to *signaled*.

```
BOOL    RtosSetEvent
          (
          HANDLE    hEvent
          );
```

PARAMETERS

hEvent

[in] - Identifies the event object. The ***RtosCreateEvent*** or RtosOpenEvent function returns this handle.

RETURN VALUE

If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE. To get extended error information, call ***GetLastError***.

REMARKS

The state of an auto-reset event object remains signaled until a single waiting thread is released, at which time the system automatically sets the state to non-signaled. If no threads are waiting, the event object's state remains signaled.

Important: Only auto-reset events are supported.

If you call ***RtosSetEvent*** (Windows context), you should only wait for that event in VxWorks (***RtosWaitForEvent***). You should not wait for the event in the Windows context because, if you do, the resulting system behavior is unpredictable.

Using ***RtosSetEvent*** does not force a context change; that is, the event will be noticed in the real-time system after the context has been switched to VxWorks for some other reason, such as the occurrence of a real-time interrupt.

PROGRAM DETAIL

Windows header:	rtosdrvif.h
Windows code:	UploadRTOS.dll

EXAMPLE

```
HANDLE hEvent;
BOOL bSuccess;

hEvent = RtosCreateEvent( NULL, FALSE, FALSE, "MyEvent" );
bSuccess = RtosSetEvent( hEvent );
```

RtosWaitForSingleObject

This function waits until the specified object is in the *signaled* state or the specified timeout interval has elapsed.

```
DWORD    RtosWaitForSingleObject
           (
           HANDLE    hObject,
           DWORD     dwMilliseconds
           );
```

PARAMETERS

hObject

[in] - Identifies the event object. The RtosCreateEvent or RtosOpenEvent function returns this handle.

dwMilliseconds

[in] - The time-out interval, in milliseconds. The function returns if the interval elapses, even if the object's state is non-signaled. If Milliseconds is zero, the function tests the object's state and returns immediately. If Milliseconds is INFINITE, the function's time-out interval never elapses.

RETURN VALUE

- If the function succeeds, the return value indicates the event that caused the function to return.
- If the function fails, the return value is WAIT_FAILED. To get extended error information, call ***GetLastError***.
- Upon success, the return value is one of the following values
  - WAIT_OBJECT_0: The specified object was set to 'signaled'.
  - WAIT_TIMEOUT: The time-out interval elapsed, and the object's state is non-signaled.

REMARKS

- ***RtosWaitForSingleObject*** checks the current state of the specified object. If the object's state is non-signaled, the calling thread enters an efficient wait state. The thread consumes very little processor time while waiting for the object state to become signaled or the time-out interval to elapse.
- ***RtosWaitForSingleObject*** can wait for event objects. The ***RtosCreateEvent*** or ***RtosOpenEvent*** function return the handle. An event object's state is set explicitly to signaled by the ***RtosSetEvent*** function.
- For an auto-reset event object, the wait function resets the object's state to non-signaled before returning.

Important: Only auto-reset events are supported. If you call ***RtosWaitForSingleObject*** in Windows XP you must be waiting for an ***RtosSetEvent*** call from VxWorks. You should not try to signal the event from the Windows XP context; if you do, the function's behavior is unpredictable.

PROGRAM DETAIL

Windows header:	rtosdrvif.h
Windows code:	UploadRTOS.dll

EXAMPLE

```
HANDLE hEvent;
BOOL bSuccess;

hEvent = RtosCreateEvent (NULL, FALSE, FALSE, "MyEvent");
bSuccess = RtosWaitForSingleObject (hEvent, INFINITE);
```

18.4.4 Shared Events under VxWorks

RtosCreateEvent

This function creates a named event object for communicating with a thread/task under Windows. VxWin only supports named objects.

```
SEM_ID    RtosCreateEvent
            (
            char*     szName
            );
```

PARAMETERS

szName

[in] - A pointer to a null-terminated string specifying the name of the event object. The name can contain any character except the backslash path-separator character (\). Name comparison is case-sensitive.
If szName matches the name of an existing named event object, this function will receive access to the existing object.
But because event, mutex and semaphore objects all share the same Windows XP namespace, if szName matches the name of an existing mutex, semaphore, or shared memory object, the function will fail.
Note: The event may not be named NULL.

RETURN VALUE

If the function succeeds, the returned value is a SEM_ID (Binary Semaphore Identifier) to the event object. If the function fails, the returned value is NULL.

REMARKS

- The SEM_ID returned by ***RtosCreateEvent*** permits multiple tasks to access the new event object. It can be used in any function that requires the SEM_ID.
- Use the ***RtosSetEvent*** function in the VxWorks context to signal an event object.
- Any Windows XP thread/task can specify the SEM_ID in a call to ***RtosWaitForSingleObject***. This wait function returns when the state of the specified object has been signaled. When it returns, the waiting task is free to continue executing.
- Unless other Windows tasks are waiting for the same object, the event object will remain *signaled* until the last waiting task has received the signal. When the last task that was waiting for its auto-reset event object has been signaled, the system will set the state to non-signaled.
- Use ***RtosCloseEvent*** to close the event. The system automatically destroys the event if the last process that created it is terminated or closes it (*RtosCloseEvent*).
Important: Only auto-reset events are supported. Upon initial creation, the condition is non-signaled.

PROGRAM DETAIL

VxWorks header:	rtosLib.h

EXAMPLE

```
SEM_ID semID;

semID = RtosCreateEvent( “MyEvent” );
```

RtosOpenEvent

This function returns the SEM_ID of an existing named event object.

```
SEM_ID    RtosOpenEvent
              (
              char*    szName
              );
```

PARAMETERS

szName

[in] - A pointer to a null-terminated string specifying the name of the event object.

RETURN VALUE

If the function succeeds, the return value is a SEM_ID to the event object. If the function fails, the return value is NULL.

REMARKS

- ***RtosOpenEvent*** enables multiple processes to open SEM_ID's of the same event object. The function succeeds only if some process in Windows XP has already created the event by using ***RtosCreateEvent***. The calling process can use the returned SEM_ID in any function that requires a SEM_ID of an event object, such as the wait function (RtosWaitForEvent).

- Use ***RtosCloseEvent*** to close the event. If the last process that created/opened the event terminates, the system will automatically close the event.

PROGRAM DETAIL

VxWorks header: rtosLib.h

EXAMPLE

```
SEM_ID semID;

semID = RtosOpenEvent("MyEvent" );
```

RtosCloseEvent

This function closes an open object event.

```
STATUS    RtosCloseEvent
              (
              SEM_ID    semId
              );
```

PARAMETERS

semId

[in] - An open object SEM_ID.

RETURN VALUE

If the function succeeds, the return value is SUCCESS. If the function fails, the return value is ERROR.

REMARKS

RtosCloseEvent closes SEM_ID's to Rtos event objects. It invalidates the specified object SEM_ID, decrements the object's SEM_ID use-count, and performs object retention checks.

PROGRAM DETAIL

VxWorks header: rtosLib.h

EXAMPLE

```
SEM_ID semID;
STATUS bSuccess;

semID = RtosCreateEvent( “MyEvent” );
bSuccess = RtosCloseEvent( semID );
```

RtosSetEvent

This function sets the state of the specified event object to signaled.

```
STATUS    RtosSetEvent
            (
            SEM_ID    semId
            );
```

PARAMETERS

semId

[in] - Identifies the event object. The ***RtosCreateEvent*** or ***RtosOpenEvent*** function returns this SEM_ID.

RETURN VALUE

If the function succeeds, the return value is SUCCESS. If the function fails, the return value is ERROR.

REMARKS

The state of an auto-reset event object remains signaled until a Windows process, using RtosWaitForSingleObject, is released to run, at which time the system automatically sets the object’s state to non-signaled. If no processes happens to be waiting for the object, the event object's state remains signaled.

Important: Only auto-reset events are supported. And if you are calling ***RtosSetEvent*** in VxWorks you can only use *RtosWaitForSingleObject* in Windows XP. You may not wait for the same event in VxWorks. If you attempt this anyway, the result is unpredictable.

Note: A call to ***RtosSetEvent*** is a non-deterministic action. The signal will be detected under Windows only after control has been returned to it by the usual means.

PROGRAM DETAIL

VxWorks header: rtosLib.h

EXAMPLE

```
SEM_ID semID;
STATUS bSuccess;

semID = RtosOpenEvent( “MyEvent” );
bSuccess = RtosSetEvent( semID );
```

RtosWaitForEvent

This function is used to wait until the specified object has been set to the signaled state in the Windows context or until the specified timeout elapses.

```
STATUS    RtosWaitForEvent
            (
            SEM_ID              semId,
            long unsigned int   dwMilliseconds
            );
```

PARAMETERS

semId

[in] - Identifies the event object. This SEM_ID is acquired through use of the ***RtosOpenEvent*** function.

dwMilliseconds

[in] - The time-out interval in milliseconds. The function will returns if the interval elapses, even if the object's state is non-signaled. If dwMilliseconds is zero, the function tests the object's state and returns immediately. If dwMilliseconds is set to INFINITE, the function's time-out interval can never elapse.

RETURN VALUE

If the function succeeds, the Return Value indicates the event that caused the function to return:

- WAIT_OBJECT_0: The specified object had been signaled (same value as OK).
- WAIT_TIMEOUT: The time-out interval has elapsed; the object's state is non-signaled.
- WAIT_FAILED: The event was not defined or has ceased to exist.

REMARKS

- ***RtosWaitForEvent*** checks the current state of the specified object. If the object's state is non-signaled, the calling process enters a wait-state. Note: Waiting consumes little processor time.
- If the time-out occurs, the wait-state will be terminated and the function returns control to the calling routine with the Return Value set accordingly.
- Use the ***RtosOpenEvent*** function to get the required SEM_ID.
- In the Windows context, an event object's state must be set to *signaled* by the ***RtosSetEvent*** function.
- For an auto-reset event object (the only supported type) this function resets the object's state to non-signaled before returning.

Important: Only auto-reset events are supported. If you use this function, the event should be set sometime by ***RtosSetEvent*** in the Windows context. You should not try to set the same named event under VxWorks; if you do, the resulting behavior is unpredictable.

PROGRAM DETAIL

VxWorks header: rtosLib.h

EXAMPLE

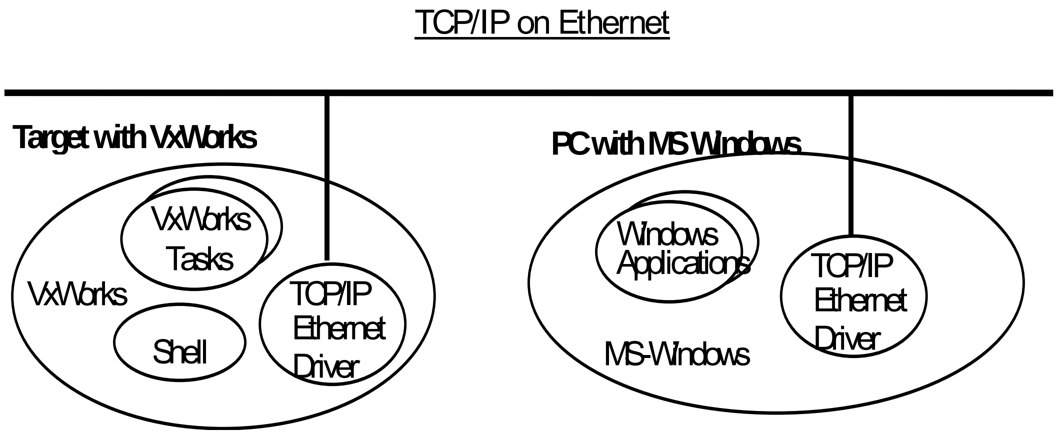
```
SEM_ID semID;
STATUS bSuccess;

semID = RtosOpenEvent( “MyEvent” );
bSuccess = RtosWaitForEvent( semID, INFINITE );
```

18.5 Virtual Network Communications

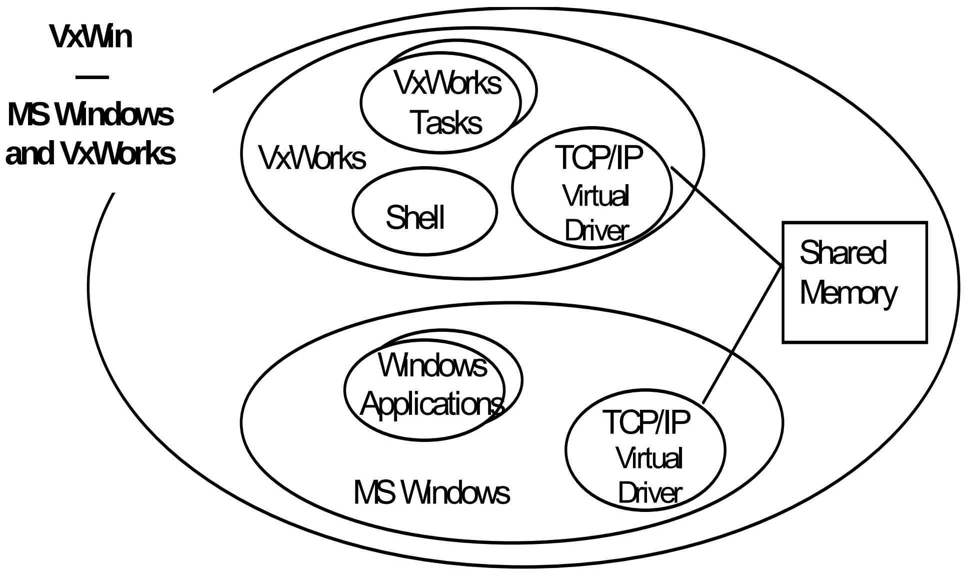
18.5.1 The Basics

The VxWin Virtual Network enables application programs running under either of the two operating systems to communicate with each other and with the external world using standard TCP/IP protocol.
While an NDIS compliant driver (Network Driver Interface Specification) was implemented for the Windows side of the system, an END driver (Enhanced Network Driver) was written for for the VxWorks side. (The END driver is part of the VxWin BSP.) Both are part of the VxWin standard release package. Since the drivers are based on sockets technology, experienced communications programmers will have no problem using them.
If a single computer is used to host VxWin, the Virtual Network communicates data between the two operating systems using a portion of shared memory as intermediate storage. If two computers are used, however, the systems will communicate over Ethernet adapters dedicated to this purpose. It is sometimes possible to share an Ethernet adapter. To learn more about this, see the discussion of Packet routing: VxWorks Uses Windows network adapter.
Once communications has been established, other Wind River or third-party products can be used in conjunction with VxWorks. The Tornado® development system, for example, can be used, whether it is on another PC or even a UNIX-based system. In this way, one can also use WindView® or third-party products to perform profiling or remote source-code debugging.
The following two diagrams graphically depict the object of the foregoing discussion.

Inter-Computer Communications



Virtual Network



How it works – a brief sketch

Preliminaries

- During the Windows boot procedure, the VxWorks Virtual Network NDIS driver is loaded. (It is loaded after the VxWin Device Driver (the Real-time Operating System Device Driver) has been loaded).
- During initialization, the NDIS Network Driver allocates a portion of the computer's memory to serve as “Shared Memory.”

- The Real-time Operating System device driver is automatically called to inform it of the start address of the Shared Memory.
- The NDIS Network Driver, which subsequently serves as the “Shared Memory Master,” initializes its portion of shared memory.
- Within the VxWorks initialization routine, Network Setup recognizes the presence of a valid Virtual Network and connects to it.
- The Virtual Network can be used as soon as VxWorks begins to run.

Windows to VxWorks

- The NDIS Driver allocates available memory from the shared memory pool and copies the data into it.
- The NDIS Driver generates an interrupt to VxWorks, informing it of the availability of incoming network data.
- Immediately invoked, VxWorks" Network Task receives the data packets.
- If the VxWorks scheduler allows VxWorks to enter the idle status, Windows runs again. If another VxWorks task is waiting, however, it will be serviced before control is given back to Windows.

VxWorks to Windows

- The VxWorks Network Task allocates available memory from the shared memory pool and copies data into it.
- The VxWorks Network Task generates an interrupt to inform Windows of the availability of data on the network.
- When VxWorks becomes inactive, Windows will again run.
- When the interrupt is recognized, the NDIS Driver is called. It then receives the data.

18.5.2 Setting up the Windows side

Virtual Network driver - General

The Virtual Network Device Driver sends and receives I/O data packets over the network. If both operating systems are on the same PC, this driver transmits and receives data to and from shared memory.

Configuring the network driver

The Virtual Network Device Driver and the TCP/IP protocol driver can be configured by activating the network icon on the Windows desktop or via Windows Device Manager.
Even though most parameters pertaining to the Virtual Network are preset in the software release to values that are expected to work for most systems, you should probably specify the size of the shared memory area to meet the requirements of your specific application.
Be certain to link a TCP/IP protocol driver to the Virtual Network Driver.
Note: Although any TCP/IP that supports the NDIS Protocol Stack can be used, Microsoft TCP/IP for Windows also fulfills this requirement.

Assigning an IP address

While you must assign an IP address to the network that agrees with that used by VxWorks, you must make certain that the network part of the address does not correspond to the network part of any other IP address in the network.

Setting up the Virtual Network Device Driver

The device-name of the Virtual Network Driver under Windows is — ***Realtime OS Virtual Network***.
The screenshot below illustrates the ***Advanced Properties*** for the Virtual Network adapter driver as listed in the ***Device Manager*** display.
To get to the display, you can either...

- Move the cursor to the ***My Computer*** icon on your desktop, then right-click on the icon. From the context menu, choose the ***Manage*** option.

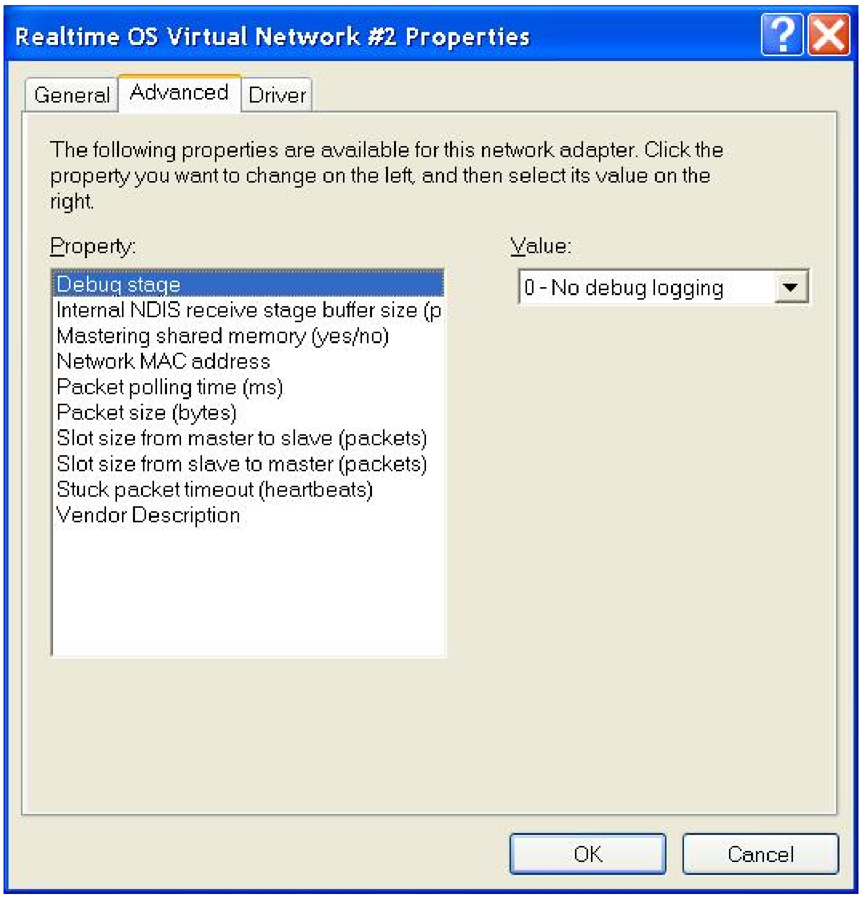
  - OR -

- Get to the System Manager by clicking on ***System*** under the ***Start*** button. Under the ***Hardware*** tab, choose ***Device Manager***, which should produce the same display

…AND then, in the ***Device Manager*** tree structure, click on ***Network Adapters,*** then ***Realtime OS Virtual Network***...

Virtual Network Driver Properties

Under the ***Network Environment***, choose ***Properties*** from the context menu. Within the list of ***Network Adapters***, select ***Realtime OS Virtual Network***. Then choose ***Properties***. After clicking the ***Configuration*** tab, a display similar to that shown below should appear.


Realtime OS Virtual Network #2 Properties

General | Advanced | Driver

The following properties are available for this network adapter. Click the property you want to change on the left, and then select its value on the right.

Property:

- Debug stage
- Internal NDIS receive stage buffer size (p
- Mastering shared memory (yes/no)
- Network MAC address
- Packet polling time (ms)
- Packet size (bytes)
- Slot size from master to slave (packets)
- Slot size from slave to master (packets)
- Stuck packet timeout (heartbeats)
- Vendor Description

Value: 0 - No debug logging

The various advanced settings are described in detail below:

- *Debug stage*
  For debugging purposes it is possible to log error information. Select among the following choices:

  0: no debug logging
  1: log only initialization messages (default)
  2: log all messages
  3: same as 1 including all DbgPrint's
  4: same as 2 including all DbgPrint's

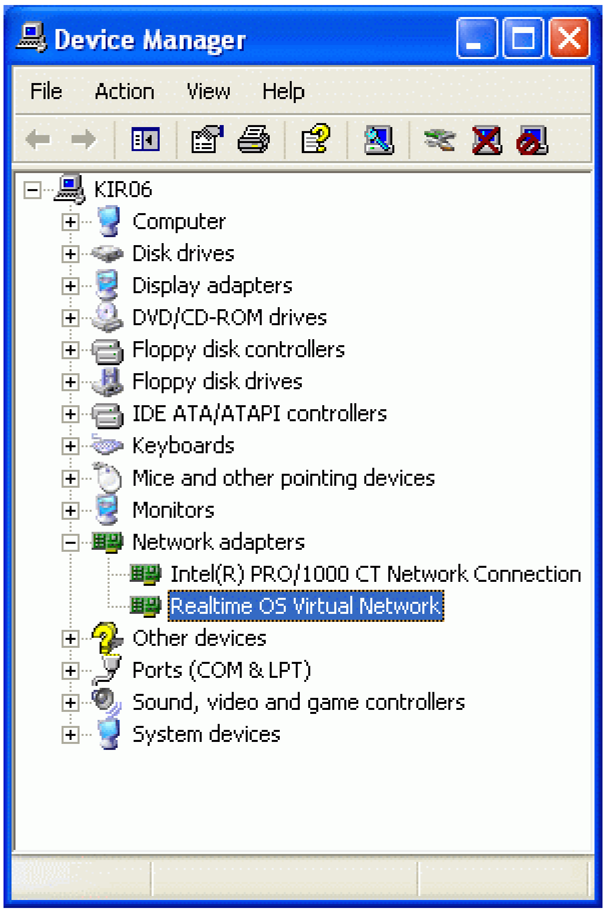
  If you have problems running the Virtual Network adapter driver, you should set this parameter to 1 (the default value) and after starting the system send the file *ndislog.txt* (in the Windows directory) to Customer Support for further analysis.

- *Internal NDIS receive stage buffer size*
  Specifies the number of received packets that can be stored in the network driver itself. This value must be greater than the sum of *Slot size from master to slave (packets)* and *Slot size from slave to master (packets)*. This parameter also informs the protocol stack how much space (number of data packets) is available for data to be received.

- *Mastering Shared Memory*
  This property, which cannot be changed by the user, indicates which of the two supported operating systems is master or slave. Under VxWin, Windows is always designated as *master* while VxWorks is *slave*.
- *Network MAC address*
  The 6-byte Ethernet address of the Windows side of the virtual network adapter. (For information on setting MAC addresses on both VxWorks and Windows, refer to Chapter 0 - Setting MAC Addresses for the Virtual Network Driver.)
- *Packet polling time (ms)*
  Windows XP polling time for the Virtual Network adapter. Every few milliseconds, in accord with this parameter, the Virtual Network adapter checks for the receipt of new network packets from VxWorks.
- *Packet size (bytes)*
  Maximum size of a packet that can be stored in the internal Virtual Network memory.
- *Slot size from master to slave (packets)*
  Number of packets sent from Windows XP to VxWorks that can be stored in the Virtual Network memory.
- *Slot size from slave to master (packets)*
  Number of packets sent from VxWorks to Windows XP that can be stored in the Virtual Network memory.
- *Stuck packet timeout (heartbeats)*
  Windows cyclically calls the network driver for maintenance checks (usually every 2 seconds), at which time the network driver increments a heartbeat counter. If its count exceeds this parameter’s value, the real-time system assumes that the Windows part is no longer running.

  This value is also used by the network driver to determine if received packets have been de-queued from the receiver queue. If some packets are not received within the given number of *heartbeats*, something that could happen if the real-time system was restarted by the Uploader, the *Stuck Packet Timeout* will occur and any such 'stuck' packets will be de-queued and deleted. This automatic mechanism helps prevent the network from hanging.
- *Vendor Description*
  This property merely identifies the software manufacturer.

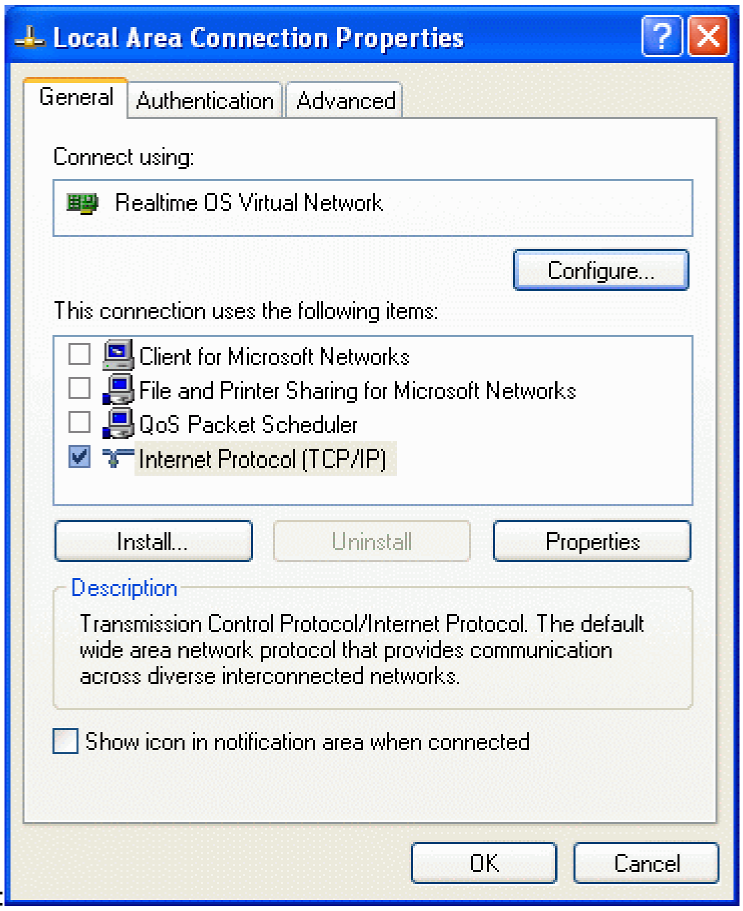
Device Manager display



KIR06
- Computer
- Disk drives
- Display adapters
- DVD/CD-ROM drives
- Floppy disk controllers
- Floppy disk drives
- IDE ATA/ATAPI controllers
- Keyboards
- Mice and other pointing devices
- Monitors
- Network adapters
  - Intel(R) PRO/1000 CT Network Connection
  - Realtime OS Virtual Network
- Other devices
- Ports (COM & LPT)
- Sound, video and game controllers
- System devices

Properties Dialog box for a LAN-Connection

The following screenshot illustrates how the dialog box would appear with TCP/IP protocol selected.


Local Area Connection Properties

General | Authentication | Advanced

Connect using:

Realtime OS Virtual Network

Configure...

This connection uses the following items:

- [ ] Client for Microsoft Networks
- [ ] File and Printer Sharing for Microsoft Networks
- [ ] QoS Packet Scheduler
- [✓] Internet Protocol (TCP/IP)

Install... | Uninstall | Properties

Description

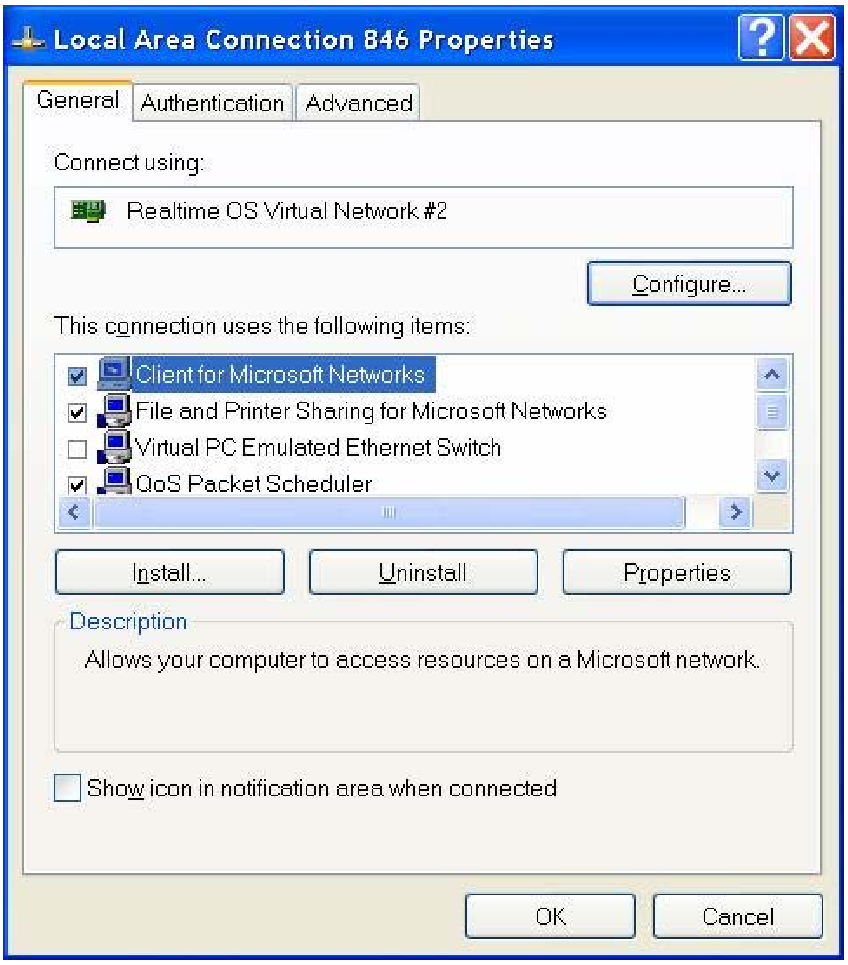
Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

- [ ] Show icon in notification area when connected

OK | Cancel

IMPORTANT: Local Area Connection for CIFS

If you are going to use the CIFS method to access the PC's hard disk, you have to choose some additional options as illustrated in the following screenshot. Please refer to Accessing the PC's hard disk, CIFS – Common Internet File System)


Local Area Connection 846 Properties

General | Authentication | Advanced

Connect using:

Realtime OS Virtual Network #2

Configure...

This connection uses the following items:

- ☑ Client for Microsoft Networks
- ☑ File and Printer Sharing for Microsoft Networks
- ☐ Virtual PC Emulated Ethernet Switch
- ☑ QoS Packet Scheduler

Install... | Uninstall | Properties

Description

Allows your computer to access resources on a Microsoft network.

☐ Show icon in notification area when connected

OK | Cancel

18.5.3 Setting up the VxWorks side

In order to use VxWorks networking facilities, the “INCLUDE_NETWORK” macro must be defined in the software configuration for VxWorks.
The VxWorks configuration is contained in a header file containing C preprocessor macros. When building a VxWorks image, i.e., compiling and linking, feature-specific libraries will be included in or excluded depending on how you define the relevant macros. You can determine which features to use or disable by declaring their status correspondingly (#define or #undef). A configuration tool, part of the Tornado/Workbench IDE, will automatically generate the header file.

To configure VxWorks for networking, the user should enter his own prototype boot line in the VxWin configuration file (***vxwin.config***). Refer to the Bootline parameter in Global parameters in the VxWin configuration file. When VxWin starts up, the Uploader Utility copies the prototype boot line from the configuration file into memory at a predefined address. Then, when VxWorks boots, it reads and interprets the network parameters in that boot line. Among the various parameters that must be specified are: network addresses, network devices, host-computer designation, script filename, and so on.
The following describes the general VxWorks boot line syntax:

```
bootdev(0,procnum)hostname:filename h=# e=# b=# g=# u=userid pw=passwd
tn=targetname s=startupscript o=other
```

Parameter		**Description**
bootdev	=	"shm" for the Virtual Network, "ultra" for the SMC Elite Ultra Ethernet adapter card, and so on. (Refer to Wind River documentation.)
procnum	=	The processor number of the Virtual Network. (Must always be 1.)
hostname	=	The name of the host computer from which the boot will be performed.
filename	=	The filename of the VxWorks image to be executed.
e	=	The IP address of the Ethernet port. This field can have an additional partial net-mask. <inet_adrs>:<subnet_mask>
b	=	The IP address of the VxWorks Virtual Network interface. This field can also have an additional partial net-mask.
h	=	The host's Internet-Address.
g	=	The Internet address of the host system's gateway. Note: When the host is on the same network, this field should be left blank.
u	=	A valid user-name on the host.
pw	=	The user's password on the host system. When specified, FTP will be used for data transfers.
tn	=	The name of the VxWorks target system.

Parameter		**Description**
s	=	The name of a script file containing commands in text form to be interpreted by VxWorks Target Shell Interpreter during startup. This script can access the following "o" parameters.
o	=	“other” – a character string that may be accessed either by commands in the –s script file or a task under VxWorks. This string may be used, e.g., to parameterize the boot process.

Note: Internet addresses are specified in “dot” notation (e.g., 90.0.0.2)
Note: The order of assigned parameters is arbitrary.
Example: The following boot line could be used to configure the Virtual Network:

```
shm(0,1)pc:VxWorks h=192.168.0.1 e=192.168.0.2 u=target pw=vxworks
```

Note: IP addresses must, of course, match those used on the Windows side of the Virtual Network.

Virtual Network adapter address

The network adapter address [also known as the MAC (Media Access Control) address] is generally set in accord with the following syntax:

AA-BB-CC-DD-EE-00

You can use a parameter in the VxWin configuration (***vxwin.config***) file to specify a network address for your Virtual Network card. Refer to NetworkAddress in Global parameters in the VxWin configuration file. For information on setting MAC addresses on both VxWorks and Windows, also refer to Setting MAC Addresses for the Virtual Network Driver.

18.6 Packet routing: VxWorks Uses Windows network adapter

There are two ways to send TCP/IP packets to and from Windows XP and VxWorks using an Ethernet cable:

- Add an Ethernet adapter to the PC on which the real-time system runs. That system and its applications will then have their own means for communicating with the outside world.
- By setting up Packet Routing, applications running under VxWorks can 'borrow' Windows Ethernet adapter card to move data onto and from the Ethernet. See "IP Forwarding" below.

18.6.1 IP forwarding

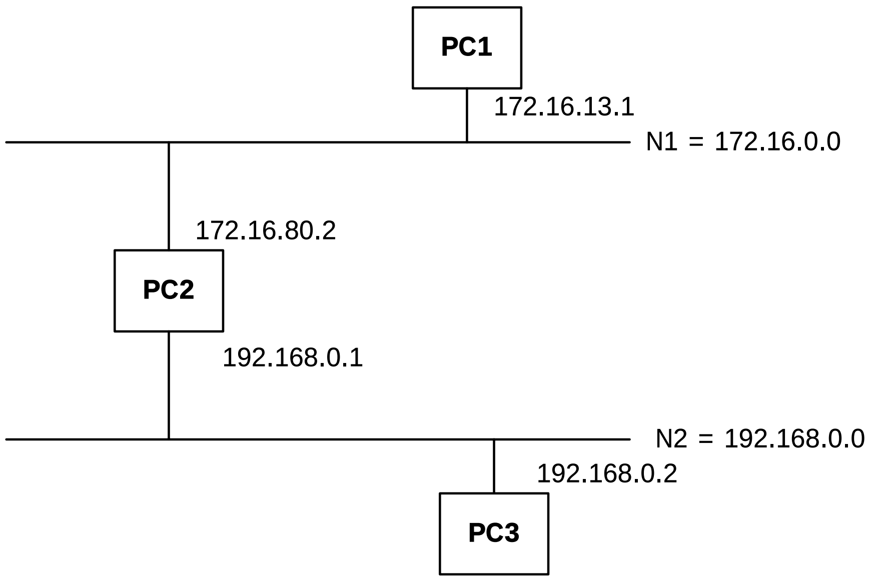
To cause Windows to forward IP packets sent from VxWorks, this feature must be explicitly enabled in Windows. Using the Windows registry editor, you should make the following changes:

```
   Key:     [HKLM\System\CurrentRoboteret\Services\Tcpip\Parameters]
   Entry:    IPEnableRouter (change value from 0 to 1)
```

Important: In VxWorks, IP packets forwarding is enabled by default. See `\target\h\netLib.h – IP_FLAGS_DFLT`.

18.6.2 IP routing table

To send packets to a destination outside your local subnet, those packets must either be sent through a default gateway or be redirected via specific routing information that you enter into Windows routing tables.
Consider the situation in the following illustration, in which it is desired to route packets from PC1 to PC3 via PC2 and vice versa:



The following assumptions are made:

- PC3 is running the VxWorks part of your system, and its IP address is 192.168.0.2.

- PC2 is running the Windows part of the system. It uses two IP addresses: 192.168.0.1 for the ***Virtual Network*** adapter and 172.16.80.2 for the Ethernet network adapter.
- PC1 is an additional PC, connected to the same network as PC2.

—

Now, while PC1 is running Windows, add a route on PC1 to PC3, using the Windows ***route*** command:

```
route [-p] add 192.168.0.2 mask 255.255.255.255 172.16.80.2
```

Note: The –p option makes the route permanent, otherwise the effect of the routing command would be lost each time the system is restarted.
Note: You can use the ***route print*** command to display currently active routes.

—

And, with VxWorks running on PC3, use the following VxWorks command to create a route to PC1:

```
mRouteAdd “172.16.13.1”,”192.168.0.1”,0xffffff00
```

Use the ***mRouteShow*** command to display currently valid routes

18.7 Accessing the PC's hard disk

Introduction

Under VxWin, the PC’s hard disk can be accessed only from the Windows context. This is not an insurmountable limitation, however, because several software methods that allow real-time applications to read or write the hard disk are nonetheless available.

Indirect Method

If you can convey information back and forth between the two operating systems by way of Shared Memory or the Virtual Network, you could then perform I/O functions to the hard disk exclusively from within the Windows context. If you use Shared Memory, you would probably also want to use Shared Events to coordinate activities, i.e., to maintain the integrity of your data. This is, however, a non-deterministic method.

FTP – File Transfer Protocol

For this method, you would have to install an FTP client/server on the Windows side of the system. The Internet Information Server (IIS) is suitable for this purpose. To communicate with the Windows client/server, you would then use ***netDrv*** a standard VxWorks driver.

Advantages:

- ***netDrv*** is a standard component of VxWorks.
- Once a file has been opened, data accesses are quick because they are performed entirely in RAM.

Disadvantages:

- Because only an entire file can buffered in RAM, the size of the files you can work with are limited by the actual size of the available memory.
- Experience has shown that rapidly opening and closing files sometimes leads to socket problems.
- Frequent opening and closing of files can slow down the overall system, because such actions cause the entire file to be read from or written to the hard disk.

NFS – Network File System

For this solution, an NFS client/server is required for the Windows context.
Advantages:

- This implements a *genuine* file system; i.e., only required portions of a data file are read or written at one time.
- The required NFS driver is a standard component of VxWorks (***nsfDrv***).

Disadvantages:

- The corresponding NFS client/server is not a standard part of Windows. It would have to be acquired from a third-party software manufacturer.
- This solution would introduce a non-standard element on the Windows side of the system.

CIFS – Common Internet File System

To use this method, a Virtual Network would have to be set up for accessing resources on a Microsoft network, and a CIFS client/server would be required for the VxWorks context. To see how to set up a Virtual Network Driver for Windows XP, see Local Area Connection for CIFS in Virtual Network Communications.
Advantages:

- This is a *genuine* network file system; i.e., only the required portions of a data file are read or written at one time.

- No additional software is required for the Windows XP context. The user must simply specify a shared directory under which Windows file(s) are to be read or written, with the same requirement for VxWorks.

- This solution makes use of standard MS Windows features.

Disadvantages:

- The corresponding CIFS software must be acquired from a third-party software vendor (such as CIFS NQ from Visuality Systems).

19 System configuration file (vxwin.config)

To characterize the VxWin system environment to suit the needs of your application, you may enter or adjust certain parameters in a configuration file. The configuration file is an ordinary text file that you can modify using a common text editor.
The default file – *vxwin.config* – is generally located in *C:\Program Files\VxWin*. (If you want to use a different file, you can specify a filename and path for it by using the *–config* option in the Uploader Utility command line. Refer to the Uploader Utility Program elsewhere in this manual.
If while reading this chapter, you wish to refer to it, we have provided an example VxWin configuration file elsewhere in this manual.
Using a syntax that should be familiar to Windows users, the configuration file contains information such as the following:

- Dynamic parameters for the VxWin Uploader Utility
- Global system settings (e.g. Shared Memory size, enable or disable tracing commands, etc.)
- A prototype boot line for VxWorks (including important parameters for communicating via the Virtual Network).

File syntax

Since the configuration file is an ordinary ASCII file, you can modify it with any text editor, such as MS Notepad. The syntax closely resembles that generally used by Windows, in the Registry for example.

General rules

- The first entry in a configuration file must be ***RtosConfig***.
- Since comments in the configuration file are introduced by a semicolon, all characters following a semicolon will be ignored.
- No line in the configuration file may exceed 256 characters in length.

Keys, Entries, and Values

Configuration settings are stored using keys. Specific settings are called entries. Each entry has both a name (ASCII characters in quotes) and a value. Each entry is logically subordinated to the key that precedes it. In some cases, a key will be followed by multiple entries (one per line).

Example:

```
[Key]
“EntryName“=EntryValue
```

Several possible types of ***EntryValue***:

- Single hexadecimal value (dword)
  `“EntryName”=dword:1F`
- Strings
  `“EntryName”=”This is a string”`
- Multiple Strings
  `“EntryName”=multi_sz:”First string”,”Second string”,”Last string”`
- Multiple hexadecimal values
  `“EntryName”=hex:XX,YY,…,ZZ`

Include statements

Configuration commands and parameters may include multiple files. You may use include statements to incorporate other configuration files into this one.

Important:

- Unless additional path information is given, the specified include-files will be assumed to be located in the same directory as this configuration file.
- Since nested includes are not supported, only include statements that appear in the main configuration file will be processed.

Examples:

```
#include "Network.config"
#include "VirtNetworkAdapter.config"
```

Dynamic boot line

As you may have read elsewhere in this manual, VxWorks uses a boot line to characterize itself when it starts up. If you did not already do so, you may want to read Prototype Boot Line for VxWorks now.

Each time VxWin is booted, the user has an opportunity to dynamically configure the VxWorks boot parameters. To accomplish this, you must first create or modify boot line entries under the ***[rtosbsp]*** key in the system configuration file (vxwin.config).

When starting up, if VxWin finds no ***[rtosbsp]*** key in the configuration file, the default boot line will be used.

The default boot line assumes that the VxWorks Virtual Network driver is installed and that its IP address is **`192.168.0.1`** on the Windows side and

`192.168.0.2` on the VxWorks side. A default boot line is illustrated as follows:

```
        Bootline = shm(0,1)pc:vxWorks h=192.168.0.1
                        e=192.168.0.2 u=target pw=vxworks
```

Global parameters

Configuration parameters for various VxWin processes are stored in this configuration file. Parameters for the Uploader Utility, the Real-time System Driver and for the Real-time BSP may all be found under corresponding keys. Important: Only one particular format to express numeric parameter values is allowed in the configuration file. If a value is strictly numeric it must be expressed as a hexadecimal number without any prefix or any suffix characters. That means, for example, to express the decimal number 100, you enter 64; to express decimal 10, you enter A.

The Upload Key

The user may define the following entries for the VxWin Uploader Utility program under the ***[Upload]*** key:

Entry Name	**Type**	**Description**
RtosMemoryStartAddress	dword	This parameter specifies the first address of main memory allocated to VxWorks. Shipped preset to 16 MB (0x1000000). Warning: By changing the default value of this parameter, you will lose compatibility to the BSP in the product delivery package. Any change to this value becomes effective only after rebooting the system.
RtosTotalMemorySize	dword	This parameter specifies the total amount of main memory allocated to VxWorks and the Shared Memory. Shipped preset to 16 MB (0x1000000), it must be expressed as a multiple of 4096 (i.e., 0x1000). Warning: For a changed value to become effective, you must reboot.
UserShmSize	dword	Shared memory size in KB for user-defined communication between Windows and VxWin. Default is 100 kilobytes (0x64). Values entered must be expressed as multiples of 4096 (0x1000). Maximum size depends on total extent of physical memory. See Using Shared Memory elsewhere in this manual. Warning: A changed value becomes effective only after a system reboot.

Entry Name	**Type**	**Description**
InternalShmSize	dword	Shared memory size in KB for VxWin internal functions. This memory is used for tracing. All VxWin debug messages are stored in the internal shared memory. Maximum size is 1 MB. Default is 100 kilobytes (0x64). Values entered must be expressed as multiples of 4096 (0x1000). Warning: A changed value becomes effective only after a system reboot.
Trace	dword	Disable tracing = 0 (default). Enable tracing = 1. If tracing is enabled, all debug messages are stored in the internal shared memory.
TraceWrap	dword	If TraceWrap is set to 0 (default), tracing will stop when the trace buffer is full. If set to 1, tracing will wrap around each time the trace memory fills.
PrintTrace	dword	If PrintTrace is set to 0 (default), traced data won't be printed. If PrintTrace is set to 1, the Uploader Utility will write all trace information on the user's screen. If no trace data is generated in VxWorks within the number of seconds specified in the entry TraceDuration, the writing will stop
TraceFile	string	If tracing is enabled, all trace data will be stored in this named file. Default = "trace.txt". If no trace data is generated in VxWorks for the number of seconds specified in the entry TraceDuration, tracing will stop
TraceDuration	dword	Duration of recording trace data. Default = 10 (0xA).
RtosEntrypointOffset	dword	Offset to RTOS entry-point in RTOS memory (relative to *RtosMemoryStartAddress*). Default is 32 MB (0x8000).
RtosUploadImageOffset	dword	Offset to RTOS image in RTOS memory (relative to *RtosMemoryStartAddress*). Default is 32 MB (0x8000).
WaitForRtosCommSubsystems	dword	Disabled = 0. Enabled = 1(default). If enabled: upon starting VxWorks, the system will wait for communication subsystems to be initialized before beginning normal real-time operations. Enabling this parameter can increase the start time. Disabling it could cause synchronization problems. The *VxwinWaitforCommSubsystems* function must be called to assure that the communication subsystems are ready before any use is made of them.
CommTimeoutWin2RTOS	dword	Communication timeout in seconds (Windows waiting for RTOS). Default = 5. Used by *VxWinCreateProcess* and *RtosSetEvent.*
CommTimeoutRTOS2Win	dword	Communication timeout in seconds (RTOS waiting for Windows). Default = 5. Used by RtosSetEvent.

Entry Name	Type	Description
MaxNumUserEvents	dword	Defines the maximum number of user-events. (Default = 10 events, written: A)
LaunchRtosControl	dword	This parameter causes the ***RtosControl*** system tray application to be launched. 1 = (default) launch. 0 = do not launch.
ClockSyncEnable	dword	0 = clock synchronization disabled. 1 = (default) clock synchronization enabled. - - - IMPORTANT: Before clock synchronization can begin, ***RtosStartTimeSync()*** must be called from the VxWorks context. See Real-time Clocks and Time.
RtosClockMaster	dword	0 = (default) Windows serves as clock master. 1 = VxWorks serves as clock master. See Real-time Clocks and Time/Calendar functions.
InitializeRtosClock	dword	This parameter determines how the VxWorks time base will be set upon starting VxWorks. 0 = (default) the VxWorks time and date must be initialized by some other means to be determined by the user. 1 = the Windows time and date values will be transferred to VxWorks once when the real-time system is started. See Real-time Clocks and Time/Calendar functions.
ClockSyncInterval	dword	Specifies the interval in seconds between synchronizing updates. Default = 2. Each time this interval expires, time and date information between the two systems will be compared, if they differ, the master clock's values will be transferred to the subordinated system. See Real-time Clocks and Time/Calendar functions.
RtosProcessor	dword	In a multiprocessor system, this parameter specifies the processor on which the real-time operating system will run. A value of 0 (default) selects the first CPU (CPU1). In a single-CPU system, this value must be set to 0 (zero).

The rtosbsp Key

The user may define the following entries for the real-time Board Support Package (BSP) under the [***rtosbsp***] key:

Entry Name	Type	Description

Entry Name	Type	Description
Bootline	string	This parameter provides the user an opportunity to specify or override the default VxWorks boot line. See following text*.
NetworkAddress	string	This parameter permits the user to specify or override the default Network Adapter Address / MAC address. See following text**
atsmnetNumCluster	dword	This parameter determines the number of network clusters for the virtual network device.

*Bootline

As shown in the following example, the user may override the default VxWorks boot line by coding his/her own under the *[rtosbsp]* key:

```
[rtosbsp]
"Bootline" = "shm(0,1)pc:vxWorks  h=192.168.0.1
             e=192.168.0.2 u=target pw=vxworks"
```

**NetworkAddress

The Virtual Network adapter address [also known as the MAC (Media Control Address) address] is generally specified in the following format:

AA-BB-CC-DD-EE-02

During system startup, you can override the default (BSP) MAC address by providing the *NetworkAddress* parameter under the *[rtosbsp]* key. In the following example, the MAC network address is set to the value

00-02-E2-00-00-01

```
[rtosbsp]
"NetworkAddress" = "00-02-E2-00-00-01"
```

20 Interrupt configuration file (interrupt.config)

By default all interrupt settings are determined automatically.
Optionally you can either turn off automatic configuration at all or override one or more single interrupt configuration settings.
The properties for each real-time interrupt can be set herein. The entries are to be made in the interrupt configuration file ***interrupt.config***. You can edit this ordinary ASCII text file with any basic text editor.
For each interrupt you intend to use there are three configuration parameters:

Interrupt priority

Interrupts must be assigned a priority between the values of 0 and 12 (hexadecimal 0 to C). The lowest priority is zero (0); the highest is twelve (C). Lower priority interrupts can only be interrupted by higher priority interrupts.

Important: Each interrupt level must be assigned a unique priority; i.e., do not assign the same priority to more than one interrupt.

Trigger mode

With this parameter, you specify whether an interrupt is level-triggered (value = 1) or edge-triggered (value = 0).

Polarity

For level-triggered interrupts, specifying a polarity value of 0 implies that the interrupt will be expressed when the input signal falls below a certain threshold level. Correspondingly, assigning a polarity value of 1 to a level-triggered interrupt, causes the interrupt to be expressed when the input signal rises above the threshold.

For edge-triggered interrupts, specifying the mode as 0, implies that the interrupt will be expressed at the falling edge of the input signal as it transitions from high to low. If you specify the polarity value as 1, the interrupt will be expressed at the rising edge of the signal.

—

The following table describes the syntax of the various entries in the ***interrupt.config*** file. All parameters must be entered under an [*interrupt*] keyword:

Entry Name	Type	Description
AutoConfig	dword	If set to 1 the parameters of all interrupts are determined automatically.
PriorityInterrupt#	dword	This parameter identifies an interrupt number and its priority. To specify the interrupt number, add the number to the entry name, replacing the ‘#’ sign with a number from 0 to 23. The priority must then be written as a hexadecimal value ranging between 0 and C. Example: To assign a priority of 6 to interrupt number 18, you would make the following entry: ”PriorityInterrupt18” = dword:6
TriggerModeInterrupt#	dword	The trigger mode is specified in a fashion similar to the interrupt priority (described above). To specify an <u>edge-triggered</u> interrupt, enter a value of 0; to specify a <u>level-triggered</u> interrupt, enter 1. Example: To specify edge-triggering for interrupt number 22, you would enter the following: ”TriggerModeInterrupt22” = dword:1
PolarityInterrupt#	dword	The polarity for a specific interrupt is specified much like the two preceding parameters. To indicate that an interrupt is to become active when its input signal changes from low to high, enter a value of 1; when an interrupt should be activated when its input signal changes from high to low, enter 0. Example: Interrupt 8 is to become active when its input signal goes from high to low, you would enter the following: ”PolarityInterrupt8” = dword:0

21 Uploader Utility Program

21.1 Introduction to Uploader Utility

After VxWin has been installed, you can use the Uploader Utility program to start the real-time system. By taking advantage of command-line parameters, you can cause it to perform a variety of functions.
The tool consists of two files:

- ***UploadRTOS.exe*** (Command line program)
- ***UploadRTOS.dll***

—

When the Uploader Utility is called to start VxWin, it reads and processes its own command line options. It also reads and processes configuration parameters from the system configuration file - ***vxwin.config*** (default). Among other things, it extracts the VxWorks prototype boot line from the configuration file and writes it into the RAM variable ***sysBootLine*** (defined in the VxWorks library – sysLib.c), which VxWorks uses to configure itself. After the executable VxWorks image has been loaded, it is given control.
Here is a summary of the services that the Uploader Utility performs:

- Reserves memory for VxWorks,
- Loads a VxWorks image into memory,
- Defines physical memory areas for use by VxWorks,
- Dynamically creates a VxWorks' boot line by using information extracted from the configuration file,
- Initiates execution of the VxWorks image(*),
- Terminates a running VxWorks session and releases any system resources VxWin may have acquired.

21.2 Uploader Utility – Command Line Options

The following commandline options are available:
Command-line Syntax: UploadRTOS [Options] Image

Option	Description
Image	Specifies the file-name of the VxWorks image to be loaded into program memory. **Information**: An image file may be in a.out or binary format. The format will be automatically detected.
-?	Show help: Displays the Uploader's command-line syntax and options on the user's screen.
-x	Terminates VxWorks and releases the system resources used by the real-time system.
-config[FILENAME]	Specifies the filename of a VxWin system configuration file. See System configuration file (vxwin.config). The default filename is *vxwin.config*
-vxwin	To run VxWin RT, this option must be set.
-nosleep	No waiting time before the Uploader Utility has finished running.

Examples:

- In the following example, the specified VxWorks image will be loaded:

```
UploadRTOS.exe –vxwin C:\Tornado\target\config\VxWin-RT\vxWorks
```

- In this example, the specified VxWorks image will be loaded. A configuration file is also specified.

```
UploadRTOS.exe –vxwin –config c:\Config\vxwin.config   C:\test\vxWorks
```

- And in this example, the VxWorks that is currently running will be terminated and any memory it had acquired will be released.

```
UploadRTOS.exe -vxwin –x
```

21.3 Developing your own Uploader Utility

The Uploader Utility is a small executable (.exe) program that transfers command line options to a predefined data structure, to make it available to a function in the upload library (DLL).
Because some users may want to closely integrate the functions provided by the VxWin Uploader Utility with their own applications, all project files required to do so are supplied in the standard VxWin software release. The files are laid out for the Microsoft ***Visual C++*** programming environment.

Files pertinent to Uploader utility

The files listed in the following table are located in the following directory:
[Install Directory] /Samples/Windows/UserUpload

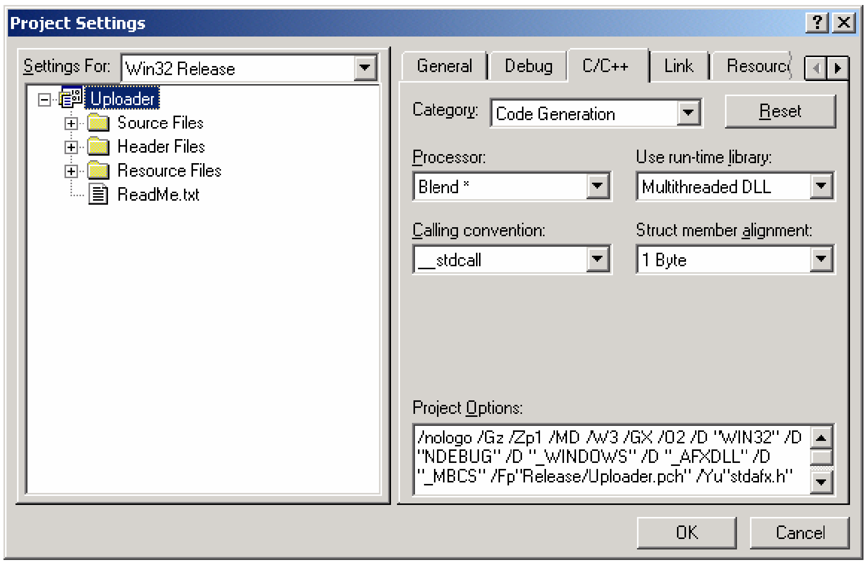
File	Description
UploadRtosExe.dsw	MS-Developer Studio 6 Workspace to generate a Demo-Uploader.
UploadRtosExe.dsp	MS-Developer Studio 6 Project to generate a Demo-Uploader.
Upvxwin32.cpp	Source code to VxWin RT Uploader Utility.
Upvxwin32.h	Header file for VxWin RT Uploader Utility.
UpRtos32.rc	Resources for VxWin RT Uploader Utility.
Resource.h	Header file for Uprtos32.rc

The Demo-Uploader source code - *Upvxwin32.cpp* - corresponds to the Uploader - *UploadRtos.exe* - installed in the *BIN* directory.

Considerations for proprietary Loader

- The development project was created using Microsoft Visual Studio Version 6.0, C++. If you use a different development environment, such as *Microsoft .NET*, the project may have to be modified or entirely refashioned.
- Structures must be defined with 1-byte alignments.
- The Uploader DLL delivered with VxWin RT, does not support the VxWin RTAcc real-time environment based on the NMI signal.
- Earlier versions of the Upload VxWin function are still supported to facilitate the porting of existing C/C++ source code. Within the Uploader DLL, however, all such calls are translated to the latest Uploader function, ***UploadVxWin4***.
- If you are writing an custom Uploader program for the first time, be sure to use ***UploadVxWin4.***

The most important ***Project Settings*** for generating a proprietary Uploader are illustrated in the following screenshot. Note ***Project Options*** under the C/C++ tab, ***Category*** (Code Generation), ***Calling convention*** (_stdcall) and ***Struct member alignment*** (1 Byte)...



Project Settings

Settings For: Win32 Release

Uploader
Source Files
Header Files
Resource Files
ReadMe.txt

General | Debug | C/C++ | Link | Resourc

Category: Code Generation

Reset

Processor: Blend *

Use run-time library: Multithreaded DLL

Calling convention: __stdcall

Struct member alignment: 1 Byte

Project Options:

```
/nologo /Gz /Zp1 /MD /W3 /GX /O2 /D "WIN32" /D
"NDEBUG" /D "_WINDOWS" /D "_AFXDLL" /D
"_MBCS" /Fp"Release/Uploader.pch" /Yu"stdafx.h"
```

OK Cancel

21.4 User-Interface to the UploadVxWin DLL

UploadVxWin4

This function carries out one of the four related actions in the structure-element ***wCommand***, as shown below.

```
DWORD    UploadVxWin4
          (
          UPLOAD_PARAMETERS_4*    pUploadPar,
          ostream*          pOutputStream
          );
```

PARAMETERS

pUploadPar

[in] – A pointer to a multi-element structure as shown below.

pOutputStream

[in] – A pointer to standard output device.

RETURN

OK, if successfully executed; otherwise, an Error Code will be returned.
All possible Error Codes are listed in the header file rte.h.

PROGRAM DETAIL

Windows header:	upvxwin32.h
Windows code:	UploadRTOS.dll

—

Upload Parameters Structure (UPLOAD_PARAMETERS_4)

WORD wCommand

Action to be performed:

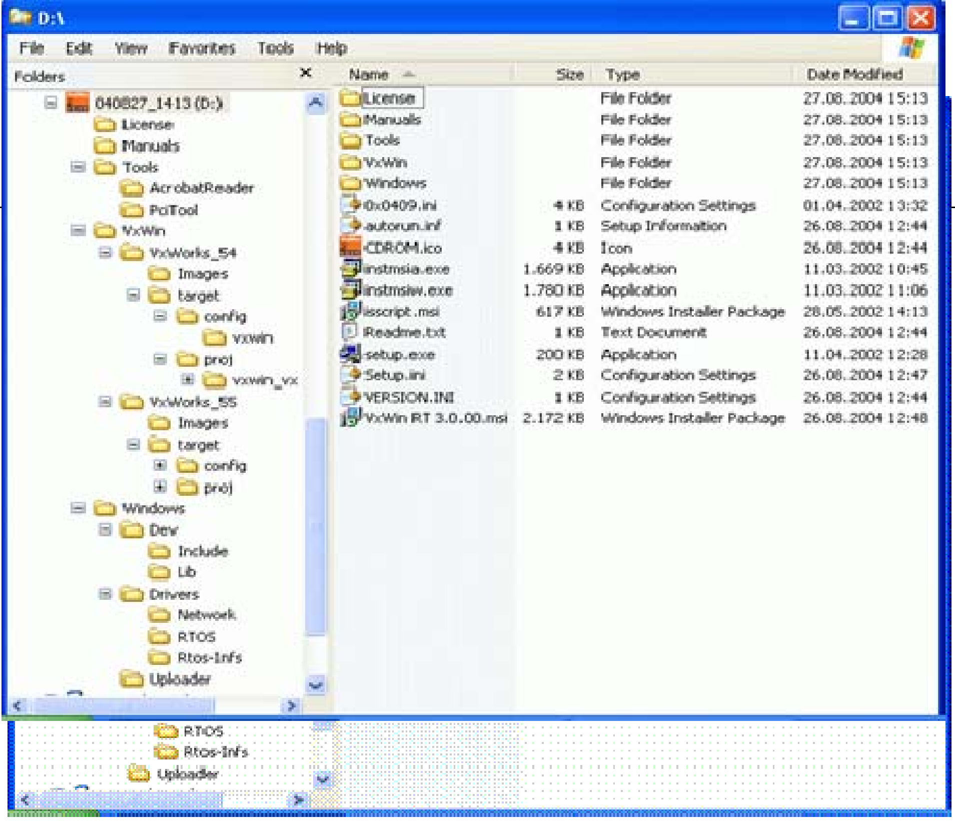
- COMMAND_COLDSTART (0) = coldstart VxWorks
- COMMAND_TERMINATE (1) = terminate VxWorks

int nOsVersion
Version of real-time OS: 2 (for VxWorks)

char* szImageName
Filename of VxWorks coldstart-image, valid for COMMAND_COLDSTART

char* szConfigFile
Filename of VxWin configuration file, e.g., *vxwin.config*

DWORD adwSpare[10]
Unused. Reserved for future applications.



```
⊟ 040827_1413 (D:)
    License
    Manuals
  ⊟ Tools
      AcrobatReader
      PciTool
  ⊟ VxWin
    ⊟ VxWorks_54
        Images
      ⊟ target
        ⊟ config
            vxwin
        ⊟ proj
          ⊞ vxwin_vx
    ⊟ VxWorks_55
        Images
      ⊟ target
        ⊞ config
        ⊞ proj
  ⊟ Windows
    ⊟ Dev
        Include
        Lib
    ⊟ Drivers
        Network
        RTOS
        Rtos-Infs
      Uploader
```

Name	Size	Type	Date Modified
License		File Folder	27.08.2004 15:13
Manuals		File Folder	27.08.2004 15:13
Tools		File Folder	27.08.2004 15:13
VxWin		File Folder	27.08.2004 15:13
Windows		File Folder	27.08.2004 15:13
0x0409.ini	4 KB	Configuration Settings	01.04.2002 13:32
autorun.inf	1 KB	Setup Information	26.08.2004 12:44
CDROM.ico	4 KB	Icon	26.08.2004 12:44
instmsia.exe	1.669 KB	Application	11.03.2002 10:45
instmsiw.exe	1.780 KB	Application	11.03.2002 11:06
isscript.msi	617 KB	Windows Installer Package	28.05.2002 14:13
Readme.txt	1 KB	Text Document	26.08.2004 12:44
setup.exe	200 KB	Application	11.04.2002 12:28
Setup.ini	2 KB	Configuration Settings	26.08.2004 12:47
VERSION.INI	1 KB	Configuration Settings	26.08.2004 12:44
VxWin RT 3.0.00.msi	2.172 KB	Windows Installer Package	26.08.2004 12:48

```
        RTOS
        Rtos-Infs
      Uploader
```

22 Preparing for Installation

Before you begin

Before installing VxWin on your computer system, if you have not already done so, you might want to review the technical information presented in Principles of Operation, System requirements and Development Environments of this manual.

You can start the installation procedure by executing ***setup.exe***, located in the root folder of the installation medium. The installation procedure automatically installs all product components from the ***Programs*** folder onto the Target System.

Setting global parameters

By adding or changing entries in the system configuration file (default: ***vxwin.config***), you may modify or set certain important system parameters without having to rebuild VxWorks. This provides a good degree of flexibility and can save you much time when trying out various settings. It is even possible that by maintaining two or more configuration files, you can implement differing system requirements without having to maintain a corresponding number of loadable images.

Each time VxWin is started, the various keys and parameter entries in the configuration file are taken to be used in place of the corresponding default parameters stored in the executable image. The configuration file information is utilized by the Uploader Utility, the real-time OS System Driver and the real-time BSP.
The VxWin configuration file is a simple text file that can be revised using a simple text editor such as MS Notepad. See System configuration file (vxwin.config).

23 Installing VxWin

This chapter leads you through various steps required to install VxWin on your target system. It takes into account both a first-time installation and an installation for a target system on which VxWin was previously installed.

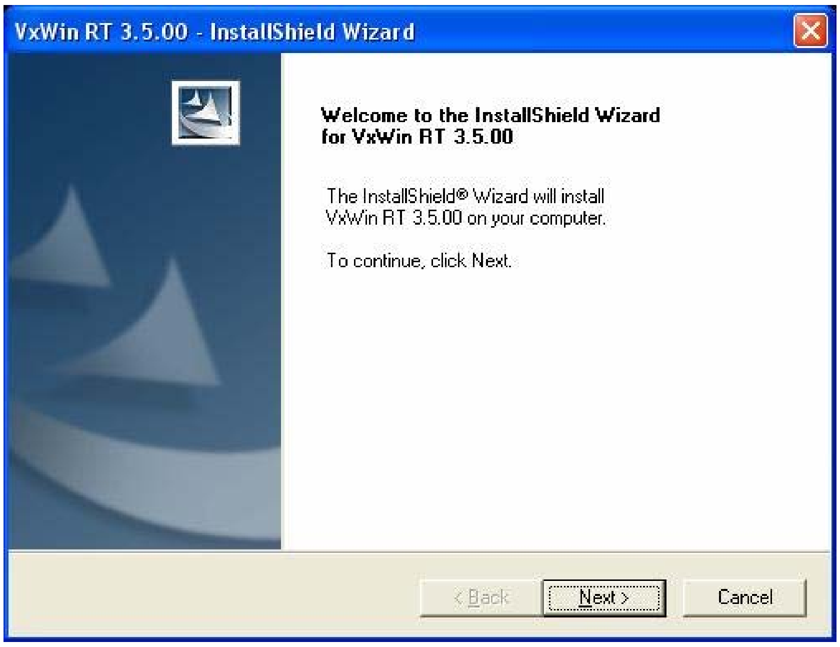
Note 1: Administrator privileges are required to install or uninstall VxWin.

Note 2: A Board Support Package can only be installed if the appropriate version of Wind River® Tornado® or Wind River Workbench® has already been installed.

Important: If an earlier version of VxWin currently resides on the target system, it is imperative that it be removed before installing a new version. The deinstallation procedure provided on the original product-delivery medium may be used for this purpose.

23.1 Automatic installation of VxWin from CD-ROM

To install VxWin, insert the installation CD-ROM in the target system's drive. Using Windows Explorer, display the contents of the root folder on the CD drive. Double-click on ***setup.exe***. After loading, the installation procedure should display the following welcome screen:

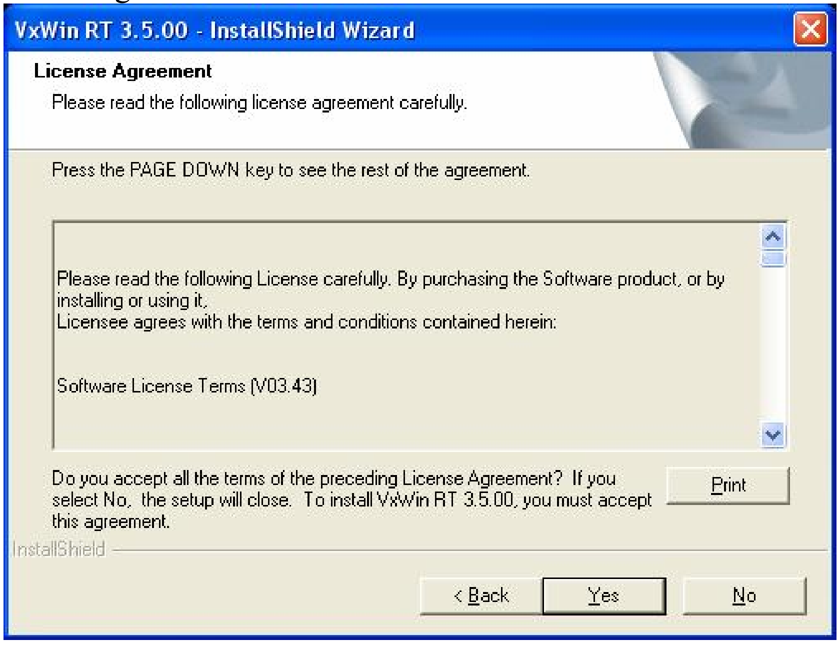

VxWin RT 3.5.00 - InstallShield Wizard

Welcome to the InstallShield Wizard for VxWin RT 3.5.00

The InstallShield® Wizard will install VxWin RT 3.5.00 on your computer.

To continue, click Next.

To continue, click the [***Next*** >] button. The next screen to appear indicates the license agreement under which VxWin is distributed.


VxWin RT 3.5.00 - InstallShield Wizard

License Agreement

Please read the following license agreement carefully.

Press the PAGE DOWN key to see the rest of the agreement.

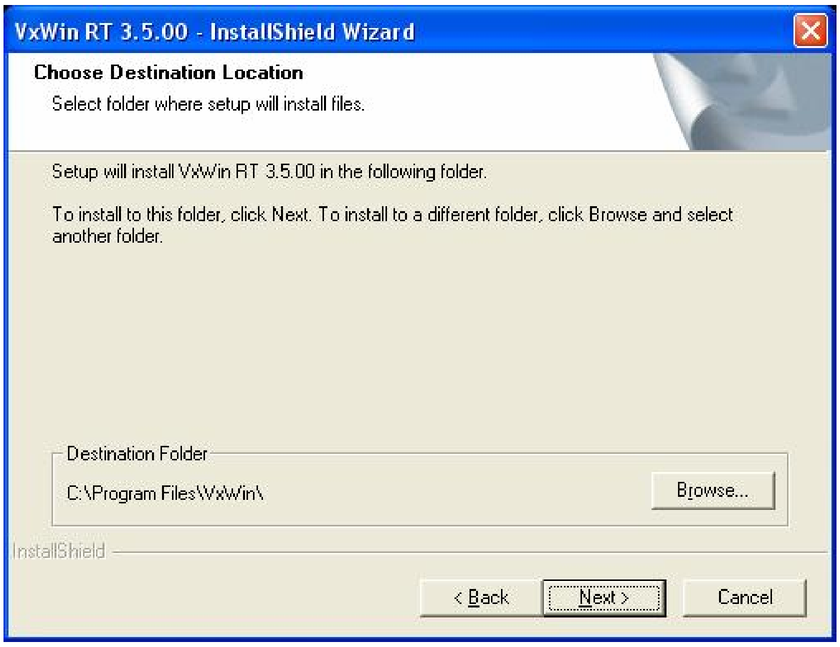
Please read the following License carefully. By purchasing the Software product, or by installing or using it,
Licensee agrees with the terms and conditions contained herein:

Software License Terms (V03.43)

Do you accept all the terms of the preceding License Agreement? If you select No, the setup will close. To install VxWin RT 3.5.00, you must accept this agreement.

Please read all terms of the agreement. Clicking ***No*** will cause the installation to terminate. By clicking ***Yes***, you declare your agreement to the terms of the license and the installation will continue.

In the next dialog box, you specify the folder in which you desire the run-time components of VxWin to be installed. If you choose to load the documentation onto your system later, it will also be written to this folder.


VxWin RT 3.5.00 - InstallShield Wizard

Choose Destination Location

Select folder where setup will install files.

Setup will install VxWin RT 3.5.00 in the following folder.

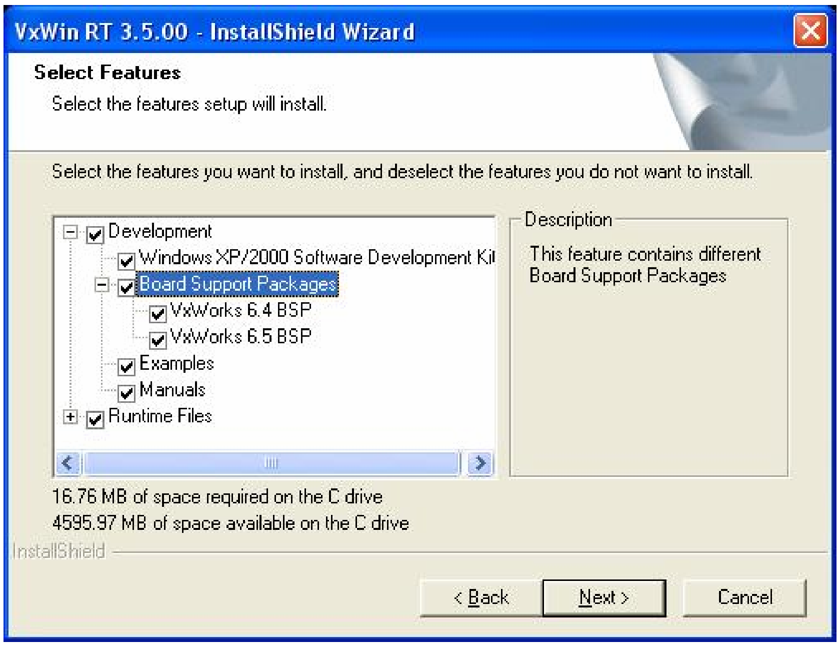
To install to this folder, click Next. To install to a different folder, click Browse and select another folder.

Destination Folder

C:\Program Files\VxWin\

If you don't accept the suggested destination folder, you may specify one of your own choice. The browse button will lead you to a screen that offers help in choosing a folder. Should you specify a path/folder that does not yet exist on the indicated drive or partition, it will automatically be created when you leave the dialog box.

After clicking ***OK***, the following dialog box appears. Here you can specify which product components you want to install. You can also choose which BSPs to install. By clicking the appropriate box, you can include or exclude an individual component from the installation. It is not possible, however, to exclude run-time components.


VxWin RT 3.5.00 - InstallShield Wizard

Select Features

Select the features setup will install.

Select the features you want to install, and deselect the features you do not want to install.

- ☑ Development
  - ☑ Windows XP/2000 Software Development Kit
  - ☑ Board Support Packages
    - ☑ VxWorks 6.4 BSP
    - ☑ VxWorks 6.5 BSP
  - ☑ Examples
  - ☑ Manuals
- ☑ Runtime Files

Description

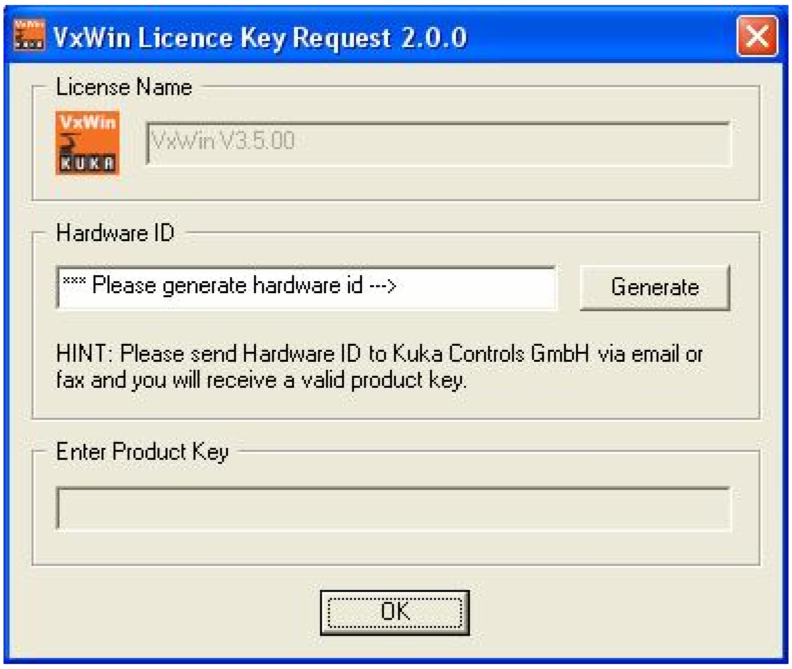
This feature contains different Board Support Packages

16.76 MB of space required on the C drive

4595.97 MB of space available on the C drive

Important: The Setup procedure checks to see which development environment(s) are installed on your system. If the development environment corresponding to a particular BSP is not installed, Setup will not load the corresponding BSP, even if you checked it.

After you click ***Next***, the VxWin license dialog will be displayed as shown in the next screenshot.


VxWin Licence Key Request 2.0.0

License Name: VxWin V3.5.00

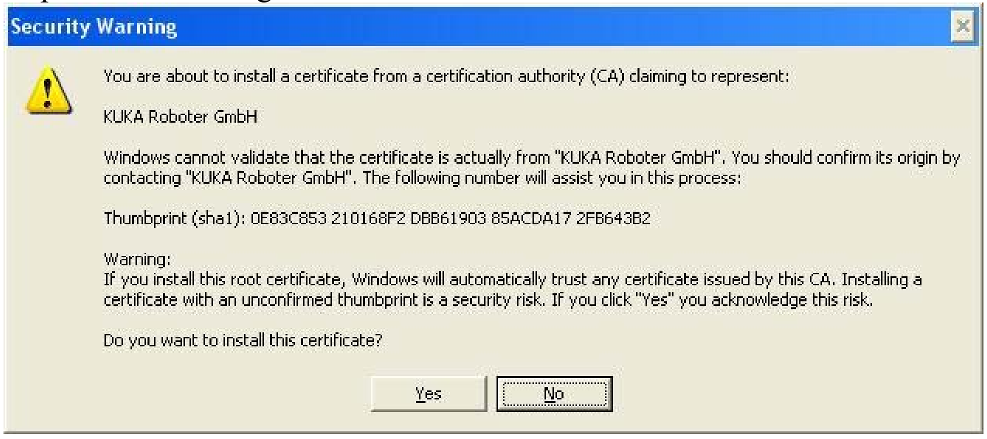
Hardware ID: *** Please generate hardware id --->

HINT: Please send Hardware ID to Kuka Controls GmbH via email or fax and you will receive a valid product key.

Enter Product Key

Important: Although the demo version requires no license, each development license must be locked onto a specific host computer.

Click the ***Generate*** button to produce the specific Hardware ID for your machine. Once you have done that, contact Kuka Roboter GmbH or your local representative to request the corresponding **Product Key**. This key enables the software to run on your computer.
After you have received the **Product Key**, type it in the information and click ***OK***. This enables *Setup* to proceed.
If you elected to install run-time components, you will at this time be prompted to install a ***certificate*** on your computer. This certificate is required to install device drivers for VxWin. When the following screenshot appears, you may respond to the dialog box with "Yes".



You are about to install a certificate from a certification authority (CA) claiming to represent:

KUKA Roboter GmbH

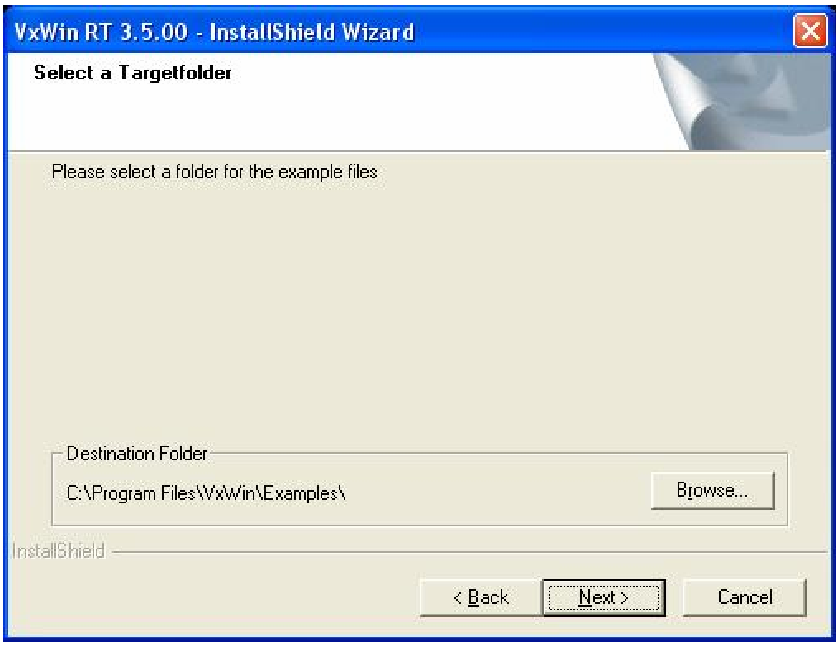
Windows cannot validate that the certificate is actually from "KUKA Roboter GmbH". You should confirm its origin by contacting "KUKA Roboter GmbH". The following number will assist you in this process:

Thumbprint (sha1): 0E83C853 210168F2 DBB61903 85ACDA17 2FB643B2

Warning:
If you install this root certificate, Windows will automatically trust any certificate issued by this CA. Installing a certificate with an unconfirmed thumbprint is a security risk. If you click "Yes" you acknowledge this risk.

Do you want to install this certificate?

Click on ***Yes*** to proceed to the next dialog window.
If you indicated earlier that you want to load the example programs, you will also be asked to accept or specify a destination directory, as indicated in the following screenshot.

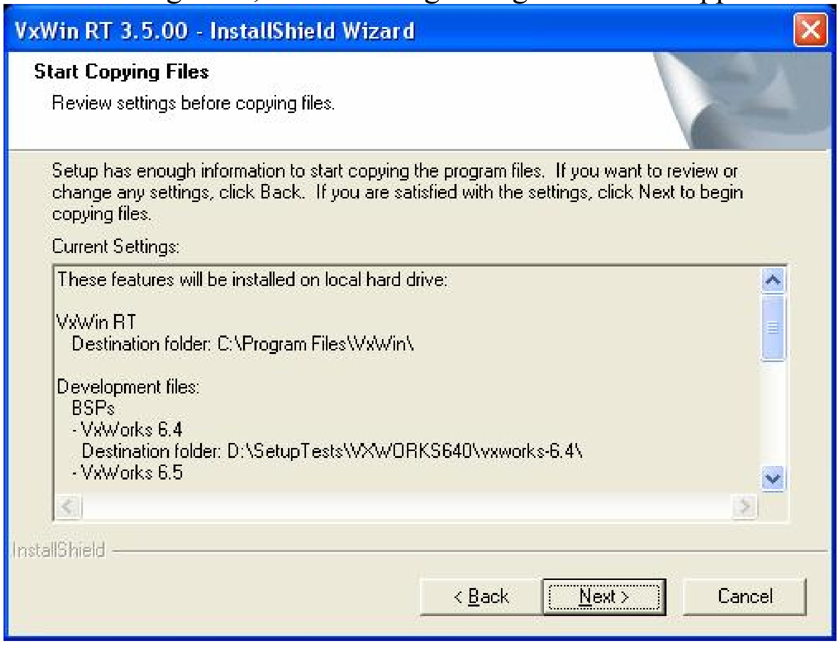


Select a Targetfolder

Please select a folder for the example files

Destination Folder
C:\Program Files\VxWin\Examples\

After clicking ***Next***, the following dialog box should appear:


VxWin RT 3.5.00 - InstallShield Wizard

Start Copying Files

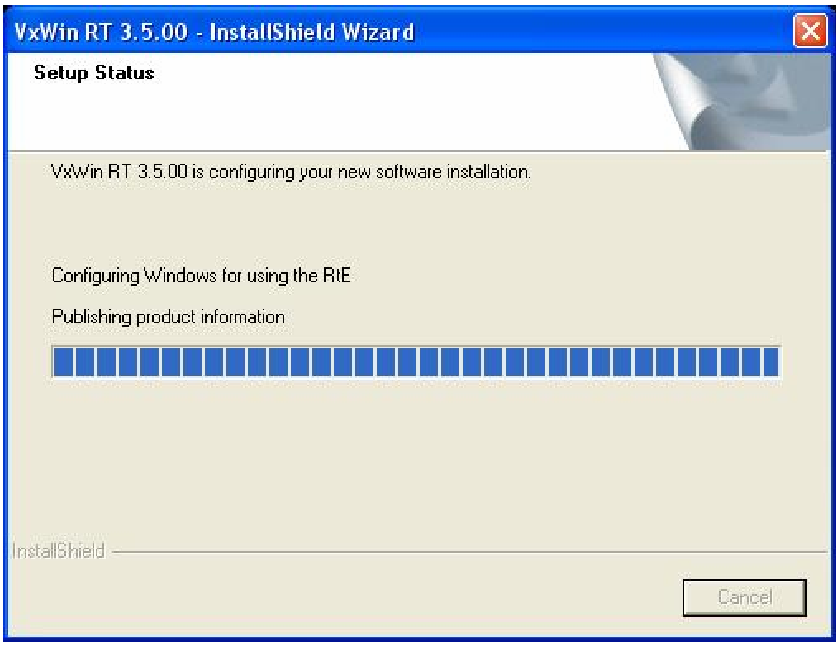
Review settings before copying files.

Setup has enough information to start copying the program files. If you want to review or change any settings, click Back. If you are satisfied with the settings, click Next to begin copying files.

Current Settings:

```
These features will be installed on local hard drive:

VxWin RT
  Destination folder: C:\Program Files\VxWin\

Development files:
  BSPs
  - VxWorks 6.4
    Destination folder: D:\SetupTests\VXWORKS640\vxworks-6.4\
  - VxWorks 6.5
```

When the following window appears, the physical installation has begun. Files will be copied into their corresponding folders. If a folder specified earlier does not yet exist, it will be created now. Both the Real-time System Driver and the Virtual Network Driver will also be installed at this time.


VxWin RT 3.5.00 - InstallShield Wizard

Setup Status

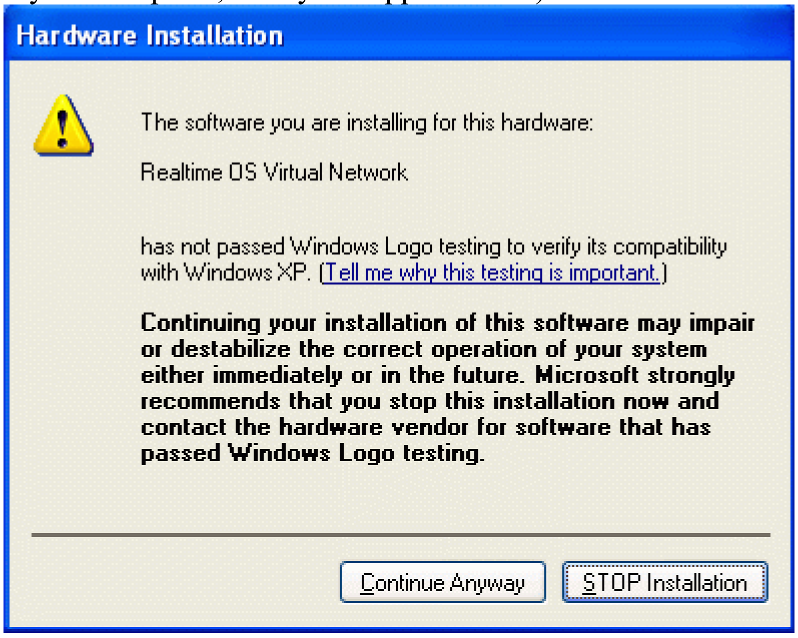
VxWin RT 3.5.00 is configuring your new software installation.

Configuring Windows for using the RtE

Publishing product information

As the foregoing Window progresses, you can follow various stages of the software installation.

If at the time of installation Microsoft has not yet provided KUKA Roboter with an official Windows logo certification, it is possible that the following message will appear on your monitor. (If Windows test for unsigned drivers has been disabled on your computer, it may not appear at all.)



The software you are installing for this hardware:

Realtime OS Virtual Network

has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)

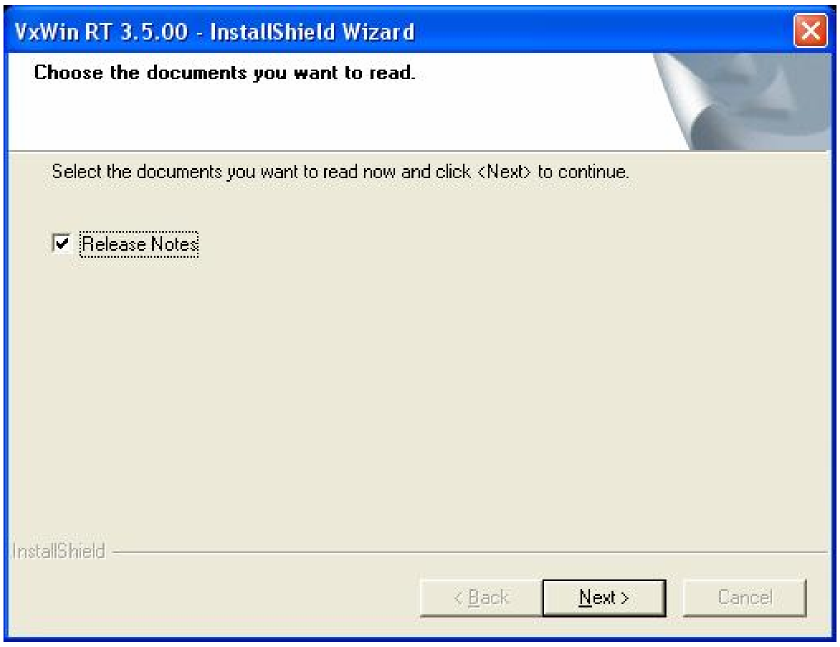
Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.

To announce the installation of the Real-time System Driver and finally the configuration of TCP/IP for the Virtual Network connection, a message will then appear. The following addresses will automatically be assigned to the IP connection:

```
    Windows XP:        "192.168.0.1"
    VxWorks:           "192.168.0.2"
```

—

You will then be asked if you would like to read the current Release Notes.

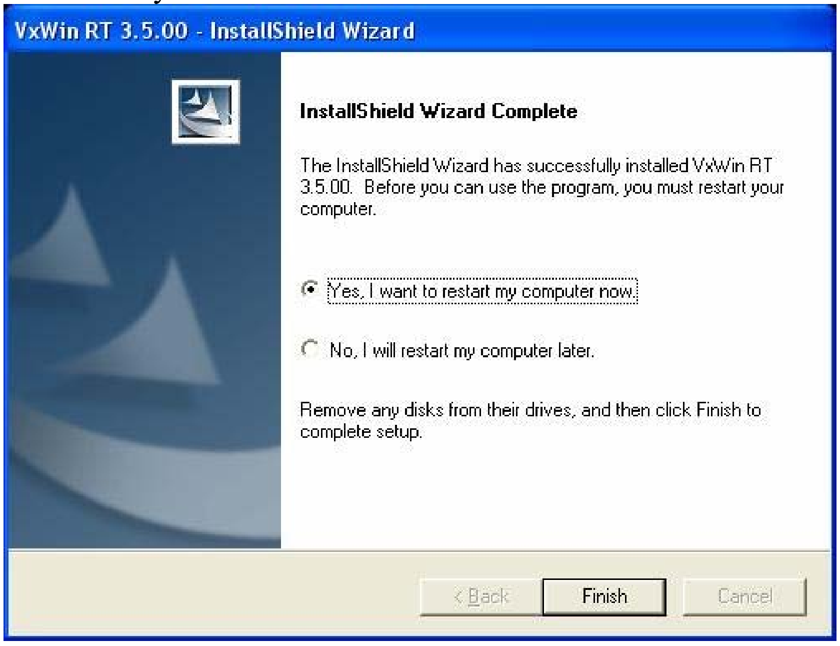

VxWin RT 3.5.00 - InstallShield Wizard

Choose the documents you want to read.

Select the documents you want to read now and click <Next> to continue.

☑ Release Notes

When you have finished with them, close the Release Notes window and the following dialog box will appear. Here you are given an opportunity either to reboot the system now or defer it until later.


VxWin RT 3.5.00 - InstallShield Wizard

InstallShield Wizard Complete

The InstallShield Wizard has successfully installed VxWin RT 3.5.00. Before you can use the program, you must restart your computer.

[✓ Yes, I want to restart my computer now.]

○ No, I will restart my computer later.

Remove any disks from their drives, and then click Finish to complete setup.

Booting causes VxWin memory allocations to become effective and the VxWin drivers to be loaded and started. After Windows has finished booting, VxWin is ready to run.

23.2 Manual product installation

23.2.1 Setting up registry entries

To be able to run VxWin, certain entries must be made in the Windows registry.

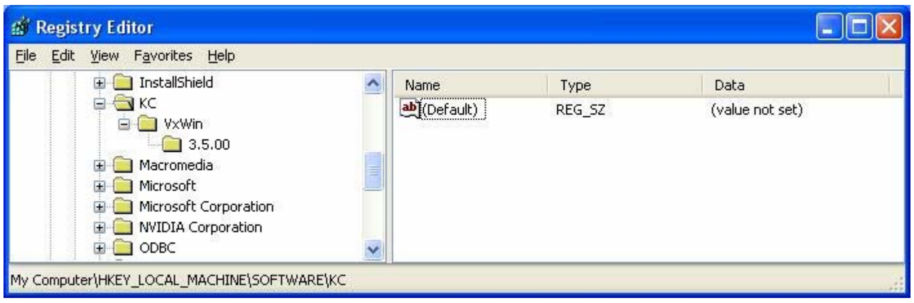
VxWin registry key

All VxWin registry settings are located in HKEY_LOCAL_MACHINE\SOFTWARE\KC\VxWin\x.y.z where x.y.z is the VxWin version. The version number can be found in the following file on the software release CD-ROM: *CD:\version.ini*.

Sample - VERSION.INI:

```
[Version]
Name=VxWin
Version=V3.5.00
Customer=KUKA
Date=20070821
Build=1
```

In the screenshot below, you can see registry settings that are located in HKEY_LOCAL_MACHINE\SOFTWARE\KC\VxWin\3.5.00:

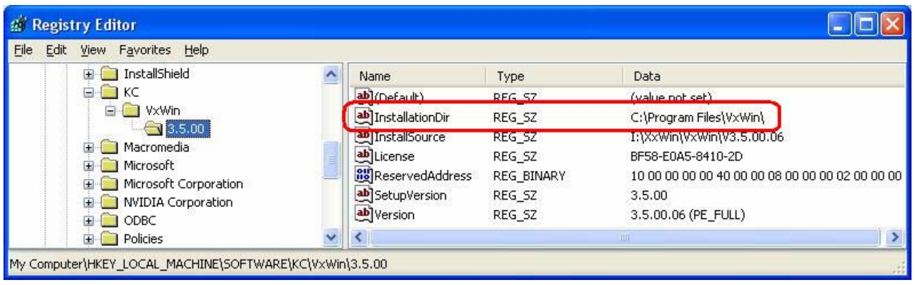


Name	Type	Data
(Default)	REG_SZ	(value not set)

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\KC

Installation Directory Information

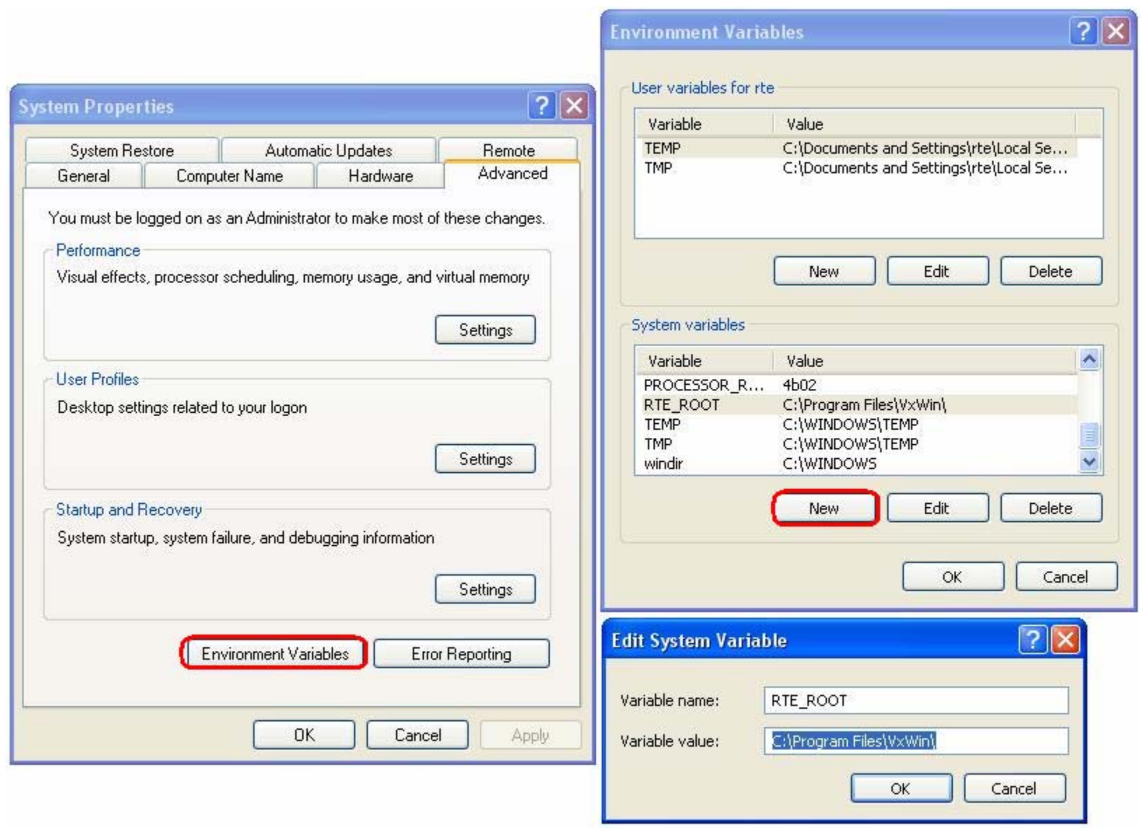
Later in this section we will copy all required files into the directory *C:\Program Files\VxWin* (the VxWin installation directory). This information must be stored in the registry key: *InstallationDir*:



Name	Type	Data
(Default)	REG_SZ	(value not set)
InstallationDir	REG_SZ	C:\Program Files\VxWin\
InstallSource	REG_SZ	I:\XxWin\VxWin\V3.5.00.06
License	REG_SZ	BF58-E0A5-8410-2D
ReservedAddress	REG_BINARY	10 00 00 00 00 40 00 00 08 00 00 00 02 00 00 00
SetupVersion	REG_SZ	3.5.00
Version	REG_SZ	3.5.00.06 (PE_FULL)

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\KC\VxWin\3.5.00

23.2.2 Setting up environment variable



User variables for rte

Variable	Value
TEMP	C:\Documents and Settings\rte\Local Se...
TMP	C:\Documents and Settings\rte\Local Se...

System variables

Variable	Value
PROCESSOR_R...	4b02
RTE_ROOT	C:\Program Files\VxWin\
TEMP	C:\WINDOWS\TEMP
TMP	C:\WINDOWS\TEMP
windir	C:\WINDOWS

Variable name: RTE_ROOT

Variable value: C:\Program Files\VxWin\

RTE-Root environment variable

For some situations it is required that a environment variable named "RTE_ROOT" is set to the VxWin installation directory. This can be done at the Windows System Properties as shown on the picture.

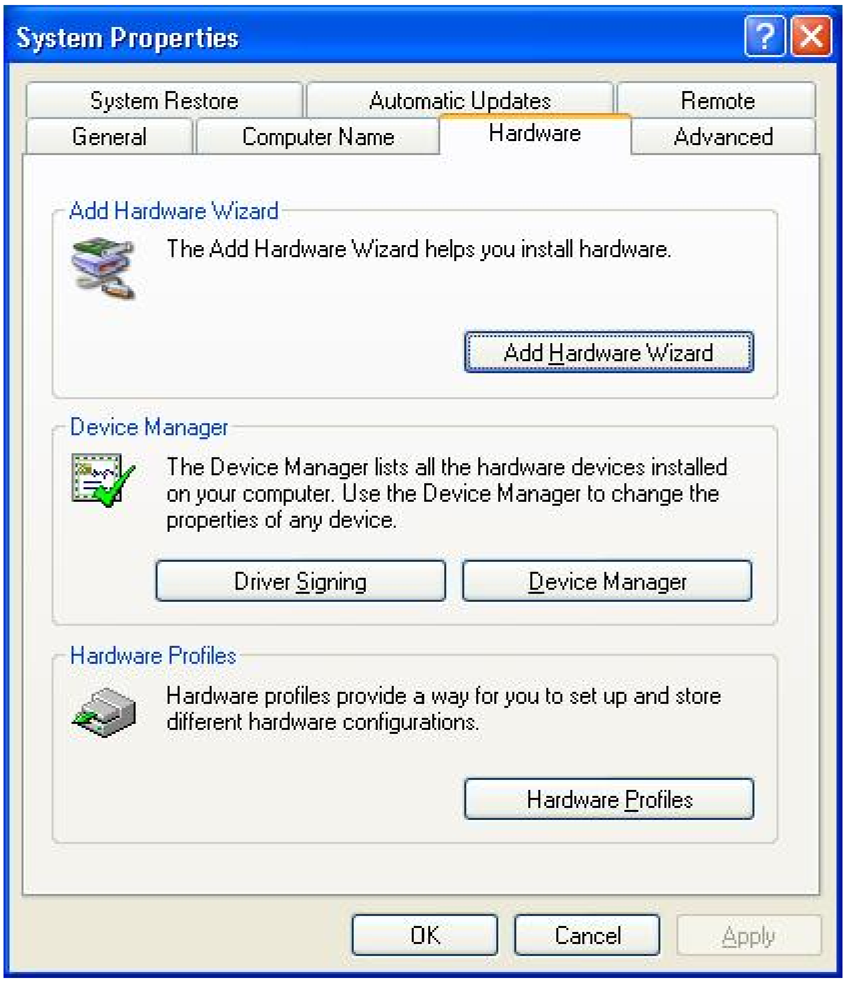
23.2.3 Setting up device drivers

Installing the Operating System Device Driver

To refresh your knowledge of the System Device Driver and the roll it plays in the VxWin system, you may now wish to review the information presented in the following chapter: Operating System device driver.
In any case, the following procedure should help you install this important software driver.

—

1 Start the ***System Properties*** dialog box and click ***Add Hardware Wizard***. A dialog box similar to that in the following screenshot should appear:


System Properties

System Restore | Automatic Updates | Remote | General | Computer Name | Hardware | Advanced

Add Hardware Wizard

The Add Hardware Wizard helps you install hardware.

Add Hardware Wizard

Device Manager

The Device Manager lists all the hardware devices installed on your computer. Use the Device Manager to change the properties of any device.

Driver Signing | Device Manager

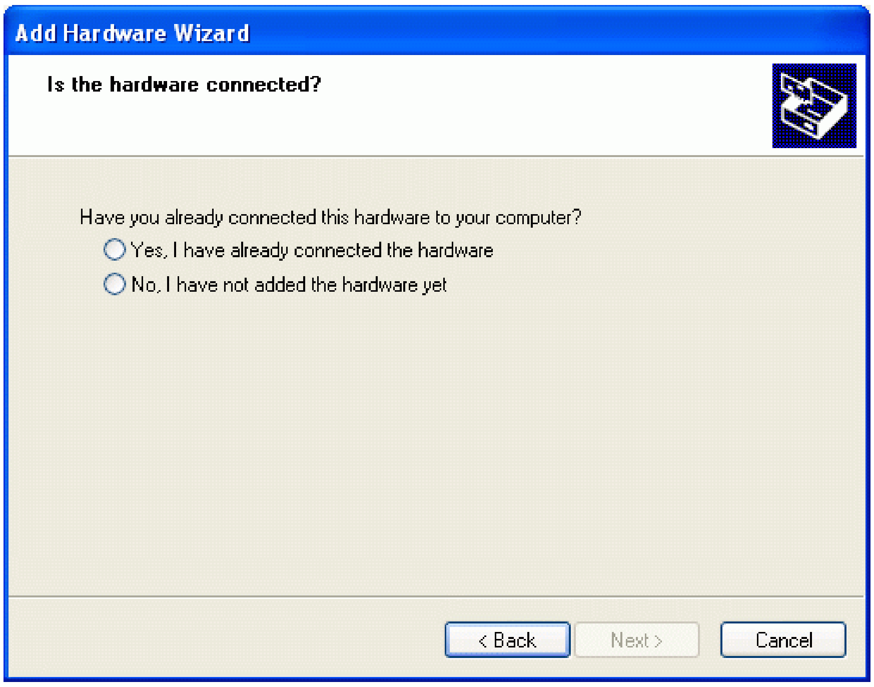
Hardware Profiles

Hardware profiles provide a way for you to set up and store different hardware configurations.

Hardware Profiles

OK | Cancel | Apply

2 The install wizard will search for newly installed hardware. At the next dialog box, select ***Yes, I have already connected the hardware***.

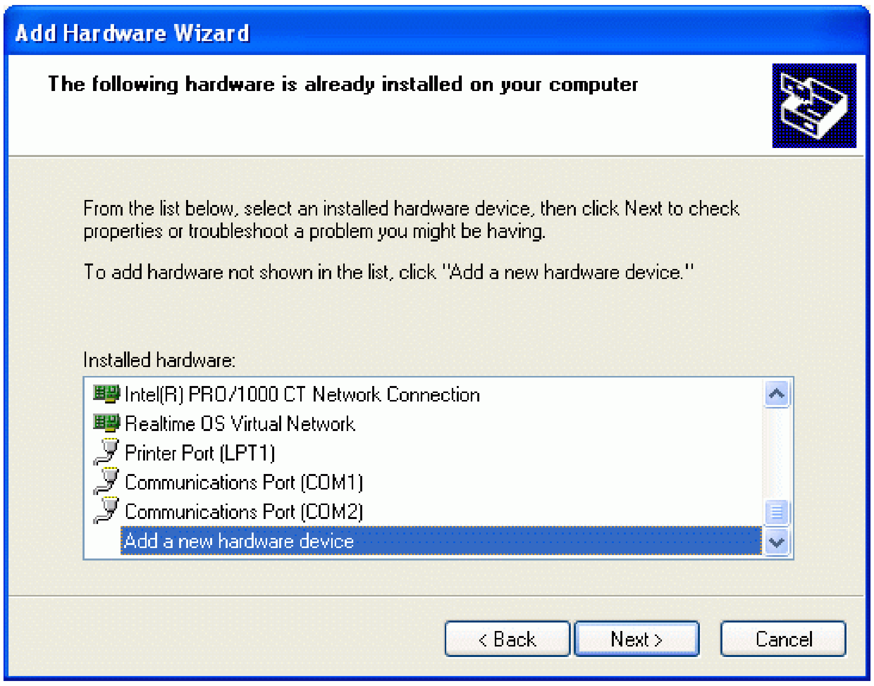

Add Hardware Wizard

Is the hardware connected?

Have you already connected this hardware to your computer?

- ◯ Yes, I have already connected the hardware
- ◯ No, I have not added the hardware yet

3 You will then be shown a list of existing hardware. Scroll to the bottom of the list, and select, ***Add a new hardware device***.


Add Hardware Wizard

The following hardware is already installed on your computer

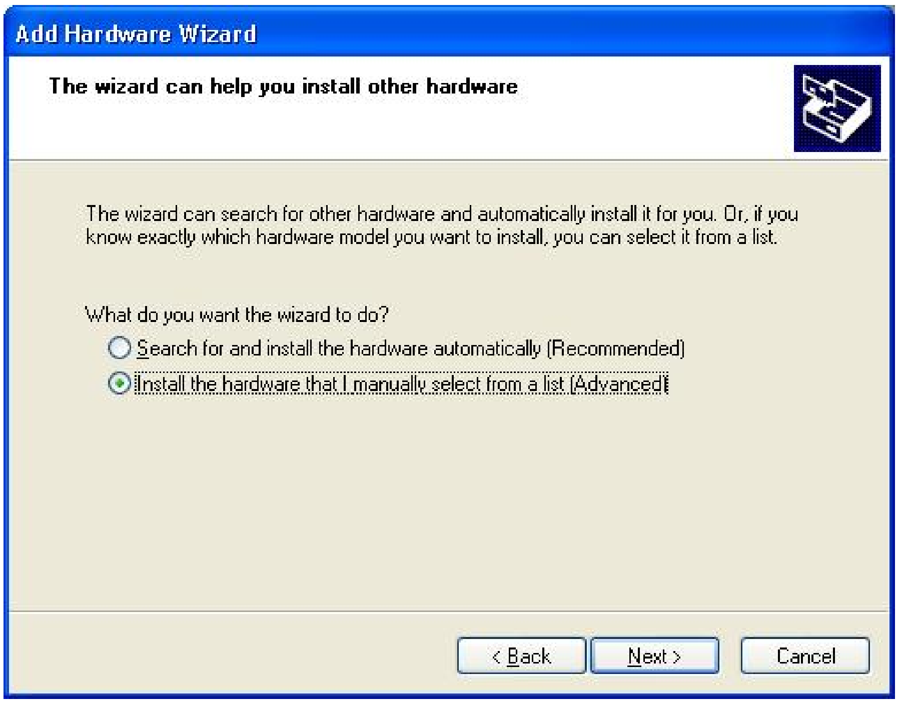
From the list below, select an installed hardware device, then click Next to check properties or troubleshoot a problem you might be having.

To add hardware not shown in the list, click "Add a new hardware device."

Installed hardware:

- Intel(R) PRO/1000 CT Network Connection
- Realtime OS Virtual Network
- Printer Port (LPT1)
- Communications Port (COM1)
- Communications Port (COM2)
- Add a new hardware device

4 Now choose the option to manually ***Install the hardware…***. Click the option as shown in the following dialog box. Click ***Next***.

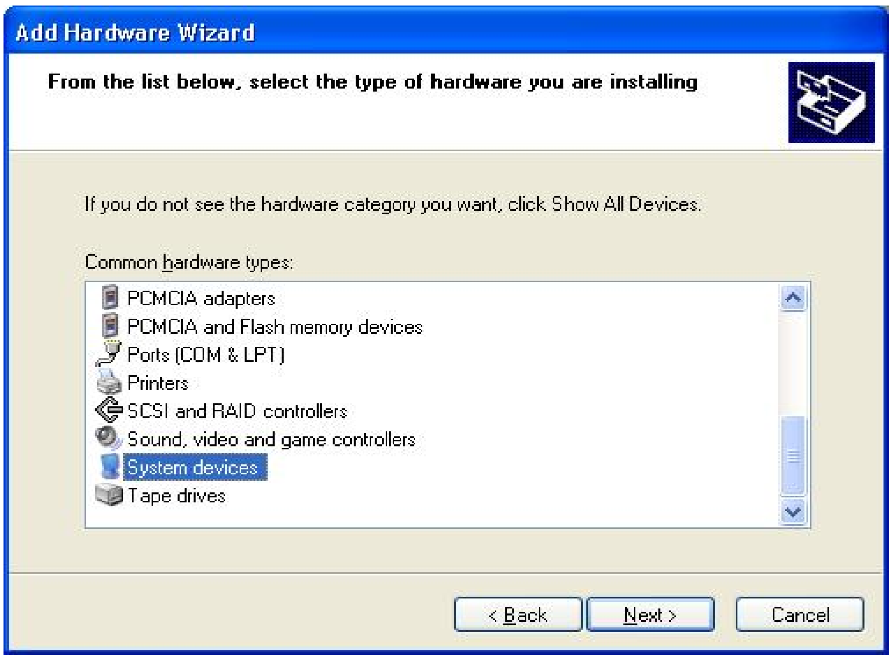


The wizard can search for other hardware and automatically install it for you. Or, if you know exactly which hardware model you want to install, you can select it from a list.

What do you want the wizard to do?

- Search for and install the hardware automatically (Recommended)
- [✓ Install the hardware that I manually select from a list (Advanced)]

5 Click ***System devices***.

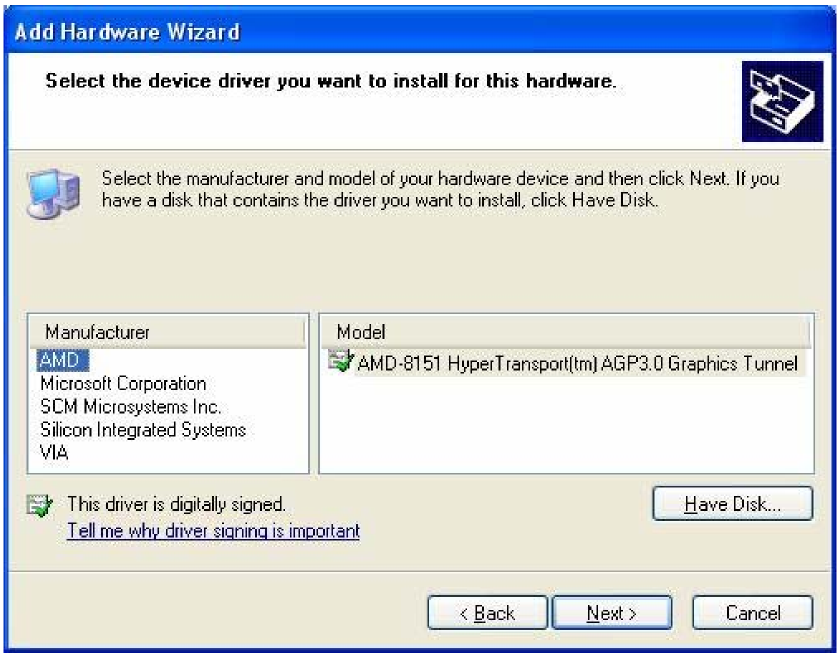

Add Hardware Wizard
From the list below, select the type of hardware you are installing
< Back
Next >
Cancel

If you do not see the hardware category you want, click Show All Devices.

Common hardware types:

- PCMCIA adapters
- PCMCIA and Flash memory devices
- Ports (COM & LPT)
- Printers
- SCSI and RAID controllers
- Sound, video and game controllers
- [✓ System devices]
- Tape drives

6. Now get the device driver from the CD-ROM; click ***Have Disk*** and then ***Next***.



Add Hardware Wizard

Select the device driver you want to install for this hardware.

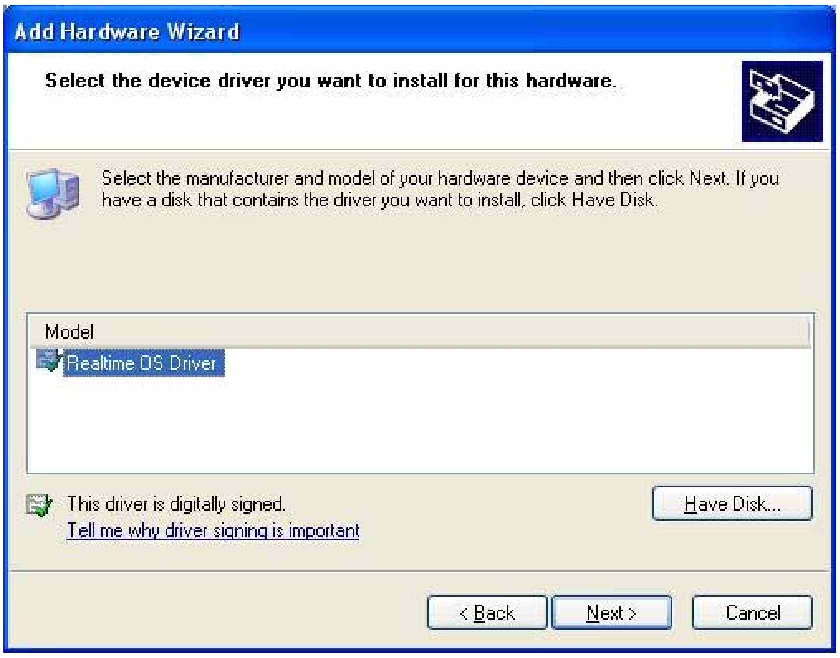
Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk.

Manufacturer	Model
AMD	AMD-8151 HyperTransport(tm) AGP3.0 Graphics Tunnel
Microsoft Corporation	
SCM Microsystems Inc.	
Silicon Integrated Systems	
VIA	

This driver is digitally signed.
Tell me why driver signing is important

Note: The driver is located in ***CD:\Windows\Drivers\RTOS***.

7 Now click ***Realtime OS Driver*** and then ***Next***.



Add Hardware Wizard

Select the device driver you want to install for this hardware.

Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk.

Model
Realtime OS Driver

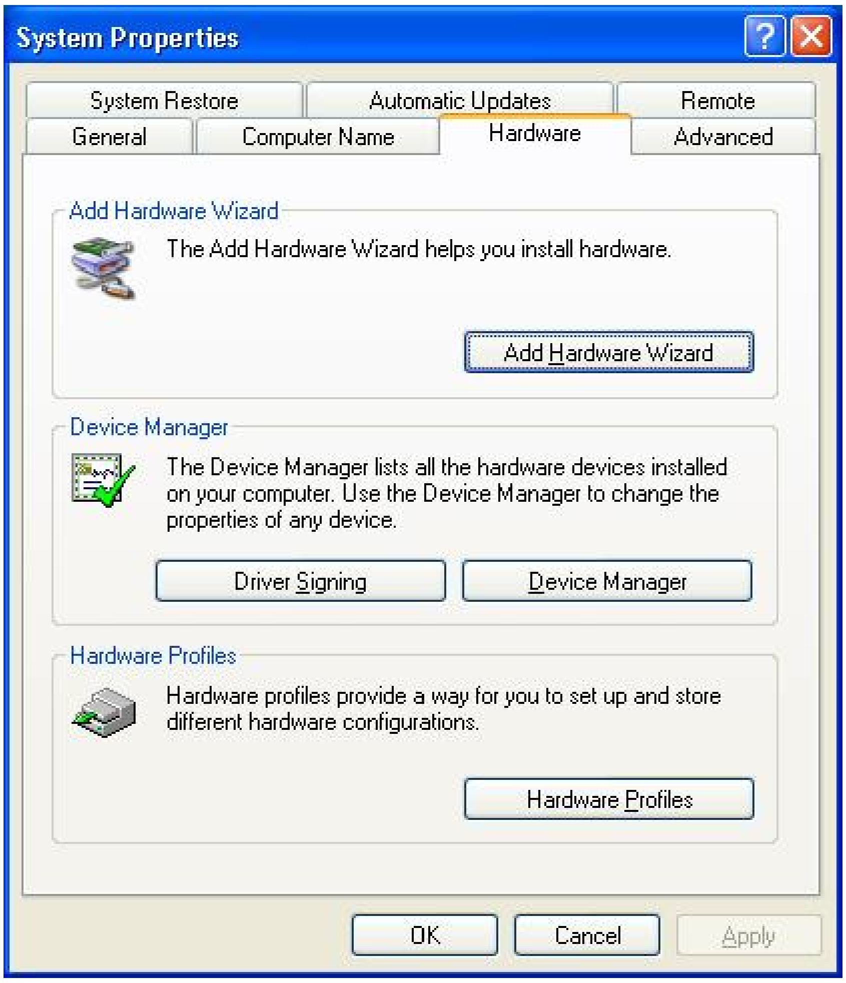
This driver is digitally signed.
Tell me why driver signing is important

Note: Although the driver may be unsigned when you install it, you should nonetheless proceed. The formal process of registering a signature with Microsoft takes some time.

Installing the Real-time Virtual Network Driver

To refresh your knowledge of the Virtual Network Driver, read the following section of this manual: Virtual Network Driver.
You are now ready to install the Virtual Network.

1. Start the System Properties dialog box and select the ***Add Hardware Wizard***.



System Properties

System Restore | Automatic Updates | Remote
General | Computer Name | Hardware | Advanced

Add Hardware Wizard

The Add Hardware Wizard helps you install hardware.

Add Hardware Wizard

Device Manager

The Device Manager lists all the hardware devices installed on your computer. Use the Device Manager to change the properties of any device.

Driver Signing | Device Manager

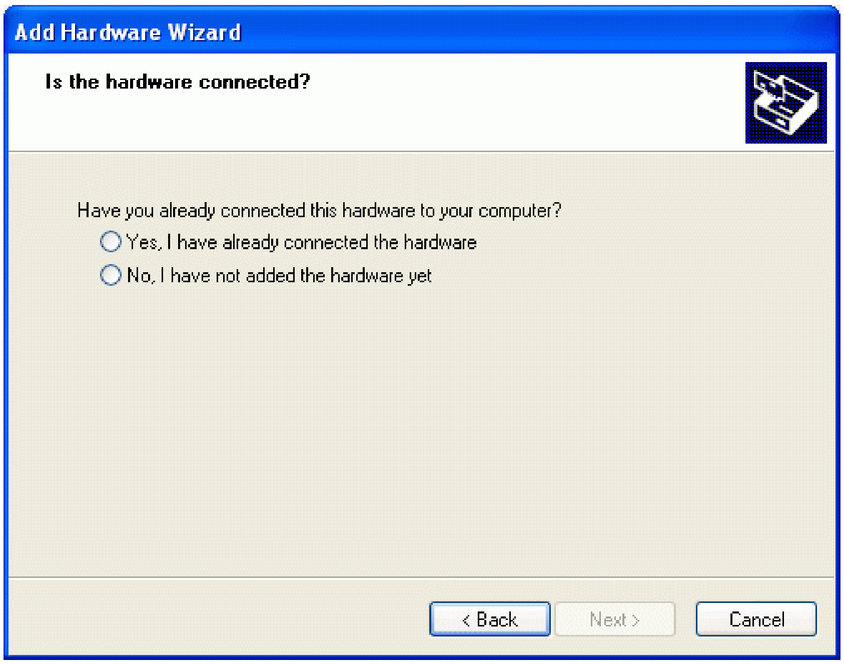
Hardware Profiles

Hardware profiles provide a way for you to set up and store different hardware configurations.

Hardware Profiles

OK | Cancel | Apply

2. The install wizard will search for newly installed hardware. At the next dialog box, select ***Yes, I have already connected the hardware***.



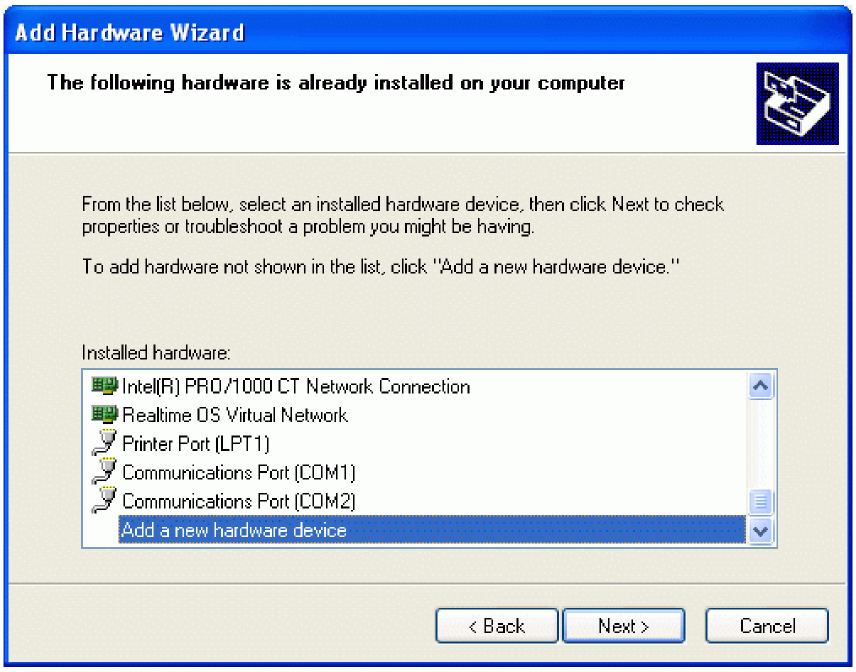
Add Hardware Wizard

Is the hardware connected?

Have you already connected this hardware to your computer?

- ○ Yes, I have already connected the hardware
- ○ No, I have not added the hardware yet

3. You will then be shown a list of existing hardware. Scroll to the bottom of the list, and select, ***Add a new hardware device***.



Add Hardware Wizard

The following hardware is already installed on your computer

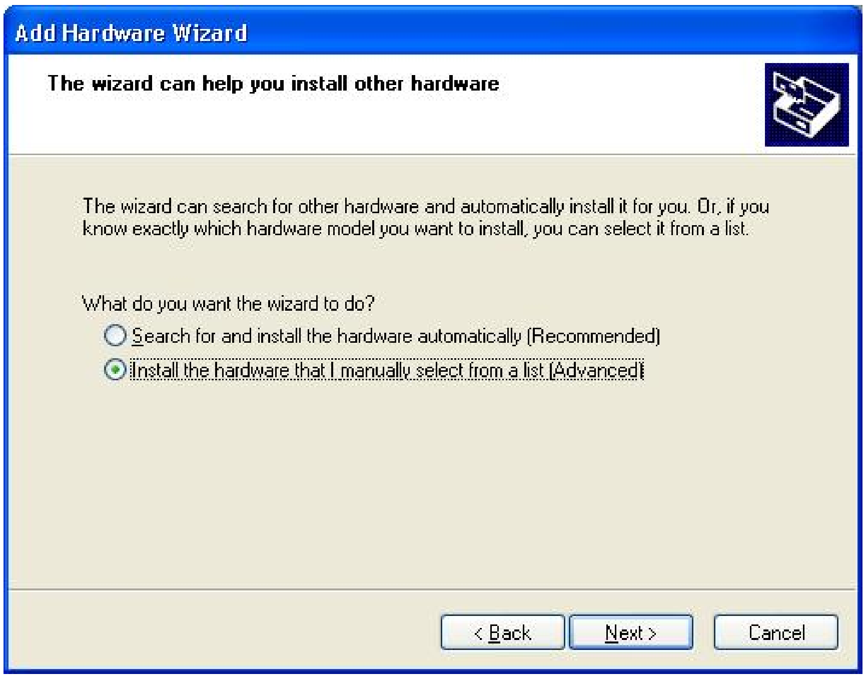
From the list below, select an installed hardware device, then click Next to check properties or troubleshoot a problem you might be having.

To add hardware not shown in the list, click "Add a new hardware device."

Installed hardware:

- Intel(R) PRO/1000 CT Network Connection
- Realtime OS Virtual Network
- Printer Port (LPT1)
- Communications Port (COM1)
- Communications Port (COM2)
- Add a new hardware device

4. Now, as indicated in the following screenshot, select ***Install the hardware manually*** and click ***Next***.


Add Hardware Wizard

The wizard can help you install other hardware

The wizard can search for other hardware and automatically install it for you. Or, if you know exactly which hardware model you want to install, you can select it from a list.

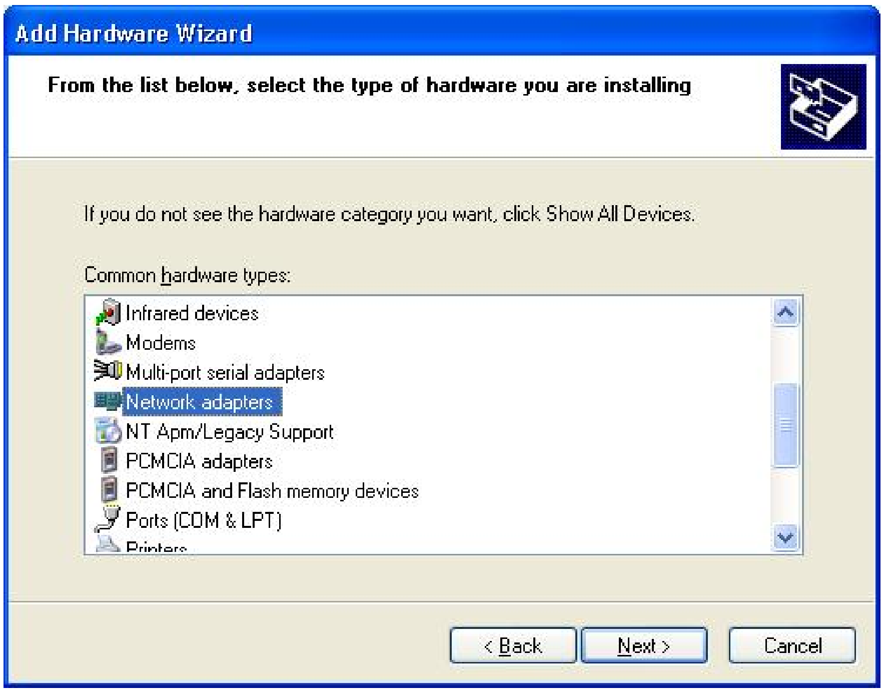
What do you want the wizard to do?

○ Search for and install the hardware automatically (Recommended)

● Install the hardware that I manually select from a list (Advanced)

< Back | Next > | Cancel

5. Click ***Network adapters***.


Add Hardware Wizard

From the list below, select the type of hardware you are installing

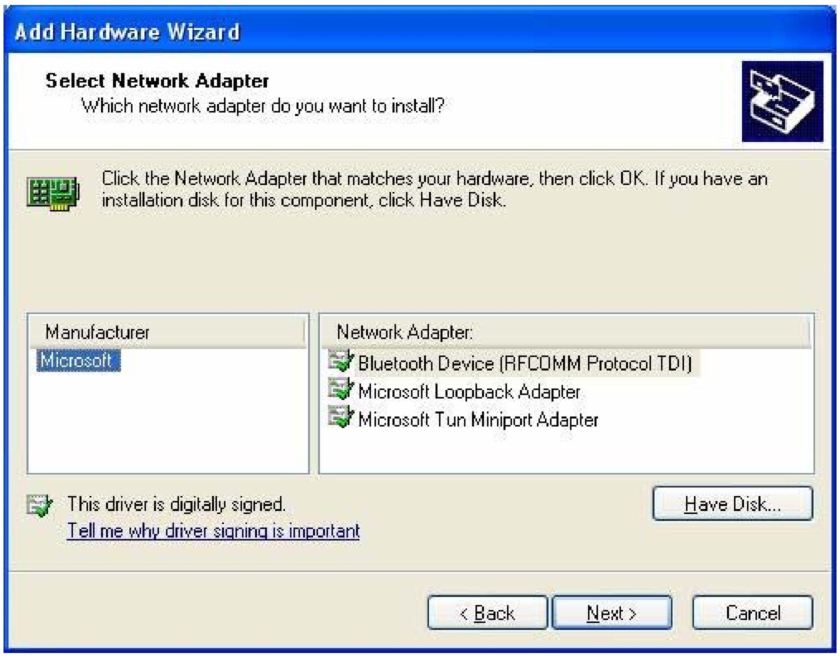
If you do not see the hardware category you want, click Show All Devices.

Common hardware types:

- Infrared devices
- Modems
- Multi-port serial adapters
- Network adapters
- NT Apm/Legacy Support
- PCMCIA adapters
- PCMCIA and Flash memory devices
- Ports (COM & LPT)
- Printers

< Back | Next > | Cancel

6. Now you have to get the device driver from the CD-ROM, click ***Have Disk*** . The driver is located in ***CD:\Windows\Drivers\Network***.


Add Hardware Wizard

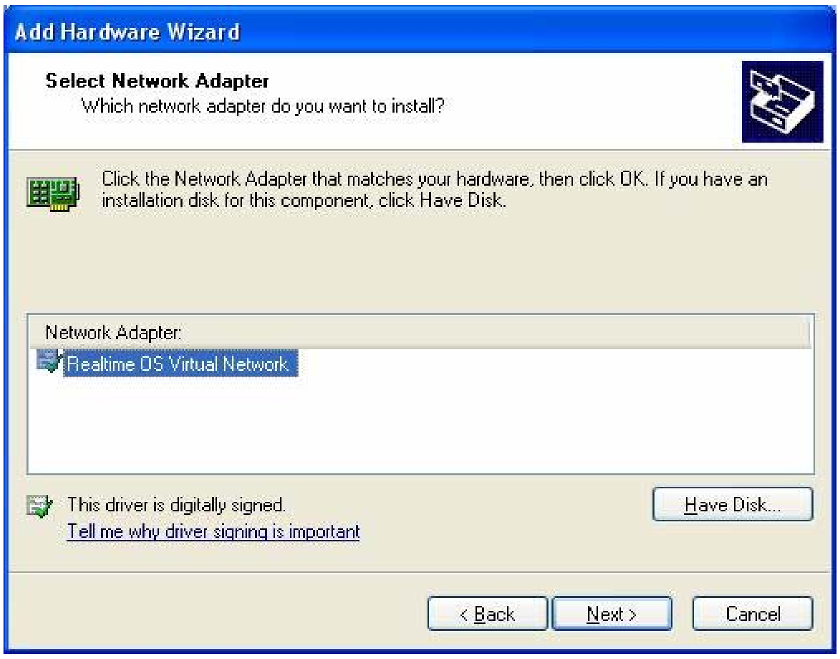
Select Network Adapter
Which network adapter do you want to install?

Click the Network Adapter that matches your hardware, then click OK. If you have an installation disk for this component, click Have Disk.

Manufacturer	Network Adapter:
Microsoft	Bluetooth Device (RFCOMM Protocol TDI)
	Microsoft Loopback Adapter
	Microsoft Tun Miniport Adapter

This driver is digitally signed.
Tell me why driver signing is important

7. Now click ***Realtime OS Virtual Network***.


Add Hardware Wizard

Select Network Adapter
Which network adapter do you want to install?

Click the Network Adapter that matches your hardware, then click OK. If you have an installation disk for this component, click Have Disk.

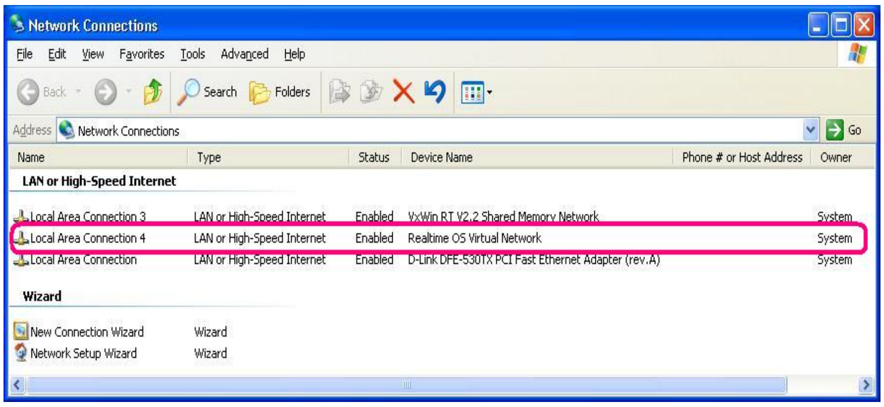
Network Adapter:
Realtime OS Virtual Network

This driver is digitally signed.
Tell me why driver signing is important

8. Reboot the computer.

9. Now It is necessary to adjust the ***IP address*** of the ***Virtual Network***:

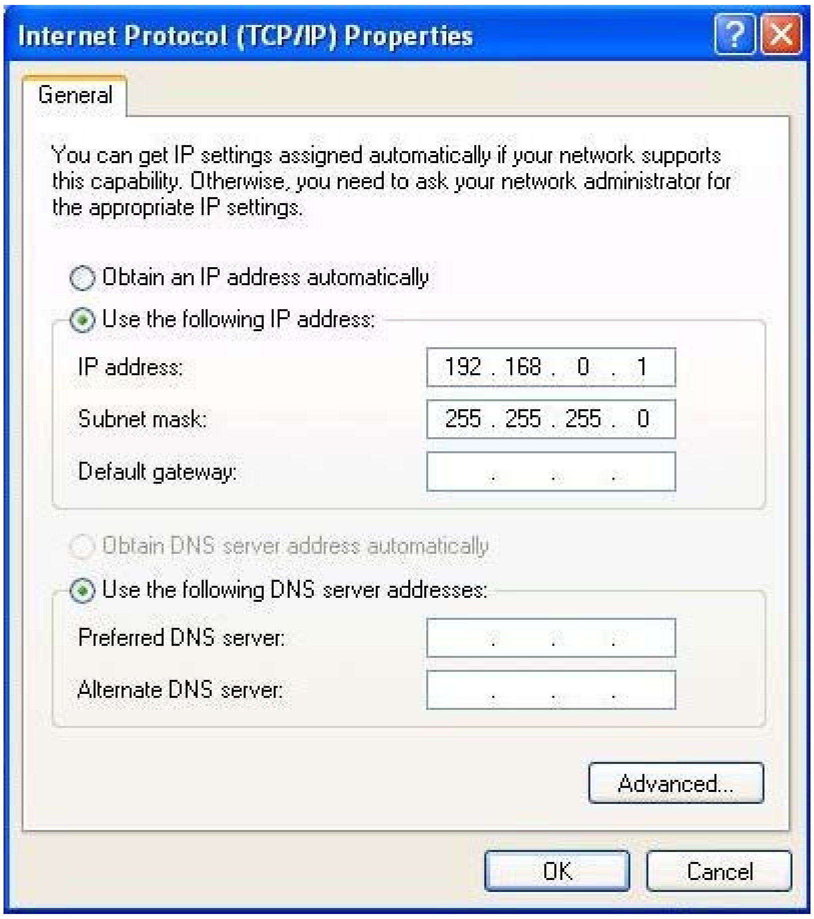
10. Open the ***Network Connections*** Window.



Address: Network Connections

Name	Type	Status	Device Name	Phone # or Host Address	Owner
LAN or High-Speed Internet					
Local Area Connection 3	LAN or High-Speed Internet	Enabled	VxWin RT V2.2 Shared Memory Network		System
Local Area Connection 4	LAN or High-Speed Internet	Enabled	Realtime OS Virtual Network		System
Local Area Connection	LAN or High-Speed Internet	Enabled	D-Link DFE-530TX PCI Fast Ethernet Adapter (rev.A)		System
Wizard					
New Connection Wizard	Wizard				
Network Setup Wizard	Wizard				

11. Adjust the IP address to the desired value (default: 192.168.0.1).



You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

Obtain an IP address automatically [✓ Use the following IP address:]

IP address: 192 . 168 . 0 . 1

Subnet mask: 255 . 255 . 255 . 0

Default gateway: . . .

Obtain DNS server address automatically [✓ Use the following DNS server addresses:]

Preferred DNS server: . . .

Alternate DNS server: . . .

Setting MAC Addresses for the Virtual Network Driver

Before the Virtual Network can run, you must assign specific MAC (Media Control Address) addresses to the Virtual Network drivers on both the VxWorks and the Windows sides of the system. These MAC addresses, written in the following format: AA-BB-CC-DD-EE-FF, must be assigned before you start VxWin.
Windows side:
On the Windows side of the system, open the Device Manager, then under "Network Adapters" select "Realtime OS Virtual Network" and click on the Advanced Properties tab. In the next dialog window, click on "Network MAC address." You can then change the MAC address to the desired value. The value you enter must be written as in the following example: FA-0E-02-E2-55-01. After entering the address, close the Device Manager.
VxWorks side:
To configure the MAC address for the VxWorks Virtual Network driver, enter an appropriate value for the ***NetworkAddress*** parameter under the [***rtosbsp***] keyword in the main VxWin configuration file, nominally ***vxwin.config***. Refer to NetworkAddress in Global parameters in the VxWin configuration file.
Example: "NetworkAddress"="00-0E-02-E2-00-02"

Once you have set MAC addresses for both sides of the system, make certain Windows Device Manager is closed before you start VxWin.

To reconfirm the MAC addresses:
Windows: To view the current MAC address on the Windows side, start the Windows command-line interpreter (DOS box) and use the "arp –a" command to display the Windows setting.
VxWorks: To view the MAC address for VxWorks, open a Telnet session and display the current setting by means of Wind River's ***arptabShow*** command. (For more information on this and other VxWorks commands, please see the appropriate Wind River Documentation, e.g., ***VxWorks Programmer's Guide*** or the ***VxWorks OS Libraries***.)

23.2.4 Installing and uninstalling the RTOS service

The real-time operating system service program that manages time, date and time-zone synchronization is located in the *VxWin* directory, typically: *C:\Program Files\VxWin*.

You can use the command “RTOSService install” to install this service and “RTOSService start” to start it without reboot. Similarly, you can uninstall it by using the commands “RTOSService uninstall” provided you have first terminated its execution. To terminate execution, you can either use the command “RTOSService stop”, in the above referenced directory or go to the Windows Administrative Tools dialog[6] and stop it there.

Without regard for these manual procedures, every time the system is re-booted, the service program will automatically be started.

23.2.5 Copy the Uploader to the hard disk

Create a new directory *C:\Program Files\VxWin*.

Copy all files located in *CD:\Windows\Uploader* into this directory.

23.2.6 Setting up Windows Firewall

The VxWin Setup creates the following firewall exceptions, which should be manually added if required:

- RtE FTP: TCP Port 21
- RtE HTTP:TCP Port 80
- RtE License Server Service C:\Program Files\VxWin\LicenseServer.exe
- RtE Remote Debug: UDP Port 17185
- RtE Telnet: TCP Port 23

All RtE exceptions are normally limited to the subnet 192.168.0.0/255.255.255.0

Setting up Memory Reservation

VxWin needs to reserve some memory from Windows. This can be initiated by calling

<UploadRtos.exe –vxwin –memcfg “-a” -nowait>

VxWin will reserve as much memory as configured in vxwin.config. After changing the memory reservation a reboot is required before the configuration is valid.

[6] To open Services, click **Start**, point to **Settings**, and then click **Control Panel**. Double-click **Administrative Tools**, and then double-click **Services**.

To remove the reservation call

<UploadRtos.exe –vxwin –memcfg “-u” -nowait>

The configuration also becomes valid after the next reboot.

Any changes to the configuration will be updated through the uploader automatically.

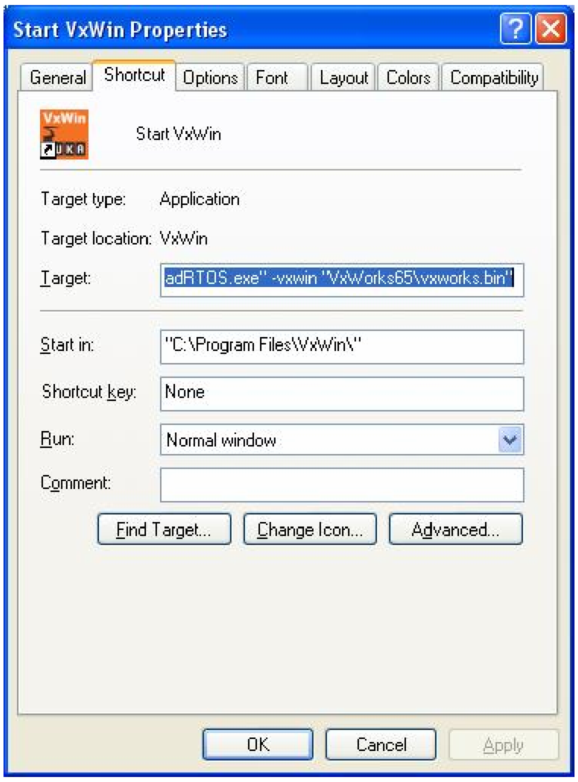
Copy the VxWorks image to the hard disk

Identify the VxWorks image file *VxWorks.bin* in the following directory on the software release CD-ROM: *CD:\VxWin\VxWokrs_xx\Images* . Then copy it into *C:\Program Files\VxWin* on your hard disk.

Create shortcuts on your desktop

a) Shortcut to start execution of VxWorks

– location:	C:\Program FilesVxWin\UploadRTOS.exe
– name:	Start VxWorks
– properties:	*target = C:\Program Files\VxWin\ UploadRTOS.exe –vxwin "VxWorks65\vxWorks.bin"*



Start VxWin

Target type: Application

Target location: VxWin

Target: adRTOS.exe" -vxwin "VxWorks65\vxworks.bin"

Start in: "C:\Program Files\VxWin\"

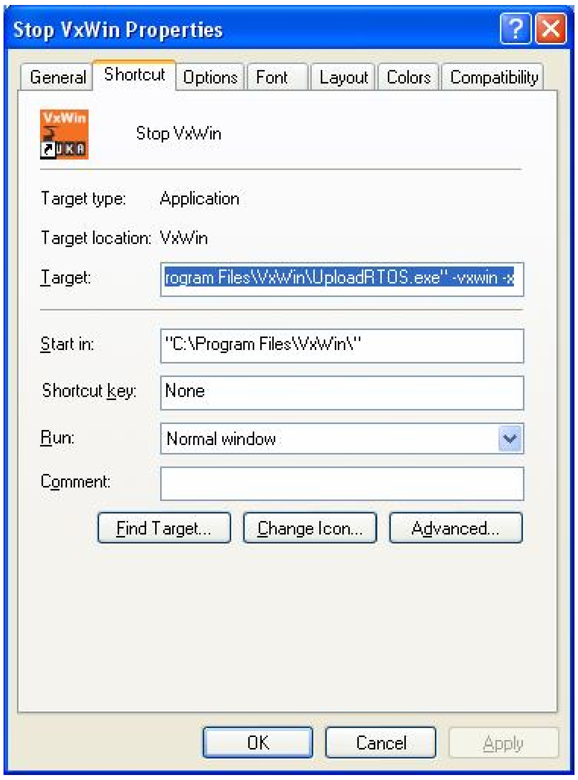
Shortcut key: None

Run: Normal window

Comment:

b) Shortcut to stop execution of VxWorks

– location:	C:\Program Files\VxWin\UploadRTOS.exe
– name:	Stop VxWorks
– properties:	target = C:\Program Files\VxWin\UploadRTOS.exe –vxwin –x


Stop VxWin Properties

Stop VxWin

Target type: Application

Target location: VxWin

Target: rogram Files\VxWin\UploadRTOS.exe" -vxwin -x

Start in: "C:\Program Files\VxWin\"

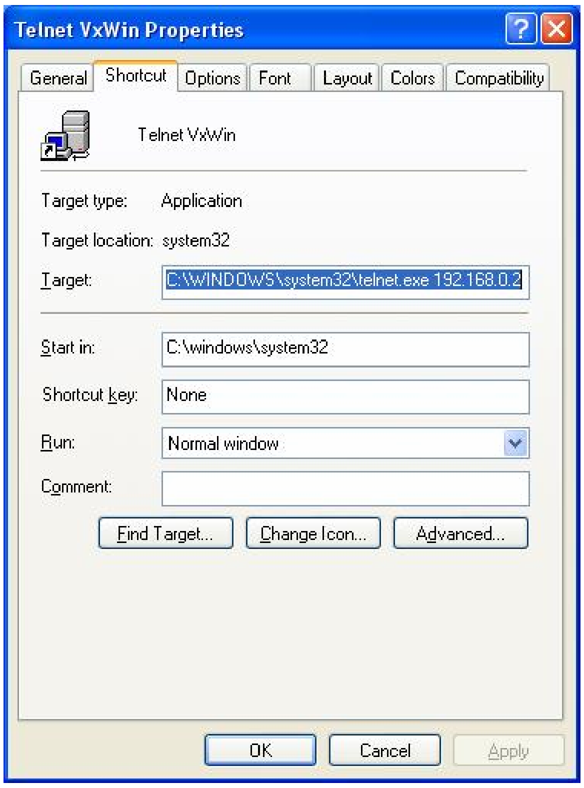
Shortcut key: None

Run: Normal window

Comment:

c) Shortcut to create a Telnet session to access VxWorks

– location:	C:\Windows\system32\Telnet.exe
– name:	Telnet VxWorks
– properties:	target = C:\Windows\system32\Telnet.exe 192.168.0.2



Telnet VxWin Properties

General | Shortcut | Options | Font | Layout | Colors | Compatibility

Telnet VxWin

Target type: Application

Target location: system32

Target: C:\WINDOWS\system32\telnet.exe 192.168.0.2

Start in: C:\windows\system32

Shortcut key: None

Run: Normal window

Comment:

Find Target... | Change Icon... | Advanced...

OK | Cancel | Apply

24 Starting VxWin

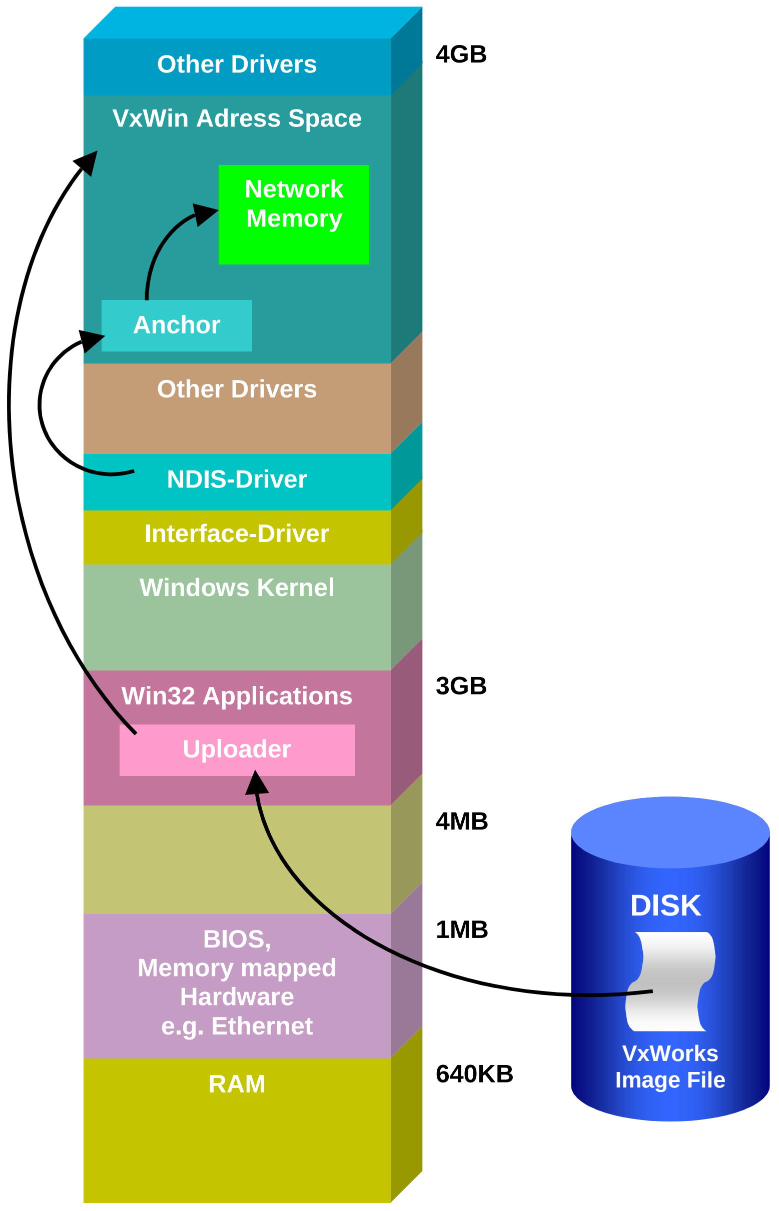
24.1 VxWin tray-icon application (RtosControl.exe)

If you start the demo version of VxWin, the VxWin tray-icon application will automatically run. When it does, a tray icon will automatically appear in Windows taskbar. After it has run for 30 minutes, VxWorks stops it and a dialog window is displayed on the PC monitor. At this point, you can either command the program to run again or terminate it.
If your proprietary icon (rtoscontrol.ico) file is in the same directory as the ***RtosControl*** program, it will be displayed in place of KUKA Control's icon. Otherwise, the KUKA icon will appear.
Even if you have installed a full version of VxWin, you can let this little demo program run, perhaps to demonstrate to yourself that the system has been properly installed. In general, setting the ***LaunchRtosControl*** parameter in the VxWin configuration file to one (1) causes ***RtosControl*** to run automatically (default). Setting it to zero (0), inhibits it from running. In a mature system, you might like to leave the parameter set to zero.
Refer to the ***LaunchRtosControl*** parameter in the System configuration file (vxwin.config).

24.2 Loading and starting VxWin / VxWorks

Having successfully completed the installation procedure described elsewhere (see Installing VxWin), you can now load and start the real-time part of the system. To run on the same computer with Windows, VxWorks requires VxWin; for this reason, the two are always loaded together.
Under the Windows desktop ***Start*** button and the menu point ***VxWin RT***, the ***Start*** and ***Stop*** entries probably appear more than once. Each pair corresponds to a different version of the VxWorks operating system. Both ***Start*** and ***Stop*** commands invoke the VxWin Uploader Utility program (***UploadRTOS.exe***) but

with different command-line parameters. To learn more about these parameters, see the Uploader Utility Program.
If you want to place start and stop shortcuts on your Windows desktop, see Using Desktop Icons.
Immediately after starting VxWin, the Virtual Network connection between Windows and the real-time system can be used. If you want to interact with VxWorks via the VxWorks Command Shell, you can open a ***telnet*** session to it (***telnet 192.168.0.2***). Optionally, you can ping VxWorks to verify that it is running.
The following illustration may help you visualize the VxWin/VxWorks startup process:



24.3 Using Desktop Icons

To make it easier to use VxWin, you can place START and STOP shortcuts on your Windows desktop.
After installing VxWin, go to the list of ***Programs*** under the ***Start*** button on your Windows' desktop. Then position the cursor over the appropriate ***Start*** VxWin entry and use the context menu (right-hand mouse button) to create a shortcut. Finally, drag the shortcut onto your desktop. You can put the ***Stop*** function on your desktop in a similar fashion.

You may wish to examine the shortcut's properties, for example:

Shortcut to start execution of VxWorks

- Location: ***C:\Program Files\VxWin\UploadRTOS.exe***
- Executable filename: ***Start VxWin***
- Properties: target = ***C:\Program Files\VxWin\UploadRTOS.exe –vxwin vxworks***

Shortcut to stop execution of VxWorks

- Location: ***C:\Program Files\VxWin\UploadRTOS.exe***
- Name: ***Stop VxWin***
- Properties: target = ***C:\Program Files\VxWin\UploadRTOS.exe –vxwin –x***

24.4 Starting VxWin – Step by Step

During the start-up phase of VxWin, the VxWin Operating System Device Driver initializes VxWin, VxWorks, and any real-time device drivers and real-time interrupts. By the time the Uploader Utility terminates, VxWorks is already running alongside Windows. From that point on, Windows can be used in the conventional fashion while VxWorks is driven by the interrupts that were assigned to it.

The following list summarizes the main steps comprising startup:

1. Windows' Bootstrap begins running.

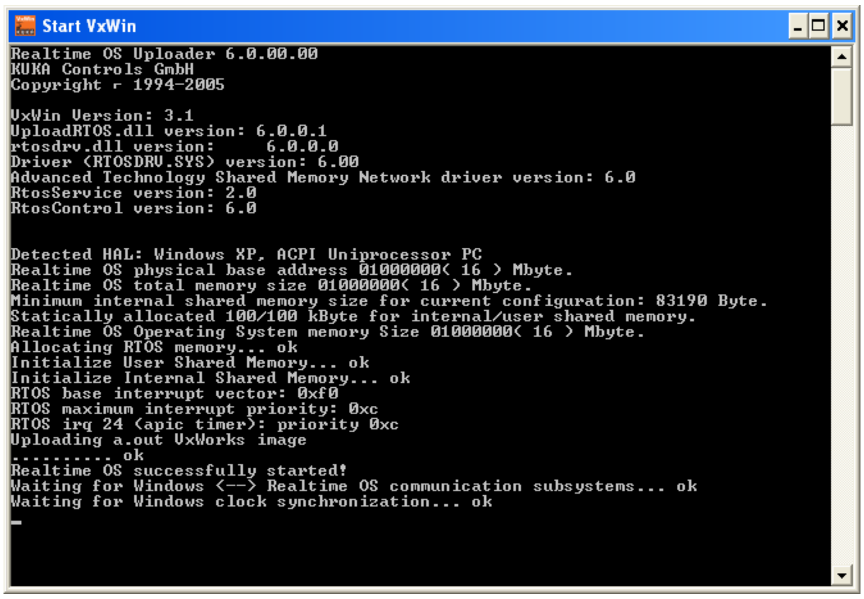
2. Windows loads the VxWin Device Driver.

3 Windows Bootstrap concludes.

4 The customary Windows desktop is displayed on the PC's monitor.

5. The user clicks on the ***Start VxWin*** icon (or uses the program menu under the Windows ***Start*** button). This causes the Uploader Utility program to execute.

6. A DOS-style information window similar to the following screenshot should appear:



```
Realtime OS Uploader 6.0.00.00
KUKA Controls GmbH
Copyright r 1994-2005

VxWin Version: 3.1
UploadRTOS.dll version: 6.0.0.1
rtosdrv.dll version:     6.0.0.0
Driver (RTOSDRV.SYS) version: 6.00
Advanced Technology Shared Memory Network driver version: 6.0
RtosService version: 2.0
RtosControl version: 6.0


Detected HAL: Windows XP, ACPI Uniprocessor PC
Realtime OS physical base address 01000000( 16 ) Mbyte.
Realtime OS total memory size 01000000( 16 ) Mbyte.
Minimum internal shared memory size for current configuration: 83190 Byte.
Statically allocated 100/100 kByte for internal/user shared memory.
Realtime OS Operating System memory Size 01000000( 16 ) Mbyte.
Allocating RTOS memory... ok
Initialize User Shared Memory... ok
Initialize Internal Shared Memory... ok
RTOS base interrupt vector: 0xf0
RTOS maximum interrupt priority: 0xc
RTOS irq 24 (apic timer): priority 0xc
Uploading a.out VxWorks image
.......... ok
Realtime OS successfully started!
Waiting for Windows <--> Realtime OS communication subsystems... ok
Waiting for Windows clock synchronization... ok
_
```

7. The Uploader Utility allocates memory for VxWorks.

8. Starting at RtosMemoryStartAddress, it copies the VxWorks image into the RAM area allocated for it. See Program Memory Layout.

9. It transfers the VxWorks Boot Line from ***vxwin.config*** into a predefined memory location. See Prototype Boot Line for VxWorks, System configuration file (vxwin.config) Boot line parameters in the configuration take precedence over those in the VxWorks Image (see Dynamic Boot Line).

10. The Uploader Utility then calls the VxWorks initialization routine ***sysInit.***

11. ***sysVxwinHwInit()*** is called, a routine that gives the user an opportunity, for example, to initialize custom PCI devices. See Initializing the API for PCI Devices.

12. Real-time interrupts for VxWorks are initialized.

13. Now ***sysVxwinHwInit2()*** is called, giving the user's code an opportunity to complete PCI card initialization. See Initializing the API for PCI Devices.

14. The VxWorks initialization routine creates a new ***Interrupt Descriptor Table*** (**IDT**).

15. The Uploader Utility returns control to Windows, which carries on with its normal operations.

16. At this point, VxWorks, the Shared Memory and Virtual Network Drivers are running.

17. Now an interrupt from any of several possible hardware sources might occur, sources such as the system clock, a serial communication port, or any external hardware interrupt.

18. If such an interrupt occurs, the VxWin ***Operating System Device Driver*** is given control. Because it knows which operating system context was active at the time the interrupt occurred and which IRQ was activated, it determines whether to remain in the current context or to transfer control to the *other* operating system.

19. In either case, the ***Operating System Device Driver*** calls the appropriate ***Interrupt Service Routine*** (**ISR**) in behalf of Windows or VxWorks.

20. Assuming that a VxWorks ISR runs; when it concludes, VxWorks determines whether or not another real-time task is ready to run.

21. Since VxWorks and its tasks always have the highest priorities in a VxWin system, it will continue to run until no more real-time tasks are waiting to run. That is, if other real-time interrupts occur while a VxWorks task is running, those interrupts will be serviced before processing time is granted to Windows.

22. Finally, when the real-time system requires no more CPU time – i.e., there are no more outstanding tasks and no new real-time interrupts – VxWorks enters its idle-routine.

23. As a result, the VxWin ***Operating System Device Driver*** restores the Windows context, including the saved **Interrupt Descriptor Table**, and returns control to Windows.

24. Windows again runs at VxWorks idle level, while VxWorks is ready to respond to more real-time interrupts.

24.5 Common start-up problems

A variety of common errors can occur when starting up VxWin for the first time. A few are mentioned below:

ERROR: 0x01 Device Driver could not be initialized

This means that the Real-time System Driver could not be found or the wrong path to it was specified. Check the following points:

- Did you re-start the system after completing the set-up procedure?
- Did you check to see if the VxWin RT system driver was entered in the Device Manager as a System Device and that no error is indicated?

—

ERROR: There is no TCP/IP connection to VxWorks

If pinging VxWorks does not succeed; a ***telnet*** connection cannot be made.

- Do the IP addresses of Windows and VxWorks agree? (See Virtual Network Communications, Installing the Real-time Virtual Network Driver, and System configuration file (vxwin.config).
- Is the TCP/IP protocol connected to the Virtual Network?
- Is a network adapter installed? Does it run correctly?
- The Virtual Network Driver may not have been entered in the VxWorks boot line; it may only be designated as a second network in the BSP configuration.

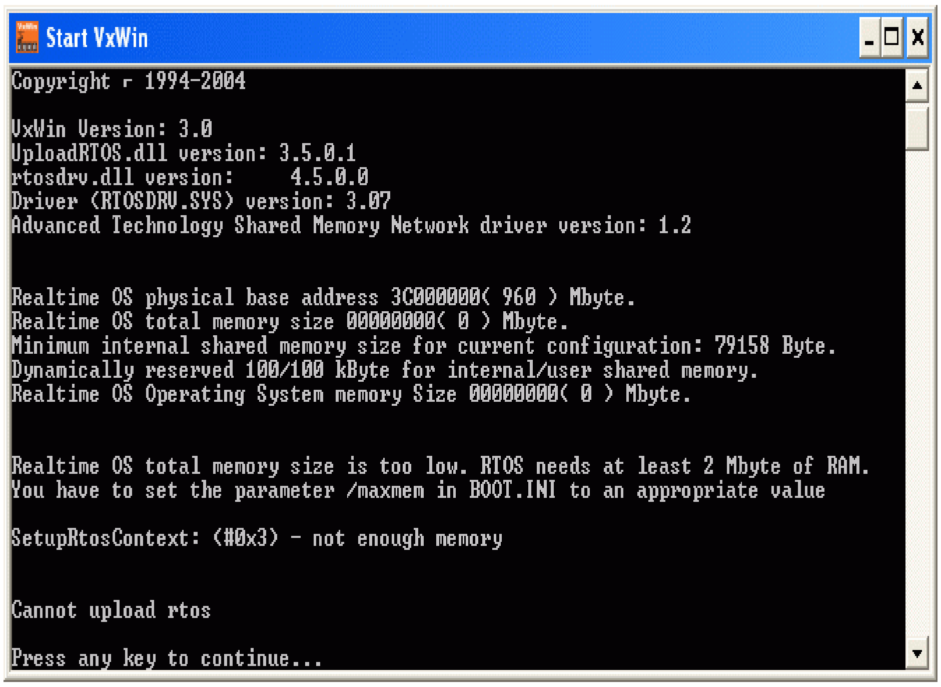
—

ERRORS: #19, #1B, #1E, or #1F memory configuration

These errors occur when the VxWorks image you are trying to load does not fit into the current Windows RAM memory configuration, i.e., the space allocated for it.
To learn more about RAM configuration, see Program Memory Layout.
Note: The PC host must have at least 128 megabytes of RAM to run VxWin.

—

The following screenshot is typical of what may appear on your monitor when an error occurs.



```
Copyright r 1994-2004

VxWin Version: 3.0
UploadRTOS.dll version: 3.5.0.1
rtosdrv.dll version:     4.5.0.0
Driver (RTOSDRV.SYS) version: 3.07
Advanced Technology Shared Memory Network driver version: 1.2


Realtime OS physical base address 3C000000( 960 ) Mbyte.
Realtime OS total memory size 00000000( 0 ) Mbyte.
Minimum internal shared memory size for current configuration: 79158 Byte.
Dynamically reserved 100/100 kByte for internal/user shared memory.
Realtime OS Operating System memory Size 00000000( 0 ) Mbyte.


Realtime OS total memory size is too low. RTOS needs at least 2 Mbyte of RAM.
You have to set the parameter /maxmem in BOOT.INI to an appropriate value

SetupRtosContext: (#0x3) - not enough memory


Cannot upload rtos

Press any key to continue...
```

24.6 Starting Debug Target Server

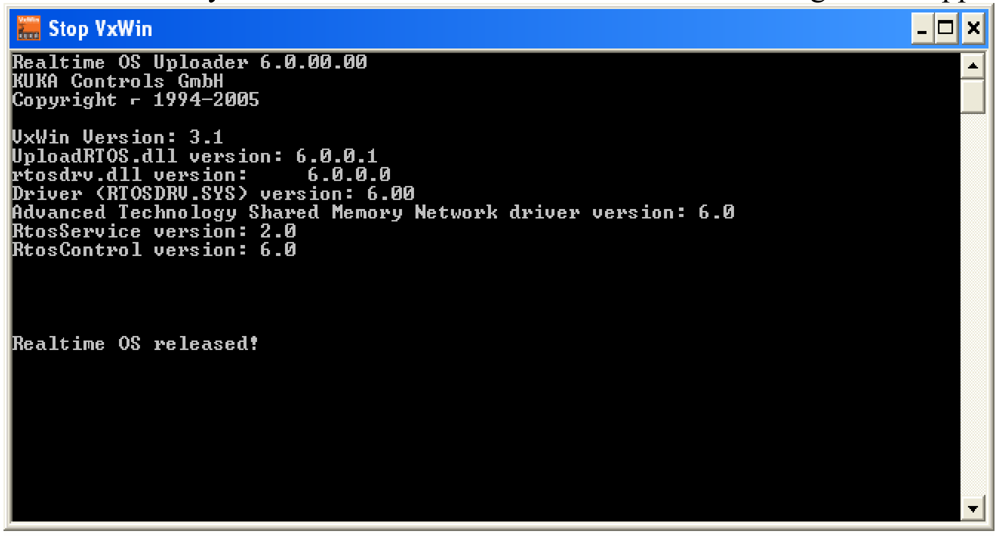
Important: If you are planning to run a debugger on the same PC on which VxWorks is being run, please pay particular attention to the following point: When using two network adapter cards on the target PC - an Ethernet network adapter and the VxWorks Virtual Network adapter - you have to connect both adapters to the TCP/IP protocol. But because of an error in Microsoft's Winsock DLL, however, the target server's IP address can only be specified as a numeric value, not as its symbolic equivalent.
For detailed information about configuring and running target servers, including commands such as that below, see the following Wind River Systems publications: *WindView User's Guide* and *Tornado User's Guide -or-* the *SystemViewer User's Guide* and *Workbench User's Guide*.

Example using a numeric IP address:
Make certain that the file name of the running VxWorks system is called out by the –c option in the command line.
Example:

```
tgtsvr  192.168.0.2 -V -c C:\Tornado20\target\config\VxWin\vxWorks
```

25 Stopping VxWin

To terminate VxWin operation, click on the ***Stop VxWin*** icon on your Windows desktop. The process will be terminated and any resources that may have been in use will be released.
A DOS-style information window similar to the following should appear:



```
Realtime OS Uploader 6.0.00.00
KUKA Controls GmbH
Copyright r 1994-2005

VxWin Version: 3.1
UploadRTOS.dll version: 6.0.0.1
rtosdrv.dll version:     6.0.0.0
Driver (RTOSDRV.SYS) version: 6.00
Advanced Technology Shared Memory Network driver version: 6.0
RtosService version: 2.0
RtosControl version: 6.0



Realtime OS released!
```

For more detailed information, refer to Uploader Utility Program and Using Desktop Icons.

26 Demonstration Programs

Included in the software release are several short application programs that demonstrate important VxWin features. Delivered with source code and project files, they may also serve as coding examples.
Caution: Because Tornado/Workbench is unable to process path names that contain blank characters, e.g. "Program Files," before attempting to work with demo programs, copy them to a different directory.

26.1 Shared Memory Demo

Applications in VxWin may use Shared Memory to exchange data between Windows and VxWorks contexts. (See Using Shared Memory.) The ***ShmDemo*** application is comprised of two parts:

1) *ShmDemo.exe*

An executable program for the Windows context. The source code is at:
C:\Program Files\VxWin\Samples\Windows\ShmDemo\

Important: Since Tornado/Workbench is unable to process path names that contain blank characters, such as: ***Program Files,*** before attempting to work with the demo program, copy it into a different directory.

You have to compile the source code yourself with Microsoft's Visual C++.

2) *ShmDemo.o*

This application is provided in source form only and has to be compiled first.

How to run ShmDemo

Start VxWin as described elsewhere in this manual. See Starting VxWin.
Start an FTP server to connect to the Virtual Network. You may find it convenient to use the FTP server included with the Tornado environment.

Configure it to accept connections from VxWorks (username:*target* password:*vxworks*).

Important: Since Tornado/Workbench is unable to process path names that contain blank characters, in this case: ***Program Files***, before attempting to work with the demo program, copy it to a different directory.

Start a ***telnet*** session. From the ***telnet*** console, change the working directory for to the appropriate directory where the object is located.

Then load the VxWorks application with the command:

```
            ld < ShmDemo.o
```

Open a command window on the Windows side and change the current directory to:

```
        C:\Program Files\VxWin\Samples\
                  Windows\ShmDemo\Release.
```

Start the Windows application from the command line.

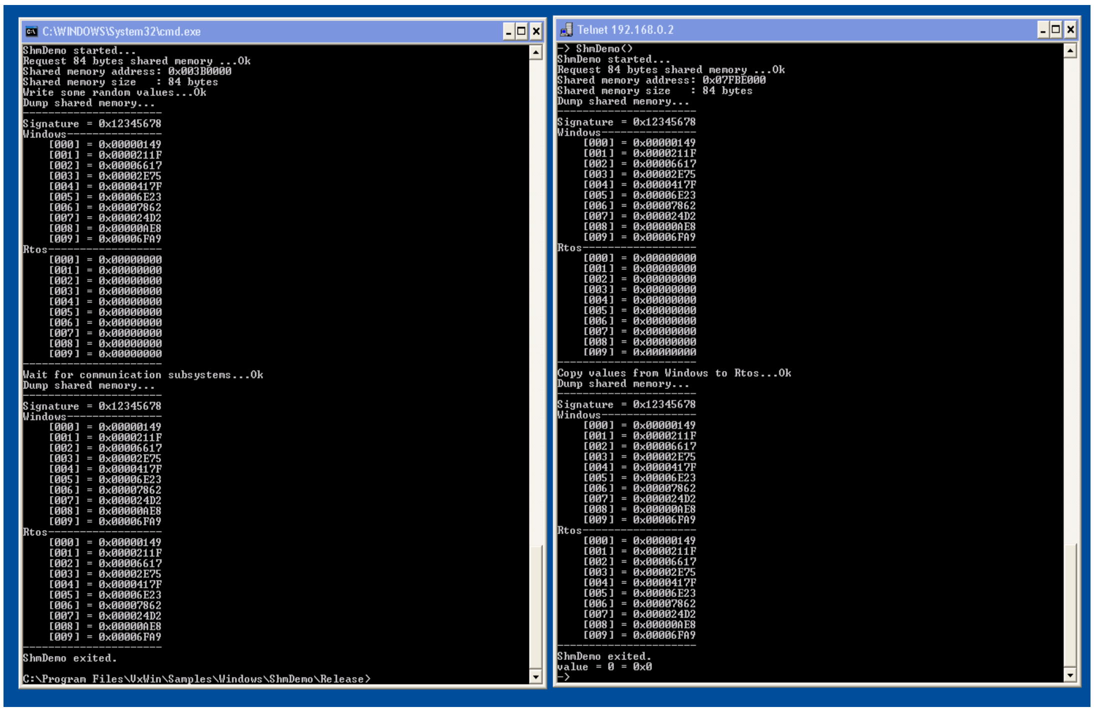
When running, the Windows application fills half of the shared memory with data and then waits for the VxWorks application to copy that data into the other half of Shared Memory.

Start the VxWorks application from the ***telnet*** console:

```
        ->ShmDemo()
```

Both applications run quickly to completion.

The output of the two applications are illustrated in the following screenshot. The Windows command window is on the left; the VxWorks telnet console is on the right. You see the monitor screen after both parts of the ***ShmDemo*** program have finished execution. Notice that both applications show the same data in Shared Memory.



```
ShmDemo started...
Request 84 bytes shared memory ...Ok
Shared memory address: 0x003B0000
Shared memory size    : 84 bytes
Write some random values...Ok
Dump shared memory...
-----------------------
Signature = 0x12345678
Windows---------------
    [000] = 0x00000149
    [001] = 0x0000211F
    [002] = 0x00006617
    [003] = 0x00002E75
    [004] = 0x0000417F
    [005] = 0x00006E23
    [006] = 0x00007862
    [007] = 0x000024D2
    [008] = 0x00000AE8
    [009] = 0x00006FA9
Rtos------------------
    [000] = 0x00000000
    [001] = 0x00000000
    [002] = 0x00000000
    [003] = 0x00000000
    [004] = 0x00000000
    [005] = 0x00000000
    [006] = 0x00000000
    [007] = 0x00000000
    [008] = 0x00000000
    [009] = 0x00000000
-----------------------
Wait for communication subsystems...Ok
Dump shared memory...
-----------------------
Signature = 0x12345678
Windows---------------
    [000] = 0x00000149
    [001] = 0x0000211F
    [002] = 0x00006617
    [003] = 0x00002E75
    [004] = 0x0000417F
    [005] = 0x00006E23
    [006] = 0x00007862
    [007] = 0x000024D2
    [008] = 0x00000AE8
    [009] = 0x00006FA9
Rtos------------------
    [000] = 0x00000149
    [001] = 0x0000211F
    [002] = 0x00006617
    [003] = 0x00002E75
    [004] = 0x0000417F
    [005] = 0x00006E23
    [006] = 0x00007862
    [007] = 0x000024D2
    [008] = 0x00000AE8
    [009] = 0x00006FA9
-----------------------
ShmDemo exited.

C:\Program Files\VxWin\Samples\Windows\ShmDemo\Release>
```

```
-> ShmDemo()
ShmDemo started...
Request 84 bytes shared memory ...Ok
Shared memory address: 0x07FBE000
Shared memory size    : 84 bytes
Dump shared memory...
-----------------------
Signature = 0x12345678
Windows---------------
    [000] = 0x00000149
    [001] = 0x0000211F
    [002] = 0x00006617
    [003] = 0x00002E75
    [004] = 0x0000417F
    [005] = 0x00006E23
    [006] = 0x00007862
    [007] = 0x000024D2
    [008] = 0x00000AE8
    [009] = 0x00006FA9
Rtos------------------
    [000] = 0x00000000
    [001] = 0x00000000
    [002] = 0x00000000
    [003] = 0x00000000
    [004] = 0x00000000
    [005] = 0x00000000
    [006] = 0x00000000
    [007] = 0x00000000
    [008] = 0x00000000
    [009] = 0x00000000
-----------------------
Copy values from Windows to Rtos...Ok
Dump shared memory...
-----------------------
Signature = 0x12345678
Windows---------------
    [000] = 0x00000149
    [001] = 0x0000211F
    [002] = 0x00006617
    [003] = 0x00002E75
    [004] = 0x0000417F
    [005] = 0x00006E23
    [006] = 0x00007862
    [007] = 0x000024D2
    [008] = 0x00000AE8
    [009] = 0x00006FA9
Rtos------------------
    [000] = 0x00000149
    [001] = 0x0000211F
    [002] = 0x00006617
    [003] = 0x00002E75
    [004] = 0x0000417F
    [005] = 0x00006E23
    [006] = 0x00007862
    [007] = 0x000024D2
    [008] = 0x00000AE8
    [009] = 0x00006FA9
-----------------------
ShmDemo exited.
value = 0 = 0x0
->
```

26.2 Shared Events Demo

Shared Events are program *events* that can be signaled in the Windows context and received in the VxWorks context, or vice versa. This demo illustrates their use. (See Shared Events.) The demo program consists of two parts, a windows application and a VxWorks application. Each part sends ten signals to the other side of the system.

1) *EventDemo.exe*

The Windows application sends the first signal to the VxWorks application and then waits for a signal to be received in response. This occurs ten times.
The source for the Windows context may be found in:
`C:\Program Files\VxWin\Samples\Windows\EventDemo`

You have to compile the source code yourself with Microsoft's Visual C++.

2) *EventDemo.o*

This application is provided in source form only and has to be compiled first.

How to run EventDemo

Start VxWin as described elsewhere in Starting VxWin.

Start an FTP server to connect to the Virtual Network. You may find it convenient to use the FTP server included with the Tornado development environment. Configure it to accept connections from VxWorks (username:***target*** password:***vxworks***).

Start a ***telnet*** session. From the ***telnet*** console, change the working directory for to the appropriate directory where the object is located.

Load the VxWorks application with the command:

```
ld < EventDemo.o
```

Open a command window on the Windows side and change to the following directory:

```
C:\Program Files\VxWin\Samples\Windows
                    \EventDemo\Release
```

Start the Windows part of the application at the Windows command line.

At this point, the Windows application sends a signal to a Shared Event on the VxWorks side and then waits for the VxWorks application to send a signal in return.

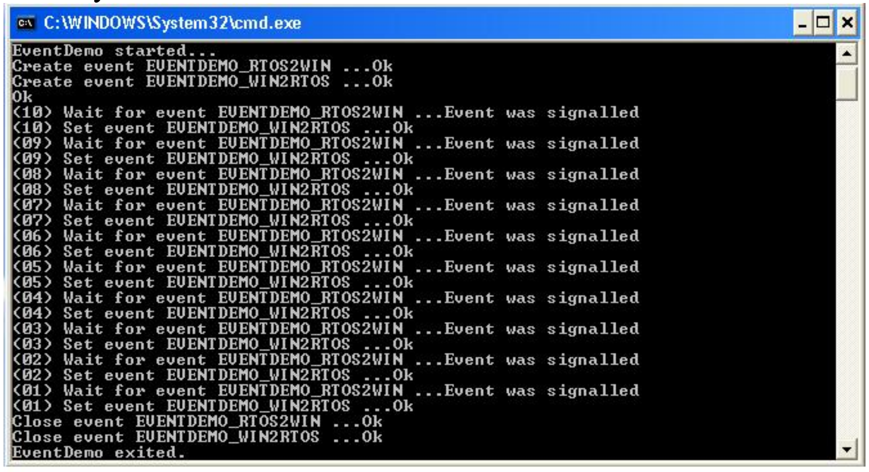
Caution: If you do not start the VxWorks application promptly, the Windows application will time out.

Start the VxWorks application from the *telnet* console:

```
->EventDemo()
```

Both applications run quickly to completion.

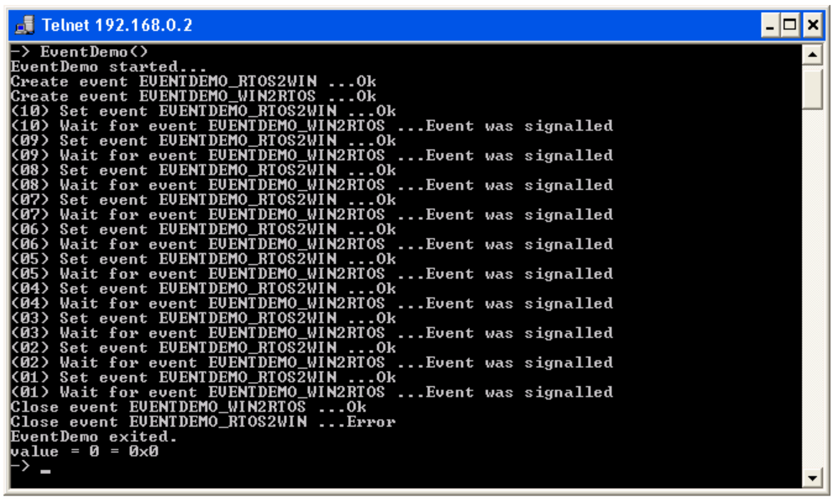
The following screenshot illustrates that ten events were sent to the real-time system and ten received:



C:\WINDOWS\System32\cmd.exe

```
EventDemo started...
Create event EVENTDEMO_RTOS2WIN ...Ok
Create event EVENTDEMO_WIN2RTOS ...Ok
Ok
(10) Wait for event EVENTDEMO_RTOS2WIN ...Event was signalled
(10) Set event EVENTDEMO_WIN2RTOS ...Ok
(09) Wait for event EVENTDEMO_RTOS2WIN ...Event was signalled
(09) Set event EVENTDEMO_WIN2RTOS ...Ok
(08) Wait for event EVENTDEMO_RTOS2WIN ...Event was signalled
(08) Set event EVENTDEMO_WIN2RTOS ...Ok
(07) Wait for event EVENTDEMO_RTOS2WIN ...Event was signalled
(07) Set event EVENTDEMO_WIN2RTOS ...Ok
(06) Wait for event EVENTDEMO_RTOS2WIN ...Event was signalled
(06) Set event EVENTDEMO_WIN2RTOS ...Ok
(05) Wait for event EVENTDEMO_RTOS2WIN ...Event was signalled
(05) Set event EVENTDEMO_WIN2RTOS ...Ok
(04) Wait for event EVENTDEMO_RTOS2WIN ...Event was signalled
(04) Set event EVENTDEMO_WIN2RTOS ...Ok
(03) Wait for event EVENTDEMO_RTOS2WIN ...Event was signalled
(03) Set event EVENTDEMO_WIN2RTOS ...Ok
(02) Wait for event EVENTDEMO_RTOS2WIN ...Event was signalled
(02) Set event EVENTDEMO_WIN2RTOS ...Ok
(01) Wait for event EVENTDEMO_RTOS2WIN ...Event was signalled
(01) Set event EVENTDEMO_WIN2RTOS ...Ok
Close event EVENTDEMO_RTOS2WIN ...Ok
Close event EVENTDEMO_WIN2RTOS ...Ok
EventDemo exited.
```

The next screenshot, showing the ***telnet*** console, also shows ten events sent and ten received.



```
-> EventDemo()
EventDemo started...
Create event EVENTDEMO_RTOS2WIN ...Ok
Create event EVENTDEMO_WIN2RTOS ...Ok
(10) Set event EVENTDEMO_RTOS2WIN ...Ok
(10) Wait for event EVENTDEMO_WIN2RTOS ...Event was signalled
(09) Set event EVENTDEMO_RTOS2WIN ...Ok
(09) Wait for event EVENTDEMO_WIN2RTOS ...Event was signalled
(08) Set event EVENTDEMO_RTOS2WIN ...Ok
(08) Wait for event EVENTDEMO_WIN2RTOS ...Event was signalled
(07) Set event EVENTDEMO_RTOS2WIN ...Ok
(07) Wait for event EVENTDEMO_WIN2RTOS ...Event was signalled
(06) Set event EVENTDEMO_RTOS2WIN ...Ok
(06) Wait for event EVENTDEMO_WIN2RTOS ...Event was signalled
(05) Set event EVENTDEMO_RTOS2WIN ...Ok
(05) Wait for event EVENTDEMO_WIN2RTOS ...Event was signalled
(04) Set event EVENTDEMO_RTOS2WIN ...Ok
(04) Wait for event EVENTDEMO_WIN2RTOS ...Event was signalled
(03) Set event EVENTDEMO_RTOS2WIN ...Ok
(03) Wait for event EVENTDEMO_WIN2RTOS ...Event was signalled
(02) Set event EVENTDEMO_RTOS2WIN ...Ok
(02) Wait for event EVENTDEMO_WIN2RTOS ...Event was signalled
(01) Set event EVENTDEMO_RTOS2WIN ...Ok
(01) Wait for event EVENTDEMO_WIN2RTOS ...Event was signalled
Close event EVENTDEMO_WIN2RTOS ...Ok
Close event EVENTDEMO_RTOS2WIN ...Error
EventDemo exited.
value = 0 = 0x0
-> _
```

26.3 Blue Screen Demo

Most Windows users are familiar with the blue-screen error, by which serious error conditions under Windows cause a text message to be displayed over a blue background on the users console. Except by rebooting, it is not possible for Windows to recover from this type of error. But, depending on the nature of the application, it may be unwise or even catastrophic to reboot a real-time system while it is working.

VxWin is therefore equipped with a mechanism that allows the real-time system to continue to run even after the Windows side of the system has detected a blue-screen error. Once the real-time application is informed that Windows has reached a blue-screen condition, it can choose either to continue running or close down its real-time operations in a safe and orderly fashion, thus protecting the integrity of any parts, machines, products or data under its control.

Under VxWin, if Windows reaches a bluescreen condition, the real-time system can continue to run for as long as the user wishes. For more information about this mechanism, please read: Blue Screen Error Management and Blue Screen Errors Module.

Important: Since Tornado/Workbench is unable to process path names that contain blank characters, in this case: ***Program Files***, before attempting to work with the demo program, copy it to a different directory.

This application - ***BsodDemo.o*** – in source form only and has to be compiled first.

How to run BSOD demo

1. Start VxWin as described elsewhere in this manual (Starting VxWin).

2. Start an FTP server to connect to the Virtual Network. You may find it convenient to use the FTP server included with the Tornado environment. Configure it to accept connections from VxWorks (username:***target*** password:***vxworks***).

3. Start a ***telnet*** session. From the ***telnet*** console, change the working directory for to the appropriate directory where the object is located.
4. Then load the VxWorks application with the command:

```
ld < BsodDemo.o
```

5. A blue-screen error will occur on the Windows side after about 30 seconds. The duration of the programmed delay was chosen arbitrarily; you may modify it to be of any desired length. The delay is meant to demonstrate that when Windows encounters a blue-screen exception, the real-time system can continue running for any length of time.

27 VxWin – Board Support Package

27.1 Introduction

The VxWin RT board support package (BSP) provides special adaptations for Wind River's VxWorks to the run-time environment of VxWin RT.
Several files and functions are designated by Wind River Systems as platform or run-time specific such as the ***sysLib.c*** file.
This manual describes such components and additional extensions delivered with VxWin RT to support the operation of VxWorks concurrently with Windows.
Some of the drivers that Wind River Systems delivers with their PC386 BSP may also be used without change under VxWin.
The communications driver for the Intel 8052 is in this category, for example, as are several Ethernet network drivers such as those for the following adapters: SMC Elite Ultra, 3Com Etherlink, NE2000.

27.2 Important files in the VxWin BSP

In the following sections, some important files included in VxWin Board Support Package have been listed along with brief explanations of their purpose.

The files are grouped primarily to reflect their general role in the system and secondarily to reflect their file-type. Only files that were developed or modified by KUKA Roboter are listed here.

VxWin Source Files

File	Description
RtosLib.h	All definitions pertinent to VxWin Interface, including those for shared events and shared-memory API.

Additional VxWorks Test Tools

File	Description
demo.c	A demo-program

VxWin Library

File	Description
RtosXxYy.a	VxWin kernel library for vxworks version Xx and tool Yy (diag or gnu)
RtosXxYy.so	VxWin shared object library for vxworks version Xx and tool Yy (diag or gnu)

27.3 VxWorks / Software Configuration

VxWorks is to be configured by using its own workspace configuration components. See *Tornado User's Guide* or *Workbench User's Guide*.
The following VxWorks configuration items may not be used:

- INCLUDE_IDE
- INCLUDE_ATA
- INCLUDE_TFFS
- INCLUDE_FD
- INCLUDE_SM_OBJ

27.4 VxWorks / PC Serial Ports (COM Ports)

To use COM ports under VxWorks, one must first assign appropriate values to two symbolic constants:

- The number of COM ports to be initialized must be specified by assigning a value of 0, 1 or 2 to the symbol *N_UART_CHANNELS* in the file ***pc.h*** ;

- The number of COM ports must also be specified by assigning a corresponding value to the symbol *NUM_TTY* in ***config.h*** .

Both constants will generally have the same value.
When *N_UART_CHANNELS* is set to 0, VxWorks will not use any serial ports, and Windows can use all COM ports for its own applications. When it is set to 1, VxWorks can exclusively use the COM1 port for its own purposes, possibly for communication with the target-resident shell. When set to 2, VxWorks is assured the exclusive use of both the COM1 and COM2 ports.

28 System Library Functions (sysLib.c, sysALib.s)

sysToMonitor

This function will be called when VxWorks has come to a point in its execution at which it has no choice except to reboot. See VxWorks / System overload.

```
STATUS    sysToMonitor
           (
           int    startType
           );
```

PARAMETERS

startType

[in] - Specifies the boot-type (see sysLib.h for possible values).

RETURN VALUE

If the function fails, the return value is ERROR.

If successful, it will not return.

REMARKS

VxWorks will call ***sysToMonitor*** only to handle an emergency situation.

If, for example,an excessive rate of interrupts causes VxWorks internal work queue to overrun, the real-time system will probably be unable to proceed. And since there is no ROM-Monitor under VxWin, when this function is called, all interrupts will be deactivated and control will remain in this function forever.

PROGRAM DETAIL

Library: sysLib.c

sysFPExceptHandler

This routine declares the Floating Point Exception Handler. It will automatically be called each time the system detects a floating point exception. Refer to VxWorks Floating-Point elsewhere in this manual.

```
void           sysFPExceptHandler ( );
```

PARAMETERS

[in] –

[out] –

RETURN VALUE

–

PROGRAM DETAIL

VxWorks header: fppExc.h

sysTaskFPExceptionAdd

This routine must be called in behalf of every task that is to be supported by floating-point exception processing. (Refer to VxWorks Floating-Point elsewhere in this manual. Exception processing may be uniquely defined for each task context, i.e., fow which floating point errors the exception handler should be called.

```
STATUS    TaskFPExceptionAdd
           (
           int     tid,
           DWORD     dwFPControlSet
           );
```

PARAMETERS

tid

[in] - Indicates a task (via task ID) that is to be supported by floating point exception handling. A value of 0 implies the ID of the task calling this function.

dwFPControlSet

[in] - A double-word mask that expresses which floating point errors will be processed as exceptions. When this function is called the mask will be transferred to the floating point unit's control register.

The value of this field may be established by combining the following defines using a logical OR function. (For detailed information see Intel IA32 Intel Architecture Software Developer's Manual, Vol.1: Basic Architecture):

- FP_EXC_NONE
  FP_EXC_INVALID
- FP_EXC_DENORM_OPERAND
- FP_EXC_ZERO_DIVIDE
- FP_EXC_OVERFLOW
- FP_EXC_UNDERFLOW
- FP_EXC_PRECISION
- FP_EXC_ALL

RETURN VALUE

OK when floating point exception support properly installed; otherwise **ERROR**.

PROGRAM DETAIL

VxWorks header: fppExc.h

RtosIdle

Regardless of the priority of the VxWorks task from which this function is called, *RtosIdle* forces the execution context to switch immediately from VxWorks to Windows. Should you have a task that tends to monopolize the CPU time, you can use *RtosIdle* to make certain that Windows XP receives additional CPU time. Upon the occurrence of the next real-time interrupt - from the real-time clock, for example - control will be returned to the real-time system.
Note: This function can be called from any task at any time.
For more discussion of the practical use of this function, refer to the following:
Forcing context change — VxWorks to Windows

```
void            RtosIdle  (void);
```

PARAMETERS
RETURN VALUE

N/A.

PROGRAM DETAIL

VxWorks header: rtosLib.h

EXAMPLE

```
/* standard VxWin Idle Task */
/* ======================== */
void vxwinIdleTaskStart( int nPriority)
{
    taskSpawn ("RtosIdle",              /* task name */
                nPriority,               /* priority */
                VX_FP_TASK,              /* options */
                0x2000,                  /* stack size */
                (FUNCPTR)vxwinIdleTask,  /* entry */
                0,0,0,0,0,               /* params */
                0,0,0,0,0);              /* params */
}

void vxwinIdleTask(void)
{
    for(;;)
    {
        RtosIdle();
    }
}

/* RTOS Idle Task with time slice */
/* =============================== */
taskid_IdleTask = taskSpawn ("tIdleTask",0 /*prio 0*/,
     0,8000,runIdleTask,winweight,vxweight,
          ticks,0,0,0,0,0,0,0);

int runIdleTask (int winweight,int vxweight,int ticks)
{
     ULONG aktick,firsttick;
     int debugcounter = 99;

   for (;;)
   {
```

```
                firsttick = tickGet();

                /* Change to Windows for the choosen weight */
                do
                {
                     vxwinIdle( );
                     aktick = tickGet();

                } while ( (aktick - firsttick) < (ticks*winweight
                           /(winweight+vxweight)) );

                /* Stay in VxWorks for the chosen weight */
                firsttick = tickGet();

                do
                {
                     taskDelay(TICKS_DELAY);
                     aktick = tickGet();
                } while ( (aktick - firsttick) < (ticks*vxweight
                          /(winweight+vxweight))  );
              }
                return 0;
           }
```

28.1 Blue Screen Errors Module

The following functions implement Blue Screen Error Handling as discussed in Blue Screen Error Management.

RtosBsodSetMode

This function sets the Bluescreen mode for the real-time system. The mode is used to determine the next actions whenever the bluescreen condition occurs.

```
STATUS    RtosBsodSetMode
          (
          long unsigned int    dwBsodMode,
          SEM_ID*              psemId
          );
```

PARAMETERS

dwBsodMode

[in] - The following options may be used to specify 'mode':

BSOD_STANDARD

The course of events will not be influenced. Windows displays a blue screen and halts. The real-time system is not notified.

BSOD_RTSAFE

After a Bluescreen exception occurs, Windows will be stopped until the real-time system has reached a safe state.
The semaphore – semId – is set to the signaled state.
The user's real-time bluescreen processing task, which was waiting for the semaphore semID, will run. This routine takes whatever action the programmer thought necessary, and when it is finished, it calls ***RtosBsodExit*** .
Sometimes the user's Bluescreen task will record data so that upon re-boot, the real-time work can take up its work in an advanced state; sometimes it will bring an external system into a safe state; sometimes it will send a signal on an external network.

psemId

[out] - The ID number of the semaphore that will be set.

RETURN VALUE

OK when function successfully executed;otherwise **ERROR**.

PROGRAM DETAIL

VxWorks header: rtosLib.h

RtosBsodExit

Upon notification of a blue screen exception, the real-time system performs whatever process it cares to in order to end its job in a graceful fashion. When that has been done, it should call this function. Although the real-time system will then be stopped, Windows will run further to display its blue-screen error message. See Blue Screen Error Management.

```
void            RtosBsodExit (void);
```

PARAMETERS

None.

RETURN VALUE

None.

PROGRAM DETAIL

VxWorks header: rtosLib.h