



acontis technologies GmbH

SOFTWARE

# RTOS Virtual Machine

User Manual  
Edition: 2019-03-28

# Content

<b>1</b>	<b>ACONTIS RTOS VIRTUAL MACHINE OVERVIEW</b> .....	<b>5</b>
1.1	SHARED MODE OPERATION.....	5
1.2	EXCLUSIVE MODE OPERATION.....	6
<b>2</b>	<b>INSTALLATION</b> .....	<b>7</b>
2.1	SETUP.....	7
2.2	CONFIGURE AND START A RTOS.....	9
2.3	ASSIGN A DEVICE TO A RTOS .....	11
<b>3</b>	<b>VIRTUAL MACHINE FRAMEWORK</b> .....	<b>12</b>
3.1.1	<i>VMF Architecture</i> .....	13
3.1.2	<i>Basic VMF Services (Hardware Abstraction Layer)</i> .....	14
3.2	PORTABILITY.....	15
3.3	VMF MANAGEMENT ANCHOR.....	15
3.4	MEMORY LAYOUT.....	16
3.5	THE RTOS LIBRARY .....	17
<b>4</b>	<b>REAL-TIME DEVICE MANAGEMENT</b> .....	<b>18</b>
4.1	OVERVIEW.....	18
4.2	ASSIGN A DEVICE TO A RTOS .....	19
4.2.1	<i>Using System Manager</i> .....	19
4.2.2	<i>Using RtosUpload.exe or RtosLib API</i> .....	20
4.3	INTERRUPT SHARING CONFLICTS .....	21
4.3.1	<i>Principle</i> .....	21
4.3.2	<i>Understanding interrupt conflicts</i> .....	21
4.3.3	<i>Resolving interrupt conflicts</i> .....	23
4.4	CONFIGURATION .....	24
4.4.1	<i>Properties dialog</i> .....	24
4.4.2	<i>RtosUpload.exe / RtosLib API</i> .....	26
4.4.3	<i>Windows INF file</i> .....	26
4.4.4	<i>RTOS config file</i> .....	32
4.5	DRIVER SIGNING .....	33
4.5.1	<i>Driver Package Signing</i> .....	33
4.5.2	<i>Certificate Pre-Installation</i> .....	35
<b>5</b>	<b>RTOS OPERATION MODE</b> .....	<b>37</b>
5.1	SHARED MODE OPERATION (SINGLE CORE) .....	38
5.2	SHARED MODE OPERATION (MULTI CORE).....	39
5.3	EXCLUSIVE MODE OPERATION .....	40
5.4	SMP EXCLUSIVE MODE OPERATION.....	41
5.5	SMP SHARED MODE OPERATION.....	42
<b>6</b>	<b>RTOS VM CONFIGURATION FILES (*.CONFIG FILES)</b> .....	<b>43</b>
6.1	PROCESSOR CONFIGURATION (RTOS).....	43
6.2	INTERRUPT PROCESSOR VECTOR RANGES.....	44
6.3	MEMORY CONFIGURATION .....	45
6.4	TIME/DATE AND TIMEZONE SYNCHRONIZATION .....	47
6.4.1	<i>Windows</i> .....	48
6.4.2	<i>RTOS</i> .....	48
6.4.3	<i>Windows / RTOS</i> .....	48
6.5	SECTION [VMF] .....	49
6.6	SECTION [UPLOAD] .....	51
6.7	MULTI PURPOSE SHARED MEMORY .....	53
6.8	OS COMMUNICATION .....	54
6.8.1	<i>Settings</i> .....	54
6.8.2	<i>Enable / Disable Comm Interrupt</i> .....	55
6.8.3	<i>System does not boot using configuring Comm interrupt</i> .....	56
6.8.4	<i>Comm configuration error</i> .....	56
6.8.5	<i>Interrupt conflict</i> .....	57
<b>7</b>	<b>START/STOP THE RTOS: UPLOADER UTILITY</b> .....	<b>59</b>
7.1	INTRODUCTION .....	59
7.2	UPLOADER OPERATION, COMMAND LINE OPTIONS.....	60
<b>8</b>	<b>RTOS TRAY-ICON APPLICATION (RTOSCONTROL.EXE)</b> .....	<b>62</b>
<b>9</b>	<b>THE RTOS SERVICE APPLICATION (RTOSSERVICE.EXE)</b> .....	<b>63</b>
9.1	CLOCK CORRECTION .....	63
9.2	DATE AND TIME SYNCHRONIZATION .....	63

9.3	RTOS FILE SERVER FOR RTOSFILE SUPPORT .....	63
9.4	RTOS GATEWAY.....	63
<b>10</b>	<b>THE REALTIME OS VIRTUAL NETWORK ADAPTER (RTOSVNET.SYS) .....</b>	<b>65</b>
10.1	CONFIGURATION.....	65
10.1.1	Windows .....	65
10.1.2	RTOS .....	70
<b>11</b>	<b>THE RTOS LIBRARY .....</b>	<b>71</b>
11.1	RTOS LIBRARY – APPLICATION LAYER API.....	71
11.1.1	Windows applications.....	71
11.1.2	RTOS applications.....	71
11.1.3	RTOS Library – initialization and shutdown.....	72
11.1.4	RTOS Library – events .....	77
11.1.5	RTOS Library – interlocked data access.....	82
11.1.6	RTOS Library – shared memory.....	86
11.1.7	RTOS Library – date and time synchronization (clock synchronization).....	91
11.1.8	RTOS Library – OS scheduling .....	93
11.1.9	RTOS Library – notification events.....	94
11.1.10	RTOS Library – uploader API.....	98
11.1.11	RTOS Library – result value.....	104
11.1.12	RTOS Library – licensing.....	106
11.1.13	RTOS Library – file server .....	107
11.1.14	RTOS Library – files.....	108
11.1.15	RTOS Library – files advanced (6.1).....	123
11.1.16	RTOS Library – generic object functions .....	137
11.1.17	RTOS Library – message queue functions.....	141
11.1.18	RTOS Library – pipe functions.....	144
11.1.19	RTOS Library – socket functions.....	147
11.1.20	RTOS Library – device functions.....	155
11.1.21	RTOS Library – memory reservation functions.....	159
11.1.22	RTOS Library – virtual I/O (VIO) functions.....	160
11.2	RTOS LIBRARY EXAMPLE APPLICATIONS.....	161
11.3	RTOS LIBRARY – COMPATIBILITY ISSUES FOR VxWIN AND CEWIN 3.5 .....	162
11.3.1	Compatibility mode .....	162
11.3.2	Initialization .....	162
11.3.3	Time/date and timezone synchronization.....	162
11.3.4	Function SetOutputStream .....	162
<b>12</b>	<b>EC-MASTER (MAC-ID) – LICENSING .....</b>	<b>163</b>
12.1	GENERAL.....	163
12.2	REQUIRED STEPS.....	163
<b>13</b>	<b>CODEMETER – LICENSING.....</b>	<b>164</b>
13.1	DEVELOPMENT VS. RUNTIME.....	164
13.2	LICENSE SOFTWARE “CODEMETER” .....	164
13.3	INSTALL A LICENSE.....	164
13.3.1	License received on USB dongle.....	164
13.3.2	License required for existing USB dongle.....	164
13.3.3	License for software based solution .....	167
13.4	UPDATE A LICENSE .....	170
13.4.1	Generate license request .....	170
13.4.2	Import a license update .....	172
13.5	SHARING A LICENSE .....	173
13.6	TROUBLESHOOTING.....	175
<b>14</b>	<b>RTOSWIN OEM BRANDING.....</b>	<b>176</b>
14.1	GENERAL.....	176
14.2	MODULE SPECIFIC BRANDING.....	176
14.2.1	RtosDrv.sys.....	176
14.2.2	RtosVnet.sys and RTOS_xxx.inf.....	176
14.2.3	RtosService.exe.....	176
14.2.4	RtosControl.exe.....	177
14.2.5	UploadRtos.exe (RTE <=4.x) or RtosUpload.exe (RTE >=5.x) .....	177
14.2.6	RtosPnp.sys.....	177
<b>15</b>	<b>APPENDIX A – PLATFORMS AND PERFORMANCE.....</b>	<b>178</b>
15.1	REAL TIME BEHAVIOUR AND THE RTOS-VM.....	178
15.2	PLATFORM EVALUATION .....	178
15.3	REDUCING DMA LATENCY PROBLEMS .....	179

15.4	CPU THROTTLING.....	181
15.4.1	<i>Detection</i> .....	181
15.4.2	<i>How to disable</i> .....	182
15.5	SYSTEM MANAGEMENT INTERRUPT (SMI) .....	183
<b>16</b>	<b>APPENDIX B – TROUBLESHOOTING.....</b>	<b>184</b>
16.1	SETUP FAILS.....	184
16.2	SYSTEM DOES NOT BOOT.....	185
16.3	COMMON STARTUP PROBLEMS.....	186
16.3.1	<i>IRQ sharing with Windows (10A0)</i> .....	186
16.3.2	<i>Error opening include file (107C)</i> .....	187
16.3.3	<i>Configured RTE memory range not available</i> .....	187
16.3.4	<i>Invalid memory configuration (10A4)</i> .....	187
16.4	WINDOWS CLOCK DELAY.....	188
16.5	WINDOWS NETWORK STACK.....	188
16.6	NETWORK SHARE ACCESS .....	189
16.7	TIMER FREQUENCY .....	190
16.7.1	<i>Setting a dedicated frequency</i> .....	190
16.7.2	<i>Frequency measure error</i> .....	190
16.8	INTERRUPT / TIMER LATENCY .....	192
16.9	VIRTUAL NETWORK ADAPTER PERFORMANCE.....	192
16.10	RTOSLIB EVENT / MSGQUEUE / PIPE / SOCKET PERFORMANCE .....	192
16.11	USING RTOSVM INSIDE A HYPERVISOR .....	192
<b>17</b>	<b>APPENDIX C – MANUAL INSTALLATION .....</b>	<b>193</b>
17.1	REALTIME OS DRIVER.....	194
17.2	REALTIME OS VIRTUAL NETWORK DRIVER .....	196
17.3	PRODUCT FILES .....	198
17.3.1	<i>Virtual Machine Framework</i> .....	198
17.3.2	<i>Realtime OS Service</i> .....	198
17.3.3	<i>Realtime OS Control</i> .....	199
17.3.4	<i>Configuration Files</i> .....	199
17.3.5	<i>Realtime OS Uploader</i> .....	200
17.4	DEBUG CONSOLE.....	201
17.5	OS IMAGE .....	201
17.6	CODEMETER RUNTIME ENVIRONMENT .....	201
17.7	FIRST START OF AN OS.....	202
17.8	PRODUCT SPECIFIC ADDITIONALS STEPS .....	204
<b>18</b>	<b>VERSION HISTORY .....</b>	<b>205</b>

# 1 ACONTIS RTOS Virtual Machine Overview

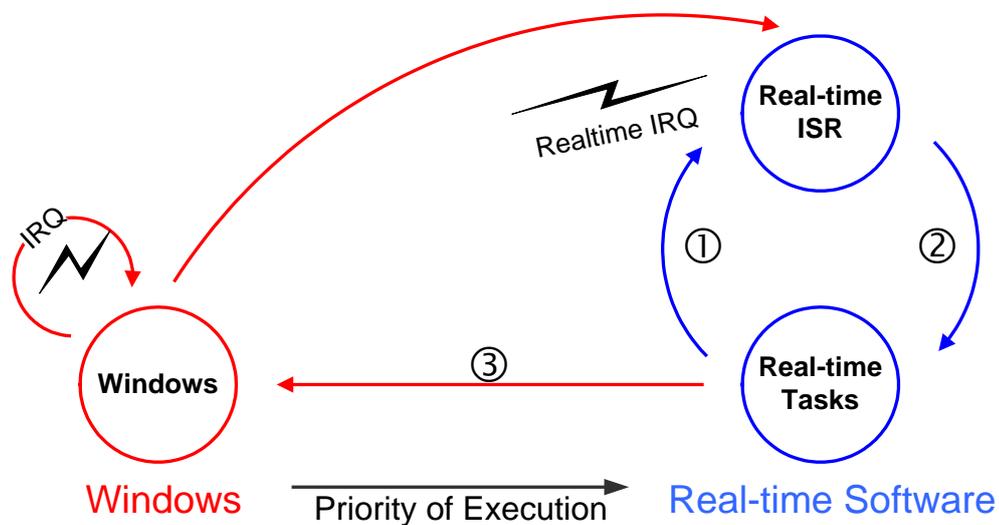
The ACONTIS RTOS-VM provides a light-weight real-time virtualization platform for Windows. On top of this platform either real-time firmware, custom or off-the-shelf real-time operating systems can be executed.

When using multicore CPUs one can choose between two general operation modes. A more detailed description about possible operation modes can also be found in section 5.

## 1.1 Shared Mode Operation

Windows shall run on all CPU cores and only one CPU core shall additionally run the real-time software. If the Windows application needs a lot of CPU power (e.g. for image processing) this will be the appropriate operation mode even on multi-core CPUs. In shared mode operation Windows (on this core) will usually only get CPU time when the real-time software is idle.

The following diagram illustrates the flow of control:



### Operating states of the RTOS-VM in shared mode

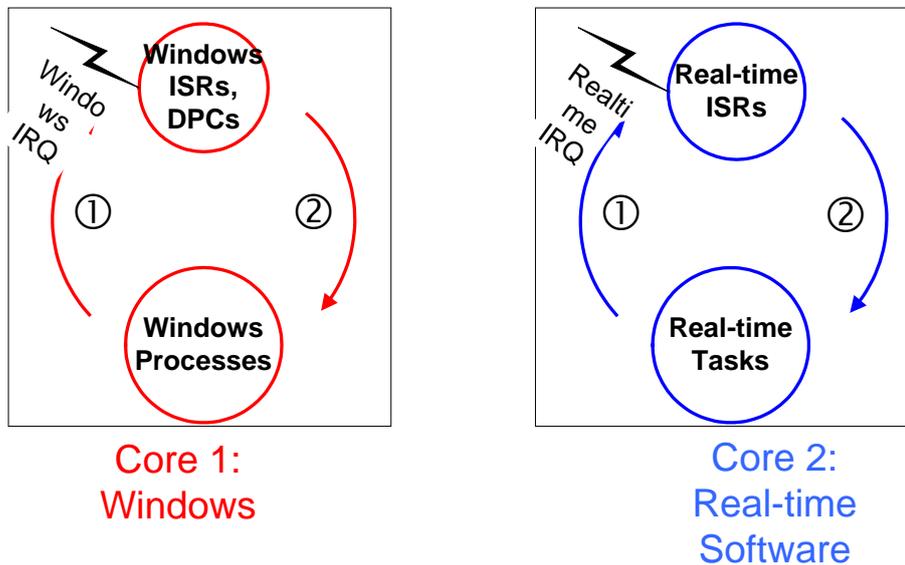
- ① Exception-handling or a higher priority interrupt becomes outstanding.
- ② Interrupt Service Routine optionally starts a new task and then finishes.
- ③ From the idle-state, VxWorks transfers control to Windows operating system.

Note: When running the RTOS-VM 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.

## 1.2 Exclusive Mode Operation

Windows and the real-time software shall run fully independently on different CPU cores. Using this mode will lead to much shorter interrupt and task latencies as there is no need to switch from Windows to the real-time software.

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



### Operating states of the RTOS-VM in exclusive mode

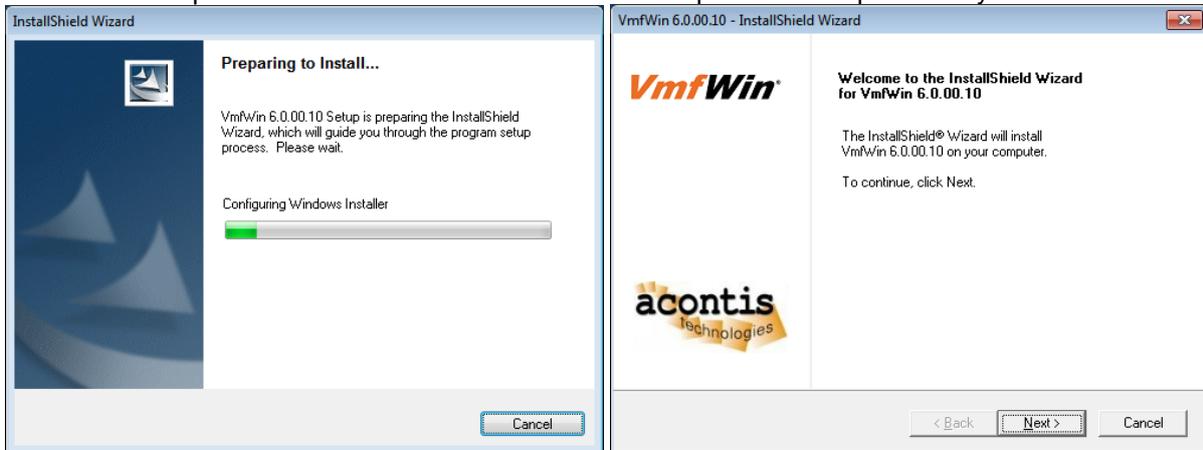
- ① Exception-handling or a higher priority interrupt becomes outstanding.
- ② Interrupt Service Routine optionally starts a new task and then finishes.

Note: When running the RTOS-VM 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 the real-time software to enter the idle state.

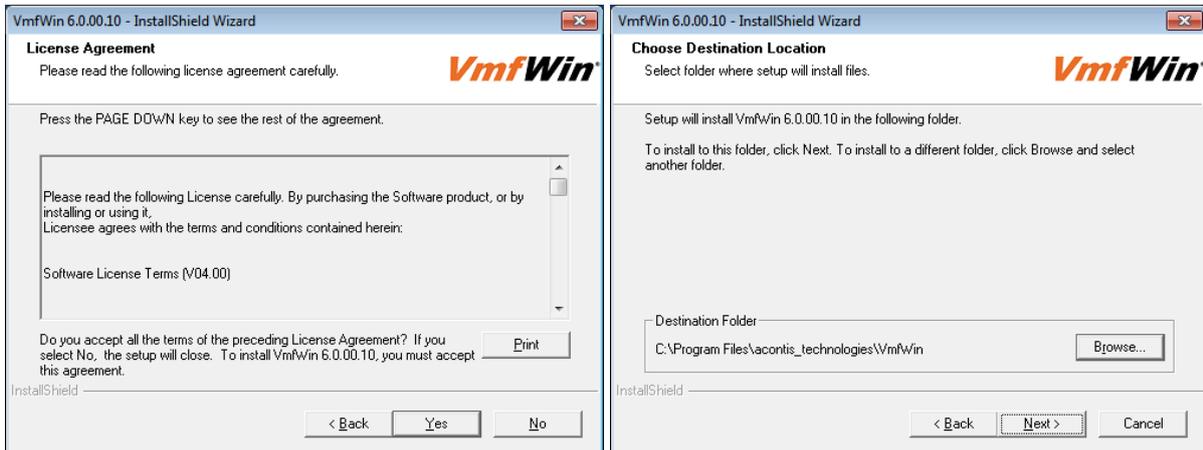
## 2 Installation

### 2.1 Setup

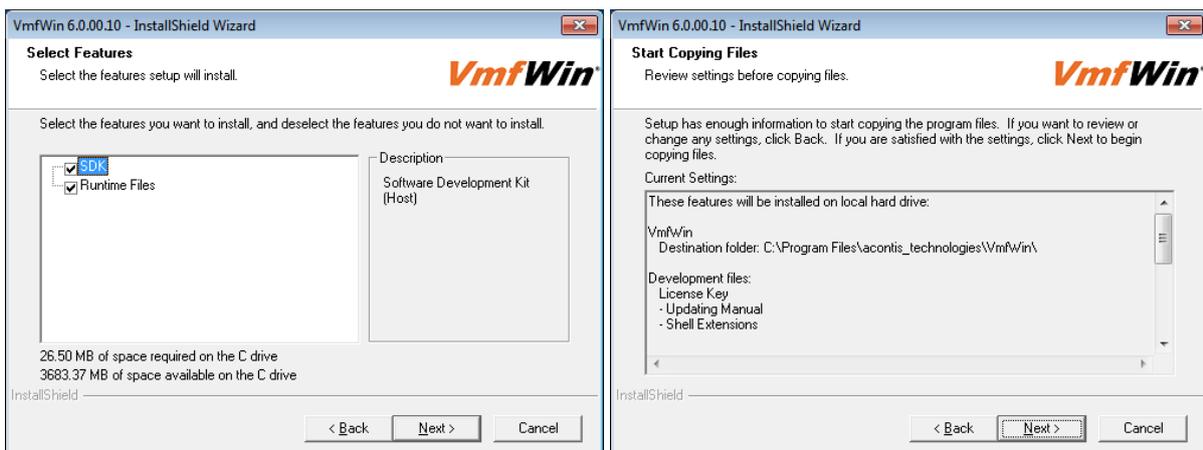
The RTOS Virtual Machine will be installed by the product setup. In this case the product VmfWin will be used but the other products setups are very similar.



After starting the Setup...



... the license has to be agreed. Optionally the installation directory can be changed.

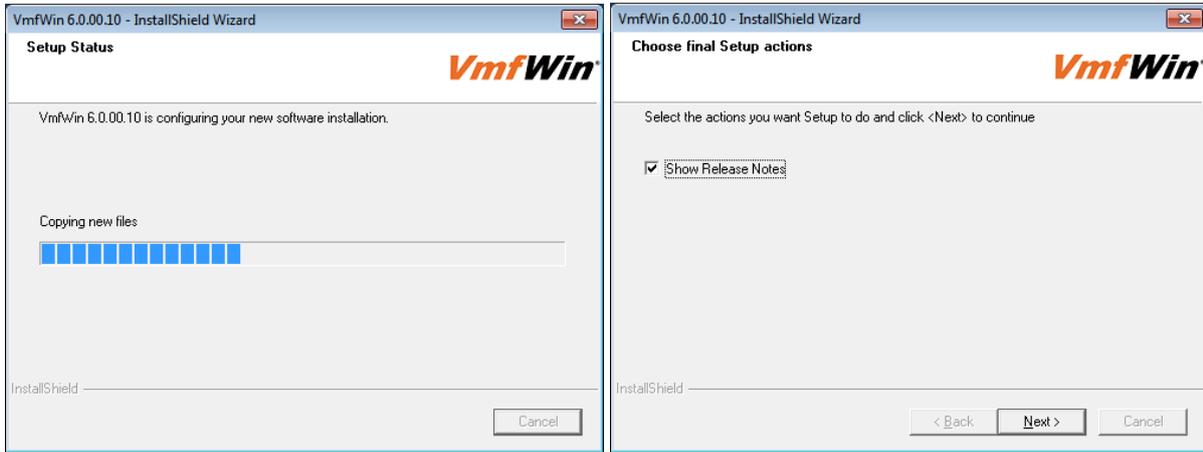


There are two features:

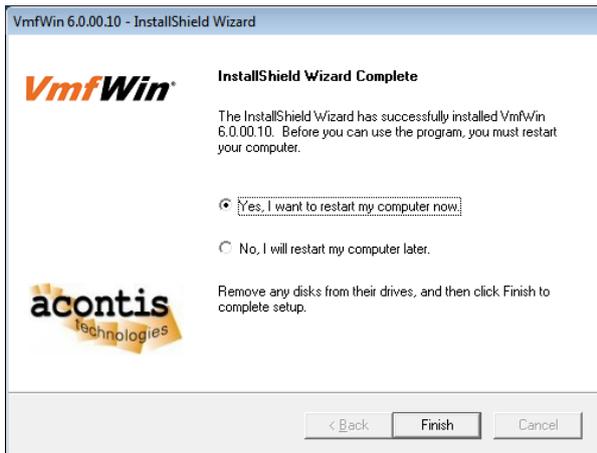
- "SDK" contains everything required to build an own application. This includes header files, libraries and examples (sources).
- "Runtime Files" contains the drivers, binaries, configuration files and applications required to start a RTOS.

In case the development PC is not the target PC the "SDK" should be installed on the development and the "Runtime Files" on the target PC.

Before setup starts an overview of the settings is shown.



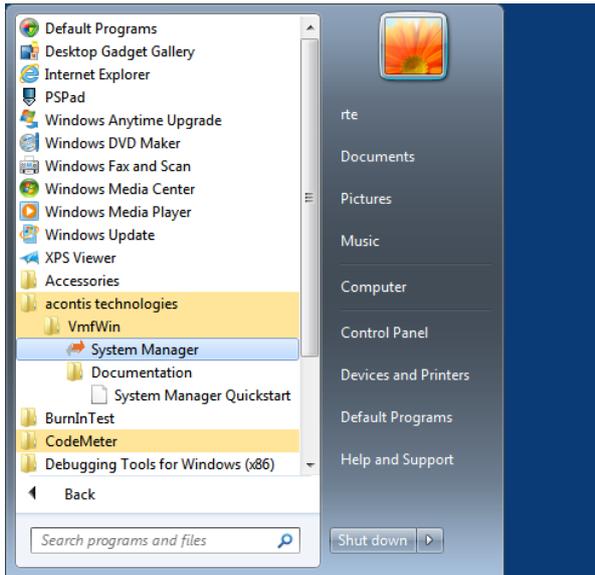
After copying the files, the release notes can be viewed.



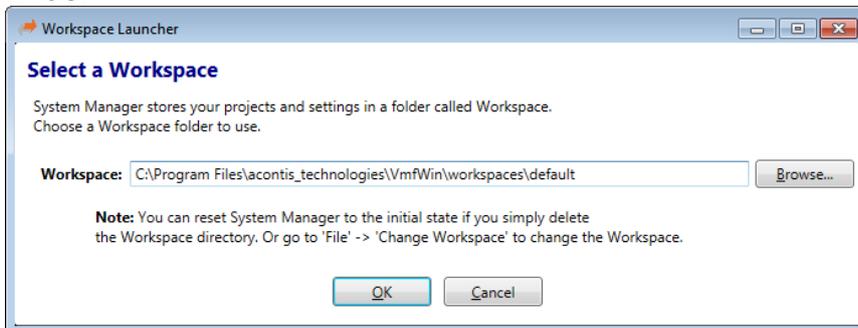
Finally, in most cases a reboot is required. This should be done before continuing.

In case a problem is encountered please check chapter "16.1 Setup fails" for additional information.

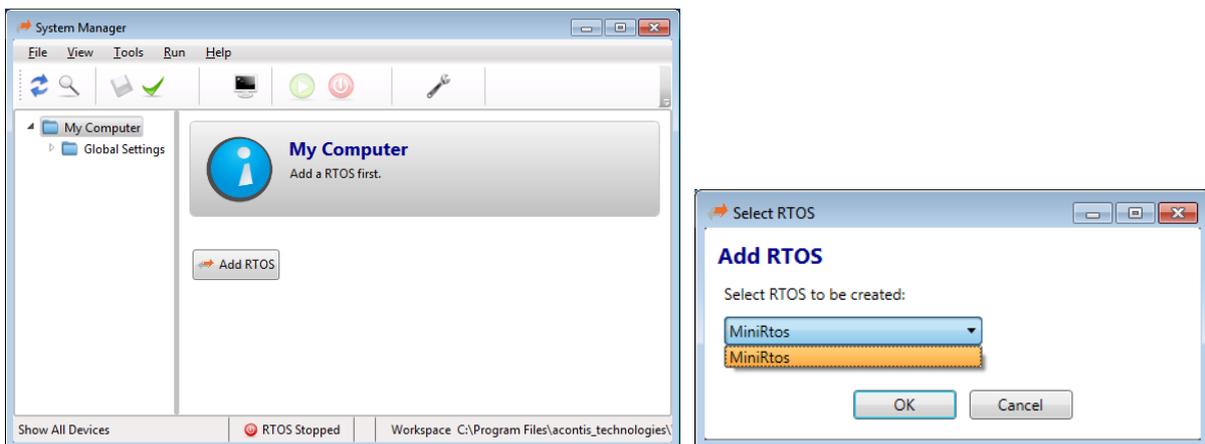
## 2.2 Configure and start a RTOS



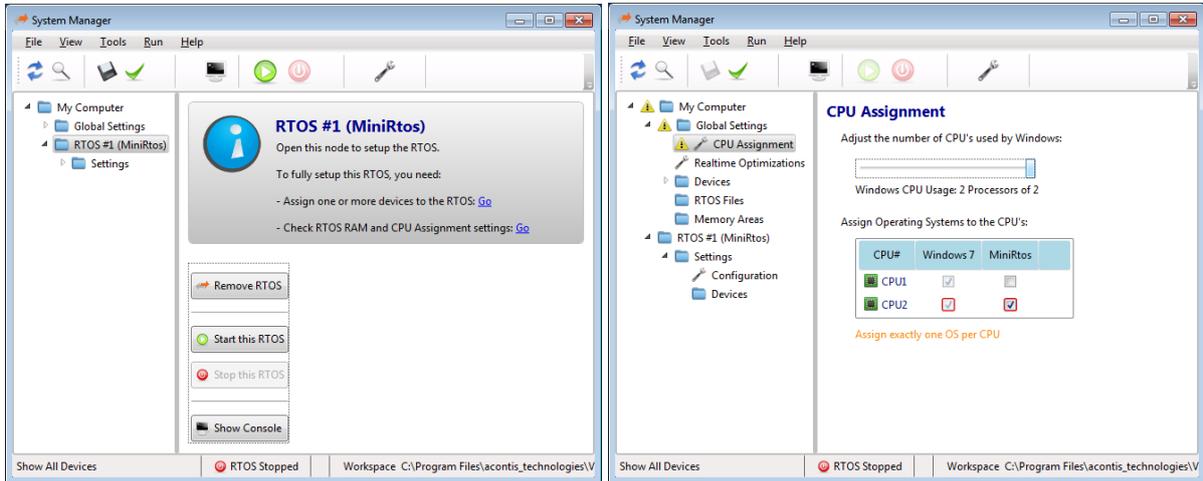
In the start menu you can find the System Manager, which can be used to configure, start and stop RTOS.



On the first launch System Manager asks where to save its workspace. All configuration data and files will be stored into that workspace. Additional workspaces can be added for different projects.



Now a RTOS can be added by "Add RTOS". Depending on the product different OS images can be selected. In case of VmfWin there is only MiniRtos.



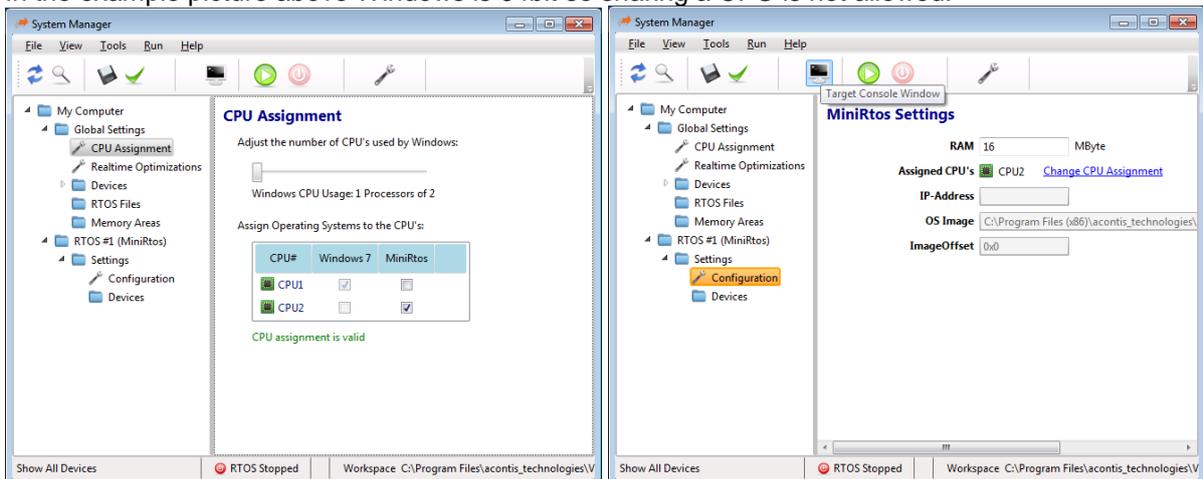
The next step is the CPU assignment.

A slider can be used to select the number of CPU's being usable by Windows  
 Checkboxes can be used to select the CPU(s) for a RTOS.

There are some restrictions:

- A 64bit OS can't share a CPU.
- An OS using multiple CPUs can't share a CPU with another OS also using multiple CPUs.
- A RTOS can't share a CPU with another RTOS.

In the example picture above Windows is 64bit so sharing a CPU is not allowed.

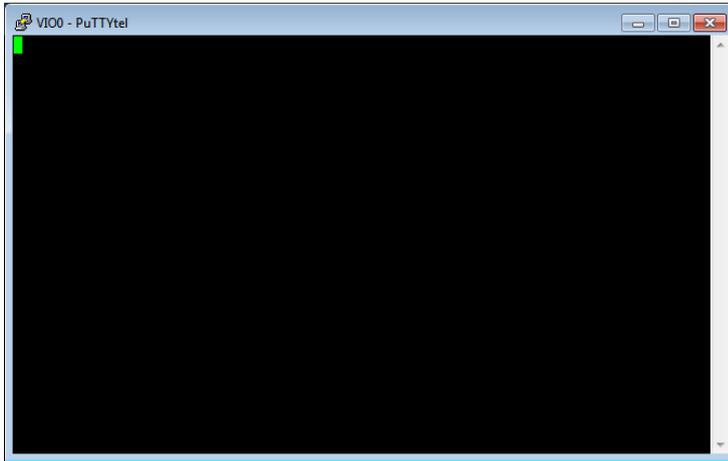


Now the CPU assignment is valid.

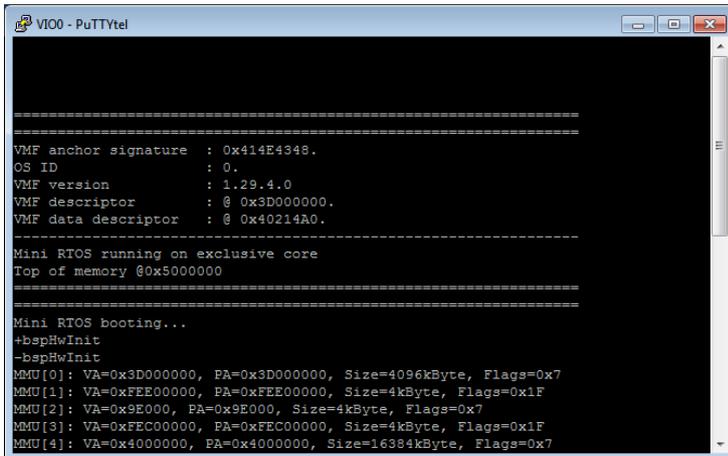
In the RTOS configuration the assigned memory (RAM) can be configured. A minimum RAM size is required to start a RTOS. The exact value depends on the RTOS to be started. The RAM size can be increased and the additional memory will automatically be used by RTOS.

Changing the RAM size will require a reboot.

Before starting a RTOS the Target Console (VIO) should be started using the screen icon in the system manager.



Now the RTOS can be started using the green “play” button. A dialog will ask to save the configuration.



Now there is some output on the Target Console so RTOS is running. Depending on the RTOS the console can be used for input, too. Also depending on the RTOS a telnet connection can be open.

A RTOS can be stopped using the red “power” button.

### 2.3 Assign a device to a RTOS

Please see chapter “4.2 Assign a device to a RTOS” for details.

### 3 Virtual Machine Framework

Using the ACONTIS RTOS-VM there is no need to understand the complex hardware of modern PC systems. The basic hardware components of the PC (architecture specific processor registers, timer, interrupt controller, memory handling/partitioning) can be accessed in the real-time software by simply calling the appropriate functions that the RTOS-VM hardware abstraction layer (HAL) provides.

Besides the HAL functions the RTOS-VM provides additional services, especially for communication with Windows:

- Shared Memory: Direct access to shared memory areas
- Shared Events: Notification using named events
- Data Access Synchronization: Interlocked Data Access
- Date and Time Synchronization
- Virtual Serial Channel
- Network Packet Library: basic Ethernet data transfer service
- RTOS configuration services (e.g. for dynamically setting the IP address of the virtual network)

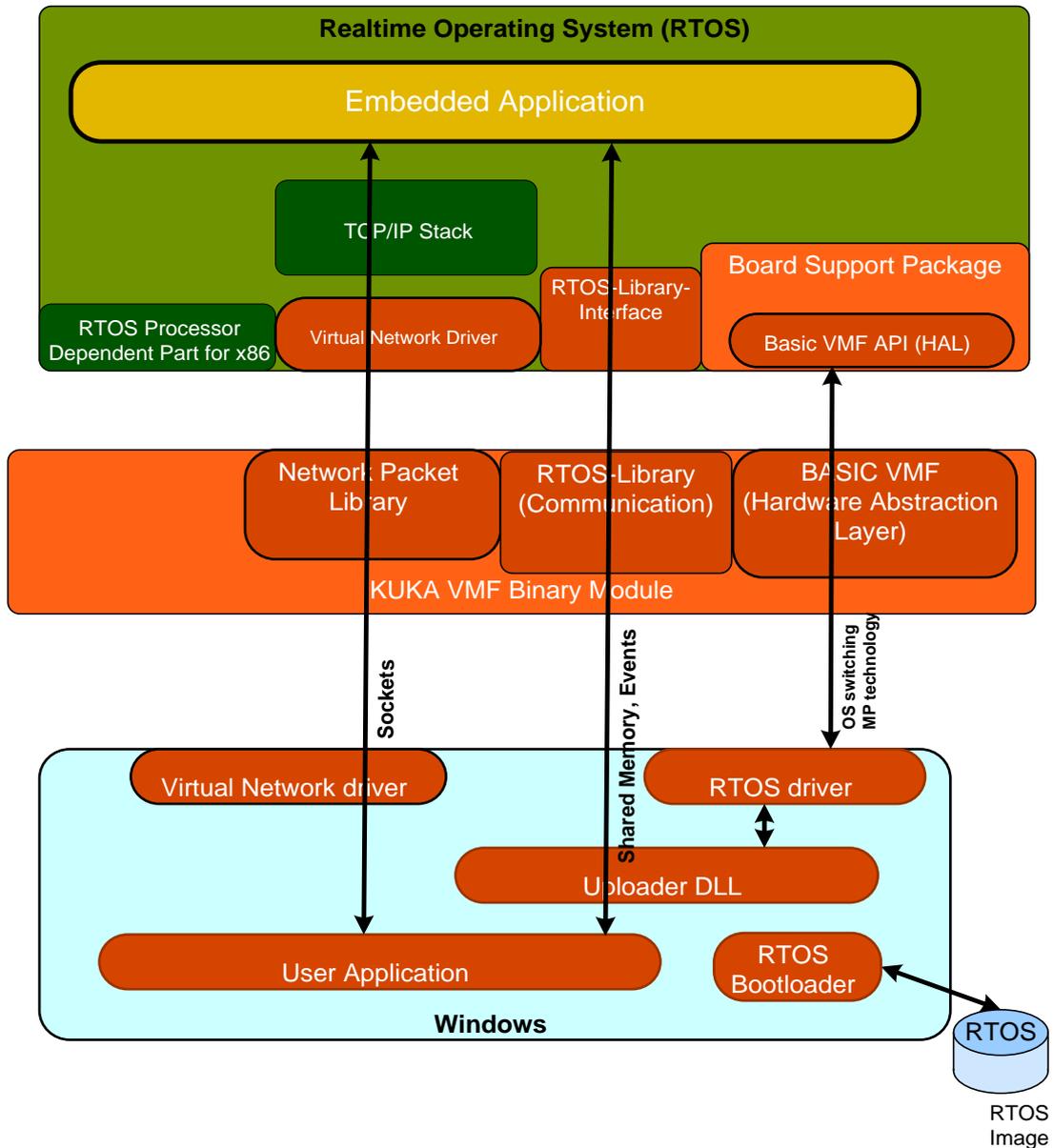
The application interface between the real-time software and the RTOS-VM is called the Virtual Machine Framework (VMF).

When calling VMF hardware functions the hardware will be directly accessed and not emulated. These functions are called the VMF Hardware Abstraction Layer (HAL) functions.

When using a product from the RTOSWin product family calling VMF functions is normally not necessary. Calling framework functions is done in the RTOS adaptation for the RTOS Virtual Machine (usually the Board Support Package).

### 3.1.1 VMF Architecture

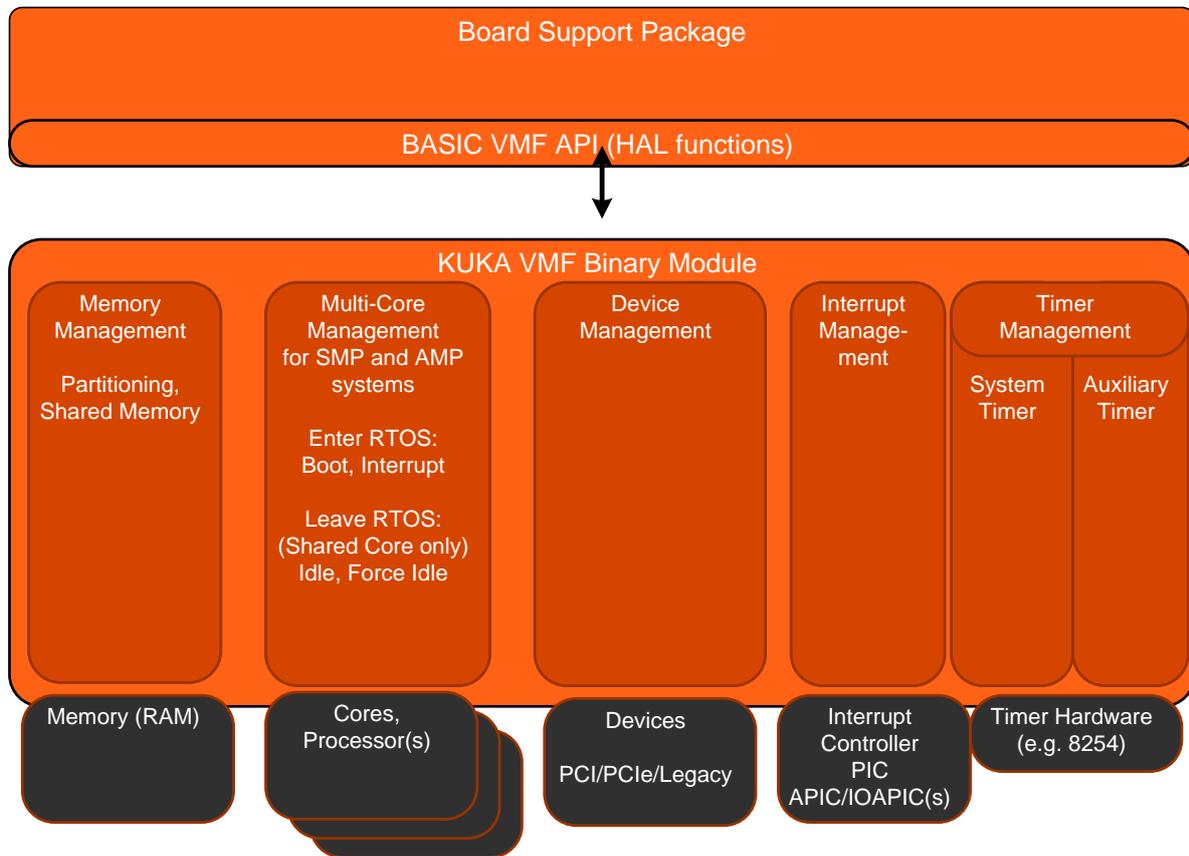
The following figure shows the general architecture of the VMF when a RTOS is embedded within Windows. Besides the basic VMF API (the HAL) which usually is required to build a RTOS BSP (Board Support Package) the VMF contains functions for communication between Windows and the RTOS (e.g. shared memory, events, network packet library). On top of the network packet library a virtual network driver can be built which will then provide a virtual network connection between Windows and the RTOS.



### 3.1.2 Basic VMF Services (Hardware Abstraction Layer)

The basic VMF services provide a simple programming interface to access the otherwise complex PC hardware.

The following figure shows in more detail the basic VMF services which usually are used within a RTOS Board Support Package.



When adapting a RTOS to run with the ACONTIS RTOS-VM there is no need to directly access PC hardware like timers or interrupt controllers.

The VMF as well provides a generic method for booting the RTOS and for setting up the RTOS memory context (virtual memory).

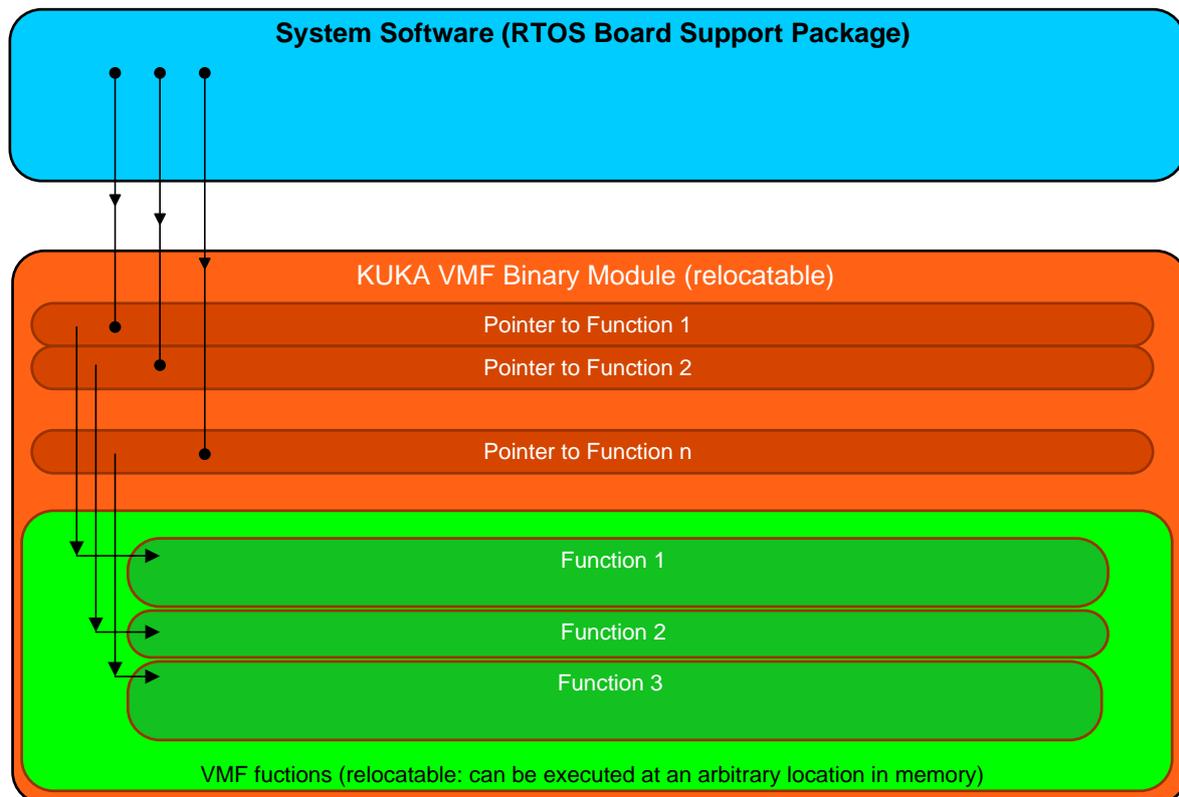
When running on multi-core systems the VMF also provides methods for executing a RTOS which supports Symmetric Multiprocessing (SMP).

Summarized, using the VMF one gets the following advantages:

- Fully virtualized hardware access (via Hardware Abstraction Layer functions). No need to understand the complex PC hardware.
- Either run the RTOS and Windows together on one single core or use dedicated cores exclusively for each operating system.
- The **same** RTOS image can be run either on a shared or a non-shared CPU core.
- Sophisticated Multi Core Support
  - Run the RTOS on one single or on multiple cores (SMP)
  - A RTOS can run in SMP mode even on dual core CPUs

### 3.2 Portability

When using standard frameworks or libraries the customer usually gets either source-code which in a first step would have to be ported to his specific environment (operating system, compiler, linker). In cases where the supplier of such a framework/library does not want to ship the source-code the customer would have to wait until a version for the framework/library is available for his environment. To avoid these implications the ACONTIS VMF is not shipped as a library or source code but as a relocatable binary module. This binary module will be loaded by the ACONTIS RTOS-VM at an arbitrary location in the memory (the VMF code can be executed at any location in memory!). Every call to a VMF function will then be redirected via well-known locations inside a jump table, this jump table is stored at a well-defined location inside the binary module. Thus, there is no need to port one single line of C language or assembly language code (and no need to add the VMF as an additional library to the customer's environment). The only requirement is to include one single header file. Within this header file the VMF functions are simply defined as macros which call the appropriate functions using the function pointer in the jump table.



Summarized, using the VMF binary module leads to the following advantages:

- No porting necessary, just include a C header file.
- No change necessary in the system software when new VMF versions are released (just exchange the binary module by the new one).
- The same binary VMF module will be used together with different RTOSes; this ensures a higher quality than if the VMF code would have been ported individually for any RTOS.

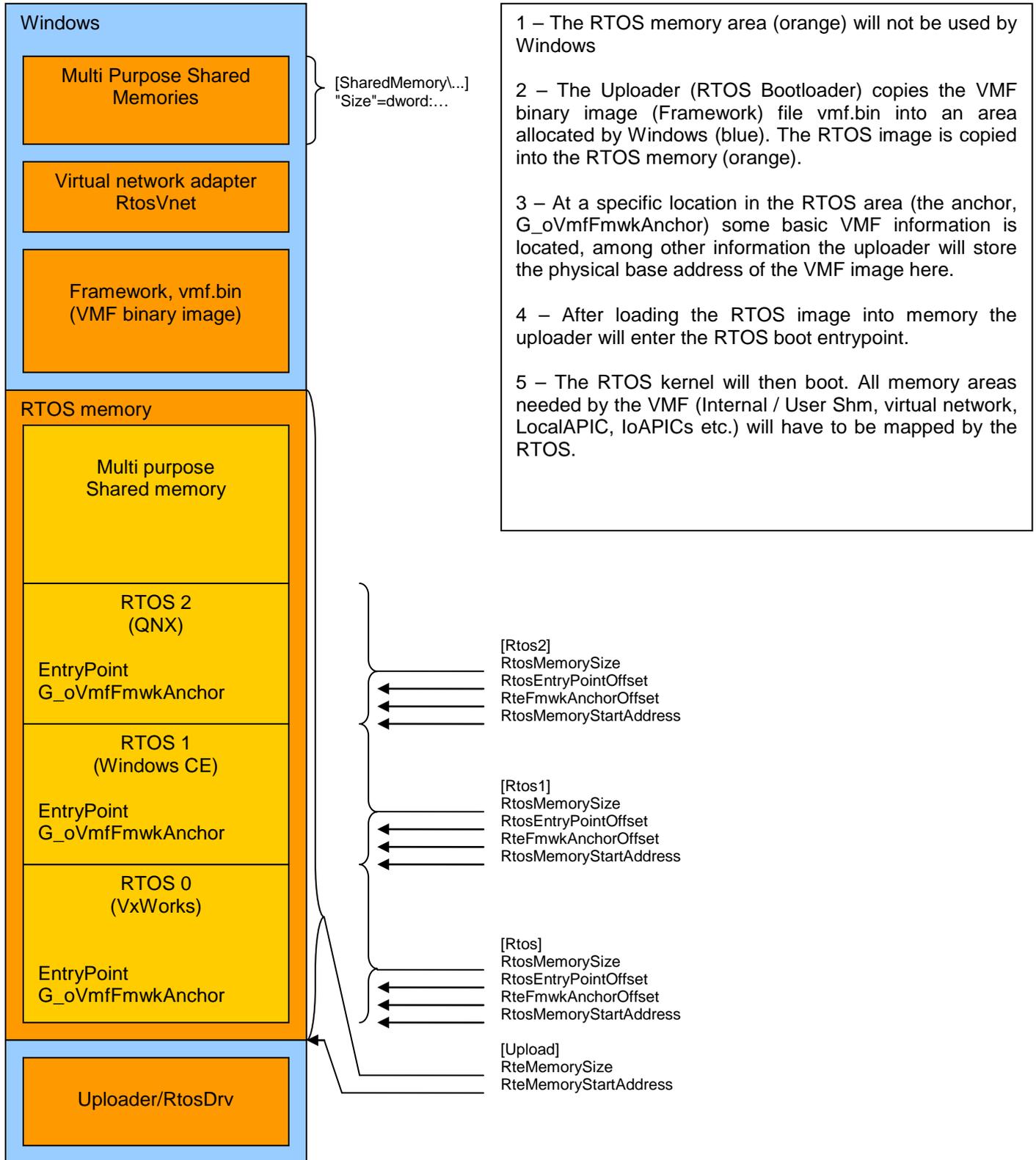
### 3.3 VMF management anchor

Some information about the VMF is needed within the RTOS, e.g. the physical base address of the framework binary image. This data is located at a specific location inside the RTOS memory. After loading the RTOS image into the memory the uploader will copy the VMF management data at the appropriate location inside the RTOS memory

### 3.4 Memory Layout

VMF = Virtual Machine Framework

RTOS Framework = RTOS interface (VMF interface functions)



### **3.5 The RTOS Library**

VMF communication service functions (e.g. for shared events) only provide basic services without any synchronization, some of them also have to be called within a well-defined memory context (ring 0 context, kernel context).

The Windows/RTOS communication services are therefore summarized within the RTOS library which is based on VMF services. This library is split into two parts, an OS independent part and an OS dependent part.

Synchronization (e.g. interrupt locking or mutexes) are part of the OS dependent part.

A detailed description of the RTOS library can be found in chapter 10.

## 4 Real-time Device Management

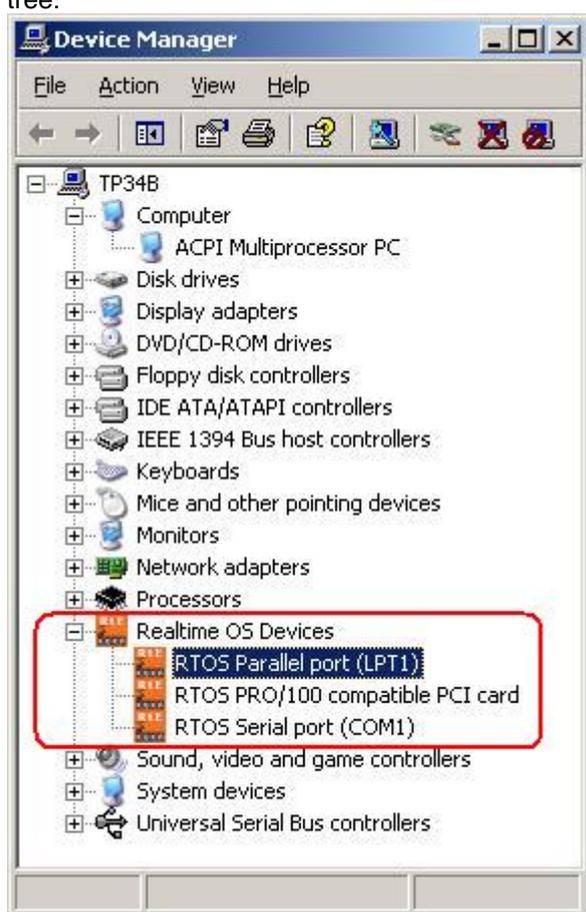
### 4.1 Overview

To achieve real-time behaviour the RTOS will have to directly access its hardware devices. In fact, hardware devices are never emulated, neither in Windows nor in the RTOS. Every specific device, e.g. a PCI network adapter card will, then either be used by Windows or by the RTOS exclusively.

In Windows all hardware devices which shall be used by the RTOS have to be managed by the generic Windows RtosPnp driver shipped with the ACONTIS RTOS-VM. It will forward all required information to RTOS.

In RTOS a device specific driver will be required.

Within the Windows Device Manager all RTOS devices will then appear in the “Realtime OS Devices” tree:



For the RTOS the Virtual Machine Framework (VMF) provides several methods to detect whether a device is assigned and usable or not:

- `vmfIdGetByName( szDeviceName, VMF_ID_DEVICE, ... )` : search by Name
- `vmfDevicePciIsForRtos( nBus, nDevice, nFunction, ... )` : search by PCI address
- `vmfDeviceIoIsForRtos( dwIoPort, ... )` : search by IO-Port

Older methods:

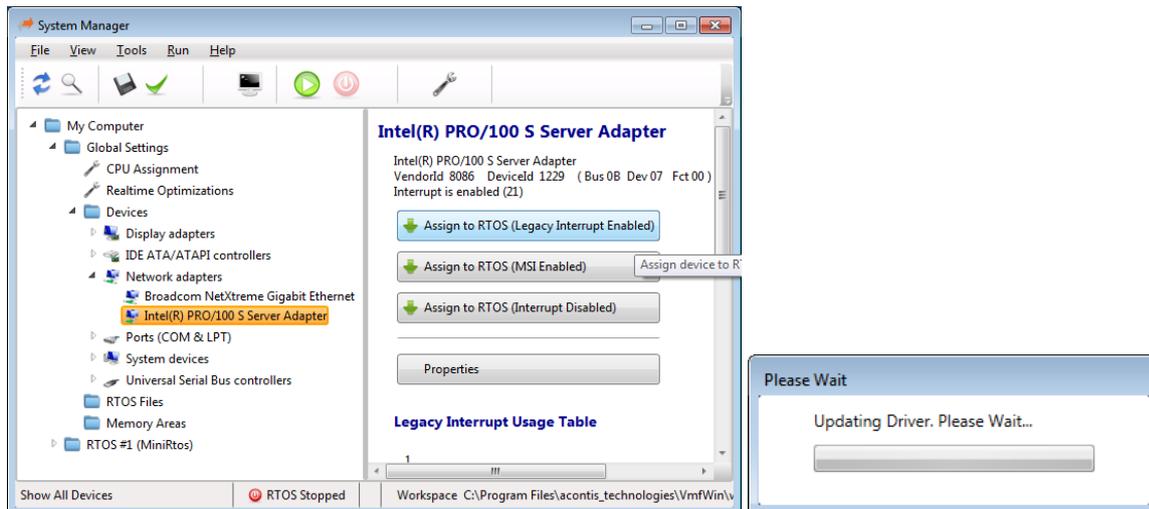
- `vmfDeviceIsForRtos( szDeviceName, ... )` : search by Name
- `vmfDeviceInterruptIdFromName( szDeviceName, ... )` : search by Name

More information about these functions can be found in the VMF documentation.

Usually the RTOS adaptation (e.g. the Board Support Package) for the RTOS-VM uses these functions internally. A RTOS user application normally doesn't need to call these functions. More information can be found in the corresponding documentation (e.g. in the VxWin, CeWin, QWin or Rtos32Win/RtVmf documentation).

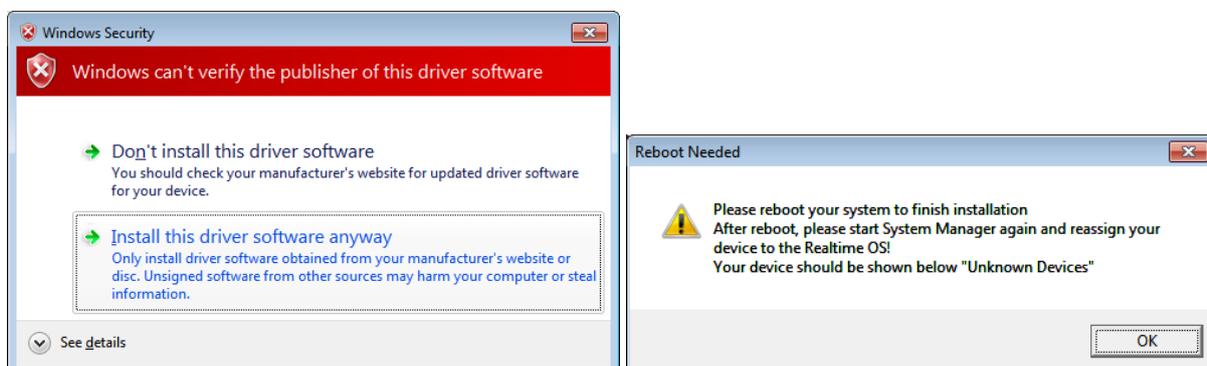
## 4.2 Assign a device to a RTOS

### 4.2.1 Using System Manager



The System Manager can be used to assign a device to a RTOS. There are three different types of assignment:

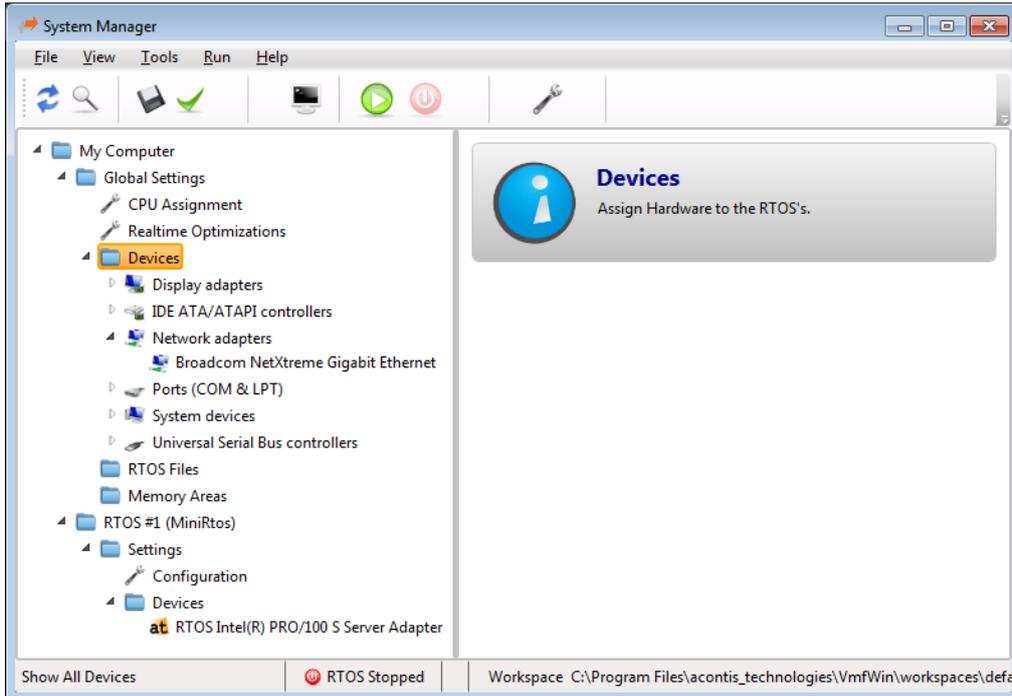
- 1) “Legacy Interrupt Enabled”  
The device will be configured to use a physical interrupt line.  
Advantage:
  - No PCI bus access required to enable or disable the interruptDisadvantage:
  - Interrupt conflicts when another card uses the same interrupt line.
- 2) “MSI Enabled”  
This option can be used for PCIe cards supporting Message Signalled Interrupts.  
Advantage:
  - No interrupt conflictsDisadvantage:
  - Enabling or disabling the interrupt requires PCI bus access.
- 3) “Interrupt Disabled”  
This option can only be used in combination with a specialized driver supporting polling mode. The acontis EC-Master Link Layers for example are such. Any regular network driver instead will always require an interrupt to work.  
Advantage:
  - No interrupt conflicts and no PCI bus accessDisadvantage:
  - Special driver required



When assigning a device to RTOS it is possible Windows asks how to continue. Please select “Install this driver software anyway” to continue.

Such a message can be prevented by providing a signed driver package, as described in chapter “4.5 Driver Signing”.

In case a reboot is required please reboot the system to continue.



After the device was assigned successfully it can be found below RTOS.

In case of starting the RTOS now brings up an interrupt conflict error please check chapter “4.3 Interrupt sharing conflicts” for possible solutions.

#### 4.2.2 Using RtosUpload.exe or RtosLib API

Device assignment can also be done

- by calling RtosUpload.exe using the option “/device”  
→ see chapter “7.2 Uploader operation, command line options”
- programmatically using RtosLib function “RtosDevice”  
→ see RtosDevice() API at chapter “11.1.20 RTOS Library – device functions”

### 4.3 Interrupt sharing conflicts

#### 4.3.1 Principle

When a physical device shall be controlled by the RTOS it will be under direct control of the RTOS. There is no virtualization of any physical device and no interference by the RTOS Virtual Machine. Thus, the regular device driver provided by the RTOS can be used (e.g. a network driver, USB driver, IEEE1394 driver, ...).

Usually the device driver will:

- a) read and/or write to device memory areas
- b) read and/or write to device I/O areas
- c) handle device interrupts

In case the device generates an interrupt, the driver's interrupt handler is responsible to acknowledge and handle the interrupt.

#### **Important:**

The same interrupt may never be used by both operating systems, Windows and the RTOS.

But why not?

Example: an Intel PRO/100 network adapter card that is using interrupt 20 shall be controlled by the RTOS and an USB host controller device which is used by Windows is also using interrupt 20. In that case, every time the USB host controller generates an interrupt the RTOS interrupt handler for the Intel PRO/100 network adapter card would be called (real-time interrupts have a higher priority than Windows interrupts). This handler is not capable to acknowledge and handle the interrupt of the USB host controller. One could now transfer control back to Windows to let the corresponding Windows handler process the interrupt. But in that case the real-time behaviour of the system would be violated. So, there is no way to both share interrupts between Windows and the RTOS and guarantee deterministic real-time behaviour for the RTOS.

#### 4.3.2 Understanding interrupt conflicts

In order to understand and finally prevent interrupt conflicts between several PCI cards, the physical arrangement of these cards should be carefully considered.

Each PCI board may generate up to four hardware interrupts on four physical interrupt lines (INTA, INTB, INTC and INTD). In most IBM-compatible PCs, each of the four interrupt lines on the PCI-bus is hard-wired to the next interrupt position, offset by one, in the neighbouring slot. This means that INTA of slot 1 is (typically) hard wired to INTB of slot 2, and to INTC of slot 3, and to INTD of slot 4.

Repeating this pattern, it is also wired to INTA of slot 5, and so on.

Since PCI card manufacturers generally lay out their boards to assert an interrupt on just one line (INTA), this has the affect of forcing the A-level interrupts of four adjacent cards to assert physically on INTA, INTB, INTC and INTD. In accord with this scheme, a fifth adjacent card would also assert its INTA on the same line as the first card. The next screenshot illustrates this principle.

**Caution:** Multi-function PCI boards may use more than one interrupt line.

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

But even when two PCI devices use different physical interrupt lines, they may be hard-wired together by the PC's *interrupt router*. The *interrupt router* maps various physical interrupt lines to the inputs of the Programmable Interrupt Controller.

The inputs of the *interrupt router* are the physical interrupt lines (refer to screenshot). The output of the *interrupt router* is connected to the interrupt controller of the PC. How many of the physical interrupt lines will be gathered together by the interrupt router depends on both the interrupt router and the number of free interrupts available at the Programmable Interrupt Controller.

If you desire, you can increase the number of free interrupt lines by using the BIOS to disable some PC components: COM ports, USB controller, or Audio/Sound controller, for example.

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 you will discover that several physical interrupts have been wired to a common interrupt on the controller.

Problems with sharing interrupts:

PCI devices that interrupt along the same interrupt pin route have no choice but to share an interrupt. The routing of interrupt pins to an *interrupt router* is system (chipset) dependent. While in many cases, finding and isolating the desired interrupt lines is not much of a problem, experience has nonetheless shown that it is not at all possible in some PCs to separate the interrupt lines as required by the RTOSWin solution.

Assume that Device A (installed under Windows) physically shares an interrupt with device B (installed under the RTOS). If Device A generates an interrupt while the RTOS is running, how could interrupt-handling software process the interrupt without impairing the ability of the real-time system to fulfil 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 the RTOS returns to its idle state and returns control to Windows. Using this approach, however, interrupts generated from device B would also be blocked for some time, a circumstance that would not be acceptable for real-time operations.
- By implementing an appropriate RTOS interrupt handler, one could prevent Device A from generating an interrupt until after the RTOS returned to Windows. Thereafter, the interrupt intended for Windows could be handled. While this theoretically could solve the dilemma, there could be no way to handle this in a general fashion. Requiring an *intelligent* real-time interrupt handler to be written for each device that shares an interrupt with a Windows device would very likely mean that the Windows device driver (depending on the device) would have to be modified, too. For this reason, this approach does not provide an adequate solution for handling shared interrupts either.

The foregoing scenarios can lead to but a single conclusion: In a dual system, such as the RTOSWin solution, Interrupt-sharing between Windows and the RTOS must be prohibited.

And that implies that PCI cards that are controlled by the RTOS may not be plugged into a PCI slot that uses the same hard-wired interrupt line (normally INTA) as does either an external or internal Windows device.

Conclusion: Windows devices can share Windows interrupts and RTOS devices can share RTOS interrupts, but never can interrupts be shared across the two operating systems.

Fortunately, most of the time, system designers can eliminate interrupt conflicts.

If it were possible to identify an otherwise unused interrupt pin route to real-time devices, the RTOS could manage multiple interrupts along that route via interrupt sharing.

Custom boards that provide a means for specifying on which of the interrupt pins – INTA#, INTB#, INTC#, or INTD# – the board's interrupts should be asserted, very well suit the scheme of having multiple-function or single-function PCI devices sharing an interrupt. Such boards allow great flexibility in combining devices that require real-time servicing with those that do not.

### 4.3.3 Resolving interrupt conflicts

Prior to start resolving interrupt conflicts the device has to be under control of the RTOS. This has to be done by installing the RtosPnp Device Driver (see section 4.4). This driver will request an exclusive interrupt from the Windows Resource Manager. If no other device requests an exclusive interrupt and if the hardware (motherboard, chipset) is also capable of providing an exclusive interrupt then a unique interrupt will be assigned for this device.

If this fails, there are several ways how to resolve interrupt conflicts then.

- a) Try to find another slot where to insert the PCI/PCIe card.
- b) If capable then configure the device to use message signalled interrupt (MSI) instead (see section 4.4.3.3)
- c) Disable all unused devices in the BIOS
- d) Disable the conflicting Windows device either by disabling it in the BIOS or by disabling it in the Windows Device Manager.

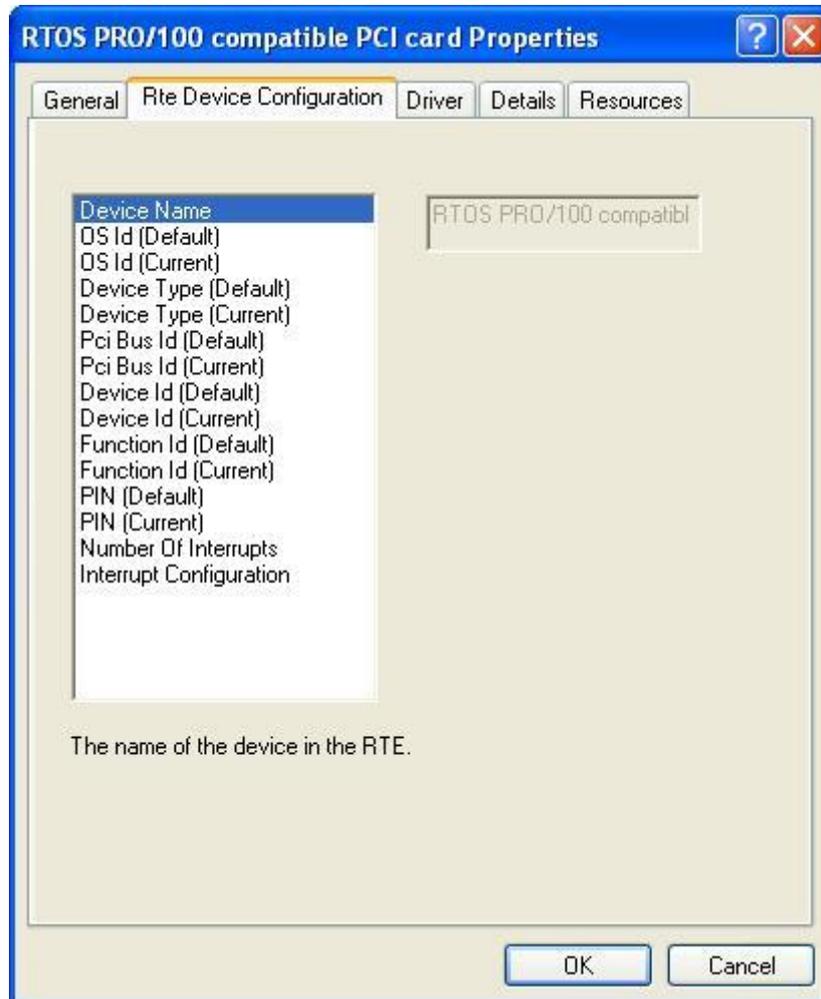
In case your RTOS device driver does not use interrupts (i.e. if used polling) you may configure the device using polling mode. This can be either done by the System Manager, RtosUpload.exe, RtosLib RtosDevice() or by modifying the standard Windows INF file that is used to assign the device to the RTOS. See section 4.4.3.3 for more information.

Sometimes a Windows device is assigned an interrupt but the device never generates an interrupt. For example, in many cases the SMBus device doesn't generate interrupts even if it is assigned one. In that case you may also ignore interrupt conflicts. But it is also necessary to modify the standard Windows INF file that is used to assign the device to the RTOS. See section 4.4.3.5 for more information.

## 4.4 Configuration

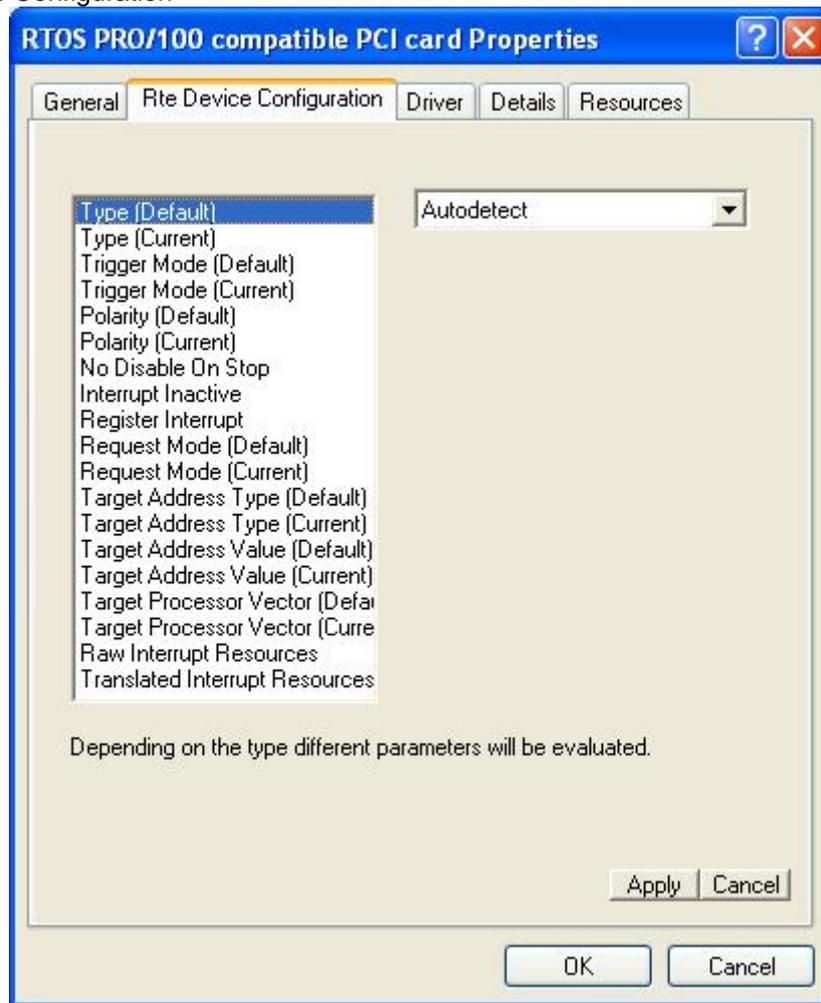
### 4.4.1 Properties dialog

Using the Windows Device Manager, it is possible to change the device configuration parameters. The meaning of the parameters is identical to the corresponding settings in section [DriverInstall\_HwAddRegUsrDef] in the INF file of the driver (see section 4.4.2).



Rte Device Configuration parameter	Corresponding entry in [DriverInstall_HwAddRegUsrDef]
OS Id	OsId
Device Type	DeviceType
Pci Bus Id	PciBus
Device Id	PciDevice
Function Id	PciFunction
PIN	PciPin

## Interrupt Source Configuration



Rte Device Configuration parameter	Corresponding entry in [DriverInstall_HwAddRegUsrDef]
Type	InterruptType
Trigger Mode	InterruptTriggerMode
Polarity	InterruptPolarity
No Disable On Stop	InterruptDontDisableOnRtosStop
Interrupt Inactive	InterruptDoesntInterrupt
Register Interrupt	InterruptRegisterToWindows
Request Mode	InterruptRequestMode
Target Address Type	InterruptTargetAddressType
Target Address Value	InterruptTargetAddress
Target Processor Vector	InterruptProcessorVector
Raw Interrupt Resources	Windows internal information (read only)
Translated Interrupt Resources	Windows internal information (read only)

## 4.4.2 RtosUpload.exe / RtosLib API

The RtosUpload.exe option "/device" and RtosLib API "RtosDevice()" allows the modification of some device configuration values like interrupt mode, device name and OS id using parameter 'rte\_configure'.

See RtosDevice() API at chapter "11.1.20 RTOS Library – device functions" for details

## 4.4.3 Windows INF file

When a device should be assigned to RTOS a Windows INF file is required to install the generic RtosPnp driver for Windows so the device information can be forwarded to RTOS.

There are several RtosPnp INF files shipped with the product, but under some circumstances it is required to create a new file:

- Support a new device and prevent the "Windows Security Warning" during driver installation
- An option should be changed to another default (for example always using polling mode) so no re-configuration is required after assigning the device

After creating or modifying an INF file it must be signed to prevent the "Windows Security Warning" during driver installation – see chapter "4.5 Driver Signing" for details.

### 4.4.3.1 Supporting a new device

In most cases the template INF file has only to be adjusted by a few changes.

#### 4.4.3.1.1 Device Names

The device name has to be adjusted, optionally several additional name string may be adjusted.

Template INF file:

```
[Strings]
DEVICEDISPLAYNAME      = "RTOS Device Name"                ; ®
DEVICECLASSNAME        = "Realtime OS Devices"          ; (o)
MFGNAME                 = < acontis technologies GmbH >    ; (o)
INSTDISKNAME           = "acontis technologies GmbH Installation Disc" ; (o)
SERVICEDISPLAYNAME     = "RTOS PnP Driver"              ; (o)
SERVICEDESCRIPTION    = "RTOS PnP Driver"              ; (o)
```

Example adjustments (device name and company name):

```
[Strings]
DEVICEDISPLAYNAME      = "My PCI Card"                ; ®
DEVICECLASSNAME        = "Realtime OS Devices"          ; (o)
MFGNAME                 = "My Company"                 ; (o)
INSTDISKNAME           = "My Company Installation Disc" ; (o)
SERVICEDISPLAYNAME     = "RTOS PnP Driver"              ; (o)
SERVICEDESCRIPTION    = "RTOS PnP Driver"              ; (o)
```

#### 4.4.3.1.2 PCI Device Identifications

A PCI device is uniquely identified by at least two identifiers: the Vendor ID (e.g. 8086 for Intel) and the Device ID (e.g. 1229 for the Intel PRO/100 device).

Some PCI devices belong to a device family (for example the Intel PRO/1000 family). In this case the same INF file may be used by several such devices.

The [DeviceList] section in the INF file contains at least entry with the appropriate vendor and device id.

Template INF file:

```
[DeviceList]
%DEVICEDISPLAYNAME%    = DriverInstall, PCI\VEN_FFFF&DEV_FFFF    ; ®
```

Example adjustments (two devices with vendor id ABCD and device ids 1234 and 5678):

```
[DeviceList]
%DEVICEDISPLAYNAME%    = DriverInstall, PCI\VEN_ABCD&DEV_1234    ; ®
%DEVICEDISPLAYNAME%    = DriverInstall, PCI\VEN_ABCD&DEV_5678    ; ®
```

#### 4.4.3.2 Forcing the RtosPnp driver to be loaded – determine interrupt sharing conflicts

In case the device which shall be controlled by the RTOS (using the RtosPnp driver) shares its interrupt with another device that is controlled by Windows two scenarios may occur:

- a) The Windows Device Manager successfully loads the RtosPnp driver
- b) The Windows Device Manager refuses to load the RtosPnp driver

The reason why sometimes the driver is not loaded is an entry in the INF file that forces the Windows Resource Manager to assign a unique interrupt to the device. If Windows cannot find a unique interrupt then it may not load the driver.

To avoid this behaviour the entry in the INF file has to be changed in a way that the Windows Resource Manager is allowed to assign an interrupt that is already used by another device.

There are several ways to change the interrupt mode to MSI:

- By modifying the devices Windows INF file to change the installation default:

Template INF file (by default the entry is not existing which means device exclusive):

```
[DriverInstall_HwAddRegUsrDef]
;HKR,"ConfigInterrupt0", "InterruptRequestMode", %REG_DWORD%, 0x00
; (o) 00=exclusive, 01=shared
```

Required adjustment (uncomment and set the value to 0x01):

```
[DriverInstall_HwAddRegUsrDef]
HKR,"ConfigInterrupt0", "InterruptRequestMode", %REG_DWORD%, 0x01
; (o) 00=exclusive, 01=shared
```

After adjusting the INF file a driver update with the new INF file has to be executed.

- Using the Windows device manager  
Select the RTOS device driver, open device properties, select the tab "Rte Device Configuration", select and click "Interrupt Configuration" and change "Request Mode (Default)" from "Exclusive" to "Shared". At last click "OK" and reboot to update the settings.

#### 4.4.3.3 Use message signalled interrupt (MSI) to solve interrupt conflicts

If a device and the chipset are capable of using message signalled interrupts (MSIs) a RTOS controlled PCI device can be configured to use MSI instead of a line interrupt.

This solves interrupt conflicts often caused by the classic line interrupt.

The advantage of MSI instead of classic line interrupt is that it needs not to be shared with other devices. The line interrupt is a wire connecting multiple slots on the mainboard and so the same wire might be shared by multiple cards. MSI on the opposite uses the PCI bus address and data lines to generate an interrupt message. The only limit for exclusivity on MSI is the number of available CPU interrupt vectors.

In case a device is configured to use MSI but it does not support MSI an error message will be shown as soon as the driver wants to enable the interrupt.

A device can be configured for using MSIs either by modifying the inf file before the device driver installation or by changing the device configuration using the Windows device manager after the installation.

There are several ways to change the interrupt mode to MSI:

- In the System Manager device options
- By using RtosDevice() API directly or by RtosUpload.exe – see RtosDevice() API for details!
- By modifying the devices Windows INF file to change the installation default:

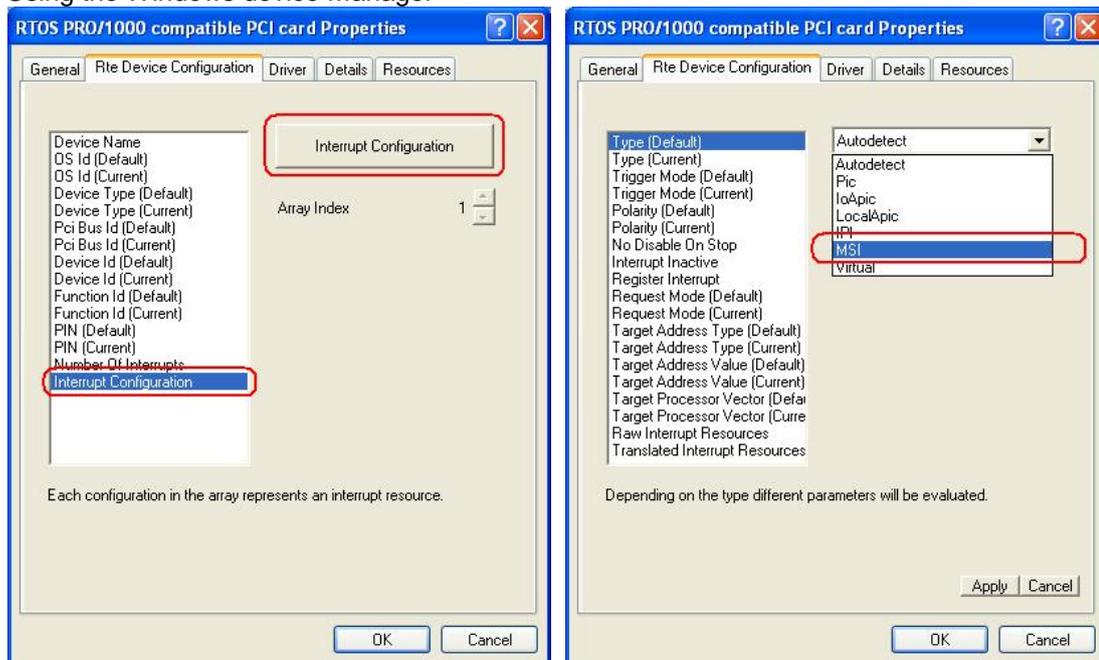
Template INF file:

```
[DriverInstall_HwAddRegUsrDef]
;HKR,"ConfigInterrupt0", "InterruptType", %REG_DWORD%, 0xFFFFFFFF
```

Required adjustment:

```
[DriverInstall_HwAddRegUsrDef]
HKR,"ConfigInterrupt0", "InterruptType", %REG_DWORD%, 0x04
```

- Using the Windows device manager



Select the RTOS device driver, open device properties, select the tab “Rte Device Configuration”, select and click “Interrupt Configuration” and change “Type (Default)” from “Autodetect” to “MSI”. At last click “OK” and reboot to update the settings.

#### 4.4.3.4 RTOS controlled PCI devices not using interrupts

Sometimes the RTOS device driver that controls the PCI device does not require to handle interrupts from this device.

In such cases it is possible that other devices which are controlled by Windows are allowed to use the same interrupt line as the RTOS device would use – normally it is not allowed that the same interrupt line is used by both, Windows and the RTOS (see section 4.2).

There are several ways to change the interrupt mode to polling:

- In the System Manager device options
- By using RtosDevice() API directly or by RtosUpload.exe – see RtosDevice() API for details!
- By modifying the devices Windows INF file to change the installation default:

Template INF file:

```
[DriverInstall_HwAddRegUsrDef]
;HKR,"ConfigInterrupt0", "InterruptDoesntInterrupt", %REG_DWORD%, 0x00
```

Required adjustment:

```
[DriverInstall_HwAddRegUsrDef]
HKR,"ConfigInterrupt0", "InterruptDoesntInterrupt", %REG_DWORD%, 0x01
```

After adjusting the INF file, a driver update with the new INF file has to be executed.

#### 4.4.3.5 RtosPnp driver Windows INF file parameters

This section describes all Windows INF file parameters of the RtosPnp driver that can be adjusted to fit to specific requirements. Shipped with the RTOSWin solution are some pre-defined INF files and two template INF files, RTOS\_Template.inf (template for one PCI device configuration) and RTOS\_MFC\_Template.inf (template for multiple PCI device configurations).

##### 4.4.3.5.1 Section [DriverInstall\_HwAddReg]

- **FriendlyName**  
If this entry exists it will override any automatically generated device friendly name. Normally the device's friendly name will be automatically generated by the RTOS Device Class Installer (RtosPnpInstaller.dll), this name is based on the DEVICEDISPLAYNAME defined in section [Strings].

##### 4.4.3.5.2 Section [Strings]

- **DEVICEDISPLAYNAME**  
Will be displayed as device name (in the Windows Device Manager)
- **DEVICECLASSNAME**  
Will be displayed as the device class name (in the Windows Device Manager)
- **MFGNAME**  
Manufacturer name
- **INSTDISKNAME**  
Name of the installation resource
- **SERVICEDISPLAYNAME**  
Name of the driver
- **SERVICEDescription**  
Description of the driver

##### 4.4.3.5.3 Section [RtelInstall]

- **AutoDeviceFriendlyName**  
The RTOS Device Class Installer uses this entry to determine how to create the device's friendly name (which is shown in the Windows Device Manager).  
Bit 0: create a friendly name if set to 1 (otherwise DEVICEDISPLAYNAME will be used by Windows)  
Bit 1: if set to 1 then append the DOS device (COMx, LPTx)  
Bit 2: if set to 1 then append a unique ID (assure that all device names are unique)

#### 4.4.3.5.4 Section [DriverInstall.RtelInstall]

- MatchBusDevFunc = 0xFFFFFFFF00  
this INF file will only be valid for PCI devices which are matching the given pattern.  
Value pattern 0xBBDDFFUU:
  - BB = PCI bus index
  - DD = PCI device index
  - FF = PCI function index
  - UU = unused (reserved)If the value for BB, DD or FF is set to 0xFF then all such devices will match, otherwise only the specified one will match.  
Example: 0xFFFF0100 → only the PCI device with function 01 will match (bus and device don't care).  
This entry is required in case that two identical PCI cards shall be controlled by the RTOS and different settings in the INF file shall be used (e.g. one device generates an interrupt and the second does not). In this case either two different INF are required or one INF file with two different configurations has to be created.

#### 4.4.3.5.5 Section [DeviceList]

This section contains at least one condition which PCI device shall use the RtosPnp driver. The PCI device is determined by its PCI vendor and device ID. If more than one PCI device shall use the RtosPnp driver then multiple of these entries have to be made.

- %DEVICEDISPLAYNAME% = DriverInstall, PCI\VEN\_XXXX&DEV\_YYYY  
PCI devices with vendor id XXXX and device id YYYY will use this INF file.

#### 4.4.3.5.6 Section [DriverInstall\_HwAddRegUsrDef]

- **OsId**  
determines the OS the device should be assigned to: FFFFFFFFE=auto, FFFFFFFFD=OsIndependent FFFFFFFFC=Host(Windows), else=value
- **DeviceType**  
determines the device type: FFFFFFFFE=auto, 00=other, 01=virtual, 02=PCI
- **PciBus**  
determines the PCI bus index value: FFFFFFFFE=auto, FFFFFFFF=unused, else=value
- **PciDevice**  
determines the PCI device index value: FFFFFFFFE=auto, FFFFFFFF=unused, else=value
- **PciFunction**  
determines the PCI function index value: FFFFFFFFE=auto, FFFFFFFF=unused, else=value
- **PciPin**  
determines the PCI interrupt pin value: FFFFFFFFE=auto, FFFFFFFF=unused, else=value
- **ConfigInterrupt0**  
These entries describe the source of the first interrupt on the device. If a device is capable to generate more than one interrupt (e.g. PCIe devices using MSIs) then additional ConfigInterruptX entries exist (ConfigInterrupt1, ConfigInterrupt2, ...)
  - **InterruptType**  
FFFFFFFFFE=auto detect, normally you should not change this value (00=Pic, 01=IoApic, 02=LocalApic, 03=IPI, 04=MSI, 05=Virtual)
  - **InterruptTriggerMode**  
FFFFFFFFFE=auto detect (PCI = level, legacy = edge)  
00: set to level triggered  
01: set to edge triggered
  - **InterruptPolarity**  
FFFFFFFFFE=auto detect (PCI = low/falling, legacy = high/rising)  
00: low level when level triggered or falling edge when edge triggered  
01: high level when level triggered or rising edge when edge triggered
  - **InterruptDontDisableOnRtosStop**  
0: disable interrupts after the RTOS is stopped (to avoid system crash)  
1: don't disable interrupt on RTOS stop (requires a handler on Windows)
  - **InterruptDoesntInterrupt**  
0: device generates interrupts  
1: device doesn't generate interrupts (allow interrupt sharing with Windows)
  - **InterruptRegisterToWindows**  
0: don't register a interrupt handler to Windows  
1: register a interrupt handler to Windows (normally this is required if the device generate interrupts)  
Note: if InterruptDoesntInterrupt is set to 1 no handler will be registered, the value set for InterruptRegisterToWindows is ignored
  - **InterruptRequestMode**  
00: exclusive  
01: shared  
Note: if InterruptDoesntInterrupt is set to 1 the value set for InterruptRequestMode is ignored
  - **InterruptTargetAddressType**  
FFFFFFFFFE=auto detect, normally you should not change this value  
00=Processor bit mask,  
01=Local Apic Id,  
02=Logical Id (Flat),  
03=Logical Id (Cluster)  
Currently only the Local APIC is supported (value 01)!
  - **InterruptTargetAddress**  
Interrupt target address, meaning depends on InterruptTargetAddressType (01 = local APIC ID, this value will be determined automatically)
  - **InterruptProcessorVector**  
Interrupt vector to be used.  
FFFFFFFFFE=auto detect  
00 .. FF: manually determined interrupt vector  
The following restrictions for the interrupt vector exist:  
a) If Windows is using one single CPU core and the RTOS is running in exclusive

mode, values from 0x00 up to 0xFF are allowed. Note: automatic configuration will only use values between 0xE0 and 0xFF.  
b) If Windows is using more than one CPU core, values have to be 0xF0 or higher.  
c) If the RTOS is running in shared mode and Windows is using only one single CPU core, values have to be 0xE0 or higher.

#### 4.4.4 RTOS config file

##### 4.4.4.1 Windows controlled PCI devices not using interrupts

Sometimes a conflicting Windows device is known to never really generate interrupts. In such cases it is possible that other devices which are controlled by the RTOS are allowed to use the same interrupt line as the Windows device would use – normally it is not allowed that the same interrupt line is used by both, Windows and the RTOS (see section 4.2).

If the RTOS is started by the Windows Uploader it will first check for interrupt conflicts. If such conflicts are detected an error message will be shown and the RTOS will not be started. The error message identifies the conflicting devices.

Example:

Interrupt conflict between a PRO/100 card that shall be controlled by the RTOS and a USB host controller that shall be controlled by Windows. The following error message will be displayed.

```
ERROR: Device Configuration - conflicting devices for interrupt (16)
- Intel® 6300ESB USB universal host controller - 25A9
  (PCI\VEN_8086&DEV_25A9&SUBSYS_25A18086&REV_02\3&267A616A&0&E8)
- RTOS PRO/100 compatible PCI card
  (PCI\VEN_8086&DEV_1229&SUBSYS_000C8086&REV_08\4&3ABFD0AC&0&00F0)
```

The USB host controller is identified by the following device string:

```
PCI\VEN_8086&DEV_25A9&SUBSYS_25A18086&REV_02\3&267A616A&0&E8
```

In case you know that this USB host controller will never generate interrupts you may insert the following configuration setting into the RTOS configuration file:

```
[WindowsDevices]
"MaxInterruptShareDeviceIndex"=dword:01
"InterruptShareDevice0"="PCI\VEN_8086&DEV_25A9&SUBSYS_25A18086&REV_02\3&267A616A&0&E8"
```

The value of InterruptShareDevice0 must be identical to the device name shown in the Uploader error message. Wildcards '\*' for multiple and '!' for a single character are supported.

## 4.5 Driver Signing

Digital signatures are used to prevent viruses from installing or manipulating drivers by ensuring their integrity. To be able to install a driver without a signing warning or error the signatures must be correct.

Since Windows Vista it is possible for a software publisher to sign a driver using a KernelMode CodeSigning certificate, which can be bought from a Certificate Authority (CA).

### 4.5.1 Driver Package Signing

A RtosPnp driver package typically contains:

- RtosPnp.sys	Driver
- RtosPnpInstaller.dll	Class(Co)Installer
- WdfCoInstaller01009.dll	Driver Framework
- MyDriver.inf	Inf-File
- MyDriver.cat	Catalog file

RtosPnp.sys, RtosPnpInstaller and WdfCoInstaller01009.dll are signed to prevent code manipulation. MyDriver.inf is not signed – its integrity is ensured by the catalog file.

The catalog file contains the filename and hash information about all other files which are part of the driver package. To protect against any changes the file is signed.

After modifying an Inf-File it is required to rebuild and sign the catalog file. The catalog filename is defined by the Inf-File entry “CatalogFile”.

Prerequisites:

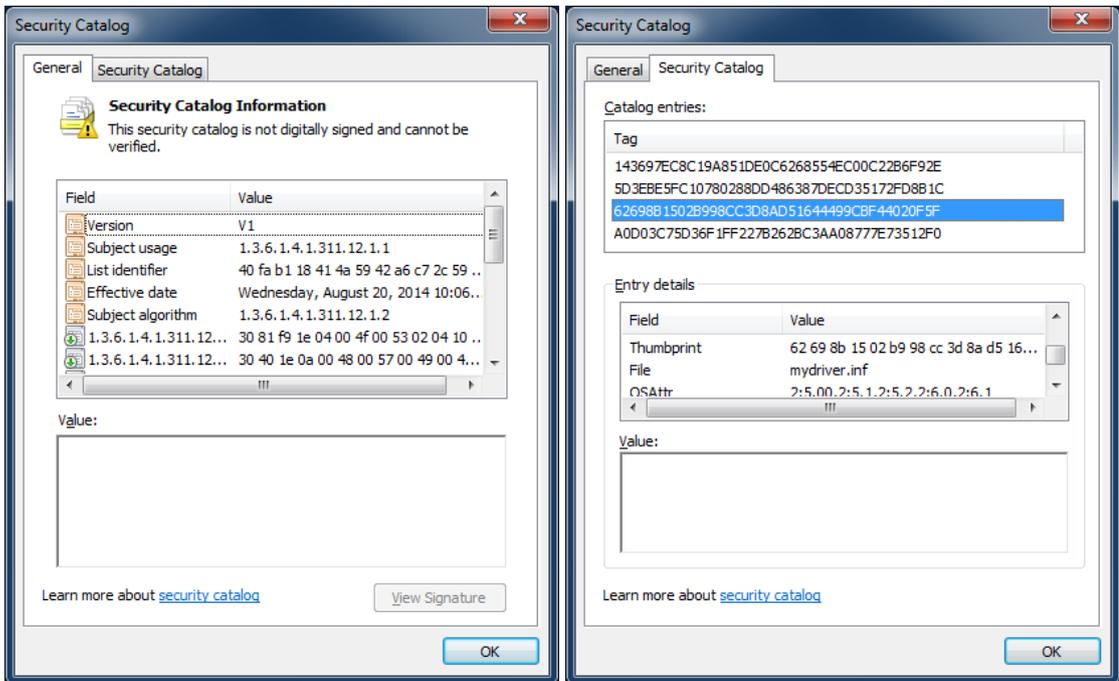
- Windows Driver Kit for the code signing tools
- A “Kernel Mode Code Signing” certificate issued for example from DigiCert, GlobalSign, Thawte, VeriSign or any other authority listed in Microsoft “Cross-Certificates for Kernel Mode Code Signing”
- A Microsoft Cross-Certificate corresponding to the “Kernel Mode Code Signing” certificate.  
A list can be found in the MSDN “Cross-Certificates for Kernel Mode Code Signing”  
[http://msdn.microsoft.com/en-us/library/windows/hardware/dn170454\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/dn170454(v=vs.85).aspx)
- The driver package to be signed

Signing:

- 1) Open a WDK Build Environment – for example Windows 7 “x86 Free Build Environment”.
- 2) Change into the directory containing all your driver package files.
- 3) Ensure the Inf-File contains all required modifications.  
Tip: The Inf-File name and “CatalogFile” entry should be different from their original.
- 4) Delete any existing catalog file in the driver package directory.
- 5) Creating a new catalog file by calling:

```
C:\MyDriverPackage>inf2cat.exe /driver:"."  
/os:2000,XP_X86,XP_X64,Server2003_X86,Server2003_X64,Vista_X86,Vista_X64,Server2008_X86,Server2008_X64,7_X86,7_X64,Server2008R2_X64 /verbose
```

You may add or remove supported OS as required. A list of possible options will be shown calling: “inf2cat.exe /?”

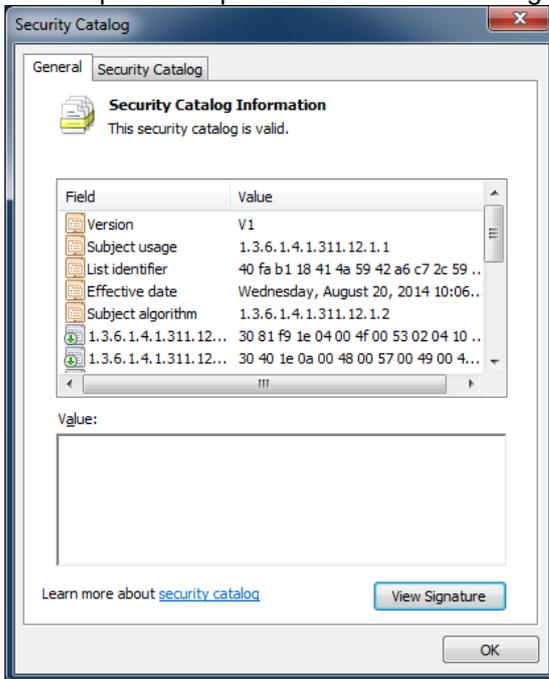


The created catalog file can be open by a double click. It is currently unsigned. The Security Catalog lists all covered files.

- 6) Sign a catalog file by calling:

```
C:\MyDriverPackage>signtool.exe sign /v /ac "C:\MyCerts\MyCrossSignCert.cer" /f
"C:\MyCerts\MyCodeSignCert.pfx" /p MyCodeSignCertPassword /t
http://timestamp.verisign.com/scripts/timestamp.dll MyDriver.cat
```

A list of possible options will be shown calling: "signtool.exe /?"



After signing the catalog file it should be shown as valid and contain a signature.

- 7) Verifying a catalog file for "default authenticode signing policy" by calling:

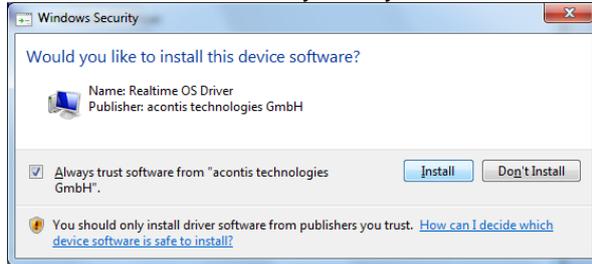
```
C:\MyDriverPackage>signtool.exe verify /tw /pa /v MyDriver.cat
```

Please remember that this has to be done separately for 32- and 64-bit driver package.

## 4.5.2 Certificate Pre-Installation

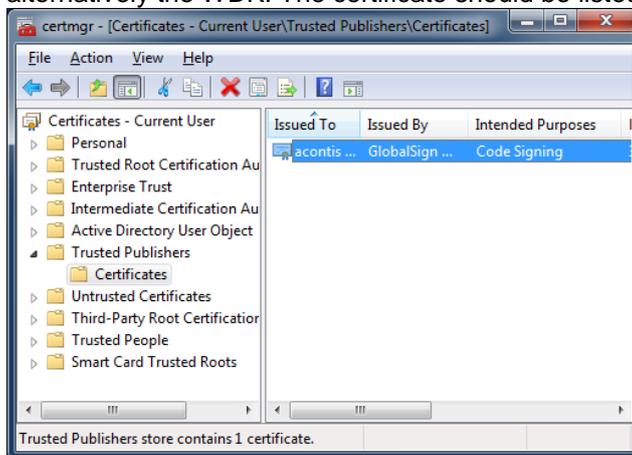
To prevent the “Do you trust this publisher” question during driver installation the certificate has to be installed in the “Trusted Publishers Certificate Store”.

This is done automatically after you once select “Always trust software from ...”.

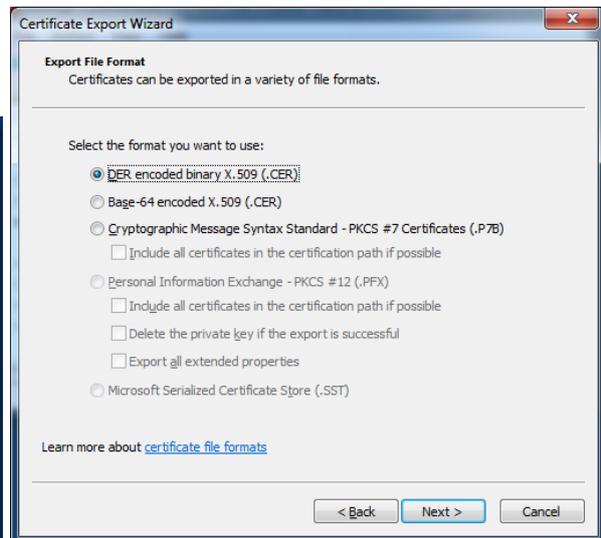
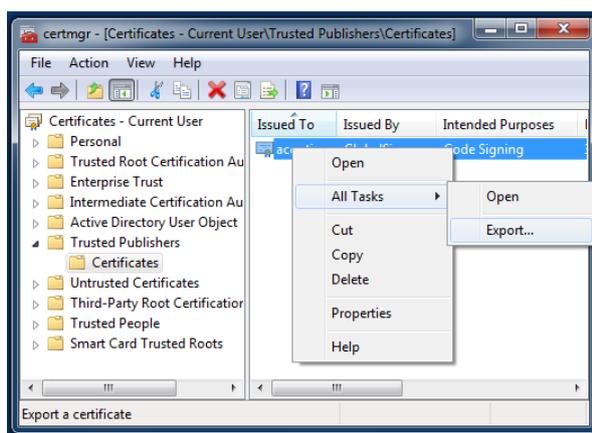


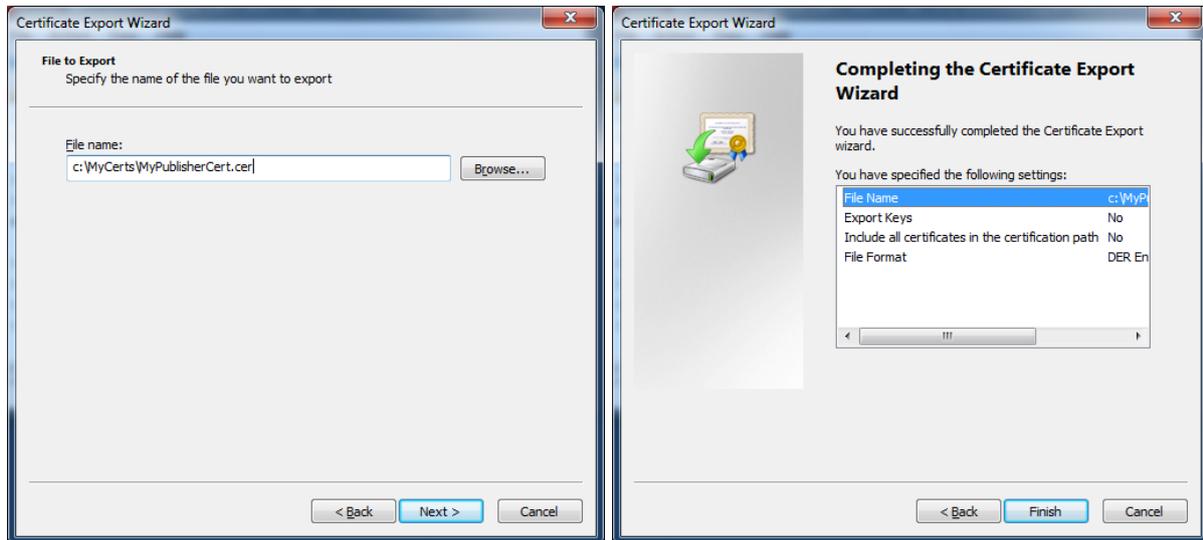
As an alternative you can pre-install the certificate in the store and so prevent the question.

- 1) Install the driver on a system and select “Always trust software from....”
- 2) Call “certmgr” from an administrator command line. The tool is part of Windows 7 or alternatively the WDK. The certificate should be listed below “Trusted Publishers”:



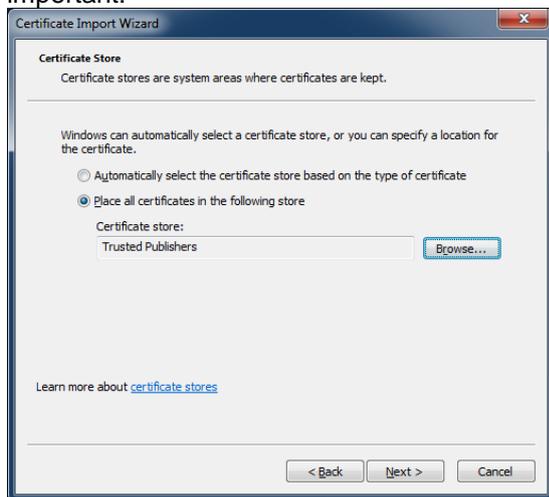
- 3) Export the certificate into a file:





This file can now be used for pre-install the certificate on any PC and prevent the dialog during driver installation.

The pre-installation can be done by right-click the file and select "install". The correct store is important:



Alternatively, the certmgr.exe tool can be used to automatically install the certificate:  
 C:\MyCerts>certmgr.exe -add MyPublisherCert.cer -s -r localMachine trustedpublisher

A programmatically solution is also possible. The following calls will be required:

- CertOpenStore
- CertEnumCertificatesInStore
- CertAddCertificateContextToStore
- CertFreeCertificateContext
- CertCloseStore

## 5 RTOS Operation Mode

The basic decision that has to be made is the RTOS Operation Mode that shall be used.

The following variants are available:

- Shared Mode operation (single core system):  
Windows and the RTOS both run on one single CPU core. Windows will only get CPU time when the RTOS becomes idle.
- Shared Mode operation (multi core system):  
Windows utilizes all CPU cores in the system, the RTOS may run on an arbitrary CPU core. Thus, the core where the RTOS is running will be shared by Windows and the RTOS. Windows will only get CPU time on this core when the RTOS becomes idle.  
**Note:** if the RTOS doesn't become idle all Windows activities on that core will cease which will also block all other Windows cores to operate correctly.
- Exclusive Mode Operation  
On a system with n CPU cores Windows will use the first (n-1) cores and the RTOS will use the last CPU core. Both operating systems run completely independent from each other.
- SMP Exclusive mode operation  
On a system with n CPU cores Windows will use the first w cores and the RTOS will use the remaining r CPU cores (where  $r > 1$ ). The RTOS thus will use more than one core and run in SMP mode (Symmetric Multiprocessing mode). Both operating systems run completely independent from each other.
- SMP Shared mode operation  
On a multi core system Windows utilizes only the first CPU core, the RTOS will use all other CPU cores in SMP mode. Thus, the first core will be shared by Windows and the RTOS. Windows will only get CPU time when the RTOS becomes idle on this core.

The operation mode can be determined as follows.

- a) The number of CPU cores used by Windows is determined by the Windows boot configuration.

Windows XP

Appending the parameter ***/NUMPROC=n*** into the Windows boot configuration file boot.ini causes Windows to always use the first n CPU cores.

Boot.ini usually is located in C:\ and can be edited using the Windows Notepad editor.

Example entry (Windows will only use one CPU core):

```
multi(0)disk(0)rdisk(0)partition(2)\WINDOWS="Windows XP"  
/noexecute=optin /fastdetect /numproc=1
```

Windows Vista / 7 / 8 / 10

The boot configuration has to be edited using "BCDEdit" from the command line.

- Open a command line (cmd) with administrator rights (right click, start as administrator)

- enter "***bcdedit /set numproc n***" to configure Windows to use 'n' processor(s).

The setting can be removed with "bcdedit /deletevalue numproc"

To be able to start RTOS on an exclusive core on Windows Vista or newer the Uploader

***automatically*** sets the following entry: "bcdedit /set firstmegabytepolicy useall"

This will ***not*** be removed when the product gets uninstalled.

The entry can be removed ***manually*** by calling "bcdedit /deletevalue firstmegabytepolicy"

- b) The cpu cores that shall be used by the RTOS are determined by the configuration parameter ***ProcessorMask*** in section ***[Rtos]***. Bit 0 represents the first cpu core, bit 1 the second etc..

Examples:

ProcessorMask = 1: RTOS will run on core 0 (first core)

ProcessorMask = 2: RTOS will run on core 1 (second core)

ProcessorMask = 3: RTOS will run on core 0 and 1 (first two cores)

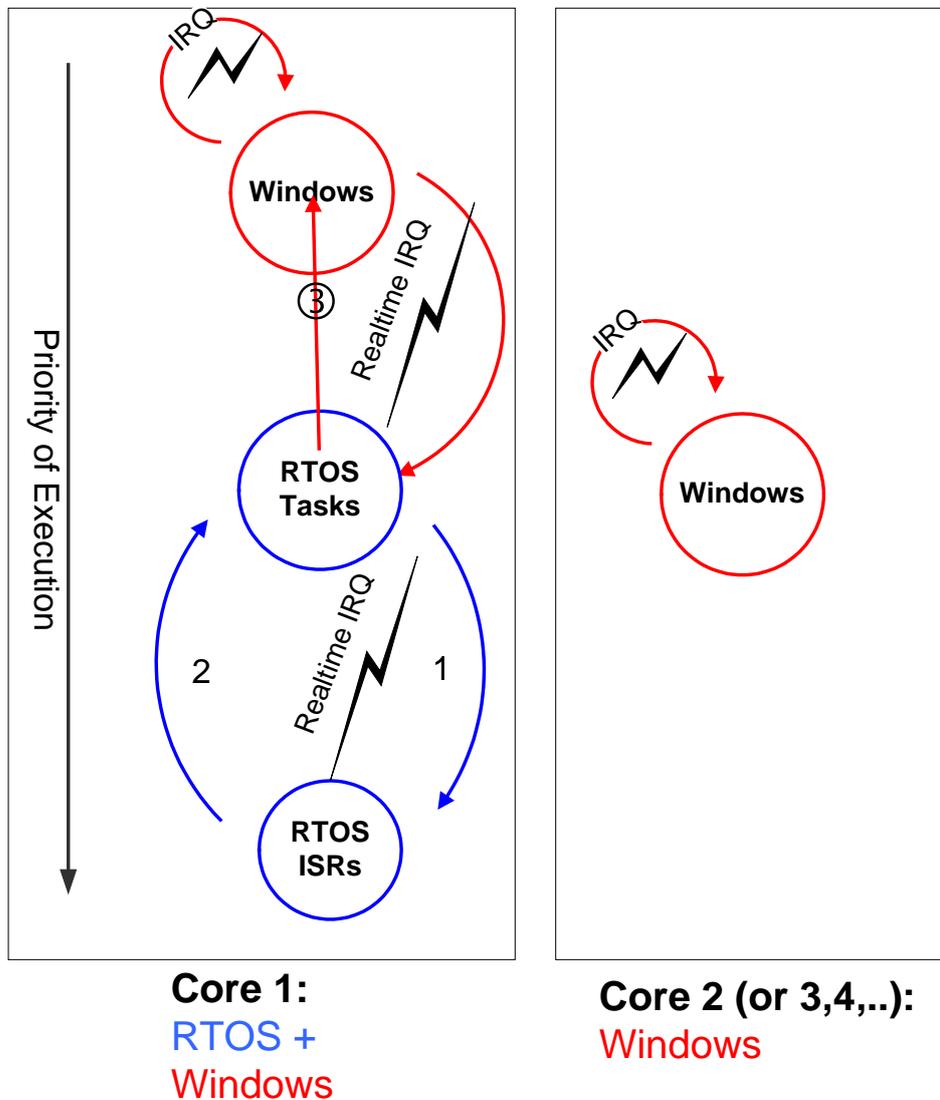
ProcessorMask = C: RTOS will run on core 2 and 3



## 5.2 Shared Mode operation (multi core)

Windows utilizes all CPU cores in the system, the RTOS may run on an arbitrary CPU core. Thus, the core where the RTOS is running will be shared by Windows and the RTOS. Windows will only get CPU time on this core when the RTOS becomes idle.

**Note:** if the RTOS doesn't become idle all Windows activities on that core will cease which will also block all other Windows cores to operate correctly.



### Configuration example 1 (RTOS running on first core, see picture)

Windows uses all CPU cores, the RTOS uses the first CPU core.

Boot configuration: no adjustment is necessary

ProcessorMask: 1

### Configuration example 2 (RTOS running on second core)

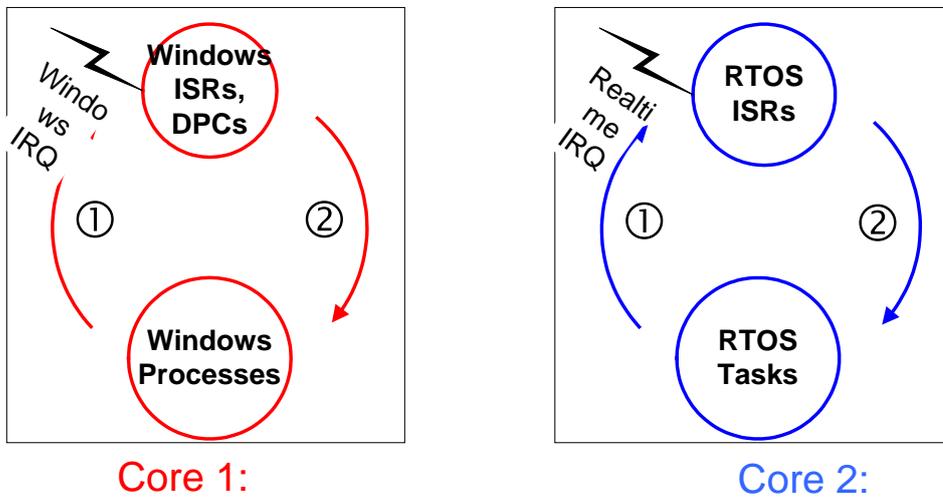
Windows uses all CPU cores, the RTOS uses the second CPU core.

Boot configuration: no adjustment is necessary

ProcessorMask: 2

### 5.3 Exclusive Mode operation

On a system with n CPU cores Windows will use the first (n-1) cores and the RTOS will use the last CPU core. Both operating systems run completely independent from each other.



#### **Configuration example 1 (dual core system, see picture)**

Windows uses the first CPU core, the RTOS uses the second CPU core.

Boot configuration: set NUMPROC to 1

ProcessorMask: 2

#### **Configuration example 2 (quad core system)**

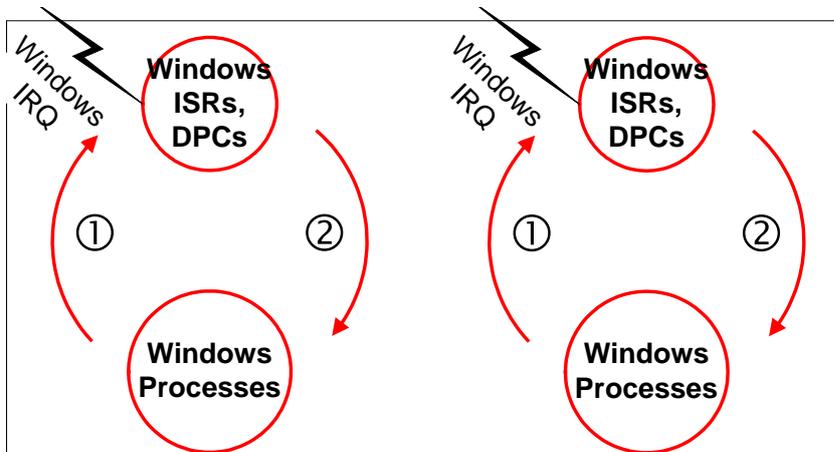
Windows uses the first 3 CPU cores, the RTOS uses the fourth CPU core.

Boot configuration: set NUMPROC to 3

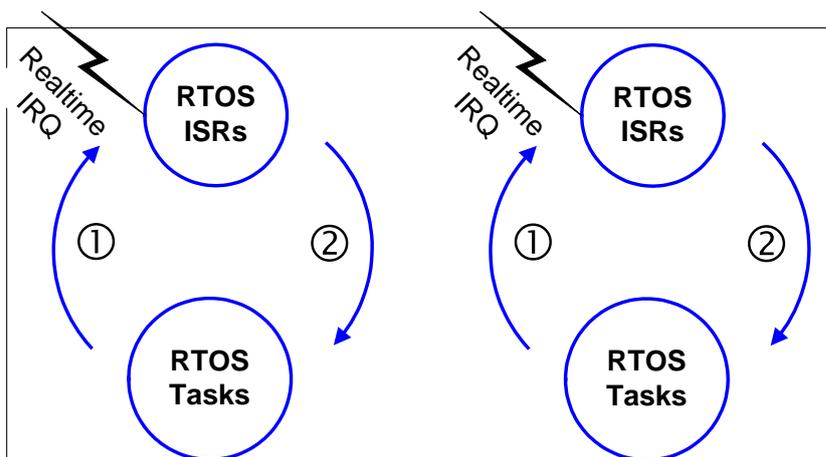
ProcessorMask: 8

## 5.4 SMP Exclusive Mode operation

On a system with  $n$  CPU cores Windows will use the first  $w$  cores and the RTOS will use the remaining  $r$  CPU cores (where  $r > 1$ ). The RTOS thus will use more than one core and run in SMP mode (Symmetric Multiprocessing mode). Both operating systems run completely independent from each other.



Cores 1 + 2: Windows in SMP Mode



Cores 3 + 4: RTOS in SMP Mode

### Configuration example 1 (quad core system, see picture)

Windows uses the first 2 CPU cores, the RTOS uses the last 2 CPU cores.

Boot configuration: set NUMPROC to 2

ProcessorMask: C

### Configuration example 2 (octal core system)

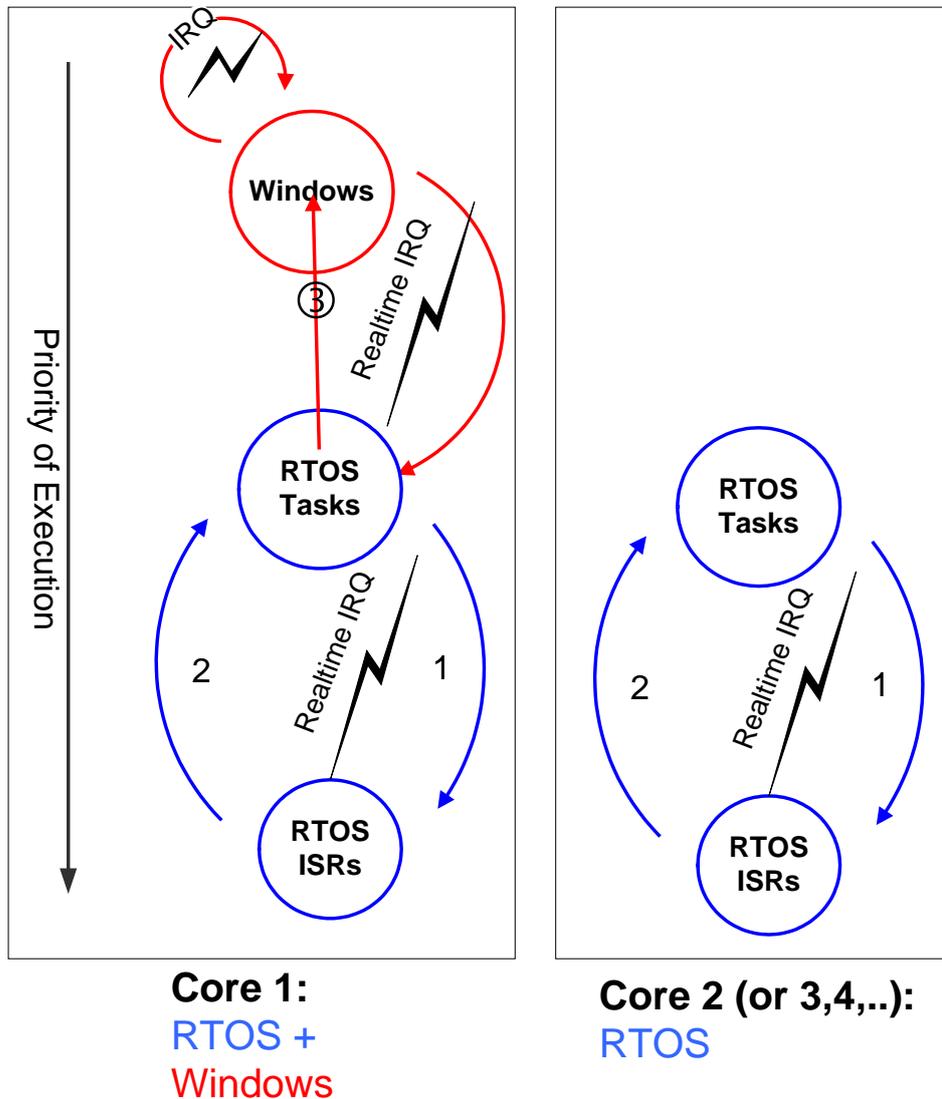
Windows uses the first 4 CPU cores, the RTOS uses the last 4 CPU cores.

Boot configuration: set NUMPROC to 4

ProcessorMask: F0

## 5.5 SMP Shared Mode operation

On a multi core system Windows utilizes only the first CPU core, the RTOS will use all other CPU cores in SMP mode. Thus, the first core will be shared by Windows and the RTOS. Windows will only get CPU time when the RTOS becomes idle on this core.



### Configuration example 1 (dual core system, see picture)

Windows uses the first CPU core, the RTOS uses all CPU cores.

Boot configuration: set NUMPROC to 1  
ProcessorMask: 3

### Configuration example 2 (quad core system)

Windows uses the first CPU core, the RTOS uses all 4 CPU cores.

Boot configuration: set NUMPROC to 1  
ProcessorMask: F

## 6 RTOS VM configuration files (\*.config files)

Configuration of the RTOS Virtual Machine is controlled using ASCII type config files. Additional RTOS specific settings will also be stored in these files.

A config file has to start with the following header / signature:

```
RtosConfig
```

Another config file can be included (nested includes are possible) using:

```
#include "AnotherConfigFile.config"
```

A comment can be made for a whole line or at the end of a line

```
; This is a comment example
```

```
#include "AnotherConfigFile.config" ; This is another comment example
```

A config file should end with a new line to prevent compatibility problems with some versions.

This manual only covers generic settings valid for all RTOSWin solutions.

The configuration file is divided into the following main sections:

- [Upload] = Upload settings
- [Vmf] = Virtual Machine Framework settings
- [Windows] = Windows configuration settings
- [Rtos] = RTOS 1 configuration settings for Oslid 0
- [Rtos1] = RTOS 2 configuration settings for Oslid 1
- [Rtos2] = RTOS 3 configuration settings for Oslid 2

...

When document refers to [Rtos] this is also applicable for [Rtos1], [Rtos2], ... .

### 6.1 Processor configuration (RTOS)

Please see chapter "5 RTOS Operation Mode" for additional details.

Section [Rtos]

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
ProcessorMask	Dword	Each bit represents a CPU to be used by a RTOS. 0x1 means 1 <sup>st</sup> CPU, 0x2 means 2 <sup>nd</sup> CPU, 0x5 means 1 <sup>st</sup> and 3 <sup>rd</sup> CPU  Limitations: - Different Rtos can't share CPU(s). - An OS using multiple CPUs can't share CPU(s) with another OS also using multiple CPUs.

## 6.2 Interrupt Processor Vector Ranges

At the default configuration the RTOS interrupt processor vector range is always above the Windows interrupt processor vector range – for compatibility even on exclusive core configuration.

Example 1 (QuadCore):

Windows Boot configuration: set NUMPROC to 1  
 Rtos Config file: [Rtos] "ProcessorMask"=dword:0F  
 → Windows vector range: 0x20-0xDF  
 → RTOS vector range: 0xE0-0xFF

Example 2 (QuadCore):

Windows Boot configuration: set NUMPROC to 2  
 Rtos Config file: [Rtos] "ProcessorMask"=dword:0C  
 → Windows vector range: 0x20-0xEF  
 → RTOS vector range: 0xE0-0xFF

At example 2 Windows uses the first two cores and RTOS the other two cores. Because the vector ranges overlap it must be ensured that the RTOS can handle incoming broadcast interrupts. This feature (Multi-SMP with Windows 7) is supported since version 5.0.00.31.

**When Windows uses only one processor which is not shared with RTOS** it might be possible to allow RTOS to use all available vectors from lowest 0x20 to highest 0xFF. This can be done at the [Rtos] section in the config file. Because of compatibility with old versions this is not the default configuration.

Example 3 (QuadCore + processor vector settings):

Windows Boot configuration: set NUMPROC to 1  
 Rtos Config file: [Rtos] "ProcessorMask"=dword:0E  
 → Windows vector range: 0x20-0xEF  
 → RTOS vector range: 0x20-0xDF

Section [Rtos]

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
ProcessorVectorLowest	Dword	Optional; Set lowest vector allowed for range determination Warning: Do not change this value without request. An invalid value causes system instability.
ProcessorVectorHighest	Dword	Optional; Set highest vector allowed for range determination Warning: Do not change this value without request. An invalid value causes system instability.

### 6.3 Memory configuration

Each RTOS has to be configured to use an individual, specific physical memory range.

The individual memory range of each RTOS must be defined in the RTOS specific section. Section [Rtos] for first, section [Rtos1] for second RTOS and so on.

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
MemoryStartAddress	Dword	RTOS memory physical base address (in bytes). omitted = use allocated memory x = use this base address
MemorySize	Dword	RTOS memory size
Alignment	dword	Physical memory address alignment (in bytes). Only relevant when "MemoryStartAddress" = 0. 0 = no alignment used Omitted equals 0
AddressMax	dword	Highest physical address usable (in bytes). Only relevant when "MemoryStartAddress" = 0. Omitted equals 0xFFFFFFFF
MemoryType	dword	0 = Autodetermination 1 = (reserved) 2 = RA; not usable on EFI systems 3 = BCD; Maximum for x86=~512MB and x64=~3MB 4 = MEM; Allocated memory (contiguous) Omitted equals 0

MemoryType autodetermination uses:

- Type 4 if MemoryStartAddress is omitted.
- Type 3 on EFI systems.  
If MemorySize is above BCD limit an additional SharedMemory named "OsXHeap", where X is OS-ID, will be allocated:  
If MemorySize is <=16MB the SharedMemory will be of the same size – if not its size will be (MemorySize – BCD limit). The MemorySize will be set to the BCD limit.
- Type 3 on systems with minimum Windows 7Sp1 and MemorySize below BCD limits.
- Type 2 if not an EFI system.
- Returns an error in all other cases.

The global physical memory range, defined in the [Upload] section, has been superseded by redesigned memory management and may only be useful to override automatic determination.

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
RteMemoryStartAddress	Dword	Physical base address where the first RTOS is located
RteMemorySize	Dword	Memory size to be reserved for all RTOS

The following RTOS specific settings are deprecated and have been superseded by VMF\_OSIMAGE\_INFO

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
ImageOffset	Dword	Offset where the RTOS image has to be copied by the uploader
EntryPointOffset	Dword	Boot endpoint offset of the RTOS

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
VmfAnchorOffset	Dword	VMF management anchor offset. After loading the RTOS image the uploader will copy the VMF management information data at this location.
VmfVersionOffset	Dword	VMF version offset. After loading the RTOS image the uploader will check the VMF version used by the RTOS.

## 6.4 Time/Date and Timezone synchronization

The RTOSWin solution supports time/date and time zone synchronization to assure that all participating operating systems show the same date and time. Each operating system may decide to either use its own date, time and time zone or shall be synchronized by another OS.

The config file settings regarding the time/date and time zone synchronization are OS specific and have to be made in an OS section like [Windows\TimeSync] or [Rtos\TimeSync].

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
ModeGet	dword	0 Auto (=RTC) 1 Time from RTC (Chipset) will be used 2 Time from Soft-RTC (OS) will be used omitted = 0 = Default = RTC This setting is currently evaluated by Windows only.
ModeSet	dword	0 Auto (=UTC when available else Local) 1 UTC time will be set 2 Local time will be set omitted = 0 = Default = UTC
TaskEnabled	dword	0 Task will not be started 1 Task will be started 2 Task will be started, init time and timezone once and then finish. omitted = 1 = Task will be started
TaskPriority	dword	This defines the task priority for the time/date and timezone synchronization thread. The values are OS specific. The delivered config file contains OS specific information about lowest and highest possible priority.  If the entry does not exist then the OS uses an OS specific priority compatible to older versions.
TimeSyncMaster	string	Identification string of the OS which is the source of the time and date.  Possible values are "Windows" and "Rtos".  If the entry does not exist then the OS will act as master and use its own time/date.
TimezoneSyncMaster	string	Identification string of the OS which is the source of the timezone.  Possible values are "Windows" and "Rtos".  If the entry does not exist then the OS will act as master and use its own timezone.

Shortly said every OS has to know where to take its info from – from itself (→ master) or another OS (→ slave). When no configuration is found an OS takes its infos from itself.

### 6.4.1 Windows

By default Windows uses its own clock and is not being synchronized with another clock. If the (first) RTOS shall be used as time and timezone master the following entries have to be activated:

```
[Windows\TimeSync]
"TimeSyncMaster"="Rtos"
"TimezoneSyncMaster"="Rtos"
```

```
[Rtos\TimeSync]
"TimeSyncMaster"="Rtos"
"TimezoneSyncMaster"="Rtos"
```

### 6.4.2 RTOS

By default the RTOS is being synchronized with the Windows clock. The following entries have to be activated:

```
[Windows\TimeSync]
"TimeSyncMaster"="Windows"
"TimezoneSyncMaster"="Windows"
```

```
[Rtos\TimeSync]
"TimeSyncMaster"="Windows"
"TimezoneSyncMaster"="Windows"
```

### 6.4.3 Windows / RTOS

Both OS are running separately without time/date and timezone synchronization. This is the default if no configuration entries are found. Alternatively, the following entries can be activated:

```
[Windows\TimeSync]
"TimeSyncMaster"="Windows"
"TimezoneSyncMaster"="Windows"
```

```
[Rtos\TimeSync]
"TimeSyncMaster"="Rtos"
"TimezoneSyncMaster"="Rtos"
```

## 6.5 Section [Vmf]

This chapter describes all settings in the [Vmf] section of the config file. It contains VMF settings valid for the whole RTOSWin system.

[Vmf]

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
AddressMax	hex	This parameter defines the maximum physical address allowed for VMF memory. When omitted "FF,FF,FF,1F,00,00,00,00" (= 0x000000001FFFFFFFF = below 512MB) will be used. The default value is "FF,FF,FF,FF,00,00,00,00" (= 0x00000000FFFFFFFF = below 4GB). Sometimes using SharedCore requires a value of "FF,FF,FF,8F,00,00,00,00" (= 0x000000008FFFFFFFF = below 2GB). Uploader will inform about this requirement. Before changing the value please affirm this will be supported by RTOS.
EventCount	dword	Maximum number of events supported. 100 (fixed) for versions < 6.1.00.06 500 (default) for versions >= 6.1.00.06 Starting with 6.1.00.06 the number can be modified using config entry "EventCount".
TimerHwInputFreq	dword	This parameter can be used to override the determined VMF hardware timer input frequency with a specific value (unit Herz).
TimerMeasureDelayLimit	dword	This parameter specifies the maximum delay in TSC ticks for reading all required timer and counter values for a single point in time during timer and TSC frequency determination. Exceeding the default limit may indicate inaccuracy in the determined frequencies. =0 Use best value – recommended for running VmWare >0 Limit in TSC ticks If omitted "0x1388" (5000) is the default.
VerbosityLevel	dword	This parameter specifies the message level to be displayed by a message box: 0 – only fatal errors 1 – also other errors 2 – also warnings If omitted "0" is the default.

[Vmf\Interrupts]

Entry Name	Type	Description
IdRenumber	dword	<p>This parameter specifies if VMF should renumber interrupt IDs. On default the IDs are given by Uploader. This numeration is not contiguous and might become values above 100. Some OS may not be compatible with such high interrupt numbers so VMF can renumber the IDs (contiguous, starting with given value).</p> <p>0 – do not renumber IDs            &gt;0 – renumber IDs equal or above given value</p> <p>If omitted “0” is the default.            For example 0x30 (48) would mean that all IDs equal or above 48 will be renumbered to contiguous ID’s starting with 48.</p>

[Vmf\MessageBox]

Entry Name	Type	Description
TextBufferLength	dword	<p>This parameter specifies the buffer size to be used for vmfMessageBox API.</p> <p>The minimum value is 522 = ( VMF_MESSAGEBOX_MAX_TEXT_SIZE + VMF_MESSAGEBOX_MAX_TITLE_SIZE + 2 ).</p> <p>When omitted the minimum will be used as default.</p>

## 6.6 Section [Upload]

This chapter describes all settings in the [Upload] section of the config file. It contains generic settings valid for the whole RTOSWin system.

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
Trace	dword	Enable or Disable the Uploader tracing facility. Disable tracing = 0 (default). Enable tracing = 1
TraceFile	string	If tracing is enabled, all trace data will be stored in this named file. Default = "trace.txt".
WaitForRtosCommSubsystems	dword	Disabled = 0. Enabled = 1 (default). If enabled: upon starting the RTOS, the Uploader will wait for communication subsystems to be initialized before finishing. Enabling this parameter can increase the start time. Disabling it could cause synchronization problems.
LaunchRtosControl	dword	This parameter causes the <b>RtosControl</b> system tray application to be launched. 0 = do not launch 1 = (default) launch RtosControl tray application 2 = Don't start RtosControl, but let RtosService do message box handling. If "MessageBoxShow" value is not configured it will be set to 0.
RtosControlPath	string	This parameter is optional and can contain a relative or absolute path with or without the executable name to <b>RtosControl</b> .
MessageBoxShow	dword	This parameter causes message box to be auto-answered without showing a dialog to ask for user input: VMF_MESSAGEBOX_ABORTRETRYIGNORE will return "Abort" VMF_MESSAGEBOX_YESNOCANCEL and VMF_MESSAGEBOX_YESNO will return "Yes" VMF_MESSAGEBOX_RETRYCANCEL will return "Cancel" all other will return "OK"  0 = auto-answer. 1 = show message box (default).
MessageBoxLog	dword	This parameter causes message box messages to be also written into application system log. This feature is independent from Microsoft Build-In feature of logging message boxes to system log.  0 = do not write log entry. 1 = write log entry. Default depends on "MessageBoxShow" setting. If a message will be shown logging is disabled – else enabled.
BootCodeReservationForce	dword	0 = (default) reservation is not forced. 1 = forces Uploader to statically reserve processor boot code memory. On default the static reservation is not forced and will be done only in case the RtosDrv could not dynamically reserve the memory. Activating static boot code reservation requires a reboot. Reservation will be removed with "-memcfg -u" (or "-ur") option only.
BootCodeBaseFallback	dword	Fallback base address to be used for processor boot code memory reservation. Default value is 0x40000.

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
SetFirstMegabytePolicy	dword	0 = force FirstMegabytePolicy not to be set. 1 = (default) set policy when required. Policy will also not be set if BootCodeReservationForce is active.
AllowMultiSmp	dword	Deprecated – this setting has been superseded by VMF_OSIMAGE_INFO flag VMF_OSIMAGE_INFO_FLAG_1_MULTISMP.  This parameter enables the option to run Windows 7 SMP with RTOS SMP (Multi-SMP).  0 = (default) Multi-SMP not allowed for Windows 7. 1 = Allowed.  Attention: RTOS must explicit support Windows 7 Multi-SMP or the system might crash! Please check release notes if Multi-SMP is supported.
Flags	dword	Bit 0: When clear (default) RtosStart will not unload VMF or OS after an error. When set RtosStart will unload VMF and/or OS after an error. But OS start failure will not unload VMF when it was previously loaded by a separate call.

## 6.7 Multi Purpose Shared Memory

This chapter describes all settings of the config file regarding multi purpose shared memories in the sections [SharedMemory\UserDefinedShmName] respectively [SharedMemory\UserDefinedShmName\AccessModes]. ("UserDefinedShmName" can be a user defined name)

Attention:

When multiple large shared memories are configured it is possible the system has not enough memory remaining to boot. Such a situation can be solved this way:

- Start the system in Safe Mode
- Change the SharedMemory configuration
- Start the RTOS to update with the new configuration
- Reboot the system

[SharedMemory\UserDefinedShmName]

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
Name	string	This parameter specifies the name of the shared memory. This name can be used to query the ID using vmfldGetByName() respectively RtosGetIdByName() function.
Description	string	A string describing the shared memory.
Base	dword	Base address of shared memory. 0 = automatic allocation (recommended) x = use this static address as base omitted = equals 0 <b>Limitations for static addresses:</b> - The address must be within the range of "RteMemoryStartAddress" and "RteMemoryStartAddress" + "RteMemorySize" - The memory can not be accessed directly from Windows.
Alignment	dword	Physical memory address alignment (in bytes). Only relevant when "Base" = 0. 0 = default = no alignment used
AddressMax	dword	Highest physical address usable (in bytes). Only relevant when "Base" = 0. 0xFFFFFFFF = default
Size	dword	Size of shared memory (in bytes)
File	string	Path and Filename to load the shared memory content from a file or save it to.
Initialize	dword	0 = don't initialize memory. The memory will nevertheless be zeroed once after allocation, but not each time VMF gets loaded. 1 = zero memory 2 = initialize memory with file 3 = as (2) but initialize with '0' if file not found instead of error 7 = as (3) but on load renames "File" to "File.loaded". Existing file will be replaced. In combination with "Save" this prevents re-use of old data e.g. after power-failure. F = as (7) but rename also read only files

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
Save	dword	Default value = 0 = don't save on stop Possible flags: 0x1 = save content to file 0x2 = overwrite also write protected file 0x4 = also save during power failure handling
AccessDefault	dword	default access mode mask (default = 1 = present,r+w,cached,no execute) Possible flags are: 0x01 = present 0x02 = readonly 0x04 = uncached 0x08 = execute

[SharedMemory\UserDefinedShmName\AccessModes]

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
"0" ... "n"	dword	Define an explicit access mode for OS X (0 ... n) This overwrites the default access mode set by "AccessDefault" for a specific OS.

## 6.8 OS Communication

### 6.8.1 Settings

The config file settings regarding the communication are OS specific and have to be made in an OS section like [Windows\Comm] or [Rtos\Comm].

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
CommTimeout	dword	Communication timeout in seconds (How long this OS should wait for another). Default = 5.
CommInterruptMode	dword	Interrupt mode. 0=Polling, 1=Interrupt, omitted= Windows: depending on "Realtime OS Driver" interrupt mode. Rtos: depending on host (Windows) Comm interrupt mode.
TaskPriority	dword	This defines the task priority for the comm thread. The values are OS specific. The delivered config file contains OS specific information about lowest and highest possible priority. If the entry does not exist then the OS uses an OS specific priority compatible to older versions.
TaskEnabled	dword	0 Task will not be started 1 Task will be started omitted = 1 = Task will be started

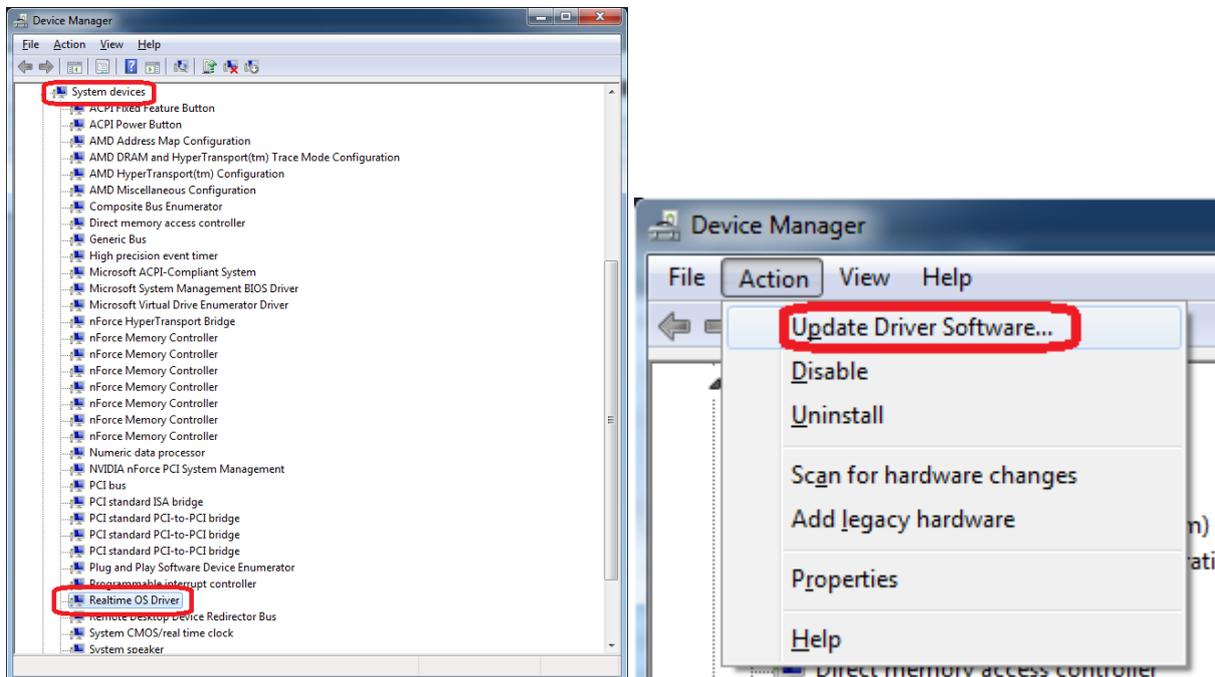
For a higher performance of Inter-OS communication interrupts can be used.  
This will minimize event round-trip times between the OS.

After changing the config file setting "CommInterruptMode" RTE should be stopped for at least 10 seconds before restarted with the new configuration setting. Without waiting between stop and start the configuration change of "CommInterruptMode" might not be completed.

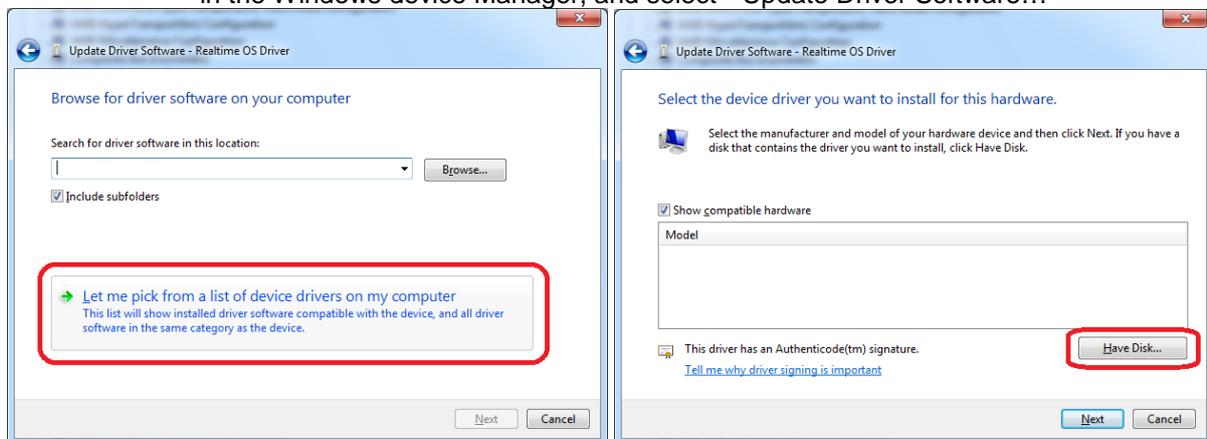
## 6.8.2 Enable / Disable Comm Interrupt

The Comm interrupt configuration depends on:

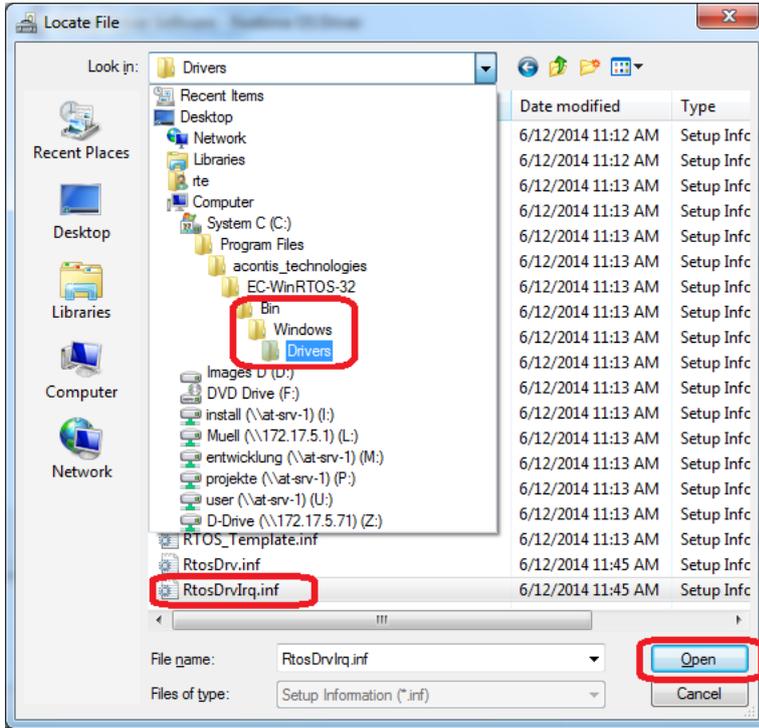
- Config file settings “CommInterruptMode”, as described above.  
This can be configured separately for Windows and each RTOS.
- Windows device manager (Windows only):



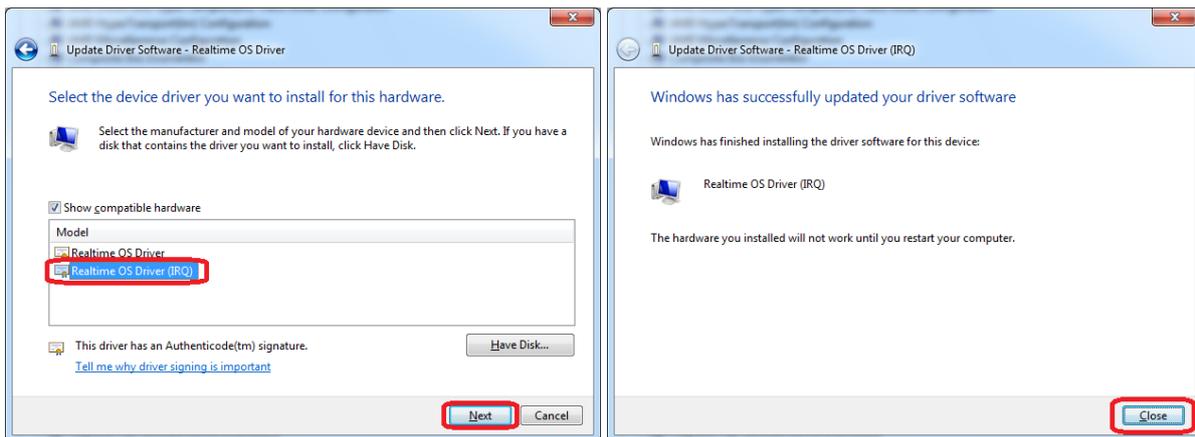
- 1) Right-Click the “Realtime OS Driver”, which can be found under the “System devices” in the Windows device Manager, and select “Update Driver Software...”



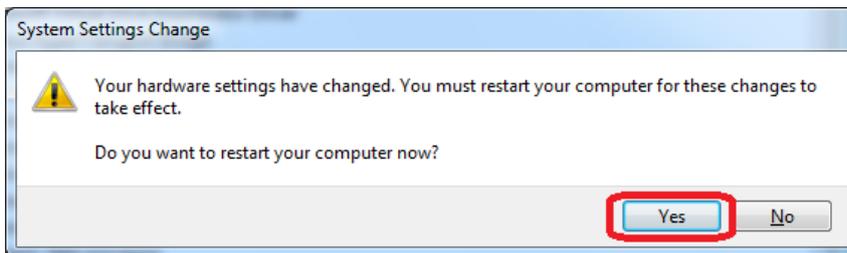
- 2) Manually pick driver from a List



3) Open the “Drivers” directory which can be found below your product installation directory and select “RtosDrvIrq.inf”



4) ENABLE: Select “Realtime OS Driver (IRQ)”  
DISABLE: Select “Realtime OS Driver”



5) Reboot the PC when required

### 6.8.3 System does not boot using configuring Comm interrupt

Because RtosDrv is loaded very early it is possible that the system doesn't start if the requested interrupt resource is not available. In this case the system can be booted in safe-mode, where no resources will be claimed by RtosDrv, to change the configuration.

### 6.8.4 Comm configuration error

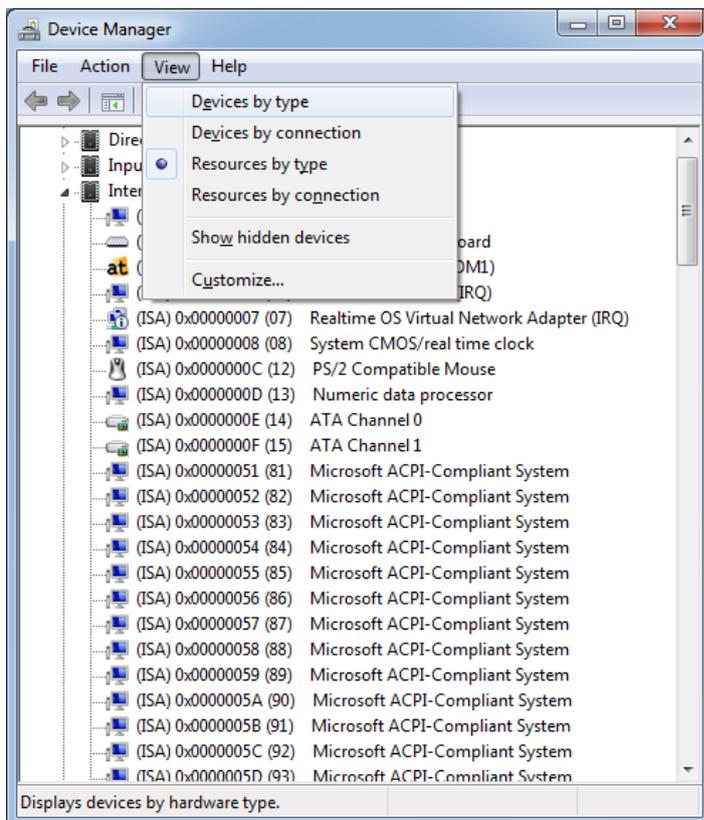
If [Windows\Comm] "CommInterruptMode"=dword:1 is set, but "Realtime OS Driver" has no interrupt configured a message box will appear when VMF is started:



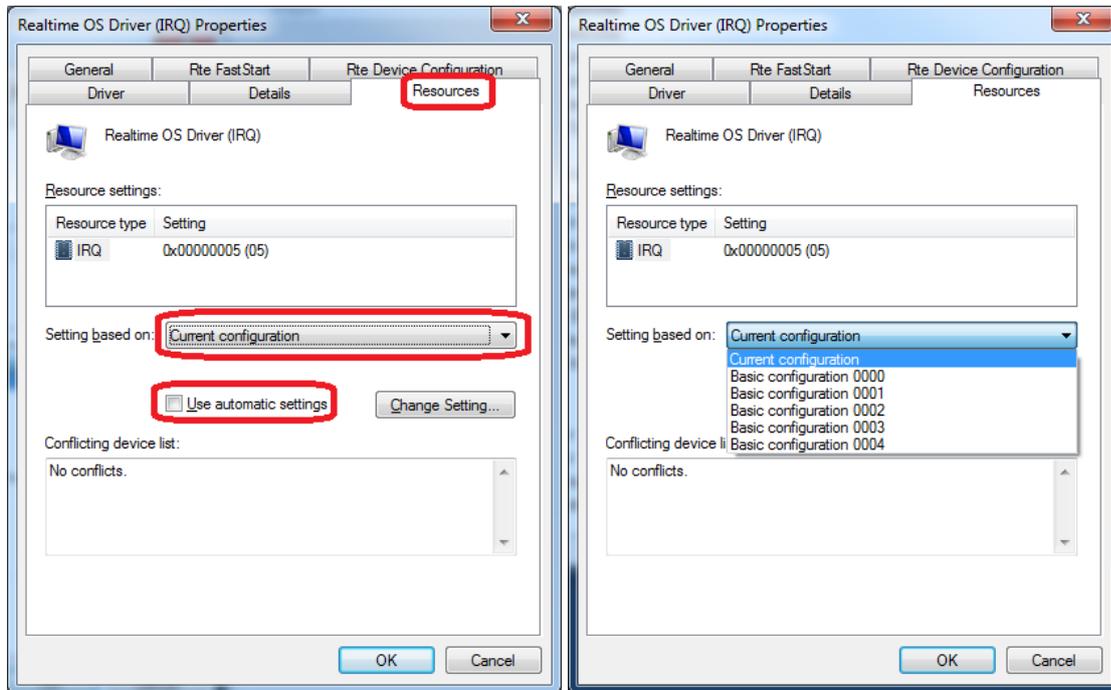
To solve this problem either the "CommInterruptMode" can be set to 0 or the "Realtime OS Driver" resources can be configured for an interrupt as described above.

### 6.8.5 Interrupt conflict

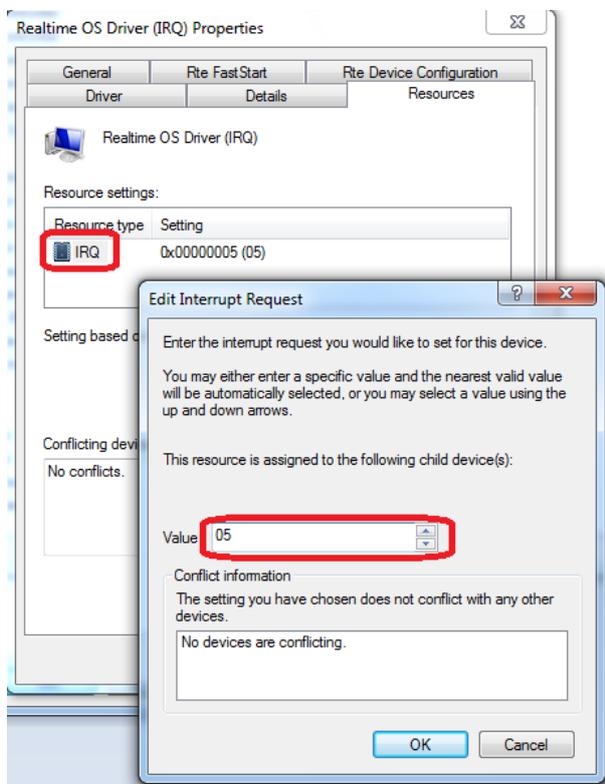
When Uploader tells about an interrupt conflict regarding "Realtime OS Driver" you can configure another interrupt using the device manager.



- 1) Open the Device Manager and change the view to "Resources by type" to check for a free interrupt number not conflicting with another device.



- 2) Double-click the “Realtime OS Driver (IRQ)”, which can be found under the “System devices” in the Windows device Manager, and select the “Resources” tab. There you can disable “Use automatic settings” and select another setting from “Setting based on:”  
Each setting contains a different interrupt number. If no configuration contains the desired number the last configuration can be selected to set an individual IRQ number.



- 3) To select an individual IRQ double-click “IRQ” and then use spin-buttons to select another interrupt or enter the value using the keyboard.  
HINT: Using the keyboard allows setting values not possible using the buttons!
- 4) After a reboot the interrupt conflict should be solved.

## 7 Start/Stop the RTOS: Uploader Utility

### 7.1 Introduction

After the RTOSWin solution 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.

For RTE 4.x The tool consists of two files:

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

For RTE 5.x The tool consists of three files:

- **RtosUpload.exe** (Command line program)
- **RtosLib32.dll** (For 32 bit applications)
- **RtosLib64.dll** (For 64 bit applications)

To run existing programs linked to UploadRtos.dll the file RtosLib32.dll can be copied to UploadRtos.dll.

—

When the Uploader Utility is called to start the RTOS, it reads and processes its own command line options. It also reads and processes configuration parameters from the system configuration file. After the RTOS image has been loaded, it is given control.

Here is a summary of the services that the Uploader Utility together with other RTOSWin components performs:

- Collects device configuration information (queries all RTOS devices),
- Loads configuration data (content of the \*.config files) into memory,
- Performs system and device configuration consistency checks (e.g. detects interrupt sharing conflicts),
- Reserves memory for the RTOS,
- Loads a RTOS image into memory,
- Defines physical memory areas for use by the RTOS,
- Initiates execution of the RTOS image,
- Terminates a running RTOS session and releases system resources.

## 7.2 Uploader operation, command line options

Command-line Syntax: RtosUpload <Mode> [Options...]

Mode (prefix can be '-' or '/')	Options / Description
/config "'f1' <options>"	Load VMF using config file 'f1'. This will stop all running OS and unload already loaded VMF. 'f1' use this file as config file possible options are: -vmf 'f2' use f2 as VMF binary (default is vmf.bin)
"f1"	Start OS 'f1'. 'f1' use this file OS image file This can be used in combination with '/config' and / or '/osid'. Default OS id is 0.
/x	Stop the RTOS given by option 'osid'. Without option 'osid' all RTOS will be stopped and VMF will be unloaded.
/osid <id>	Start or stop OS with this id
/memcfg "<params>"	Update memory configuration. Possible parameter are: 'u' uninstall 'a' install or update (requires >>/config "f1"<<) 'v' show current configuration. Also usable with 'u' or 'a'
/shmsave "Name" "f1"	Dump shared memory 'Name' into file 'f1'
/lic <parameter>	Configure runtime license Possible parameter are: <LicenseId> To install a license ID for this computer -remove To remove an installed license ID
/nosleep	Do not insert a delay after starting the RTOS
/nowait	Do not wait for a key pressed in case of errors
/noerror	Never return an error code
/device "<string>"	Call RtosDevice(<string>). See RtosDeviceA() API for details.
/?	Show help about possible options
Special Options	Special Options – not generally supported
/faststart "<opt1> <opt2> ..."	FastStart configuration Possible options are: -config 'f1' use config file 'f1' (only with <-file>) -file 'f2' use FastStart configuration file 'f2' -vmf 'f3' use VMF binary file 'f3' -os 'f4' use OS binary file 'f4' -[disable once always] disable FastStart, use only once or always -[startvmf startvmfos] start only VMF or start VMF and OS
/f "f1"	Freeze system into file 'f1'
/u "f1"	Unfreeze system from file 'f1'
/loop <count>	Option for testing. Repeat start for 'count' times - without unloading DLL.

When the Uploader is called using "/config" mode first all running OS will be stopped and a loaded VMF will be unloaded. Then the following steps are performed:

- RTOS device configuration data is collected and stored in memory.

- RTOS Device consistency checks are performed (e.g. interrupt sharing conflicts).
- RTOSWin configuration data (content of the \*.config files) is loaded into memory.
- The RTOSWin configuration is verified (e.g. detect overlapped memory areas).
- The VMF binary image is loaded into memory.
- If an OS image was specified ("f1" mode), it will now be loaded into memory and then booted.

Load VMF and start first OS (any running OS will be stopped and VMF unloaded first):

```
RtosUpload.exe /config "vxwin.config" "C:\Tornado\target\config\VxWin\vxWorks.bin"
```

Start or Restart OS 0 (VMF must already be loaded) without loading VMF again:

```
RtosUpload.exe /osid 0 "C:\MyOsImages\vxWorks.bin"
```

VMF will be loaded first (any running OS will be stopped and VMF unloaded first), followed by two OS:

```
RtosUpload.exe /config vxwin.config
```

```
RtosUpload.exe /osid 0 "C:\MyOsImages\vxWorks.bin"
```

```
RtosUpload.exe /osid 1 "C:\MyOsImages\vxWorks2.bin"
```

Using the "/x" mode in combination with "/osid" will stop the specified OS while "/x" without "/osid" will stop all OS and unload VMF.

All OS will be stopped and VMF be unloaded:

```
RtosUpload.exe /x
```

Only OS 1 will be stopped. Other OS and VMF will not be affected:

```
RtosUpload.exe /x /osid 1
```

Remove memory reservation:

```
RtosUpload.exe /memcfg "u"
```

Set or update memory configuration by config-file:

```
RtosUpload.exe /memcfg "/a" /config "C:\MyConfigFiles\MyOs.config"
```

## 8 RTOS tray-icon application (RtosControl.exe)

If you start [the demo version](#) of the RTOSWin solution, the RtosControl 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, the RTOS will be stopped 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.exe** program, it will be displayed in place of ACONTIS's icon. Otherwise, the ACONTIS icon will appear.

Even if you have installed a full version of RTOSWin, this program will be executed. The RtosControl application is responsible for showing a message box generated by the RTOS Virtual Machine adaptation (e.g. in fatal error situations).

RtosControl specific settings in config file section [Upload]

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
LaunchRtosControl	dword	This parameter causes the <b>RtosControl</b> system tray application to be launched. 0 = do not launch 1 = (default) launch RtosControl tray application 2 = Don't start RtosControl, but let RtosService do message box handling. If "MessageBoxShow" value is not configured it will be set to 0.
RtosControlPath	string	This parameter is optional and can contain a relative or absolute path with or without the executable name to <b>RtosControl</b> .
MessageBoxShow	dword	This parameter causes message box to be auto-answered without showing a dialog to ask for user input: VMF_MESSAGEBOX_ABORTRETRYIGNORE will return "Abort" VMF_MESSAGEBOX_YESNOCANCEL and VMF_MESSAGEBOX_YESNO will return "Yes" VMF_MESSAGEBOX_RETRYCANCEL will return "Cancel" all other will return "OK"  0 = auto-answer. 1 = show message box (default).
MessageBoxLog	dword	This parameter causes message box messages to be also written into application system log. This feature is independent from Microsoft Build-In feature of logging message boxes to system log.  0 = do not write log entry. 1 = write log entry. Default depends on "MessageBoxShow" setting. If a message will be shown logging is disabled – else enabled.

RtosControl specific settings in config file section [Windows\MessageBox]

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
TaskPriority	dword	This defines the task priority for the message box thread. The values are OS specific. The delivered config file contains OS specific information about lowest and highest possible priority.  If the entry does not exist then the OS uses an OS specific priority compatible to older versions.
TaskEnabled	dword	0 Task will not be started 1 Task will be started omitted = 1 = Task will be started

## 9 The RTOS service application (RtosService.exe)

When installing RTOSWin the RtosService component is added into the list of services which will be automatically started when Windows boots.

The service is responsible for the following tasks:

- Windows Clock Correction (assures the Windows clock does not run too slow)
- Date and Time synchronization between Windows and the RTOS
- RTOS File Server for RtosFile support
- RTOS Gateway

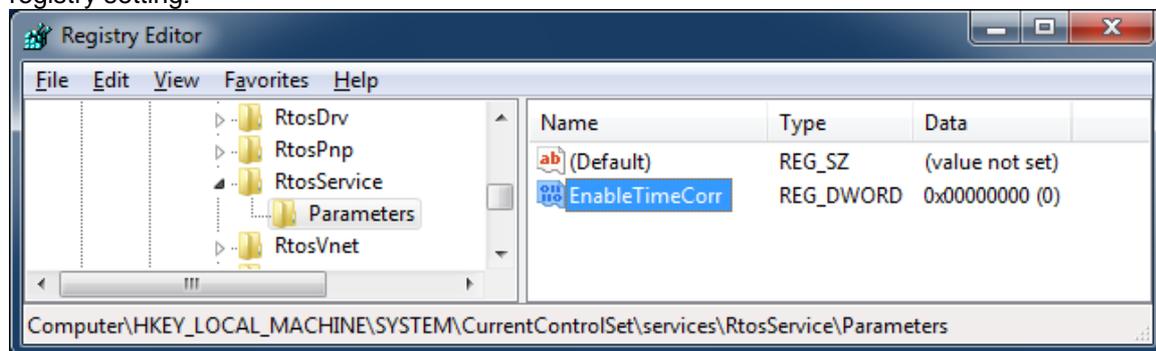
### 9.1 Clock Correction

When running in SharedCore mode Windows will lose timer interrupts when RTOS occupies the CPU longer than a timer interval. As a result, the Windows clock will be delayed

The clock correction compensates this problem by regulating the timer increments per clock tick.

The clock correction is enabled on default.

Before version 5.1.0.30 / 6.0.0.11 ClockCorretion could only be disabled by the following Windows registry setting:



Starting with 5.1.0.30 / 6.0.0.11 ClockCorretion can also be disabled by a config file entry:

ClockCorrection specific settings in config file section [Windows\ClockCorrection]

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
TaskEnabled	dword	0 Task will not be started 1 Task will be started omitted = 1 = Task will be started

### 9.2 Date and Time synchronization

See chapter “0 Time/Date and Timezone synchronization” for details.

### 9.3 RTOS File Server for RtosFile support

See chapter “11.1.14 RTOS Library – files” for details.

### 9.4 RTOS Gateway

RtosGateway forwards a RtosSocket to an IP socket.

When a RTOS application supports using RtosSocket, like for example acontis EC-Master does, the RtosGateway allows access to that application from an external PC.

In case of EC-Master it allows to visualize EtherCat data on an external PC.

RtosGateway specific settings in config file section [Windows\Gateway]

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
-------------------	-------------	--------------------

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
TaskPriority	dword	This defines the task priority for the gateway thread. The values are OS specific. The delivered config file contains OS specific information about lowest and highest possible priority.  If the entry does not exist then the OS uses an OS specific priority compatible to older versions.
TaskEnabled	dword	0 Task will not be started 1 Task will be started omitted = 1 = Task will be started
Port	dword	TCP port to be used
PipeThreadRxTimeout	dword	Timeout reading RX pipe. Do not change without request!
PipeRxBufSize	dword	RX pipe buffer size. Do not change without request!

## 10 The Realtime OS Virtual Network Adapter (RtosVnet.sys)

All OS (Windows and RTOS) can communicate using a virtual network.

The “Realtime OS Virtual Network Adapter” (RtosVnet.sys) enables Windows to access this virtual network.

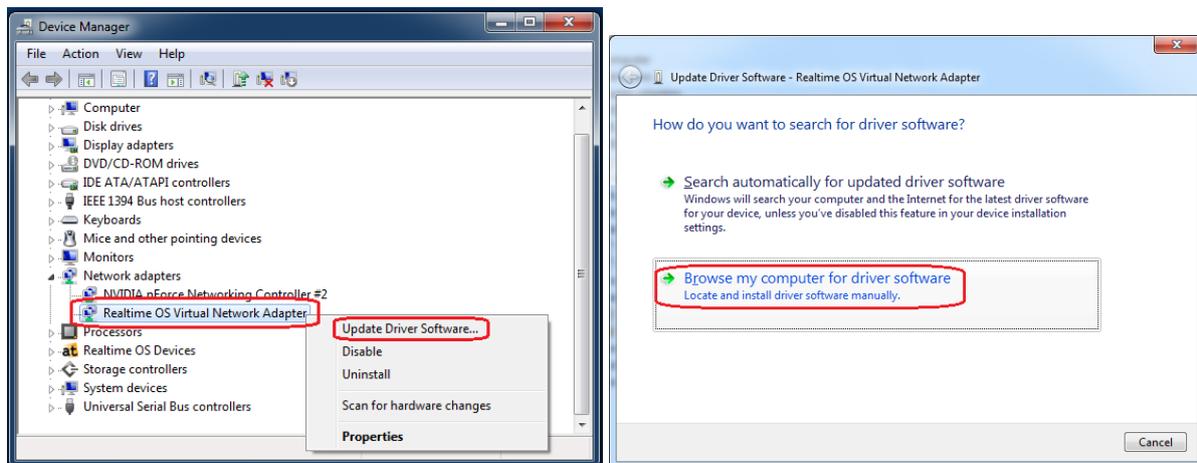
### 10.1 Configuration

It is OS dependent which features of the “Realtime OS Virtual Network Adapter” can be configured.

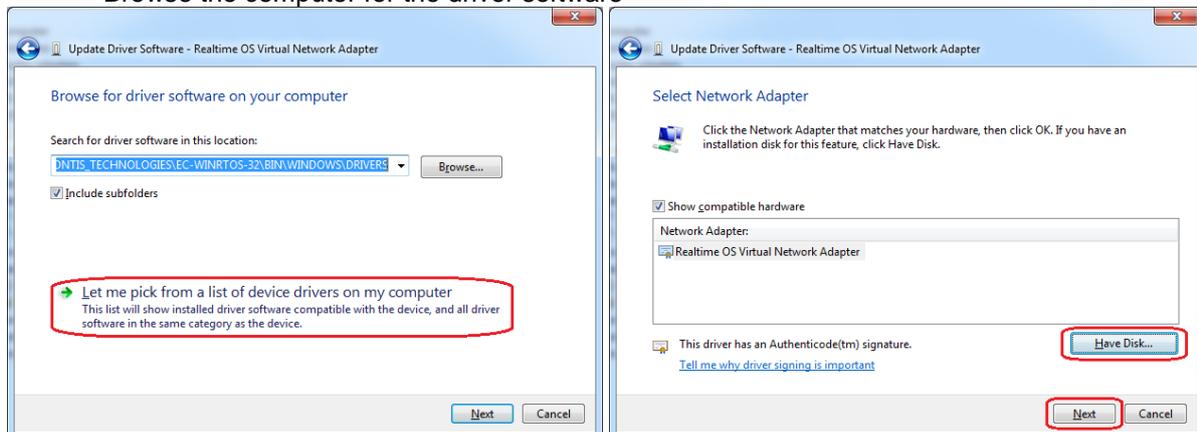
#### 10.1.1 Windows

##### 10.1.1.1 Polling or Interrupt Mode

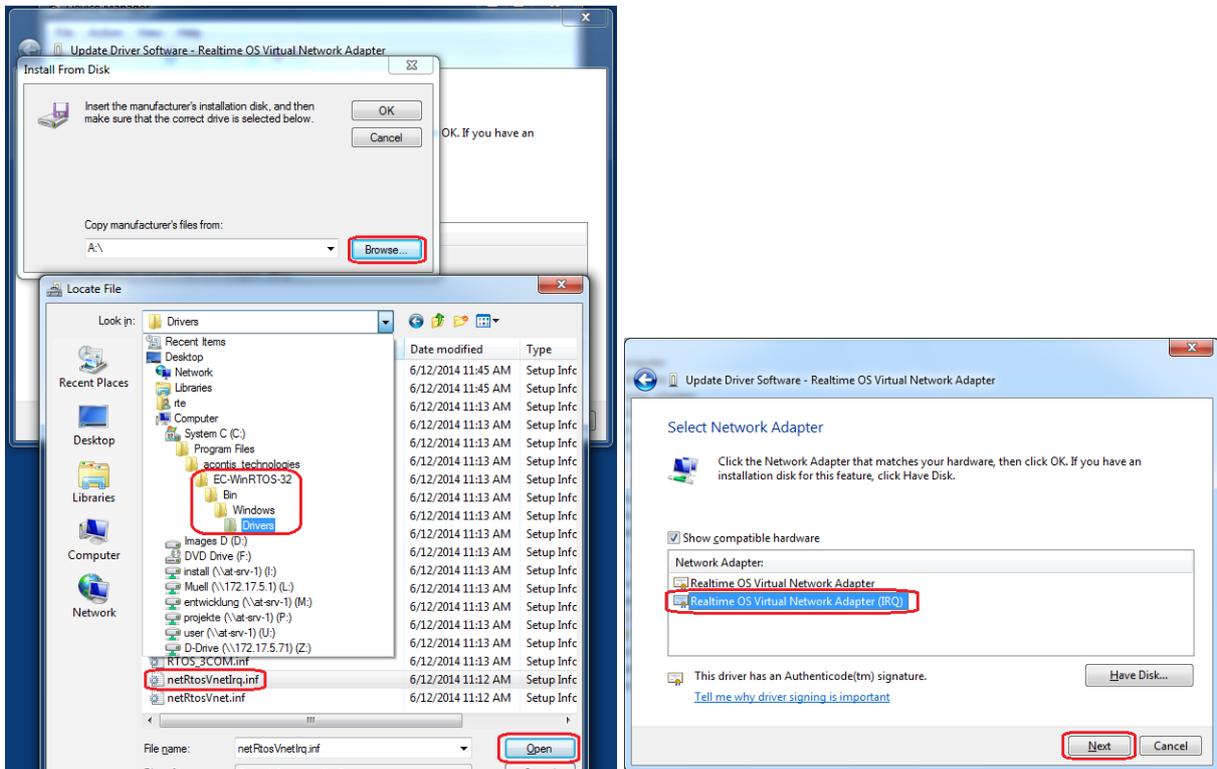
Changing from polling (default) to interrupt mode can be done using the Windows device manager.



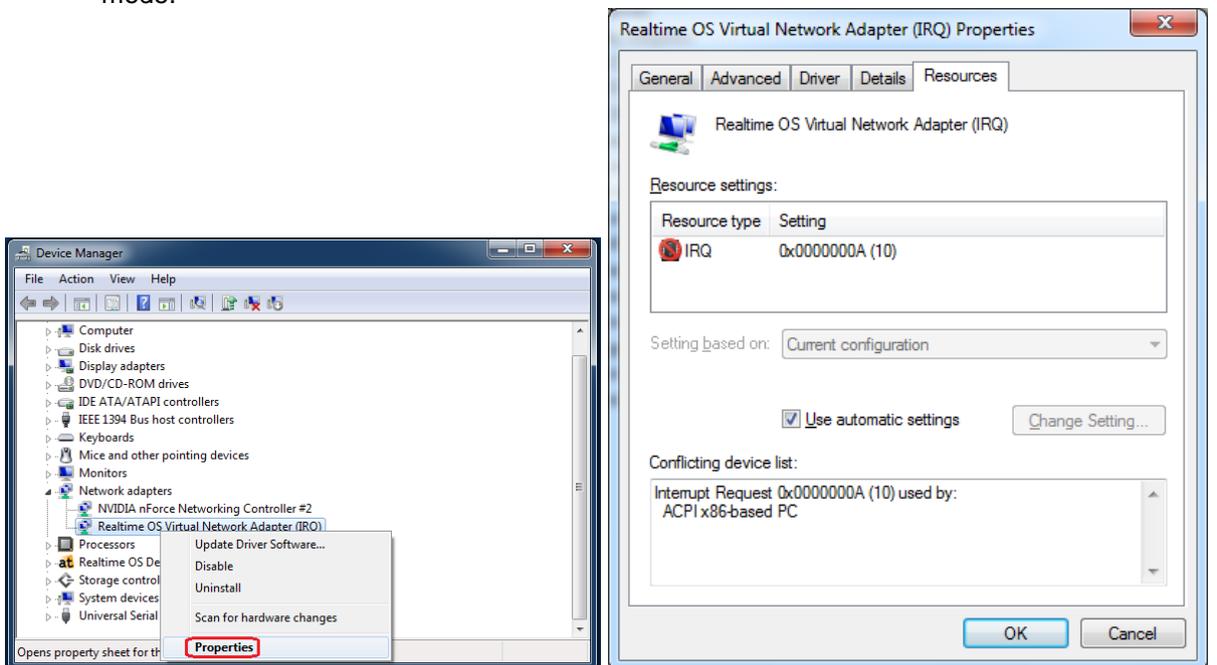
- Open the device manager, right-click “Realtime OS Virtual Network Adapter” and select “Update Driver Software...”
- Browse the computer for the driver software



- Pick a device driver from your computer by selecting “Let me pick...” and “Have Disk”



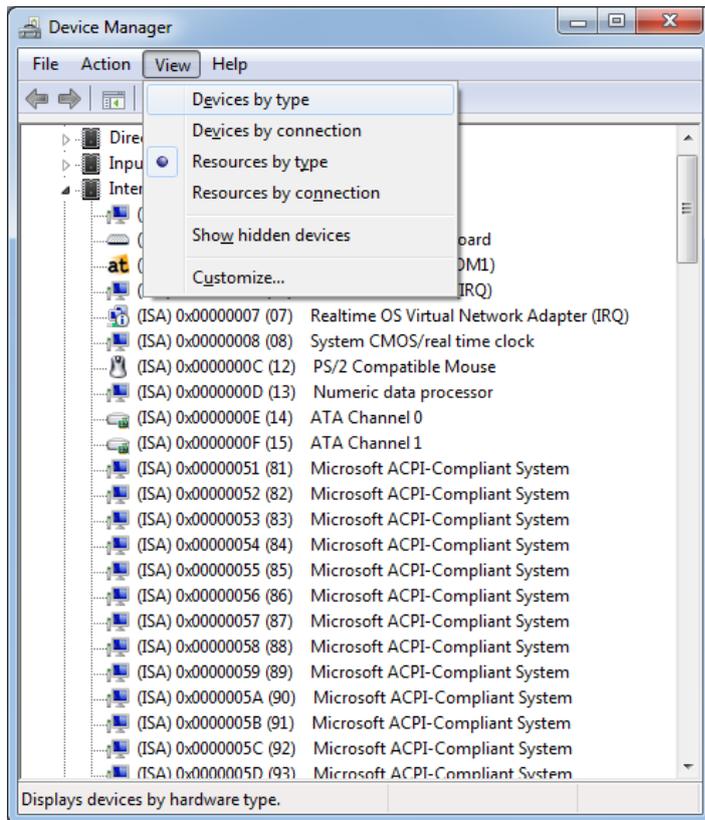
- Select the "Drivers" folder at "Bin\Windows\Drivers" below your product installation directory and choose "netRtosVnetIrq.inf". The file doesn't matter because all compatible drivers of the directory will be listed in the next step.
- Select "Realtime OS Virtual Network Adapter (IRQ)" and click "Next" to update the driver to interrupt mode. Select "Realtime OS Virtual Network Adapter" and click "Next" to update the driver to polling mode.



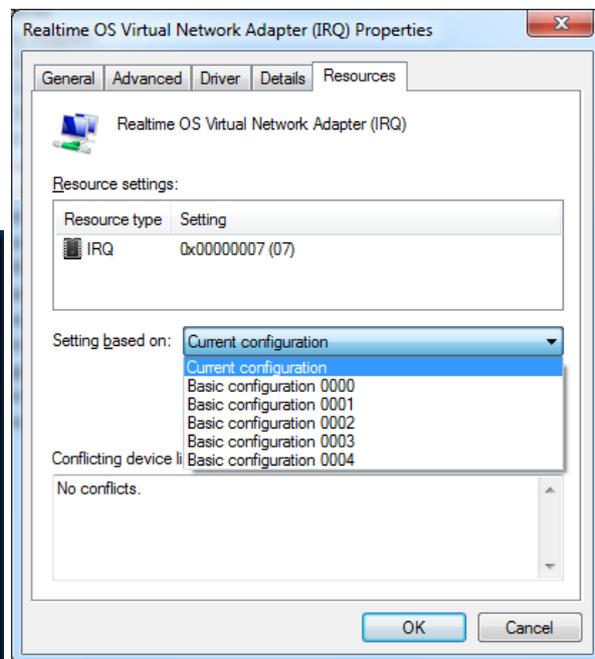
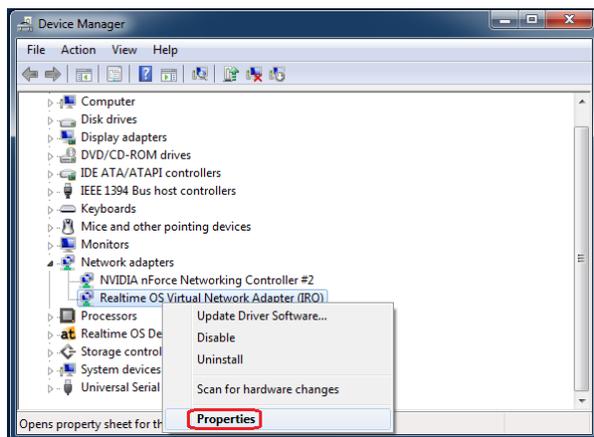
- After finishing install you can open the "Realtime OS Virtual Network Adapter (IRQ)" properties using either a double-click or a right-click and select "Properties". When you get a conflict with ACPI PC, like on the right picture, a reboot is required.

### 10.1.1.2 Solving Interrupt Conflicts

When Uploader tells about an interrupt conflict regarding “Realtime OS Virtual Network Adapter (IRQ)” you can configure another interrupt using the device manager.

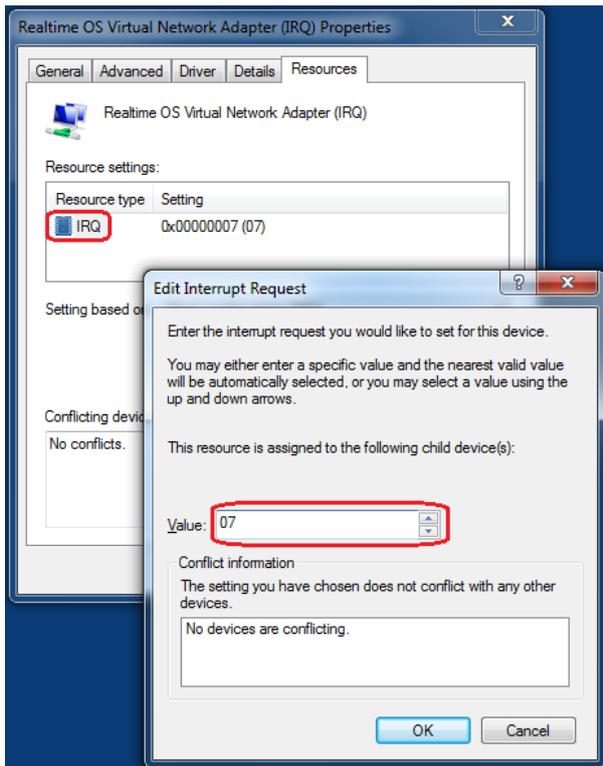


- 1) Open the Device Manager and change the view to “Resources by type” to check for a free interrupt number not conflicting with another device.



- 2) Double-click the “Realtime OS Virtual Network Adapter (IRQ)”, which can be found under the “Network adapters” in the Windows device Manager, and select the “Resources” tab.  
There you can disable “Use automatic settings” and select another setting from “Setting based on:”  
Each setting contains a different interrupt number. If no configuration contains the

desired number the last configuration can be selected to set an individual IRQ number.



- 3) To select an individual IRQ double-click "IRQ" and then use spin-buttons to select another interrupt or enter the value using the keyboard.  
HINT: Using the keyboard allows setting values not possible using the buttons!
- 4) After a reboot the interrupt conflict should be solved.

#### 10.1.1.3 Reconfiguring to polling mode

Configure polling mode is done equal to configure the interrupt mode but select "Realtime OS Virtual Network Adapter" instead of "... IRQ".

After finishing the driver update the driver properties should be checked for the "Resources" tab. If this tab is still part of the properties the driver will continue using an interrupt.

In this case the driver should be removed and then re-installed using the device manager action "Add legacy hardware" (on XP use "Add Hardware" instead of the device manager).

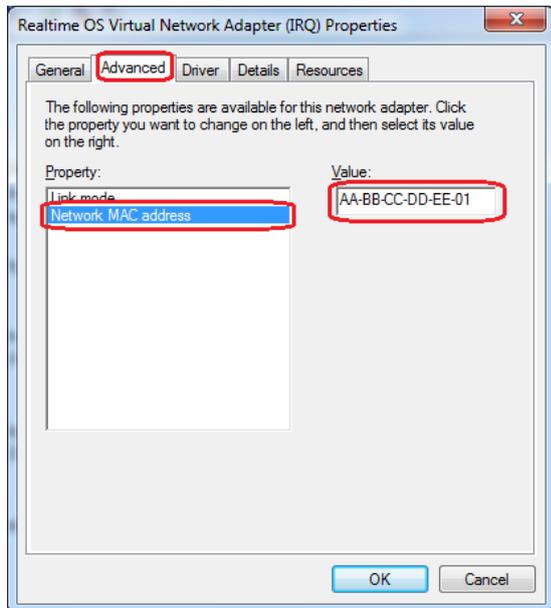
Remark:

After re-installing the network adapter its IP must be re-configured.

#### 10.1.1.4 MAC Address

The MAC address can be configured using the Windows device manager.

Open the device manager and double-click "Realtime OS Virtual Network Adapter" below "Network adapters"

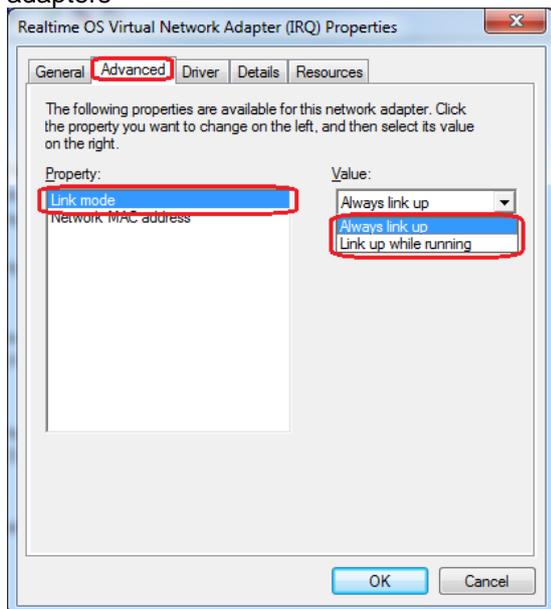


Select the “Advanced” tab and “Network MAC address” to change the MAC address value.

#### 10.1.1.5 Link Mode

The link mode can be configured using the Windows device manager.

Open the device manager and double-click “Realtime OS Virtual Network Adapter” below “Network adapters”

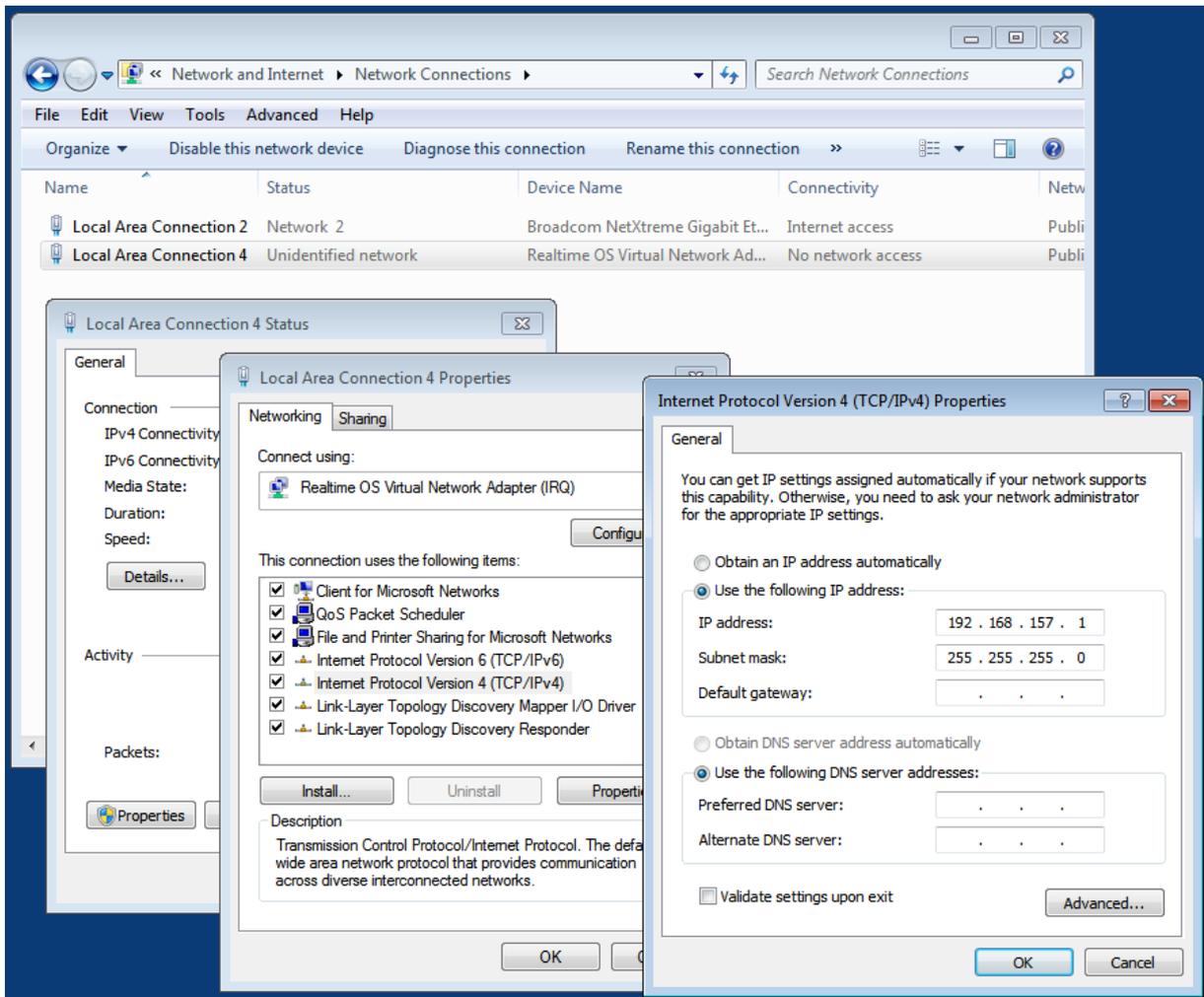


Select the “Advanced” tab and “Link mode”.

- “Always link up” means the adapter behaves like a network card always connected to a switch or hub.
- “Link up while running” means the adapter behaves like a network card connected with a crossover cable to another PC’s network card. If the other PC is turned off the link is down. When a RTOS is running the link will be up and when it’s stopped the link will go down.

#### 10.1.1.6 Network Address

The network address can be changed using the “Network Connections”.



### 10.1.2 RTOS

Please check the product specific manual for virtual network adapter configuration possibilities.

## 11 The RTOS Library

The RTOS library provides higher level communication services for synchronizing Windows with the RTOS or to exchange data between the operating systems. The RTOS library is based on VMF functions which provide the basic communication functionality.

This library is provided in source code as it has to be ported to the RTOS. To make this job easier most of the source code is written OS independently. The OS adaptation is done in a separate OS adaptation layer. VmfWin provides example implementations.

### 11.1 RTOS Library – Application Layer API

This section describes the RTOS library functions provided for the user.

User applications will have to include the RtosLib.h header file prior to using the RTOS library functions.

#### 11.1.1 Windows applications

##### 11.1.1.1 General Restriction

Before version 5.1.00.29 / 6.0.00.04 the RTOS Library API can only be called concurrently by a total number of maximum 31 threads (total sum of threads used in one or multiple executable applications).

##### 11.1.1.2 RTE Version 4.x

The Windows part of the RTOS Library is implemented in the UploadRtos.dll dynamic link library. Windows applications have to be linked together with the library UploadRtos.lib.

##### 11.1.1.3 RTE Version 5.x

The Windows part of the RTOS Library is implemented in the RtosLib32.dll (32 bit) and RtosLib64.dll (64 bit) dynamic link library.

Windows applications have to be linked together with the library RtosLib32.lib (32 bit) or RtosLib64.lib (64 bit).

To run existing programs linked to UploadRtos.dll the file RtosLib32.dll can be copied to UploadRtos.dll.

#### 11.1.2 RTOS applications

The RTOS part of the RTOS Library in most cases is implemented in a binary library. Details of the RTOS part can be found in the corresponding RTOSWin product manual.

### 11.1.3 RTOS Library – initialization and shutdown

The first step which has to be taken when the RTOS Library shall be used is calling the appropriate initialization routines of the library. After this step the RTOS Library routines are available. If the system shall be shut down, the de-initialization functions have to be called.

---

#### 11.1.3.1 RtosLibInit

Initialize the RTOS Library.

#### UINT32 RtosLibInit (VOID)

##### Parameter

–

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

Each process must call this function at least once before using RtosLib.

An internal call counter is incremented each call. The initialization will be done when the value goes from 0 to 1 and the de-initialization when it goes from 1 to 0.

There are some RtosLib functions not requiring RtosLibInit to be called:

- RtosSetOutputPrintf
- RtosResultGetModule
- RtosResultGetText
- RtosDevice
- RtosSetMemoryConfiguration

---

#### 11.1.3.2 RtosLibDeinit

De-Initialize the RTOS Library.

#### UINT32 RtosLibDeinit (VOID)

##### Parameter

–

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

A process must first close all RtosLib handles and call all Stop functions where it called Start before. An internal call counter is decremented each call. The initialization will be done when the value goes from 0 to 1 and the de-initialization when it goes from 1 to 0.

After De-Init was done no other RtosLib function except RtosLibInit should be called.

---

#### 11.1.3.3 RtosCommStart

Start the inter OS communication subsystem (e.g. required for events).

#### UINT32 RtosCommStart (VOID)

##### Parameter

–

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

The following OS will do this automatically:

- Windows: Handled by RtosDrv
- VxWorks: Done by BSP
- Windows CE: Done by RtosService

It is responsible for communication services.

Please check chapter 6.8 "OS Communication" for configuration details.

---

#### 11.1.3.4 RtosCommStop

Stop the inter OS communication subsystem.

#### UINT32 RtosCommStop (VOID)

##### Parameter

–

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

The following OS will do this automatically:

Windows: Handled by RtosDrv

VxWorks: Done by BSP

Windows CE: Done by RtosService

---

#### 11.1.3.5 RtosCommWaitForSubsystems

Wait until all required subsystems are up and running. The application has to wait until all communication subsystems are ready prior to calling the inter OS communication functions of the RTOS Library (e.g. events, interlocked data access, ...).

#### UINT32 RtosCommWaitForSubsystems (

UINT32	dwOsId,
UINT32	dwTimeout,
BOOL*	pbSubsystemsInitialized)

##### Parameter

*dwOsId*

[in] Operating System ID.

*dwTimeout*

[in] Timeout in seconds.

*pbSubsystemsInitialized*

[out] TRUE if the subsystems are initialized, FALSE if not.

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

The application wants to communicate with at least one OS (usually Windows). The value given in *dwOsId* has to be set to the appropriate value corresponding with this OS, it can be determined by *RtosGetIdByName()*. If communication with more than one OS is requested then this function has to be called for each of the corresponding operating systems.

---

### 11.1.3.6 RtosMsgBoxStart

Start the inter RTOS message box communication mechanism.

#### UINT32 RtosMsgBoxStart (VOID)

##### Parameter

–

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

If the caller's operating system will show the message box it has to be assured that the process calling RtosMsgBoxStart() is able to do this job (e.g. on the Windows XP side this function may not be called by a Windows Service as services are not able to get a Window handle).

This function is called by default on

- Windows (by RtosControl)

---

### 11.1.3.7 RtosMsgBoxStop

Stop the inter RTOS message box communication mechanism.

#### UINT32 RtosMsgBoxStop (VOID)

##### Parameter

–

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

This function is called by default on

- Windows (by RtosControl)

---

### 11.1.3.8 RtosGetIdByNameA

Get a VMF ID by a given name.

```
UINT32 RtosGetIdByNameA(  
    Const CHAR*   szName,  
    UINT32        dwIdType,  
    UINT32*       pdwId)
```

#### Parameter

*szName*

[in] Name of the desired element

*dwIdType*

[in] Element type to get an ID of. See comment for supported types (RTOS\_ID\_XXX).

*pdwOsId*

[out] requested ID.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

RTOS\_ID\_OS:

Every operating system which is part of the RTOS VM (at least Windows and one RTOS) is required to have a unique name by which it can be identified. This name will be determined by setting the appropriate entry "Name" in the OS config section in the RTOS VM configuration files. The RTOS VM then assigns a unique OS ID to internally identify the OS.

RTOS\_ID\_DEVICE:

Each device has an OS unique name (devices with same name may be assigned to another OS). The RTOS VM assigns a unique device ID to internally identify the device.

RTOS\_ID\_SHM:

Each shared memory requires a unique name. The RTOS VM assigns a unique shared memory ID to internally identify the shared memory.

The macro RtosGetIdByName can be used to call RtosGetIdByNameW if UNICODE is defined and RtosGetIdByNameA if not.

---

### 11.1.3.9 RtosGetIdByNameW

Get a VMF ID by a given name.

```
UINT32 RtosGetIdByNameW (  
    Const WCHAR*   wszName,  
    UINT32        dwIdType,  
    UINT32*       pdwId)
```

#### Parameter

*szName*

[in] Unicode name of the desired element

*dwIdType*

[in] Element type to get an ID of. See comment for supported types (RTOS\_ID\_XXX).

*pdwOsId*

[out] requested ID.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

See RtosGetIdByNameA().

The macro RtosGetIdByName can be used to call RtosGetIdByNameW if UNICODE is defined and RtosGetIdByNameA if not.

#### 11.1.4 RTOS Library – events

This feature makes it possible for RTOS applications (thread or process) to asynchronously notify an application under Windows XP of the occurrence of some event. *Shared Events* supports signaling in both directions.

A typical use for *Shared Events* would be, for example, to coordinate activities between the two operating systems. For example, if two applications use a shared memory area to communicate data back and forth, the *shared events* mechanism could be used to synchronize the reading and writing, i.e., to protect the integrity of the data.

In the rare case that events require a short round trip time Communication interrupt mode can be enabled as described in chapter 6.8 “OS Communication”.

Starting with version 6.1.00.06 the formally fixed total number of events (100) has changed to 500 per default and can be modified using

[Vmf]

"EventCount"=dword:X

See chapter “6.5

Section [Vmf]” for details.

---

### 11.1.4.1 RtosCreateEventA

Create a named event.

```
UINT32 RtosCreateEventA (  
    BOOL          bManualReset,  
    BOOL          bInitialState,  
    const CHAR*   szName,  
    RTOSLIB_HANDLE* phEvent)
```

#### Parameter

*bManualReset*  
[in] Currently only auto-reset events are supported (bManualReset = FALSE).

*bInitialState*  
[in] Currently the initial-state has always set to FALSE (non-signaled).

*szName*  
[in] Event name

*phEvent*  
[out] Event handle

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

If the event already exists a handle to the existing event object will be returned.

The handle returned by **RtosCreateEvent** may be used for all subsequent accesses to the new event object.

Any thread in the operating system that expects notification of an event can use the event's object-handle in a call to **RtosWaitForEvent**. Correspondingly, any thread in the other operating system that wishes to send notification may use **RtosSetEvent** to send the signal. The setting and waiting functions work in either direction. When the state of the specified object is triggered, the **RtosWaitForEvent** function returns, and the waiting thread will continue executing.

Even though the initial state of the event object is formally specified by the *bInitialState* parameter, the only valid initial state is the non-signaled state.

All Shared Event objects are auto-reset objects. If the initial state of a event is not set to *non-signaled*, an error will be returned. After the event has been signaled (**RtosSetEvent**), it remains signaled only until the waiting thread is released. Thereupon, the system will reset the state to non-signaled. If no threads are waiting, the event object remains signaled.

Use **RtosCloseEvent** to close the handle. The application is responsible for resource reclamation.

*Shared events* only work between the RTOS and Windows XP. Setting an event under Windows XP, can only wake a thread in the RTOS and vice versa. It is impossible to use shared Events to wake up a thread in the operating system under which the **RtosSetEvent** has been issued.

The macro RtosCreateEvent can be used to call RtosCreateEventW if UNICODE is defined and RtosCreateEventA if not.

---

#### 11.1.4.2 RtosCreateEventW

Create a named event.

```
UINT32 RtosCreateEventW (  
    BOOL                bManualReset,  
    BOOL                bInitialState,  
    const WCHAR*       wszName,  
    RTOSLIB_HANDLE*    phEvent)
```

##### Parameter

*bManualReset*

[in] Currently only auto-reset events are supported (bManualReset = FALSE).

*bInitialState*

[in] Currently the initial-state has always set to FALSE (non-signaled).

*szName*

[in] Unicode event name

*phEvent*

[out] Event handle

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

See RtosCreateEventA().

The macro RtosCreateEvent can be used to call RtosCreateEventW if UNICODE is defined and RtosCreateEventA if not.

---

#### 11.1.4.3 RtosOpenEventA

Open an already existing event object.

```
UINT32 RtosOpenEventA (  
    const CHAR*         szName,  
    RTOSLIB_HANDLE*    phEvent)
```

##### Parameter

*szName*

[in] Event name

*phEvent*

[out] Event handle

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

If the event already exists a handle to the existing event object will be returned.

The macro RtosOpenEvent can be used to call RtosOpenEventW if UNICODE is defined and RtosOpenEventA if not.

---

#### 11.1.4.4 RtosOpenEventW

Open an already existing event object.

```
UINT32 RtosOpenEventW (  
    const WCHAR*      wszName,  
    RTOSLIB_HANDLE*  phEvent)
```

##### Parameter

*szName*  
    [in] Unicode event name  
*phEvent*  
    [out] Event handle

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

See RtosOpenEventA().  
The macro RtosOpenEvent can be used to call RtosOpenEventW if UNICODE is defined and RtosOpenEventA if not.

---

#### 11.1.4.5 RtosCloseEvent

Close access to a already existing event object.

```
UINT32 RtosCloseEvent (  
    RTOSLIB_HANDLE  hEvent)
```

##### Parameter

*hEvent*  
    [in] Event handle

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

A call to this function closes a handle to the event object.  
**RtosCloseEvent** invalidates the specified object handle, decrements the object's handle count. Once the last handle to an object is closed, the object is removed from the operating system.

---

#### 11.1.4.6 RtosSetEvent

Signal a new event.

**UINT32 RtosSetEvent (**  
    **RTOSLIB\_HANDLE**        **hEvent)**

**Parameter**

*hEvent*  
    [in]        Event handle

**Return**

RTE\_SUCCESS on success and an error-code on failure.

**Comment**

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.

Note 1: Only auto-reset events are supported.

Note 2: If you call *RtosSetEvent* in the RTOS, you can only call *RtosWaitForEvent* in Windows XP. That is, you cannot wait for the same event in the RTOS. If you do, the result is unpredictable.

Note 3: A call to *RtosSetEvent* will not immediately set the event into the counter-part OS due to interactions between the two operating systems.

Note 4: Multiple calls to the *RtosSetEvent* function may be made, however, if the counterpart does not consume one of them, a subsequent *RtosSetEvent* function call will replace the former unconsumed event.

---

#### 11.1.4.7 RtosWaitForEvent

Signal a new event.

**UINT32 RtosWaitForEvent (**  
    **RTOSLIB\_HANDLE**        **hEvent,**  
    **DWORD**                  **dwTimeout)**

**Parameter**

*hEvent*  
    [in]        Event handle

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

**Return**

RTE\_SUCCESS on success (event was signaled) and an error-code on failure (RTE\_ERROR\_TIMEOUT on timeout).

**Comment**

*RtosWaitForEvent* checks the current state of the specified event object. If the object's state is *non-signaled*, the calling thread will block.

Since only auto-reset event objects are used, the wait function that successfully receives the signal resets the object's state to non-signaled before returning.

## 11.1.5 RTOS Library – interlocked data access

Interlocked data access is required whenever an atomic access to a shared data variable shall be performed.

---

### 11.1.5.1 RtosInterlockedCompareExchange

The function performs an atomic comparison of the specified values and exchanges the values, based on the outcome of the comparison. The function prevents more than one thread inside the callers OS as well as multiple operating systems from simultaneously accessing the same variable.

#### UINT32 RtosInterlockedCompareExchange (

    UINT32\* volatile   pdwDestination,  
    UINT32            dwExchange,  
    UINT32            dwComparand,  
    UINT32\*           pdwInitial)

#### Parameter

*pdwDestination*

    [in]   Pointer to the destination value.

*dwExchange*

    [in]   Exchange value.

*dwComparand*

    [in]   Value to compare to destination.

*pdwInitial*

    [out]   Destination value found at start of compare.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The function performs an atomic comparison of the *Destination* value with the comparand value. If the destination value is equal to the comparand value, the exchange value is stored in the address specified by *pnDestination*. Otherwise, no operation is performed.

The interlocked functions provide a simple mechanism for synchronizing access to a variable that is shared by multiple threads or operating systems. The threads of different processes or different operating systems can use this mechanism if the variable is in shared memory.

The parameters for this function must be aligned on a 32-bit boundary; otherwise, the function will behave unpredictably.

#### Example (how to synchronize access to a shared memory region)

```
INT32* pnShmValue;           /* points into shared memory area */
INT32  nTimeout;
UINT32 dwRes;
UINT32 dwInitVal;           /* initial value when starting exchange */
BOOL   bAccess = FALSE;

// when set to 0 no one owns the shared memory
// when set to 1 the counterpart may access shared memory
// when set to 2 we may access shared memory
for( nTimeout = 1000; 0 < nTimeout; nTimeout-- )
{
    dwRes = RtosInterlockedCompareExchange( pnShmValue, 2, 0, & dwInitVal );
    if( RTE_SUCCESS != dwRes )
    {
        /* error during call */
        goto Exit;
    }

    if( 0 == dwInitVal )
    {
        /* success: value exchanged */
        bAccess = TRUE;
        break;
    }
    Sleep(1);
}
```

```

}

/* check for access */
if( TRUE == bAccess )
{
    // now access the shared memory region
    :   :   :
    :   :   :
    // now release access to the shared memory region
    *pnShmValue = 0;
}

```

---

### 11.1.5.2 RtosInterlockedExchangeAdd

The function performs an atomic addition. The function prevents more than one thread inside the callers OS as well as multiple operating systems from simultaneously accessing the same variable.

#### UINT32 RtosInterlockedExchangeAdd (

```

    INT32* volatile  pnDestination,
    INT32          nValue,
    INT32*          pnInitial)

```

#### Parameter

*pnDestination*  
[in] Pointer to the destination value.

*nValue*  
[in] The value to be added (might be negative).

*pnInitial*  
[out] Destination value found before addition.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The function performs an atomic addition.  
The interlocked functions provide a simple mechanism for synchronizing access to a variable that is shared by multiple threads or operating systems. The threads of different processes or different operating systems can use this mechanism if the variable is in shared memory.  
The parameters for this function must be aligned on a 32-bit boundary; otherwise, the function will behave unpredictably.

---

### 11.1.5.3 RtosInterlockedAnd

The function performs an atomic AND operation. The function prevents more than one thread inside the callers OS as well as multiple operating systems from simultaneously accessing the same variable.

**UINT32 RtosInterlockedAnd (**  
    **UINT32\* volatile pdwDestination,**  
    **UINT32 dwValue,**  
    **UINT32\* pdwInitial)**

#### Parameter

*pdwDestination*

[in] Pointer to the destination value.

*dwValue*

[in] The value used for the logical operation with the destination value.

*pdwInitial*

[out] Destination value found before logical operation.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The function performs an atomic AND operation.

The interlocked functions provide a simple mechanism for synchronizing access to a variable that is shared by multiple threads or operating systems. The threads of different processes or different operating systems can use this mechanism if the variable is in shared memory.

The parameters for this function must be aligned on a 32-bit boundary; otherwise, the function will behave unpredictably.

---

### 11.1.5.4 RtosInterlockedOr

The function performs an atomic OR operation. The function prevents more than one thread inside the callers OS as well as multiple operating systems from simultaneously accessing the same variable.

**UINT32 RtosInterlockedOr (**  
    **UINT32\* volatile pdwDestination,**  
    **UINT32 dwValue,**  
    **UINT32\* pdwInitial)**

#### Parameter

*pdwDestination*

[in] Pointer to the destination value.

*dwValue*

[in] The value used for the logical operation with the destination value.

*pdwInitial*

[out] Destination value found before logical operation.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The function performs an atomic OR operation.

The interlocked functions provide a simple mechanism for synchronizing access to a variable that is shared by multiple threads or operating systems. The threads of different processes or different operating systems can use this mechanism if the variable is in shared memory.

The parameters for this function must be aligned on a 32-bit boundary; otherwise, the function will behave unpredictably.

---

### 11.1.5.5 RtosInterlockedXor

The function performs an atomic XOR operation. The function prevents more than one thread inside the callers OS as well as multiple operating systems from simultaneously accessing the same variable.

**UINT32 RtosInterlockedXor (**  
    **UINT32\* volatile pdwDestination,**  
    **UINT32 dwValue,**  
    **UINT32\* pdwInitial)**

#### Parameter

*pdwDestination*

[in] Pointer to the destination value.

*dwValue*

[in] The value used for the logical operation with the destination value.

*pdwInitial*

[out] Destination value found before logical operation.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The function performs an atomic XOR operation.

The interlocked functions provide a simple mechanism for synchronizing access to a variable that is shared by multiple threads or operating systems. The threads of different processes or different operating systems can use this mechanism if the variable is in shared memory.

The parameters for this function must be aligned on a 32-bit boundary; otherwise, the function will behave unpredictably.

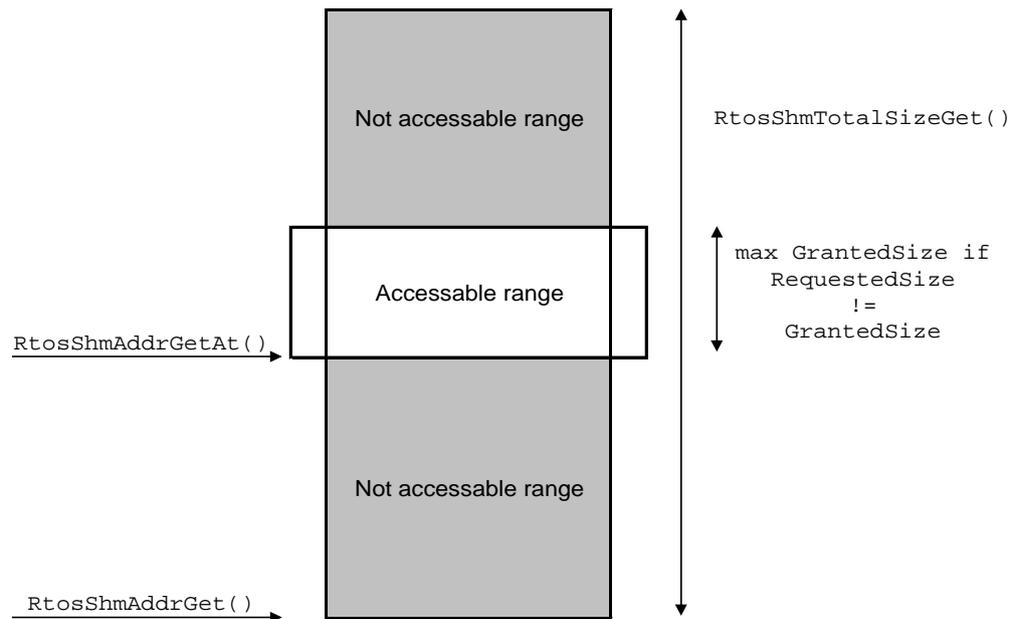
### 11.1.6 RTOS Library – shared memory

The RTOS VM platform provides one system-wide shared memory area to exchange data between multiple operating systems. Some operating systems may only provide a limited amount of the shared memory area to be mapped into the caller's memory context (e.g. Windows XP). In this case it is necessary to use windowing technique to access the whole shared memory area.

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

- using the function **RtosShmAddrGet**, an application 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 **dwRequestedSize** to the value returned by **RtosShmTotalSizeGet**, call the **RtosShmAddrGet** function.
- If the returned parameter **pdwGrantedSize** is not equal to **dwRequestedSize**, then **pdwGrantedSize** 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:** Most RTOS 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.

---

### 11.1.6.1 RtosShmAddrGet

Determine the base address of the shared memory area.

If (RTOSLIB\_API\_VERSION < 50):

```
UINT32 RtosShmAddrGet (  
    UINT32          dwRequestedSize,  
    UINT32*        pdwGrantedSize,  
    VOID**         ppvShmAddr)
```

#### Parameter

*dwRequestedSize*

[in] Requested size of shared memory to be accessed by the application.

*pdwGrantedSize*

[out] Actually granted size that is mapped into the caller's memory context.

*ppvShmAddr*

[out] Base address of the mapped shared memory area.

If (RTOSLIB\_API\_VERSION >= 50):

```
UINT32 RtosShmAddrGet (  
    UINT32          dwShmId,  
    UINT32          dwRequestedSize,  
    UINT32*        pdwGrantedSize,  
    VOID**         ppvShmAddr)
```

#### Parameter

*dwShmId*

[in] Shared memory ID – can be requested by *RtosGetIdByName(...)*

*dwRequestedSize*

[in] Requested size of shared memory to be accessed by the application.

*pdwGrantedSize*

[out] Actually granted size that is mapped into the caller's memory context.

*ppvShmAddr*

[out] Base address of the mapped shared memory area.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

Any pointer previously initialized with a call to *RtosShmAddrGet* or *RtosShmAddrGetAt* must be considered as invalid after a renewed call to those functions.

---

### 11.1.6.2 RtosShmAddrGetAt

Determine the address of the shared memory area at a given offset.

If (RTOSLIB\_API\_VERSION < 50):

```
UINT32 RtosShmAddrGetAt (  
    UINT32          dwOffset,  
    UINT32          dwRequestedSize,  
    UINT32*         pdwGrantedSize,  
    VOID**          ppvShmAddr)
```

#### Parameter

*dwOffset*  
[in] Offset from the shared memory base where to map the memory into the caller's memory context.

*dwRequestedSize*  
[in] Requested size of shared memory to be accessed by the application.

*pdwGrantedSize*  
[out] Actually granted size that is mapped into the caller's memory context.

*ppvShmAddr*  
[out] Base address of the mapped shared memory area.

If (RTOSLIB\_API\_VERSION >= 50):

```
UINT32 RtosShmAddrGetAt (  
    UINT32          dwShmId,  
    UINT32          dwOffset,  
    UINT32          dwRequestedSize,  
    UINT32*         pdwGrantedSize,  
    VOID**          ppvShmAddr)
```

#### Parameter

*dwShmId*  
[in] Shared memory ID – can be requested by RtosGetIdByName(...)

*dwOffset*  
[in] Offset from the shared memory base where to map the memory into the caller's memory context.

*dwRequestedSize*  
[in] Requested size of shared memory to be accessed by the application.

*pdwGrantedSize*  
[out] Actually granted size that is mapped into the caller's memory context.

*ppvShmAddr*  
[out] Base address of the mapped shared memory area.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

Any pointer previously initialized with a call to *RtosShmAddrGet* or *RtosShmAddrGetAt* must be considered as invalid after a renewed call to those functions.

---

### 11.1.6.3 RtosShmTotalSizeGet

Determine the total size of the system-wide shared memory area.

If (RTOSLIB\_API\_VERSION < 50):

```
UINT32 RtosShmTotalSizeGet(  
    UINT32*      pdwSize)
```

#### Parameter

*pdwSize*  
[out] Total shared memory size.

If (RTOSLIB\_API\_VERSION >= 50):

```
UINT32 RtosShmTotalSizeGet(  
    UINT32      dwShmId,  
    UINT32*     pdwSize)
```

#### Parameter

*dwShmId*  
[in] Shared memory ID – can be requested by RtosGetIdByName(...)  
*pdwSize*  
[out] Total shared memory size.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

–

---

#### 11.1.6.4 RtosShmSaveFileA

Save a shared memory to a file.

**UINT32 RtosShmSaveFileA (**  
    **UINT32**        **dwShmId,**  
    **const CHAR\***    **szFilename)**

##### Parameter

*dwShmId*  
    [in] Shared memory ID  
*szFilename*  
    [out] File to write shared memory data to.

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

A shared memory ID can be queried using "RtosGetIdByName".  
The macro RtosShmSaveFile can be used to call RtosShmSaveFileW if UNICODE is defined and RtosShmSaveFileA if not.

---

#### 11.1.6.5 RtosShmSaveFileW

Save a shared memory to a file.

**UINT32 RtosShmSaveFileA (**  
    **UINT32**        **dwShmId,**  
    **const WCHAR\***    **wszFilename)**

##### Parameter

*dwShmId*  
    [in] Shared memory ID  
*wszFilename*  
    [out] File to write shared memory data to.

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

A shared memory ID can be queried using "RtosGetIdByName".  
The macro RtosShmSaveFile can be used to call RtosShmSaveFileW if UNICODE is defined and RtosShmSaveFileA if not.

## 11.1.7 RTOS Library – date and time synchronization (clock synchronization)

---

### 11.1.7.1 RtosTimeSyncStart

Start the date and time synchronization.

#### UINT32 RtosTimeSyncStart (VOID)

##### Parameter

–

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

This function is called by default on

- Windows (by RtosService)
- Windows CE (by RtosService)
- VxWorks (by VxWin BSP)

On other OS the time and date synchronization is not running by default and the initial RTOS date and time value will have to be set to a reasonable initial value by the RTOS Board Support Package.

In the default configuration Windows is the master. Configuration is described in chapter “0

Time/Date and Timezone synchronization”.

---

### **11.1.7.2 RtosTimeSyncStop**

Stop the date and time synchronization.

#### **UINT32 RtosTimeSyncStop (VOID)**

##### **Parameter**

–

##### **Return**

RTE\_SUCCESS on success and an error-code on failure.

##### **Comment**

–

### 11.1.8 RTOS Library – OS scheduling

When running in shared mode (Windows XP and the RTOS share one single processor core) by default the RTOS will always be executed with higher priority than Windows. Windows effectively runs as idle task of the RTOS.

RTOS should never run any other task on the same or a lower priority than the idle task because it might not get any CPU time.

In case the RTOS never enters its idle state this would lead to a situation where Windows XP never gets CPU time. Therefore it is possible to control OS scheduling manually, e.g. to force the RTOS to switch back to Windows.

---

#### 11.1.8.1 RtosIdle

Regardless of the priority of the currently executing RTOS thread that calls *RtosIdle()*, this function forces the execution context to switch immediately to Windows XP. Should you have a task that monopolizes the CPU time, you can use this function to make certain that Windows XP receives additional CPU time. Upon the occurrence of the next real-time interrupt (e.g. from the real-time system timer), control will be returned to the real-time system.

#### VOID RtosIdle (VOID)

##### Parameter

–

##### Return

–

##### Comment

Before RTE version 5.0.00.02 / 4.5.00.20 / 4.1.01.13:

- If this function is called in exclusive mode it will return immediately.

Starting with RTE version 5.0.00.02 / 4.5.00.20 / 4.1.01.13:

- If this function is called in exclusive mode the processor will idle until next interrupt.

### 11.1.9 RTOS Library – notification events

The RTOS VM will notify the application about the occurrence of several events. If the RTOS application waits for such an event the RTOS VM will signal this event and wait until the RTOS application acknowledges that it has finished event handling.

For example, the RTOS application may wait for the STOP event, which will be signaled if the RTOS is stopped by the Windows Uploader tool. In that case the application may block stopping the RTOS until a safe state has been reached. After acknowledging this notification the RTOS will finally be stopped by the Windows Uploader tool.

#### Important:

**RtosNotification :** Only one single thread is allowed to wait for a specific event.

**RtosNotificationEx :** Multiple thread are allowed but the API is currently implemented for Windows only.

---

#### 11.1.9.1 RtosNotificationWait

Wait for specific notification event.

#### UINT32 RtosNotificationWait (

    UINT32                dwNotificationId  
    UINT32                dwTimeout)

#### Parameter

*dwNotificationId*

[in]	Notification identifier. The following notification identifiers may be generated:
	RTOS_NOTIFICATION_ID_BSOD                     → Windows BSOD
	RTOS_NOTIFICATION_ID_SUSPEND                 → Windows Suspend
	RTOS_NOTIFICATION_ID_RESUME                 → Windows Resume
	RTOS_NOTIFICATION_ID_STOP                    → Uploader requests RTOS Stop

*dwTimeout*

[in]      Timeout in milliseconds

#### Return

RTE\_SUCCESS on success and an error-code on failure. If another thread called RtosNotificationBreakWait with the corresponding id then RTE\_ERROR\_BREAK\_WAIT will be returned.

#### Comment

If this function is called the RTOS VM will notify the application about the corresponding event. The normal event handling in the RTOS VM will be blocked until the application calls the appropriate RtosNotificationDone() routine. For example if the application waits for the BSOD event then Windows will not enter its normal BSOD handling until RtosNotificationDone(RTOS\_NOTIFICATION\_ID\_BSOD) is called.

---

### 11.1.9.2 RtosNotificationDone

Acknowledge event handling.

**UINT32 RtosNotificationDone (**  
**UINT32 dwNotificationId)**

#### Parameter

*dwNotificationId*

[in] Notification identifier. The following notification identifiers exist:

RTOS_NOTIFICATION_ID_BSOD	→ Windows BSOD
RTOS_NOTIFICATION_ID_SUSPEND	→ Windows Suspend
RTOS_NOTIFICATION_ID_RESUME	→ Windows Resume
RTOS_NOTIFICATION_ID_STOP	→ Uploader requests RTOS Stop

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

Tell the RTOS VM that the corresponding event was processed by the application and the RTOS VM can continue its normal event handling.

---

### 11.1.9.3 RtosNotificationBreakWait

Continue the thread which is waiting for the specified event.

**UINT32 RtosNotificationBreakWait (**  
**UINT32 dwNotificationId)**

#### Parameter

*dwNotificationId*

[in] Notification identifier. The following notification identifiers exist:

RTOS_NOTIFICATION_ID_BSOD	→ Windows BSOD
RTOS_NOTIFICATION_ID_SUSPEND	→ Windows Suspend
RTOS_NOTIFICATION_ID_RESUME	→ Windows Resume
RTOS_NOTIFICATION_ID_STOP	→ Uploader requests RTOS Stop

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

In case a thread is waiting for the given notification id it will immediately return to the application. The caller then will get a return value of RTE\_ERROR\_BREAK\_WAIT.

### 11.1.9.4 BSOD handling example

```
DWORD dwRes;  
  
dwRes = RtosNotificationWait(RTOS_NOTIFICATION_ID_BSOD, RTOS_WAIT_INFINITE);  
if (dwRes == RTE_SUCCESS)  
{  
    // Handle the BSOD situation!  
    // [...]  
    // Now enter normal BSOD handling (the RTOS will be stopped now)  
    RtosNotificationDone(RTOS_NOTIFICATION_ID_BSOD);  
}
```

---

### 11.1.9.5 RtosNotificationExCreate

Create a notification object to wait for specific notification(s).

**UINT32 RtosNotificationExCreate (**  
    **UINT32**                    **dwMaskWait,**  
    **RTOSLIB\_HANDLE\***        **phObject )**

#### Parameter

*dwMaskWait*

[in] Mask of notifications to be notified. The following values exist (can be combined):

RTOSNOTIFICATION_MASK_VMFLOADED	→ VMF was loaded
RTOSNOTIFICATION_MASK_VMFUNLOAD	→ VMF will be unloaded
RTOSNOTIFICATION_MASK_VMFSUSPEND	→ VMF will be suspended
RTOSNOTIFICATION_MASK_VMFRESUMED	→ VMF was resumed
RTOSNOTIFICATION_MASK_OSSTARTED	→ OS was started
RTOSNOTIFICATION_MASK_OSSTOP	→ OS will be stopped
RTOSNOTIFICATION_MASK_OSSUSPEND	→ OS will be suspended
RTOSNOTIFICATION_MASK_OSRESUMED	→ OS was resumed
RTOSNOTIFICATION_MASK_OSBSOD	→ Windows BSOD

*phObject*

[out] Pointer to receive a RTOSLIB\_HANDLE object (notification-ex).

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The returned handle can be used with the generic object functions.

From the moment the handle all notifications will be received and must be confirmed by calling RtosNotificationExWait() again or closing the handle.

The Uploader for example will be blocked if an application registered for VMF load notification but did not confirm the notification after it was notified.

---

### 11.1.9.6 RtosNotificationExWait

Wait for a notification.

**UINT32 RtosNotificationExWait (**  
    **RTOSLIB\_HANDLE**                   **hObject,**  
    **RTOS\_NOTIFICATION\_INFO**       **pInfo,**  
    **UINT32**                           **dwTimeoutMs )**

#### Parameter

*hObject*

[in] Notification handle

*pInfo*

[out] Pointer to a RTOS\_NOTIFICATION\_INFO structure. Its dwSize must be initialized to sizeof(RTOS\_NOTIFICATION\_INFO)

*dwTimeoutMs*

[in] Timeout in milliseconds

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The function will return on timeout or when a notification occurred.

After a notification occurred it must be confirmed by calling RtosNotificationExWait again or closing the handle so the process can continue.

For any OS specific notification ( RTOSNOTIFICATION\_MASK\_OS... )

RTOS\_NOTIFICATION\_INFO.dwOsId contains the ID of the related OS.

Additionally should be pointed out that when VMF is being unloaded it is possible that

RTOSNOTIFICATION\_MASK\_VMFUNLOAD is received before

RTOSNOTIFICATION\_MASK\_OSSTOP!

---

### 11.1.9.1 RtosNotificationExBreakWait

This is a synonym for RtosObjectBreakWait. See RtosObjectBreakWait for details.

**UINT32 RtosNotificationExBreakWait (**  
    **RTOSLIB\_HANDLE**                   **hObject )**

---

### 11.1.9.1 RtosNotificationExClose

This is a synonym for RtosObjectClose. See RtosObjectClose for details.

**UINT32 RtosNotificationExClose (**  
    **RTOSLIB\_HANDLE**                   **hObject )**

## 11.1.10 RTOS Library – uploader API

These functions are provided only for the Windows version of the RTOS Library. It is related to starting and stopping the RTOS.

---

### 11.1.10.1 SetOutputBuffer

The Uploader tool will internally generate null-terminated ANSI-string messages. If the user application wants to get these messages it has to tell the Uploader where to store these messages. If the message buffer is full no further messages will be stored (no wrap).

**UINT32 SetOutputBuffer (**  
    **VOID\***            **pvOutputBuffer,**  
    **UINT32**          **dwSize)**

#### Parameter

*pvOutputBuffer*

[in] Pointer to the buffer where ANSI string messages shall be stored.

*dwSize*

[in] Size of the buffer.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

This function has been superseded by RtosSetOutputPrintf() to ensure synchronized output and solve buffer limitation problems.

---

### 11.1.10.2 RtosSetOutputPrintfA

The Uploader tool will internally generate messages. If the user application wants to get these messages it can register a printf - callback function.

**UINT32 RtosSetOutputPrintfA(**  
    **RTOSLIB\_PFN\_PRINTF\_A pfnPrintfA )**

#### Parameter

*pfnPrintfA*

[in] Pointer to a ANSI printf callback function of type  
INT32 (RTOSLIB\_CALLCONV \*RTOSLIB\_PFN\_PRINTF\_A)(const CHAR \*szFormat,  
...);

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The macro RtosSetOutputPrintf can be used to call RtosSetOutputPrintfW if UNICODE is defined and RtosSetOutputPrintfA if not.

The function does not require RtosLibInit to be called.

---

### 11.1.10.1 RtosSetOutputPrintfW

The Uploader tool will internally generate messages. If the user application wants to get these messages it can register a printf - callback function.

#### UINT32 RtosSetOutputPrintfW( RTOSLIB\_PFN\_PRINTF\_W pfnPrintfW )

##### Parameter

*pfnPrintfW*

[in] Pointer to a UNICODE printf callback function of type  
INT32 (RTOSLIB\_CALLCONV \*RTOSLIB\_PFN\_PRINTF\_W)(const WCHAR  
\*wszFormat, ...);

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

The macro RtosSetOutputPrintf can be used to call RtosSetOutputPrintfW if UNICODE is defined and RtosSetOutputPrintfA if not.

The function does not require RtosLibInit to be called.

---

### 11.1.10.2 RtosStartA

Upload and start the RTOS image.

#### UINT32 RtosStartA ( const CHAR\* szImageName, const CHAR\* szConfigFile)

##### Parameter

*szImageName*

[in] Path and filename of the RTOS image file.

*szConfigFile*

[in] Path and filename of the RTOS configuration file. The RTOS started is described in section [Rtos].

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

The macro RtosStart can be used to call RtosStartW if UNICODE is defined and RtosStartA if not.

RtosStartA internally calls RtosStartExA( szImageName, szConfigFile, 0 )

See RtosStartExA.

---

### 11.1.10.3 RtosStartW

Upload and start the RTOS image.

**UINT32 RtosStartW (**  
    **const WCHAR\****wszImageName***,**  
    **const WCHAR\****wszConfigFile***)**

#### **Parameter**

*wszImageName*

[in] Unicode path and filename of the RTOS image file.

*wszConfigFile*

[in] Unicode path and filename of the RTOS configuration file. The RTOS started is described in section [Rtos].

#### **Return**

RTE\_SUCCESS on success and an error-code on failure.

#### **Comment**

The macro RtosStart can be used to call RtosStartW if UNICODE is defined and RtosStartA if not. RtosStartW internally calls RtosStartExW( *wszImageName*, *wszConfigFile*, 0 )  
See RtosStartExA.

---

#### 11.1.10.4 RtosStartExA

Upload and start the RTOS image.

**UINT32 RtosStartExA (**  
    **const CHAR\***    **szImageName,**  
    **const CHAR\***    **szConfigFile,**  
    **UINT32**          **dwOsId)**

#### Parameter

*szImageName*  
    [in] Path and filename of the RTOS image file.  
*szConfigFile*  
    [in] Path and filename of the RTOS configuration file.  
*dwOsId*  
    [in] OS ID of the RTOS to be started. Depending on the ID the RTOS started is described in section:  
        0: [Rtos]  
        1: [Rtos1]  
        2: [Rtos2]  
        ...

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The Uploader tool acts as the RTOS bootloader. It can handle different situations:

- 1) Load VMF and start a RTOS  
    Call using *szImageName*, *szConfigFile*, *dwOsId*.  
    This will stop any running RTOS, unload the VMF, load VMF again with values from *szConfigFile* and start RTOS *szImageName*.
- 2) Load VMF but start no RTOS  
    Call using *szConfigFile* (*szImageName*=NULL, *dwOsId* will be ignored).  
    This will stop any running RTOS, unload the VMF and load VMF again with values from *szConfigFile* but not start any RTOS.
- 3) Do not load VMF but start RTOS  
    Call using *szImageName*, *dwOsId* (*szConfigFile*=NULL).  
    This will start or restart the RTOS with the given *dwOsId* but not load or reload th VMF.  
    This function will fail if no VMF is loaded (1 or 2 was not called before).

Start RTOS means this function will copy the RTOS image file into a memory area not used by Windows and then call the boot entry point of the RTOS.

The macro *RtosStartEx* can be used to call *RtosStartExW* if UNICODE is defined and *RtosStartExA* if not.

The config file string has the format "'f1' <options>".

- 'f1' use this file as config file

Possible options are:

- /vmf 'f2' use f2 as VMF binary (default is vmf.bin)

For example "'c:\MyOs.config' /vmf 'c:\MyVmf.bin'" uses the configuration file "c:\MyOs.config" and the VMF binary "c:\MyVmf.bin".

Load VMF and start OS:

- *RtosStartExA*( "c:\MyOs.bin", "'c:\MyOs.config' /vmf 'c:\MyVmf.bin'", 0 );
- *RtosStartExA*( "c:\MyOs.bin", "c:\MyOs.config", 0 );

Load VMF but do not start OS:

- *RtosStartExA*( NULL, "'c:\MyOs.config' /vmf 'c:\MyVmf.bin'", 0 );
- *RtosStartExA*( NULL, "c:\MyOs.config", 0 );

Start OS (VMF must already be loaded):

- *RtosStartExA*( "c:\MyOs.bin", NULL, 0 );

Remarks:

- Pay attention to the single and double quotation marks!
- On a 64bit system using Windows 8 or newer only a 64bit application can call this function!

---

### 11.1.10.5 RtosStartExW

Upload and start the RTOS image.

```
UINT32 RtosStartExW (  
    const WCHAR*  wszImageName,  
    const WCHAR*  wszConfigFile,  
    UINT32        dwOsId)
```

#### Parameter

*wszImageName*

[in] Unicode path and filename of the RTOS image file.

*wszConfigFile*

[in] Unicode path and filename of the RTOS configuration file.

*dwOsId*

[in] OS ID of the RTOS to be started. Depending on the ID the RTOS started is described in section:

0: [Rtos]

1: [Rtos1]

2: [Rtos2]

...

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

See RtosStartExA()

The macro RtosStartEx can be used to call RtosStartExW if UNICODE is defined and RtosStartExA if not.

---

### 11.1.10.6 RtosStop

Stop all RTOS operation and unload the VMF.

```
UINT32 RtosStop (VOID)
```

#### Parameter

—

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

All RTOS will be stopped and VMF will be unloaded.

---

### 11.1.10.7 RtosStopEx

Stop the RTOS operation without unloading the VMF.

**UINT32 RtosStopEx (**  
    **UINT32**        **dwOsId)**

**Parameter**

*dwOsId*  
    [in]    OS ID of the RTOS to be stopped.

**Return**

RTE\_SUCCESS on success and an error-code on failure.

**Comment**

The RTOS will be stopped but VMF will not be unloaded.

---

### 11.1.10.8 RtosRunning

Determine if the RTOS is running.

**BOOL RtosRunning (VOID)**

**Parameter**

—

**Return**

TRUE if the RTOS is running, FALSE if not.

**Comment**

This function will return the state of RTOS with OsId 0 only.

---

### 11.1.10.9 RtosRunningEx

Determine if the RTOS is running.

**UINT32 RtosRunningEx (**  
    **UINT32**        **dwOsId**  
    **BOOL**          **\*pbIsRunning)**

**Parameter**

*dwOsId*  
    [in]    OS ID of the RTOS to be queried.  
*pbIsRunning*  
    [out]   contains TRUE if the RTOS is running, FALSE if not.

**Return**

RTE\_SUCCESS on success and an error-code on failure.

**Comment**

—

## 11.1.11 RTOS Library – result value

---

### 11.1.11.1 RtosResultGetTextA

Get a message text for a result value.

```
UINT32 RtosResultGetTextA (  
    UINT32      dwResult,  
    CHAR*      szText,  
    UINT32*    pdwSizeInBytes)
```

#### Parameter

*dwRteResult*  
[in] Result value to query text for

*szText*  
[out] String buffer to receive the result message text

*pdwSizeInBytes*  
[in] Length of string buffer in bytes  
[out] Number of bytes used for message text

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

A text for a result value can be queried using “RtosResultGetText”.  
The macro RtosResultGetText can be used to call RtosResultGetTextW if UNICODE is defined and RtosResultGetTextA if not.  
The function does not require RtosLibInit to be called.

---

### 11.1.11.2 RtosResultGetTextW

Get a message text for a result value.

```
UINT32 RtosResultGetTextW (  
    UINT32      dwResult,  
    WCHAR*     wszText,  
    UINT32*    pdwSizeInBytes)
```

#### Parameter

*dwRteResult*  
[in] Result value to query text for

*wszText*  
[out] String buffer to receive the result message text

*pdwSizeInBytes*  
[in] Length of string buffer in bytes  
[out] Number of bytes used for message text

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

A text for a result value can be queried using “RtosResultGetText”.  
The macro RtosResultGetText can be used to call RtosResultGetTextW if UNICODE is defined and RtosResultGetTextA if not.  
The function does not require RtosLibInit to be called.

---

### 11.1.11.3 RtosResultGetModuleA

Get a module name for a result value.

**UINT32 RtosResultGetModuleA (**  
    **UINT32**        **dwResult,**  
    **CHAR\***        **szText,**  
    **UINT32\***      **pdwSizeInBytes)**

#### Parameter

*dwRteResult*  
    [in] Result value to query module name for  
*szText*  
    [out] String buffer to receive the result module name  
*pdwSizeInBytes*  
    [in] Length of string buffer in bytes  
    [out] Number of bytes used for module name

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

A module name for a result value can be queried using "RtosResultGetModule".  
The macro RtosResultGetModule can be used to call RtosResultGetModuleW if UNICODE is defined and RtosResultGetModuleA if not.  
The function does not require RtosLibInit to be called.

---

### 11.1.11.4 RtosResultGetModuleW

Get a module name for a result value.

**UINT32 RtosResultGetModuleW (**  
    **UINT32**        **dwResult,**  
    **WCHAR\***       **wszText,**  
    **UINT32\***      **pdwSizeInBytes)**

#### Parameter

*dwRteResult*  
    [in] Result value to query text for  
*wszText*  
    [out] String buffer to receive the result module name  
*pdwSizeInBytes*  
    [in] Length of string buffer in bytes  
    [out] Number of bytes used for module name

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

A module name for a result value can be queried using "RtosResultGetModule".  
The macro RtosResultGetModule can be used to call RtosResultGetModuleW if UNICODE is defined and RtosResultGetModuleA if not.  
The function does not require RtosLibInit to be called.

## 11.1.12 RTOS Library – licensing

The main licensing functionality is only supported through the Windows part of the RTOS Library. On the real-time parts of the RTOS Library some of these routines are not implemented.

### 11.1.12.1 RtosLicense

Setting, getting and removing of a license will be handled through this routine.

**UINT32 RtosLicense (**  
**UINT32        dwAction**  
**VOID\*        pvData**  
**UINT32\*      pdwSizeInBytes)**

#### Parameter

*dwAction*

[in] License action identifier. See comment for details.

*pvData*

[in] Supplied data (LicenseID) for a specific “set” action

[out] Requested data by specific “get” action

→ NULL if action = remove license

*pdwSizeInBytes*

[in] Size of supplied data for a specific “set” action

[out] Size of requested data for a specific “get” action

→ NULL if action = remove license

#### Return

RTE\_SUCCESS on success and an error-code on failure. Calling this routine on the real-time part of the RTOS Library, a RTE\_ERROR\_NOT\_IMPL will be returned.

#### Comment

This function performs actions for setting, getting and removing of a license.

dwAction	pvData	Description
RTOS_LICENSE_ACTION_SET_A	[IN] License string	Set license
RTOS_LICENSE_ACTION_SET_W	[IN] License string	As before, but UNICODE instead of ANSI
RTOS_LICENSE_ACTION_REMOVE	-	Remove license
RTOS_LICENSE_ACTION_GET_A	[OUT] License string	Get license
RTOS_LICENSE_ACTION_GET_W	[OUT] License string	As before, but UNICODE instead of ANSI
RTOS_LICENSE_ACTION_REQUESTFILE_A	[IN, OPT] Filename	Write request file
RTOS_LICENSE_ACTION_REQUESTFILE_W	[IN, OPT] Filename	As before, but UNICODE instead of ANSI
RTOS_LICENSE_ACTION_SETFILE_A	[IN] Filename	Read license from file
RTOS_LICENSE_ACTION_SETFILE_W	[IN] Filename	As before, but UNICODE instead of ANSI
RTOS_LICENSE_ACTION_GETFILE_A	[IN, OPT] Filename	Write license to file
RTOS_LICENSE_ACTION_GETFILE_W	[IN, OPT] Filename	As before, but UNICODE instead of ANSI
RTOS_LICENSE_ACTION_PARSE_A	[IN] Command + CmdData [OUT] see other [OUT]	Parse data for action and its params Actions: /set, /remove, /get, /requestfile, ...
RTOS_LICENSE_ACTION_PARSE_W	[IN] Command + CmdData [OUT] see other [OUT]	As before, but UNICODE instead of ANSI

#### Notes:

- 1) If parameter 2 is NULL parameter 3 must also be NULL - otherwise an error will be returned!
- 2) If not enough memory is supplied for the “get” action only a part of the license ID might be returned!
- 3) When using the “parse” action for “/get” the input buffer will also be used for the output. This requires the buffer size to be the maximum of input and output.
- 4) The RtosLicense parse action can be addressed using the Uploader parameter “/lic”.

Examples:

- RtosUpload.exe /lic "/requestfile 'C:\MyFile.HwldReq"
- RtosUpload.exe /lic "/setfile 'C:\MyFile.HwldLic"

### 11.1.13 RTOS Library – file server

Please check chapter 11.1.14 “RTOS Library – files” for configuration and further details.

---

#### 11.1.13.1 RtosFileServerStart

Start the file server for remote file handling.

#### UINT32 RtosFileServerStart (VOID)

##### Parameter

–

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

This function is called by default on

- Windows (by RtosService)

RtosFileServer is currently only supported by Windows.

---

#### 11.1.13.2 RtosFileServerStop

Stop the file server for remote file handling.

#### UINT32 RtosFileServerStop (VOID)

##### Parameter

–

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

This function is called by default on

- Windows (by RtosService)

RtosFileServer is currently only supported by Windows.

### 11.1.14 RTOS Library – files

The RtosLib supports file handling routines to access files on all supported platforms in a uniform way. The RtosLib files can operate in 2 different modes. The default mode accesses the files remotely on the file system of the Windows OS with a file server module of RtosService.

**HINT:** On CE or RTOS-32 platforms using the new native file system drivers should fits on the most cases and direct access to the RtosFile-API should not be needed.

In the other case the files are supplied via shared memory and acts mainly like memory-mapped files. This so-called SHM-mode has some restrictions which are discussed below.

**SHM File Mode:** To access a file through a shared memory, the file name path must contain the prefix `//SHM/<FILENAME>`.

The file name must correspond to the shared memory name (**case-sensitive!**)

Example:

```
SHM-Name:      TestFileXyZ.txt
SHM-FileName: //SHM/TestFileXyZ.txt
```

A corresponding shared memory must be defined in the config file. The memory can be initialized with zeros, without anything or with an existing file from the file system on Windows. For further details about shared memories prepared for file handling please look at chapter 6.7 Multi Purpose Shared Memory.

Example:

```
[SharedMemory\TestFileXyZ.txt]
  "Name"="TestFileXyZ.txt"
  "Description"="TestFile for SHM file access"
  "Size"=dword:0
  "File"="C:\Temp\TestFileForShmAccess.txt"
  "Initialize"=dword:2
  "Save"=dword:1
  "AccessDefault"=dword:1
[SharedMemory\TestFileXyZ.txt\AccessModes]
```

**Limitations:** The SHM file depends on a shared memory; therefore it's not possible to increase the size of a SHM file.

The initial size of the shared memory is the maximum possible file size!

Due to technical reasons it's not possible to rename or remove a SHM file with the RtosFile API!

**Remote File Mode:** Accessing a file remote is realized by the file server module of RtosService. The default operation directory of the file server is `%RTE_ROOT%\RtFiles`. If not yet available, it will be created at startup. All file operations are restricted to this directory and its subdirs by default.

The config file settings regarding the file server are OS specific and have to be made in an OS section like `[Windows\FileServer]` or `[Rtos\FileServer]`. Only a file server under Windows is supported at moment!

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
MaxOpenFiles	dword	Number of concurrently possible open RtosFiles. Default = 10.
HomeDir	string	The home directory of the file server. If no path is specified for a file, all file operations will go to this directory. If directory doesn't exist, it will be created at first access. Default: "%RTE_ROOT%\RtFiles"
RestrictedToHomeDir	dword	This value restricts the access of the file server to its home directory and its subdirs. Every file access outlying the home directory will return an error. Default: TRUE

<u>Entry Name</u>	<u>Type</u>	<u>Description</u>
TaskPriority	dword	This defines the task priority for the fileserver threads. The values are OS specific. The delivered config file contains OS specific information about lowest and highest possible priority. If the entry does not exist then the OS uses an OS specific priority compatible to older versions.
TaskEnabled	dword	0 Task will not be started 1 Task will be started omitted = 1 = Task will be started
SuppressAccessDeniedMsg	dword	0 'AccessDenied' message will be shown 1 Message will be suppressed omitted = 0 = Message will be shown
LogLevel	dword	0 No (internal) log messages of the file server will be logged. 1 Internal log messages of file server will be logged at the Windows event log. omitted = 0 = no log messages will be logged

**Limitations:** Currently 10 simultaneously open remote files are supported. This value could be adjusted with corresponding entry MaxOpenFiles in config file. See above table.  
Scan functionality is currently not available for remote files.

**Caution:** The file handles of RtosFile 6.0 API and RtosFile 6.1 API could not be mixed generally, if not stated otherwise.

Generic functions like RtosFileClose, RtosFileRead, RtosFileWrite, etc can operate on various kinds of handles.

#### Ways of accessing the file API:

For easier access to the RtosFile API, 3 different ways are supported. For some RTOS's (CE- or RTOS-32-based) we provide native file system driver, which will call the appropriate RtosFile functions and second calling the RtosFile API directly or as (partly deprecated) third method through supplied rtostdio.h.

On CE- or RTOS-32-based products the default case is using the supplied native file system drivers, which will hide the complete RtosFile-API and native fopen() or CreateFile() calls can be used.

The following rtostdio.h example should be considered deprecated (on CE- and RTOS-32-based projects) and should be only used for legacy reasons on these platforms.

Porting to native stdio.h on these platforms should be possible without much hassle.

The rtostdio.h remaps default stdio calls like fopen() to the native RtosFile API. To use the remapping functionality of rtostdio.h the include file must be added into the source file as the last include.

Minimal example:

```
// other includes...
// #include ...

// last include:
#include <rtostdio.h>

int main(int argc, char *argv[])
{
    FILE* hFile = NULL;

    RtosLibInit();

    // Create file handle
    hFile = fopen( "//SHM/TestFile.txt", "rw" );
    if( NULL == hFile )
```

```

{
    // error
}

...

if( NULL != hFile )
{
    if( fclose( hFile ) != 0 )
    {
        // error
    }
}

RtosLibDeinit();

return 0;
}

```

The following functions are supported:

```

fopen(); fclose();
fwrite(); fread(); fputs(); fgets(); fputc(); fgetc(); getc(); putc();
fflush(); feof(); rewind(); fseek(); fsetpos(); fgetpos(); ftell();
ferror(); clearerr();
setbuf(); setvbuf();
fprintf(); vfprintf();
fscanf(); vfscanf();

```

Supported only by remote file mode (**not** available in SHM mode)

```

tmpfile(); remove(); rename();

```

The following functions are **NOT** supported:

```

freopen(); ungetc(); tmpnam();

```

For Debugging and/or Logging purposes the `rtosstdio.h` exposes the underlying returned error code through a global error variable:

```

/* To be used in code to determine the underling RTE_ERROR if the function
call fails. */
extern UINT32 rte_errno = RTE_SUCCESS;

```

---

### 11.1.14.1 RtosFileCreateA

This is the main function to open an existing file or create a new file. Both remote files as well as SHM files are supported.

```
UINT32 RtosFileCreateA (  
    const CHAR*      szName,  
    const CHAR*      szOptions,  
    RTOSLIB_HANDLE*  phFileCreate )
```

#### Parameter

*szName*  
[in] Name of the file

*szOptions*  
[in] File create options. See comment.

*phFileCreate*  
[out] Pointer to receive a RTOSLIB\_HANDLE object (file).

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

"r"	<b>read:</b> Open file for read operations. The file must exist.
"w"	<b>write:</b> Create an empty file for write operations. If the file yet exists, its contents will be discarded and the file is used like a fresh new file.
"a"	<b>append:</b> Open the file for write operations at the end of file, expanding it. Reposition operations (RtosFileSeek, RtosFileRewind, etc) are ignored. If the file doesn't exist, it will be created.
"r+"	<b>read/update:</b> Open file for read/write operations. The file must exist.
"w+"	<b>write/update:</b> Create an empty file for read/write operations. If the file yet exists, its contents will be discarded and the file is used like a fresh new file.
"a+"	<b>append/update:</b> Open the file for read/write operations with write operations at the end of the file, expanding it. Reposition operations (RtosFileSeek, RtosFileRewind, etc) affects only the following read operations, but write operations move the position back to the end of file. If the file doesn't exist, it will be created.
"b"	<b>binary:</b> File should be open as binary file.
"t"	<b>text:</b> File should be open as text file.

The first character of the option string must be either "r", "w" or "a". The next and further character could be one of the optional "b" or "t" or sign "+" with resulting compound modes: "rb", "rt", "wb", "wt", "ab", "at" and "r+b", "r+t", "w+b", "w+t", "a+b", "a+t" or "rb+", "rt+", "wb+", "wt+", "ab+", "at".

With errors in format string the function will fail with RTE\_ERROR\_FILE\_INVALID\_CREATE\_OPTS (0x4461).

The macro RtosFileCreate can be used to call RtosFileCreateW if UNICODE is defined and RtosFileCreateA if not.

---

### 11.1.14.2 RtosFileCreateW

See RtosFileCreateA

```
UINT32 RtosFileCreateW (  
    const WCHAR*      wszName,  
    const WCHAR*      wszOptions,  
    RTOSLIB_HANDLE*  phFileCreate )
```

---

### 11.1.14.3 RtosFileClose

This is a synonym for RtosObjectClose. See RtosObjectClose for details.

**UINT32 RtosFileClose (**  
**RTOSLIB\_HANDLE            hObject )**

---

### 11.1.14.4 RtosFileRead

This is a synonym for RtosObjectRead. See RtosObjectRead for details.

**UINT32 RtosFileRead(**  
**RTOSLIB\_HANDLE            hObject,**  
**UINT8                      \*pBuffer,**  
**UINT32                     dwBufferSize,**  
**UINT32                     \*pdwBytesRead,**  
**UINT32                     dwTimeoutMs )**

---

### 11.1.14.5 RtosFileWrite

This is a synonym for RtosObjectWrite. See RtosObjectWrite for details.

**UINT32 RtosFileWrite(**  
**RTOSLIB\_HANDLE            hObject,**  
**const UINT8                \*pBuffer,**  
**UINT32                     dwBufferSize,**  
**UINT32                     \*pdwBytesWritten,**  
**UINT32                     dwTimeoutMs )**

---

### 11.1.14.6 RtosFileReadStrA

This function read "string" data into an ANSI string.

**UINT32 RtosFileReadStrA(**  
**RTOSLIB\_HANDLE            hFileCreate,**  
**CHAR                        \*szStr,**  
**UINT32                     dwNum,**  
**UINT32                     \*pdwNumOfCharRead )**

#### Parameter

*hFileCreate*  
[in]     Object handle returned from RtosFileCreate()  
*szStr*  
[out]    String buffer to receive the data read  
*dwNum*  
[in]     Length of string buffer in characters  
*pdwNumOfCharRead*  
[out]    Number of characters read into the string buffer

#### Return

RTE\_SUCCESS on success and an error-code on failure.

If the return value is RTE\_SUCCESS but \*pdwNumOfCharRead returned zero, then the stream should be checked for eof. In that case the szStr remains unchanged.

#### Comment

If there is no data to be read the function will return and the \*pdwNumOfCharRead will contain zero.

If data was read the function will return as soon as (dwNum-1) characters have been read or either a newline or the end-of-file is reached or the read string buffer is full, whichever happens first.

A newline character is considered as a valid character and is included into the returned string.

A terminating null character is appended automatically.

---

#### 11.1.14.7 RtosFileReadStrW

See RtosFileReadStrA.

```
UINT32 RtosFileReadStrW(
    RTOSLIB_HANDLE    hFileCreate,
    WCHAR             *wszStr,
    UINT32            dwNum,
    UINT32            *pdwNumOfCharRead )
```

---

#### 11.1.14.8 RtosFileFlush

This function performs a flush on the remote file.

```
UINT32 RtosFileFlush (
    RTOSLIB_HANDLE    hFile )
```

##### Parameter

*hFile*

[in] Object handle returned from RtosFileCreate() or RtosFileCreateEx()

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

This function is only supported on the remote file mode. It performs a flush on the real file at the file server.

---

#### 11.1.14.9 RtosFileEof

This function returns either the RTOSFILE\_EOF indicator, if end of file is reached or otherwise 0.

**UINT32 RtosFileEof (**  
    **RTOSLIB\_HANDLE**      **hFileCreate,**  
    **INT32\***              **pnEof )**

**Parameter**

*hFileCreate*

[in] Object handle returned from RtosFileCreate()

*pnEof*

[out] Pointer to receive the RTOSFILE\_EOF indicator.

**Return**

RTE\_SUCCESS on success and an error-code on failure.

**Comment**

Check returned \*pnEof value with define RTOSFILE\_EOF to indicate, if EoF is reached.

---

#### 11.1.14.10 RtosFileRewind

This function performs a rewind at the file. It sets the position indicator of the file to the beginning.

**UINT32 RtosFileRewind (**  
    **RTOSLIB\_HANDLE**      **hFileCreate )**

**Parameter**

*hFileCreate*

[in] Object handle returned from RtosFileCreate()

**Return**

RTE\_SUCCESS on success and an error-code on failure.

**Comment**

-

---

#### 11.1.14.11 RtosFileSeek

This function returns either the RTOSFILE\_EOF indicator, if end of file is reached or otherwise 0.

**UINT32 RtosFileSeek (**  
    **RTOSLIB\_HANDLE**        **hFileCreate,**  
    **INT64**                  **qnOffset,**  
    **INT32**                  **nOrigin )**

#### Parameter

*hFileCreate*  
    [in] Object handle returned from RtosFileCreate()  
*qnOffset*  
    [in] Sets the position indicator relative to this offset from origin parameter.  
*nOrigin*  
    [in] Defines the origin for the offset. Valid values see comment.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

Origin ID	Description
RTOSFILE_SEEK_SET	Sets the position indicator to the beginning of the file. If qnOffset has a positive value, the position indicator will be set to this offset from the beginning.
RTOSFILE_SEEK_CUR	Uses the qnOffset parameter to move position indicator from current position of the offset value.
RTOSFILE_SEEK_END	Sets the position indicator to the end of the file. The end of file means after the last char. In <b>SHM file mode</b> , this means, that the position indicator is beyond the valid memory range! If qnOffset has a negative value, the position indicator will be set to this offset from the end of file towards to the beginning.

---

#### 11.1.14.12 RtosFileGetPos

This function returns the current position in file.

**UINT32 RtosFileGetPos (**  
    **RTOSLIB\_HANDLE**        **hFileCreate,**  
    **INT64\***                 **pqnPos )**

#### Parameter

*hFileCreate*  
    [in] Object handle returned from RtosFileCreate()  
*pqnPos*  
    [out] Pointer to receive the current position indicator.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

-

---

#### 11.1.14.13 RtosFileSetPos

This function sets the current position in file.

**UINT32 RtosFileSetPos (**  
    **RTOSLIB\_HANDLE           hFileCreate,**  
    **INT64                   qnPos )**

**Parameter**

*hFileCreate*  
    [in]     Object handle returned from RtosFileCreate()  
*qnPos*  
    [in]     Desired value of position indicator.

**Return**

RTE\_SUCCESS on success and an error-code on failure.

**Comment**

-

---

#### 11.1.14.14 RtosFileSize

This function returns the current size of file.

**UINT32 RtosFileSize (**  
    **RTOSLIB\_HANDLE           hFile,**  
    **UINT64\*                  pqwSize )**

**Parameter**

*hFile*  
    [in]     Object handle returned from RtosFileCreate() or RtosFileCreateEx()  
*pqwSize*  
    [out]    Pointer to receive the current file size.

**Return**

RTE\_SUCCESS on success and an error-code on failure.

**Comment**

-

---

#### 11.1.14.15 RtosFileError

This function returns the current error indicator of file.

```
UINT32 RtosFileError (
    RTOSLIB_HANDLE      hFileCreate,
    INT64*              pqnError )
```

##### Parameter

*hFileCreate*  
[in] Object handle returned from RtosFileCreate()  
*pqnError*  
[out] Pointer to receive the current error indicator.

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

-

---

#### 11.1.14.16 RtosFileClearError

This function performs resets to both the error and the eof indicator of the file.

```
UINT32 RtosFileClearError (
    RTOSLIB_HANDLE      hFileCreate )
```

##### Parameter

*hFileCreate*  
[in] Object handle returned from RtosFileCreate()

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

-

---

#### 11.1.14.17 RtosFileCreateTmpfile

Creates a temporary binary file with a filename guaranteed to be different from any other existing file.

```
UINT32 RtosFileCreateTmpfile (
    RTOSLIB_HANDLE*     phFileCreate )
```

##### Parameter

*phFileCreate*  
[out] Pointer for receive a RTOSLIB\_HANDLE object (file) of a temporary file.

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

This function is not supported on SHM file mode!  
Closing this file handle with RtosFileClose() will delete the temporary file automatically.

---

#### 11.1.14.18 RtosFileRemoveA

Removes a file with the specified name.

**UINT32 RtosFileRemoveA (**  
**const CHAR\* szName )**

#### Parameter

*szName*  
[in] Name of the file to be removed

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The macro RtosFileRemove can be used to call RtosFileRemoveW if UNICODE is defined and RtosFileRemoveA if not.

This function is not supported on SHM file mode!

---

#### 11.1.14.19 RtosFileRemoveW

See RtosFileRemoveA

**UINT32 RtosFileRemoveW (**  
**const WCHAR\* wszName )**

---

#### 11.1.14.20 RtosFileRenameA

Rename a file with the specified name.

**UINT32 RtosFileRenameA (**  
**const CHAR\* szOldName,**  
**const CHAR\* szNewName )**

#### Parameter

*szOldName*  
[in] Name of the file to be renamed  
*szNewName*  
[in] New name of the file

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The macro RtosFileRename can be used to call RtosFileRenameW if UNICODE is defined and RtosFileRenameA if not.

This function is not supported on SHM file mode!

---

#### 11.1.14.21 RtosFileRenameW

See RtosFileRenameA

**UINT32 RtosFileRenameW (**  
**const WCHAR\* wszOldName,**  
**const WCHAR\* wszNewName )**

---

#### 11.1.14.22 RtosFileSetBuffer

This function sets a local buffer to the file. At moment not supported.

```
UINT32 RtosFileSetBuffer (
    RTOSLIB_HANDLE      hFileCreate,
    const UINT8*        pBuffer,
    UINT64               qwBufferSize
    INT32               nMode )
```

##### Parameter

*hFileCreate*  
[in] Object handle returned from RtosFileCreate()

*pBuffer*  
[in] Pointer to buffer to be set.

*qwBufferSize*  
[in] Size of the supplied buffer.

*nOrigin*  
[in] Operation mode. Currently don't care.

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

This function is currently not supported. It will remember the buffer, but no further action will be taken.

---

#### 11.1.14.23 RtosFilePrintfA

Works similar to the classic fprintf() function.

```
UINT32 RtosFilePrintfA (
    RTOSLIB_HANDLE      hFile,
    CHAR*               szFormat,
    UINT64*             pqwCharsWritten,
    ... )
```

##### Parameter

*hFile*  
[in] Object handle returned from RtosFileCreate() or RtosFileCreateEx()

*szFormat*  
[in] Format string. All Visual C fprintf() format sequences are supported.

*pqwCharsWritten*  
[out] Pointer to receive how many chars was written.

...  
[in] 0..n additional parameter for the format entries.

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

This function is comparable with the ANSI fprintf() function.  
The macro RtosFilePrintf can be used to call RtosFilePrintfW if UNICODE is defined and RtosFilePrintfA if not.

---

#### 11.1.14.24 RtosFilePrintfW

See RtosFilePrintfA

**UINT32 RtosFilePrintfW (**  
    **RTOSLIB\_HANDLE**        **hFile,**  
    **WCHAR\***                **wszFormat,**  
    **UINT64\***                **pqwCharsWritten,**  
    **... )**

---

#### 11.1.14.25 RtosFileVPrintfA

This function is similar to RtosFilePrintfA . The main difference is the last parameter. This function expects the parameters for the format string as a va\_list. . For further details see RtosFilePrintfA.

**UINT32 RtosFileVPrintfA (**  
    **RTOSLIB\_HANDLE**        **hFile,**  
    **CHAR\***                  **szFormat,**  
    **UINT64\***                **pqwCharsWritten,**  
    **va\_list**                 **vaList )**

---

#### 11.1.14.26 RtosFileVPrintfW

See RtosFileVPrintfA.

**UINT32 RtosFileVPrintfW (**  
    **RTOSLIB\_HANDLE**        **hFile,**  
    **WCHAR\***                **wszFormat,**  
    **UINT64\***                **pqwCharsWritten,**  
    **va\_list**                 **vaList )**

---

#### 11.1.14.27 RtosFileScanfA

Works similar to the classic fscanf() function.

```
UINT32 RtosFileScanfA (
    RTOSLIB_HANDLE      hFileCreate,
    CHAR*               szFormat,
    UINT64*              pqwItemsRead,
    ... )
```

#### Parameter

*hFileCreate*

[in] Object handle returned from RtosFileCreate()

*szFormat*

[in] Format string. All Visual C fscanf() format sequences are supported.

*pqwItemsRead*

[out] Pointer to receive how many items could be read.

...

[in/out] 0..n additional parameter for the format entries.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

This function is comparable with the ANSI fscanf() function.

The macro RtosFileScanf can be used to call RtosFileScanfW if UNICODE is defined and RtosFileScanfA if not.

**Important:** This function is currently **not** supported on remote files!  
In SHM file mode are up to 20 items supported.

---

#### 11.1.14.28 RtosFileScanfW

See RtosFileScanfA

```
UINT32 RtosFileScanfW (
    RTOSLIB_HANDLE      hFileCreate,
    WCHAR*              wszFormat,
    UINT64*              pqwItemsRead,
    ... )
```

---

#### 11.1.14.29 RtosFileVScanfA

This function is similar to RtosFileScanfA . The main difference is the last parameter. This function expects the parameters for the format string as a va\_list. . For further details see RtosFileScanfA.

```
UINT32 RtosFileVScanfA (
    RTOSLIB_HANDLE      hFileCreate,
    CHAR*               szFormat,
    UINT64*              pqwItemsRead,
    va_list              vaList )
```

---

#### 11.1.14.30 RtosFileVScanfW

See RtosFileVScanfA.

**UINT32 RtosFileVScanfW (**  
    **RTOSLIB\_HANDLE**        **hFileCreate,**  
    **WCHAR\***                **wszFormat,**  
    **UINT64\***                **pqwltemsRead,**  
    **va\_list**                **vaList )**

---

#### 11.1.14.31 RtosFileOptionSet

This is a synonym for RtosObjectOptionSet. See RtosObjectOptionSet for details.

**UINT32 RtosFileOptionSet (**  
    **RTOSLIB\_HANDLE**        **hObject,**  
    **UINT32**                  **dwOptionId,**  
    **const UINT8**            **\*pBuffer,**  
    **UINT32**                  **dwBufferSize )**

---

#### 11.1.14.32 RtosFileOptionGet

This is a synonym for RtosObjectOptionGet. See RtosObjectOptionGet for details.

**UINT32 RtosFileOptionGet (**  
    **RTOSLIB\_HANDLE**        **hObject,**  
    **UINT32**                  **dwOptionId,**  
    **const UINT8**            **\*pBuffer,**  
    **UINT32**                  **dwBufferSize )**

### 11.1.15 RTOS Library – files advanced (6.1)

The new file functions from Rte 6.1 on uses the Windows CreateFile()-API on the file server side. Therefore the returned handles are generally not compatible with Rte 6.0 file handles (fopen()-based) and could not be mixed, if not stated otherwise.

Generic functions like Close, Read, Write, etc applies to all kind of handles.

**General rule:** the name of the handle suggests, which handle are allowed.

See following table:

Parameter Name	Allowed Handles
hObject	All handle types allowed
hFile	hFileCreate or hFileCreateEx handles allowed
hFileCreate	Only hFileCreate handles allowed
hFileCreateEx	Only hFileCreateEx handles allowed
hFind	Only hFind handles allowed

**Hint:** As most of the following functions have equivalent Windows function calls on the file server side, the main limitation of these native Windows calls applies also to these functions. Please see the functional equivalent function descriptions at the MSDN.

#### 11.1.15.1 RtosFileCreateExA

This is the main function to open an existing file or create a new file. SHM files are NOT supported.

```

UINT32 RtosFileCreateExA (
    const CHAR*                szFileName,
    UINT32                    dwDesiredAccess,
    UINT32                    dwShareMode,
    RTOSLIB_SECURITY_ATTRIBUTES lpSecurityAttributes,
    UINT32                    dwCreationDisposition,
    UINT32                    dwFlagsAndAttributes,
    RTOSLIB_HANDLE            hTemplateFile,
    RTOSLIB_HANDLE*          phFileCreateEx )

```

#### Parameter

*szFileName*  
[in] Name of the file

*dwDesiredAccess*  
[in] Desired access mode. See comment.

*dwShareMode*  
[in] Shared mode. See comment.

*lpSecurityAttributes*  
[in] Currently only NULL is supported/accepted.

*dwCreationDisposition*  
[in] Creation Disposition. See comment.

*dwFlagsAndAttributes*  
[in] File create flags and attributes. See comment.

*hTemplateFile*  
[in] Currently only NULL is supported/accepted.

*phFileCreateEx*  
[out] Pointer to receive a RTOSLIB\_HANDLE object (file).

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

Desired access modes:

Value	Meaning
RTOSFILE_GENERIC_READ (0x80000000L)	Read access

RTOSFILE_GENERIC_WRITE	(0x40000000L)	Write access
RTOSFILE_GENERIC_EXECUTE	(0x20000000L)	Execute access
RTOSFILE_GENERIC_ALL	(0x10000000L)	All possible access rights

The most commonly used values are RTOSFILE\_GENERIC\_READ, RTOSFILE\_GENERIC\_WRITE, or both (RTOSFILE\_GENERIC\_READ | RTOSFILE\_GENERIC\_WRITE).

File shared mode:

Value	Meaning
0 (0x00000000)	Prevents other processes from opening a file or device if they request delete, read, or write access.
RTOSFILE_SHARE_READ (0x00000001)	Enables subsequent open operations on a file or device to request read access.
RTOSFILE_SHARE_WRITE (0x00000002)	Enables subsequent open operations on a file or device to request write access.
RTOSFILE_SHARE_DELETE (0x00000004)	Enables subsequent open operations on a file or device to request delete access.

File attributes:

Value	Meaning
RTOSFILE_ATTRIBUTE_READONLY (0x00000001)	The file is read only. Applications can read the file, but cannot write to or delete it.
RTOSFILE_ATTRIBUTE_HIDDEN (0x00000002)	The file is hidden. Do not include it in an ordinary directory listing.
RTOSFILE_ATTRIBUTE_SYSTEM (0x00000004)	The file is part of or used exclusively by an operating system.
RTOSFILE_ATTRIBUTE_DIRECTORY (0x00000010)	The handle that identifies a directory.
RTOSFILE_ATTRIBUTE_ARCHIVE (0x00000020)	The file should be archived. Applications use this attribute to mark files for backup or removal.
RTOSFILE_ATTRIBUTE_DEVICE (0x00000040)	This value is reserved for system use.
RTOSFILE_ATTRIBUTE_NORMAL (0x00000080)	The file does not have other attributes set. This attribute is valid only if used alone.
RTOSFILE_ATTRIBUTE_TEMPORARY (0x00000100)	The file is being used for temporary storage.
RTOSFILE_ATTRIBUTE_SPARSE_FILE (0x00000200)	A file that is a sparse file.
RTOSFILE_ATTRIBUTE_REPARSE_POINT (0x00000400)	A file or directory that has an associated reparse point, or a file that is a symbolic link.
RTOSFILE_ATTRIBUTE_COMPRESSED (0x00000800)	A file or directory that is compressed.
RTOSFILE_ATTRIBUTE_OFFLINE (0x00001000)	The data of a file is not immediately available.
RTOSFILE_ATTRIBUTE_NOT_CONTENT_INDEXED (0x00002000)	The file or directory is not to be indexed by the content indexing service.
RTOSFILE_ATTRIBUTE_ENCRYPTED (0x00004000)	The file or directory is encrypted.
RTOSFILE_ATTRIBUTE_VIRTUAL (0x00010000)	This value is reserved for system use.

File flags:

Value	Meaning
RTOSFILE_FLAG_WRITE_THROUGH (0x80000000)	Write operations will not go through any intermediate cache; they will go directly to disk.
RTOSFILE_FLAG_NO_BUFFERING (0x20000000)	The file or device is being opened with no system caching for data reads and writes.

RTOSFILE_FLAG_RANDOM_ACCESS (0x10000000)	Access is intended to be random. The system can use this as a hint to optimize file caching.
RTOSFILE_FLAG_SEQUENTIAL_SCAN (0x08000000)	Access is intended to be sequential from beginning to end. The system can use this as a hint to optimize file caching.
RTOSFILE_FLAG_DELETE_ON_CLOSE (0x04000000)	The file is to be deleted immediately after all of its handles are closed, which includes the specified handle and any other open or duplicated handles.
RTOSFILE_FLAG_BACKUP_SEMANTICS (0x02000000)	The file is being opened or created for a backup or restore operation.
RTOSFILE_FLAG_POSIX_SEMANTICS (0x01000000)	Access will occur according to POSIX rules.
RTOSFILE_FLAG_SESSION_AWARE (0x00800000)	The file or device is being opened with session awareness.
RTOSFILE_FLAG_OPEN_REPARSE_POINT (0x00200000)	Normal reparse point processing will not occur; RtosFileCreateEx will attempt to open the reparse point.
RTOSFILE_FLAG_OPEN_NO_RECALL (0x00100000)	The file data is requested, but it should continue to be located in remote storage.

**Hint:**

If the FILE\_FLAG\_NO\_BUFFERING flag is active, then file access buffer addresses for read and write operations should be physical sector-aligned, which means aligned on addresses in memory that are integer multiples of the volume's physical sector size.

**File creation disposition:**

Value	Meaning
RTOSFILE_CREATE_NEW (1)	Creates a new file, only if it does not already exist.
RTOSFILE_CREATE_ALWAYS (2)	Creates a new file, always.
RTOSFILE_OPEN_EXISTING (3)	Opens a file or device, only if it exists.
RTOSFILE_OPEN_ALWAYS (4)	Opens a file, always.
RTOSFILE_TRUNCATE_EXISTING (5)	Opens a file and truncates it so that its size is zero bytes, only if it exists.

---

### 11.1.15.2 RtosFileCreateExW

See RtosFileCreateExA

#### UINT32 RtosFileCreateExA (

const WCHAR*	wszFileName,
UINT32	dwDesiredAccess,
UINT32	dwShareMode,
RTOS_SECURITY_ATTRIBUTES	lpSecurityAttributes,
UINT32	dwCreationDisposition,
UINT32	dwFlagsAndAttributes,
RTOSLIB_HANDLE	hTemplateFile,
RTOSLIB_HANDLE*	phFileCreateEx )

---

### 11.1.15.3 RtosFileReadSeek

This is mainly the same as RtosObjectRead. It contains an additional offset parameter for the desired seek position. See RtosObjectRead for further details.

UINT32 RtosFileRead(  
 RTOSLIB\_HANDLE hFileCreateEx,

UINT8 \*pBuffer,  
UINT32 dwBufferSize,  
UINT32 \*pdwBytesRead,  
UINT32 dwTimeoutMs,  
UINT64 qwOffset )

**Additional Parameter**

*qwOffset*  
[in] Desired seek position from which the data should be read. The offset starts from file begin.

**Return**

RTE\_SUCCESS on success and an error-code on failure.

**Comment**

-

---

**11.1.15.4 RtosFileWriteSeek**

This is mainly the same as RtosObjectWrite. It contains an additional offset parameter for the desired seek position. See RtosObjectWrite for further details.

UINT32 RtosFileWrite(  
RTOSLIB\_HANDLE hFileCreateEx,  
const UINT8 \*pBuffer,  
UINT32 dwBufferSize,  
UINT32 \*pdwBytesWritten,  
UINT32 dwTimeoutMs,  
UINT64 qwOffset )

**Additional Parameter**

*qwOffset*  
[in] Desired seek position from were the data should be write. The offset starts from file begin.

**Return**

RTE\_SUCCESS on success and an error-code on failure.

**Comment**

-

---

**11.1.15.5 RtosFileSetEndOfFile**

This function set the current file pointer to the end of file.

UINT32 RtosFileEof (  
RTOSLIB\_HANDLE hFileCreateEx )

**Parameter**

*hFileCreateEx*  
[in] Object handle returned from RtosFileCreateEx()

**Return**

RTE\_SUCCESS on success and an error-code on failure.

**Comment**

-

---

### 11.1.15.6 RtosFileSetFilePointer

This function sets the file pointer on the desired location.

```
UINT32 RtosFileSetFilePointer (
    RTOSLIB_HANDLE      hFileCreateEx,
    INT64*              pqnDistanceToMove,
    UINT32              dwMoveMethod )
```

#### Parameter

*hFileCreateEx*

[in] Object handle returned from RtosFileCreateEx()

*pqnDistanceToMove*

[in/out] Value that specifies the number of bytes to move the file pointer.

*dwMoveMethod*

[in] The starting point for the file pointer move. See comment.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The following starting points are defined:

Value	Meaning
RTOSFILE_BEGIN (0)	The starting point is zero or the beginning of the file.
RTOSFILE_CURRENT (1)	The starting point is the current value of the file pointer.
RTOSFILE_END (2)	The starting point is the current end-of-file position.

---

### 11.1.15.7 RtosFileGetInformationByHandle

This function gets information about the file described by its handle.

```
UINT32 RtosFileGetInformationByHandle (
    RTOSLIB_HANDLE      hFileCreateEx,
    PRRTOSFILE_BY_HANDLE_INFORMATION pFileInfo)
```

#### Parameter

*hFileCreateEx*

[in] Object handle returned from RtosFileCreateEx()

*pFileInfo*

[out] Collected file information. See comment.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The following informations will be collected:

```
typedef struct _RTOSFILE_BY_HANDLE_INFORMATION {
    UINT64 qwCreationTime;      Value that specifies when a file or
                                directory was created.
    UINT64 qwLastAccessTime;   For a file, the value specifies when the
                                file was last read from, written to, or
                                for executable files, run.
                                For a directory, the structure specifies
                                when the directory is created.
```

<pre> UINT64 qwLastWriteTime; UINT64 qwFileSize; UINT64 qwFileIndex; UINT32 dwFileAttributes; UINT32 dwVolumeSerialNumber; UINT32 dwNumberOfLinks; UINT32 dwReservedFor64BitAlignment; } RTOSLIB_PACKED(8) RTOSFILE_BY_HANDLE_INFORMATION, *RTOSFILE_BY_HANDLE_INFORMATION; </pre>	<p>For a file, the value specifies when the file was last written to, truncated, or overwritten.</p> <p>For a directory, the value specifies when the directory is created.</p> <p>File size in bytes.</p> <p>A unique identifier that is associated with a file.</p> <p>The file attributes of a file. → see RtosFileCreateEx() comment.</p> <p>The serial number of the volume that contains a file.</p> <p>The number of links to this file.</p>
--	---

### 11.1.15.8 RtosFileGetTime

This function gets the file time.

```

UINT32 RtosFileGetTime (
    RTOSLIB_HANDLE          hFileCreateEx,
    UINT64*                 pqwCreationTime,
    UINT64*                 pqwLastAccessTime,
    UINT64*                 pqwLastWriteTime)

```

#### Parameter

*hFileCreateEx*  
[in] Object handle returned from RtosFileCreateEx()

*pqwCreationTime*  
[out] Get the file creation time.

*pqwLastAccessTime*  
[out] Get the file last access time.

*pqwLastWriteTime*  
[out] Get the file last write time.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

If a value is not desired, a NULL could be supplied.

### 11.1.15.9 RtosFileSetTime

This function sets the file time.

```

UINT32 RtosFileSetTime (
    RTOSLIB_HANDLE          hFileCreateEx,
    UINT64*                 pqwCreationTime,
    UINT64*                 pqwLastAccessTime,
    UINT64*                 pqwLastWriteTime)

```

#### Parameter

*hFileCreateEx*  
[in] Object handle returned from RtosFileCreateEx()

*pqwCreationTime*

[in] Set the file creation time, if not NULL.  
*pqwLastAccessTime*  
[in] Set the file last access time, if not NULL.  
*pqwLastWriteTime*  
[in] Set the file last write time, if not NULL.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

-

---

#### 11.1.15.10 RtosFileGetTempNameA

Get temp file name based on the values of the parameter.

**UINT32 RtosFileGetTempNameA (**  
    **const CHAR\***            **szPathName,**  
    **const CHAR\***            **szPrefixString,**  
    **UINT32\***                **pdwUnique,**  
    **CHAR\***                    **szTempFileName )**

#### Parameter

*szPathName*

[in] The directory path for the file name. Applications typically specify a period (.) for the current directory. The string cannot be longer than RTOSFILE\_MAX\_PATH-14 characters and not NULL.

*szPrefixString*

[in] The function uses up to the first three characters of this string as the prefix of the file name.

*pdwUnique*

[in/out] An unsigned integer to be used in creating the temporary file name.

*szTempFileName*

[out] Generated temporary file name.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The macro RtosFileGetTempName can be used to call RtosFileGetTempNameW if UNICODE is defined and RtosFileGetTempNameA if not.

If \*pdwUnique is zero, the function attempts to form a unique file name using the current system time.

If the function succeeds, the returned value in \*pdwUnique specifies the unique numeric value used in the temporary file name. If the \*pdwUnique parameter is nonzero, the return value specifies that same number.

If the function fails, the returned value in \*pdwUnique is zero.

---

#### 11.1.15.11 RtosFileGetTempNameW

See RtosFileGetTempNameA.

**UINT32 RtosFileGetTempNameW (**  
    **const WCHAR\***            **wszPathName,**  
    **const WCHAR\***            **wszPrefixString,**  
    **UINT32\***                **pdwUnique,**  
    **WCHAR\***                    **wszTempFileName )**

---

#### 11.1.15.12 RtosFileMoveA

Move a file with the specified name (across directories).

**UINT32 RtosFileMoveA (**  
    **const CHAR\***                    **szExistingFilename,**  
    **const CHAR\***                    **szNewFilename)**

##### Parameter

*szExistingFilename*

[in] Name of the file to be moved

*szNewFilename*

[in] New name and/or location of the file

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

The macro RtosFileMove can be used to call RtosFileMoveW if UNICODE is defined and RtosFileMoveA if not.

This function is not supported on SHM file mode!

---

#### 11.1.15.13 RtosFileMoveW

See RtosFileMoveA

**UINT32 RtosFileMoveW (**  
    **const WCHAR\***                    **wszExistingFilename,**  
    **const WCHAR\***                    **wszNewFilename )**

---

#### 11.1.15.14 RtosFileGetAttributesA

Get the file attributes.

**UINT32 RtosFileGetAttributesA (**  
    **const CHAR\***                    **szFileName,**  
    **UINT32\***                        **pdwFileAttributes)**

##### Parameter

*szFileName*

[in] Name of the file which attributes are desired

*pdwFileAttributes*

[out] File attributes. See RtosFileCreateEx() comment.

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

The macro RtosFileGetAttributes can be used to call RtosFileGetAttributesW if UNICODE is defined and RtosFileGetAttributesA if not.

This function is not supported on SHM file mode!

---

#### 11.1.15.15 RtosFileGetAttributesW

See RtosFileGetAttributesA

**UINT32 RtosFileGetAttributesW (**  
    **const WCHAR\***        **wszFileName,**  
    **UINT32\***            **pdwFileAttributes)**

---

#### 11.1.15.16 RtosFileSetAttributesA

Set the file attributes.

**UINT32 RtosFileSetAttributesA (**  
    **const CHAR\***        **szFileName,**  
    **UINT32**            **dwFileAttributes)**

#### Parameter

*szFileName*

[in] Name of the file which attributes should be set.

*dwFileAttributes*

[in] New file attributes. See RtosFileCreateEx() comment.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The macro RtosFileSetAttributes can be used to call RtosFileSetAttributesW if UNICODE is defined and RtosFileSetAttributesA if not.

This function is not supported on SHM file mode!

---

#### 11.1.15.17 RtosFileSetAttributesW

See RtosFileSetAttributesA

**UINT32 RtosFileSetAttributesW (**  
    **const WCHAR\***        **wszFileName,**  
    **UINT32**            **dwFileAttributes)**

---

#### 11.1.15.18 RtosFileGetDiskFreeSpaceA

Get the available free disk space.

**UINT32 RtosFileGetDiskFreeSpaceA (**  
    **const CHAR\***        **szRootPathName,**  
    **UINT32\***            **pdwSectorsPerCluster,**  
    **UINT32\***            **pdwBytesPerSector,**  
    **UINT32\***            **pdwNumberOfFreeClusters,**  
    **UINT32\***            **pdwTotalNumberOfClusters )**

#### Parameter

*szFile Name*

[in] The root directory of the disk for which information is to be returned.

*pdwSectorsPerCluster*

[out] A pointer to a variable that receives the number of sectors per cluster.

*pdwBytesPerSector*

[out] A pointer to a variable that receives the number of bytes per sector.

*pdwNumberOfFreeClusters*

[out] A pointer to a variable that receives the total number of free clusters on the disk.

*pdwTotalNumberOfClusters*

[out] A pointer to a variable that receives the total number of clusters on the disk.

#### Return

28.03.2019

Page 131/205

RTE\_SUCCESS on success and an error-code on failure.

### Comment

The macro RtosFileGetDiskFreeSpace can be used to call RtosFileGetDiskFreeSpaceW if UNICODE is defined and RtosFileGetDiskFreeSpaceA if not.

This function is not supported on SHM file mode!

---

#### 11.1.15.19 RtosFileGetDiskFreeSpaceW

See RtosFileGetDiskFreeSpaceA

```
UINT32 RtosFileGetDiskFreeSpaceW (
    const WCHAR*      wszRootPathName,
    UINT32*           pdwSectorsPerCluster,
    UINT32*           pdwBytesPerSector,
    UINT32*           pdwNumberOfFreeClusters,
    UINT32*           pdwTotalNumberOfClusters )
```

---

#### 11.1.15.20 RtosFileFindFirstA

Find first file and get find file handle for subsequent find next calls.

```
UINT32 RtosFileFindFirstA (
    const CHAR*       szName,
    RTOSFILEFIND_DATA_A pFindData,
    RTOSLIB_HANDLE*  phFind )
```

### Parameter

*szName*

[in] The directory or path, and the file name, which can include wildcard characters, for example, an asterisk (\*) or a question mark (?).

This parameter should not be NULL, an invalid string (for example, an empty string or a string that is missing the terminating null character), or end in a trailing backslash (\).

*pFindData*

[out] A pointer to the RTOSFILEFIND\_DATA structure that receives information about a found file or directory. See comment.

*phFind*

[out] Handle for subsequent calls of RtosFileFindNext(). See comment.

### Return

RTE\_SUCCESS on success and an error-code on failure.

If the function fails because no matching files can be found, the function returns RTE\_ERROR\_FILE\_NOT\_FOUND.

### Comment

The macro RtosFileFindFirst can be used to call RtosFileFindFirstW if UNICODE is defined and RtosFileFindFirstA if not.

This function is not supported on SHM file mode!

```
typedef struct _RTOSFILEFIND_DATA_A
{
    UINT64 qwCreationTime;      Value that specifies when a file or
                               directory was created.
    UINT64 qwLastAccessTime;   For a file, the value specifies when the
                               file was last read from, written to, or
                               for executable files, run.
                               For a directory, the structure specifies
                               when the directory is created.
```

```

    UINT64 qwLastWriteTime;        For a file, the value specifies when the
                                   file was last written to, truncated, or
                                   overwritten.
                                   For a directory, the value specifies
                                   when the directory is created.
    UINT64 qwFileSize;            File size in bytes.
    UINT32 dwReserved0;
    UINT32 dwReserved1;
    UINT32 dwFileAttributes;      The file attributes of a file.
                                   → see RtosFileCreateEx() comment.
    CHAR    tszFileName[RTOSFILE_MAX_PATH];
    CHAR    tszAlternateFileName[RTOSFILE_FIND_FILENAME_ALT_LENGTH];
    UINT16  wReservedFor64BitAlignment;
} RTOSLIB_PACKED(8) RTOSFILEFIND_DATA_A, *RTOSFILEFIND_DATA_A;

```

tszFileName contains the name of the file.

tszAlternateFileName contains an alternative name for the file.

This name is in the classic 8.3 file name format.

#### 11.1.15.21 RtosFileFindFirstW

See RtosFileFindFirstA

```

UINT32 RtosFileFindFirstW (
    const WCHAR*          wszName,
    RTOSLIB_HANDLE*      pFindData,
    RTOSLIB_HANDLE*      phFind )

```

#### 11.1.15.22 RtosFileFindNextA

Subsequent calls to get all files in specified directory.

```

UINT32 RtosFileFindNextA (
    RTOSLIB_HANDLE        hFind,
    RTOSLIB_HANDLE*      pFindData)

```

#### Parameter

*hFind*

[in] Handle (returned by a previous RtosFileFindFirst call) for subsequent calls of RtosFileFindNext(). See comment.

*pFindData*

[out] A pointer to the RTOSFILEFIND\_DATA structure that receives information about a found file or directory. See comment.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

If the function fails because no more matching files can be found, the function returns RTE\_ERROR\_FILE\_NO\_MORE\_FILES.

#### Comment

The macro RtosFileFindNext can be used to call RtosFileFindNextW if UNICODE is defined and RtosFileFindNextA if not.

This function is not supported on SHM file mode!

#### 11.1.15.23 RtosFileFindNextW

See RtosFileFindNextA

**UINT32 RtosFileFindNextW (**  
    **RTOSLIB\_HANDLE**          **hFind,**  
    **PRTOSEFILEFIND\_DATA\_W**  **pFindData)**

---

#### 11.1.15.24 RtosFileSetCurrentDirectoryA

Set the current directory.

**UINT32 RtosFileSetCurrentDirectoryA (**  
    **const CHAR\***              **szPathName)**

**Parameter**

*szPathName*

[in] The path to the new current directory. This parameter may specify a relative path or a full path. In either case, the full path of the specified directory is calculated and stored as the current directory. See comment

**Return**

RTE\_SUCCESS on success and an error-code on failure.

**Comment**

The string must not exceed RTOSFILE\_MAX\_PATH characters, including the terminating null character. The final character before the null character must be a backslash ('\'). If you do not specify the backslash, it will be added for you; therefore, specify RTOSFILE\_MAX\_PATH-2 characters for the path unless you include the trailing backslash, in which case, specify RTOSFILE\_MAX\_PATH-1 characters for the path.

The macro RtosFileSetCurrentDirectory can be used to call RtosFileSetCurrentDirectoryW if UNICODE is defined and RtosFileSetCurrentDirectoryA if not. This function is not supported on SHM file mode!

---

#### 11.1.15.25 RtosFileSetCurrentDirectoryW

See RtosFileSetCurrentDirectoryA

**UINT32 RtosFileSetCurrentDirectoryW (**  
    **const WCHAR\***             **wszPathName)**

#### 11.1.15.26 RtosFileGetCurrentDirectoryA

Get the current directory.

**UINT32 RtosFileGetCurrentDirectoryA (**  
    **UINT32\***                  **pdwBufferLength,**  
    **const CHAR\***             **szPathName)**

**Parameter**

*pdwBufferLength*

[in/out] Size of supplied buffer. The buffer length must include room for a terminating null character.

*szPathName*

[out] A pointer to the string buffer that receives the current directory string. See comment

**Return**

RTE\_SUCCESS on success and an error-code on failure.

**Comment**

28.03.2019

Setting parameter `szPathName` to `NULL` and supply `*pdwBufferLength = 0`, than the needed buffer size will be determined and returned by `pdwBufferLength`, in characters, including the null-terminating character.

The macro `RtosFileGetCurrentDirectory` can be used to call `RtosFileGetCurrentDirectoryW` if `UNICODE` is defined and `RtosFileGetCurrentDirectoryA` if not.  
This function is not supported on SHM file mode!

---

#### 11.1.15.27 `RtosFileGetCurrentDirectoryW`

See `RtosFileGetCurrentDirectoryA`

**UINT32** `RtosFileGetCurrentDirectoryW` (  
    **UINT32\***                    **pdwBufferLength,**  
    **const WCHAR\***             **wszPathName)**

---

#### 11.1.15.28 `RtosFileCreateDirectoryA`

Create a new directory.

**UINT32** `RtosFileCreateDirectoryA` (  
    **const CHAR\***                    **szPathName,**  
    **const PRDOS\_SECURITY\_ATTRIBUTES** **pSecurityAttributes )**

##### Parameter

*szPathName*

[in] The path of the directory to be created.

There is a default string size limit for paths of 248 characters.

*pSecurityAttributes*

[in] Currently only `NULL` is supported.

##### Return

`RTE_SUCCESS` on success and an error-code on failure.

##### Comment

The macro `RtosFileCreateDirectory` can be used to call `RtosFileCreateDirectoryW` if `UNICODE` is defined and `RtosFileCreateDirectoryA` if not.  
This function is not supported on SHM file mode!

---

#### 11.1.15.29 `RtosFileCreateDirectoryW`

See `RtosFileCreateDirectoryA`

**UINT32** `RtosFileCreateDirectoryW` (  
    **const WCHAR\***                    **wszPathName,**  
    **const PRDOS\_SECURITY\_ATTRIBUTES** **pSecurityAttributes )**

---

#### 11.1.15.30 `RtosFileRemoveDirectoryA`

Remove the specified directory.

**UINT32** `RtosFileRemoveDirectoryA` (  
    **const CHAR\***                    **szPathName)**

**const CHAR\*                    szPathName)**

**Parameter**

*szPathName*

[in]     The path of the directory to be removed. This path must specify an empty directory.

**Return**

RTE\_SUCCESS on success and an error-code on failure.

**Comment**

The macro RtosFileRemoveDirectory can be used to call RtosFileRemoveDirectoryW if UNICODE is defined and RtosFileRemoveDirectoryA if not.

This function is not supported on SHM file mode!

---

**11.1.15.31 RtosFileRemoveDirectoryW**

See RtosFileRemoveDirectoryA

**UINT32 RtosFileRemoveDirectoryW (**

**const WCHAR\*                    wszPathName)**

## 11.1.16 RTOS Library – generic object functions

RtosLib provides generic functions for handling different object types like events, files, sockets, etc.

---

### 11.1.16.1 RtosObjectRead

This is the generic function to read data into a buffer.

```
UINT32 RtosObjectRead(  
    RTOSLIB_HANDLE    hObject,  
    UINT8             *pBuffer,  
    UINT32            dwBufferSize,  
    UINT32            *pdwBytesRead,  
    UINT32            dwTimeoutMs )
```

#### Parameter

<i>hObject</i>	[in]	Object handle returned from RtosMsgQueueCreate(), RtosPipeCreate(), RtosSocketCreate(), RtosFileCreate()
<i>pBuffer</i>	[out]	Buffer to receive the data read
<i>dwBufferSize</i>	[in]	Length of string buffer in bytes
<i>pdwBytesRead</i>	[out]	Number of bytes read into the buffer
<i>dwTimeoutMs</i>	[in]	Timeout in milliseconds

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

If there is no data to be read the function will return after timeout elapsed.  
If data was read the function will return as soon as the read buffer is full or there is no more data to be read.

---

### 11.1.16.1 RtosObjectWrite

This is the generic function to write data from a buffer.

```
UINT32 RtosObjectWrite(
    RTOSLIB_HANDLE      hObject,
    const UINT8         *pBuffer,
    UINT32              dwBufferSize,
    UINT32              *pdwBytesWritten,
    UINT32              dwTimeoutMs )
```

#### Parameter

*hObject*  
[in] Object handle returned from RtosMsgQueueCreate(), RtosPipeCreate(), RtosSocketCreate(), RtosFileCreate()

*pBuffer*  
[out] Buffer to receive the data read

*dwBufferSize*  
[in] Length of string buffer in bytes

*pdwBytesWritten*  
[out] Number of bytes written

*dwTimeoutMs*  
[in] Timeout in milliseconds

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The function will return after all data was written or the timeout elapsed. If the timeout elapsed the number of bytes written will be returned in *pdwBytesWritten*.

---

### 11.1.16.2 RtosObjectWait

This is the generic function to wait for an event. The event is object specific. For details check the comment.

```
UINT32 RtosObjectWait (
    RTOSLIB_HANDLE      hObject,
    UINT32              dwTimeoutMs )
```

#### Parameter

*hObject*  
[in] Object handle returned from RtosCreateEvent(), RtosNotificationExCreate(), RtosMsgQueueCreate(), RtosPipeCreate(), RtosSocketCreate(), RtosFileCreate()

*dwTimeoutMs*  
[in] Timeout in milliseconds

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The function will return when the object was signaled or after timeout elapsed. The signal is object specific:

- Event : The event was set
- NotificationEx : A notification occurred - use RtosNotificationWait instead for details.
- MsgQueue : Reader: Read data available / Writer: Data can be send
- Pipe : Read data available
- Socket : Read data available

---

### 11.1.16.3 RtosObjectBreakWait

This is the generic function to break a waiting function.

**UINT32 RtosObjectBreakWait (**  
**RTOSLIB\_HANDLE            hObject )**

**Parameter**

*hObject*

[in]        Object handle returned from  
RtosCreateEvent(), RtosNotificationExCreate(), RtosMsgQueueCreate(),  
RtosPipeCreate(),RtosSocketCreate(),RtosFileCreate()

**Return**

RTE\_SUCCESS on success and an error-code on failure.

**Comment**

A thread could be waiting infinite for an event. If this thread should be shut down the RtosObjectBreakWait function can be used to break the wait. In this case the wait function will return the error RTE\_ERROR\_BREAK\_WAIT.

---

### 11.1.16.4 RtosObjectClose

This is the generic function to close any RTOSLIB\_HANDLE object.

**UINT32 RtosObjectClose (**  
**RTOSLIB\_HANDLE            hObject )**

**Parameter**

*hObject*

[in]        Object handle returned from  
RtosCreateEvent(), RtosNotificationExCreate(), RtosMsgQueueCreate(),  
RtosPipeCreate(),RtosSocketCreate(),RtosFileCreate()

**Return**

RTE\_SUCCESS on success and an error-code on failure.

**Comment**

The handle becomes invalid and can't be used any more – regardless of an returned error code like timeout.

---

### 11.1.16.1 RtosObjectOptionSet

This is the generic function to configure a handle option. The option is object specific. For details check the comment.

```
UINT32 RtosObjectOptionSet (  
    RTOSLIB_HANDLE      hObject,  
    UINT32              dwOptionId,  
    const UINT8         *pBuffer,  
    UINT32              dwBufferSize )
```

#### Parameter

*hObject*  
[in] Object handle returned from RtosSocketCreate(), RtosFileCreate()  
*dwOptionId*  
[in] Object specific ID  
*pBuffer*  
[in] Buffer containing the data to be set  
*dwBufferSize*  
[in] Length of buffer in bytes

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

Possible options are object specific:

Object	Option ID	Type	Description
Socket	RTOSOCKET_OPTION_RCVTIMEO	UINT32	Default Rx timeout in milliseconds
Socket	RTOSOCKET_OPTION_SNDTIMEO	UINT32	Default Tx timeout in milliseconds
File	RTOSFILE_OPTION_ACKTIMEOUT	UINT32	Default acknowledge timeout in milliseconds

---

### 11.1.16.1 RtosObjectOptionGet

This is the generic function to query a handle option. The option is object specific. See "RtosObjectOptionSet" for details.

```
UINT32 RtosObjectOptionGet (  
    RTOSLIB_HANDLE      hObject,  
    UINT32              dwOptionId,  
    const UINT8         *pBuffer,  
    UINT32              dwBufferSize )
```

#### Parameter

*hObject*  
[in] Object handle returned from RtosSocketCreate(), RtosFileCreate()  
*dwOptionId*  
[in] Object specific ID  
*pBuffer*  
[out] Buffer for receiving the option data  
*dwBufferSize*  
[in] Length of buffer in bytes

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

See "RtosObjectOptionSet" for details.

## 11.1.17 RTOS Library – message queue functions

RtosLib contains message queue functionality based on shared memory communication.

---

### 11.1.17.1 RtosMsgQueueCreateA

This function creates a message queue

```
UINT32 RtosMsgQueueCreateA (
    const CHAR*          szName,
    PRTOSMSGQUEUE_OPTIONS pOptions,
    RTOSLIB_HANDLE*      phObject )
```

#### Parameter

*szName*  
[in] Name of the message queue

*pOptions*  
[in] Message queue options. See comment.

*phObject*  
[out] Pointer to receive a RTOSLIB\_HANDLE object (message queue).

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

```
typedef struct _RTOSMSGQUEUE_OPTIONS
{
    UINT32 dwSize;           must be initialized with sizeof(RTOSMSGQUEUE_OPTIONS)
    UINT32 dwFlags;         can be
                            - RTOSMSGQUEUE_OPTIONS_FLAG_ALLOWBROKEN
                            to allow reading and writing the queue without counterpart
                            being connected
                            - RTOSMSGQUEUE_OPTIONS_FLAG_FORCE_CREATE
                            to ensure the message queue didn't exist before
                            - RTOSMSGQUEUE_OPTIONS_FLAG_FORCE_OPEN
                            to ensure the message queue existed before
                            -
                            RTOSMSGQUEUE_OPTIONS_FLAG_FORCE_EXCLUSIVE
                            to ensure being the first reader or writer (depending on
                            bReadAccess)
    UINT32 dwNumMessages;   number of messages the queue should contain.
                            0 uses the maximum available.
    UINT32 dwMsgDataSizeInBytes; number of data bytes usable in each message
    BOOL bReadAccess;       determines if the creator will be reader or writer to this queue.
} RTOSMSGQUEUE_OPTIONS, *PRTOSMSGQUEUE_OPTIONS;
```

The message queue shared memory (named "RtosLibMsgQueue") will be divided into n equal sized parts. The shared memory size per queue can be configured via config file:

```
[MessageQueue\RtosLibMsgQueue]
"MaxShmUsagePerQueue"=dword:100000
```

Note that not the entire data can be used by user data since there is also queue administrative data.

The macro RtosMsgQueueCreate can be used to call RtosMsgQueueCreateW if UNICODE is defined and RtosMsgQueueCreateA if not.

#### Hint:

Because the queue has a fixed maximum size the dwNumMessages member could be set to 0 to gain the maximum number of packets.

---

### 11.1.17.2 RtosMsgQueueCreateW

See RtosMsgQueueCreateA

```
UINT32 RtosMsgQueueCreateW (
    const WCHAR*      wszName,
    PRTOMSGQUEUE_OPTIONS pOptions,
    RTOSLIB_HANDLE*   phObject )
```

---

### 11.1.17.1 RtosMsgQueueInfoGet

This function queries information about message queue

```
UINT32 RtosMsgQueueInfoGet (
    RTOSLIB_HANDLE   hObject
    PRTOMSGQUEUE_INFO pInfo )
```

#### Parameter

*hObject*

[in] Object handle returned from  
RtosMsgQueueCreate()

*pInfo*

[in] Information structure. See comment.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

```
typedef struct _RTOSMSGQUEUE_INFO
{
    UINT32 dwSize;           must be initialized with sizeof( RTOSMSGQUEUE_INFO )
    UINT32 dwFlags;         reserved – should be 0.
    UINT32 dwNumMessages;   number of message containers
    UINT32 dwNumPending;    number of currently pending messages
    UINT32 dwNumPendingMax; maximum number of pending messages
    UINT32 dwMsgDataSizelnBytes; maximum data size per message in bytes
    UINT16 wNumReaders;     number of readers attached to this queue
    UINT16 wNumWriters;     number of writers attached to this queue
} RTOSMSGQUEUE_INFO, *RTOSMSGQUEUE_INFO;
```

---

### 11.1.17.1 RtosMsgQueueRead

This is a synonym for RtosObjectRead. See RtosObjectRead for details.

```
UINT32 RtosMsgQueueRead (
    RTOSLIB_HANDLE   hObject,
    UINT8            *pBuffer,
    UINT32           dwBufferSize,
    UINT32           *pdwBytesRead,
    UINT32           dwTimeoutMs )
```

---

#### 11.1.17.1 RtosMsgQueueWrite

This is a synonym for RtosObjectWrite. See RtosObjectWrite for details.

**UINT32 RtosMsgQueueWrite (**  
    **RTOSLIB\_HANDLE**        **hObject,**  
    **const UINT8**          **\*pBuffer,**  
    **UINT32**               **dwBufferSize,**  
    **UINT32**               **\*pdwBytesWritten,**  
    **UINT32**               **dwTimeoutMs )**

---

#### 11.1.17.1 RtosMsgQueueWait

This is a synonym for RtosObjectWait. See RtosObjectWait for details.

**UINT32 RtosMsgQueueWait (**  
    **RTOSLIB\_HANDLE**        **hObject,**  
    **UINT32**               **dwTimeoutMs )**

---

#### 11.1.17.1 RtosMsgQueueBreakWait

This is a synonym for RtosObjectBreakWait. See RtosObjectBreakWait for details.

**UINT32 RtosMsgQueueBreakWait (**  
    **RTOSLIB\_HANDLE**        **hObject )**

---

#### 11.1.17.2 RtosMsgQueueClose

This is a synonym for RtosObjectClose. See RtosObjectClose for details.

**UINT32 RtosMsgQueueClose (**  
    **RTOSLIB\_HANDLE**        **hObject )**

## 11.1.18 RTOS Library – pipe functions

RtosLib contains pipe functionality based on shared memory communication.

---

### 11.1.18.1 RtosPipeCreateA

This function creates a message queue

```
UINT32 RtosPipeCreateA (
    const CHAR*      szName,
    PRTOSPIPE_OPTIONS pOptions,
    RTOSLIB_HANDLE*  phObject )
```

#### Parameter

*szName*  
[in] Name of the pipe

*pOptions*  
[in] Pipe options. See comment.

*phObject*  
[out] Pointer to receive a RTOSLIB\_HANDLE object (pipe).

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

```
typedef struct _RTOSPIPE_OPTIONS
{
    UINT32 dwSize;           must be initialized with sizeof(RTOSPIPE_OPTIONS )
    UINT32 dwFlags;         can be
                           - RTOSPIPE_OPTIONS_FLAG_READ for read access
                           - RTOSPIPE_OPTIONS_FLAG_WRITE for write access
                           - RTOSPIPE_OPTIONS_FLAG_READWRITE for both
    UINT32 dwAvgDataSize;   Should be set to the average message data size in bytes.
                           0 = default = 1500 bytes
} RTOSPIPE_OPTIONS, *RTOSPIPE_OPTIONS;
```

A pipe contains two queues – one for reading and one for writing.  
Only the queue(s) requested (read / write / read+write) will be created.

To establish a communication using a pipe one side must call RtosPipeCreate while the other side has to call RtosPipeOpen. This is required to open the correct message queues.

The first caller – regardless of using RtosPipeCreate or RtosPipeOpen – will initialize the pipe with its average data size. The second caller's average data size parameter will be ignored.

The macro RtosPipeCreate can be used to call RtosPipeCreateW if UNICODE is defined and RtosPipeCreateA if not.

---

### 11.1.18.2 RtosPipeCreateW

See RtosPipeCreateA

```
UINT32 RtosPipeCreateW(
    const WCHAR*      wszName,
    PRTOSPIPE_OPTIONS pOptions,
    RTOSLIB_HANDLE*  phObject )
```

---

### 11.1.18.1 RtosPipeOpenA

This function opens a message queue. The counterpart has to use RtosPipeCreate. See RtosPipeCreateA for details.

```
UINT32 RtosPipeOpenA (
    const CHAR*      szName,
    PRTOSPIPE_OPTIONS pOptions,
    RTOSLIB_HANDLE* phObject )
```

---

### 11.1.18.1 RtosPipeOpenW

This function opens a message queue. The counterpart has to use RtosPipeCreate. See RtosPipeCreateA for details.

```
UINT32 RtosPipeOpenW (
    const WCHAR*      wszName,
    PRTOSPIPE_OPTIONS pOptions,
    RTOSLIB_HANDLE* phObject )
```

---

### 11.1.18.2 RtosPipeInfoGet

This function queries information about a pipe

```
UINT32 RtosPipeInfoGet (
    RTOSLIB_HANDLE hObject
    PRTOSPIPE_INFO pInfo )
```

#### Parameter

*hObject*

[in] Object handle returned from RtosPipeCreate()

*pInfo*

[in] Pointer to information structure. See comment.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

```
typedef struct _RTOSPIPE_INFO
```

```
{
```

UINT32 dwSize;	must be initialized with sizeof( RTOSPIPE_INFO )
UINT32 dwFlags;	see RTOSPIPE_OPTIONS_FLAG_...
UINT32 dwNumMessagesRx;	number of message containers for Rx queue
UINT32 dwNumMessagesTx;	number of message containers for Tx queue
UINT32 dwNumPendingRx;	number of currently pending Rx messages
UINT32 dwNumPendingTx;	number of currently pending Tx messages
UINT32 dwNumPendingMaxRx;	maximum number of pending Rx messages
UINT32 dwNumPendingMaxTx;	maximum number of pending Tx messages
UINT32 dwNumWriterRx;	number of writers attached to Rx
UINT32 dwNumReaderTx;	number of readers attached to Tx
UINT32 dwMsgDataSizeInBytes;	maximum data size per message in bytes. Larger messages will be split into packets.

```
} RTOSPIPE_INFO, *RTOSPIPE_INFO;
```

---

### 11.1.18.3 RtosPipeRead

This is a synonym for RtosObjectRead. See RtosObjectRead for details.

**UINT32 RtosPipeRead (**  
    **RTOSLIB\_HANDLE**        **hObject,**  
    **UINT8**                  **\*pBuffer,**  
    **UINT32**                 **dwBufferSize,**  
    **UINT32**                 **\*pdwBytesRead,**  
    **UINT32**                 **dwTimeoutMs )**

---

### 11.1.18.4 RtosPipeWrite

This is a synonym for RtosObjectWrite. See RtosObjectWrite for details.

**UINT32 RtosPipeWrite (**  
    **RTOSLIB\_HANDLE**        **hObject,**  
    **const UINT8**            **\*pBuffer,**  
    **UINT32**                 **dwBufferSize,**  
    **UINT32**                 **\*pdwBytesWritten,**  
    **UINT32**                 **dwTimeoutMs )**

---

### 11.1.18.5 RtosPipeWait

This is a synonym for RtosObjectWait. See RtosObjectWait for details.

**UINT32 RtosPipeWait (**  
    **RTOSLIB\_HANDLE**        **hObject,**  
    **UINT32**                 **dwTimeoutMs )**

---

### 11.1.18.6 RtosPipeBreakWait

This is a synonym for RtosObjectBreakWait. See RtosObjectBreakWait for details.

**UINT32 RtosPipeBreakWait (**  
    **RTOSLIB\_HANDLE**        **hObject )**

---

### 11.1.18.7 RtosPipeClose

This is a synonym for RtosObjectClose. See RtosObjectClose for details.

**UINT32 RtosPipeClose (**  
    **RTOSLIB\_HANDLE**        **hObject )**

## 11.1.19 RTOS Library – socket functions

RtosLib contains socket functionality based on shared memory communication.

---

### 11.1.19.1 RtosSocketCreate

This function creates a socket

```
UINT32 RtosSocketCreate(
    UINT32          dwFamily,
    UINT32          dwType,
    UINT32          dwProtocol,
    RTOSLIB_HANDLE* phObject )
```

#### Parameter

*dwFamily*  
[in] Socket family – must be RTOSSOCKET\_FAMILY\_RTE

*dwType*  
[in] Socket type – currently only RTOSSOCKET\_TYPE\_STREAM is supported.

*dwProtocol*  
[in] Protocol – can be RTOSSOCKET\_PROTOCOL\_TCP or RTOSSOCKET\_PROTOCOL\_UDP

*phObject*  
[out] Pointer to receive a RTOSLIB\_HANDLE object (socket).

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

-

---

### 11.1.19.1 RtosSocketBind

This function binds a socket

```
UINT32 RtosSocketBind(
    RTOSLIB_HANDLE hObject,
    UINT16         wPort )
```

#### Parameter

*hObject*  
[in] Object handle returned from RtosSocketCreate()

*wPort*  
[in] Port number to bind the socket to.

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

-

---

### 11.1.19.1 RtosSocketListen

This function configures a socket to listen

**UINT32 RtosSocketListen( RTOSLIB\_HANDLE hObject )**

#### Parameter

*hObject*  
[in] Object handle returned from RtosSocketCreate()

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

-

---

### 11.1.19.1 RtosSocketAccept

This function (TCP) accepts a new connection

**UINT32 RtosSocketAccept( RTOSLIB\_HANDLE hObject, RTOSLIB\_HANDLE\* phObject, RTOSSOCKET\_ADDR pAddrNew, UINT32 dwTimeoutMs )**

#### Parameter

*hObject*  
[in] Object handle returned from RtosSocketCreate()  
*phObject*  
[out] Pointer to receive a RTOSLIB\_HANDLE object (socket).  
*pAddrNew*  
[out] Pointer to a RTOSSOCKET\_ADDR structure containing the address of the new socket. Its member 'bySize' must be initialized with sizeof(RTOSSOCKET\_ADDR).  
*dwTimeoutMs*  
[in] Timeout in milliseconds

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

Before calling RtosSocketAccept pAddrNew→bySize must be initialized with sizeof(RTOSSOCKET\_ADDR).

```
typedef struct _RTOSSOCKET_ADDR
{
    UINT8 bySize;           must be initialized with sizeof( RTOSSOCKET_ADDR )
    UINT8 byFamily;        see RTOSSOCKET_FAMILY_...
    UINT16 wPort;          port number
    union _RTOSSOCKET_ADDR_FAMILY
    {
        struct _RTOSSOCKET_ADDR_FAMILY_RAW
        {
            UINT8 abyData[12];
        } Raw;
        struct _RTOSSOCKET_ADDR_FAMILY_RTE
        {
            UINT32 dwOsId;    OS id
        }
    }
}
```

```
        UINT32 adwReserved[2];
    } Rte;
} u;
} RTOSSOCKET_ADDR, *PRTOSOCKET_ADDR;
```

---

#### 11.1.19.1 RtosSocketConnect

This function (TCP) connects to a server

```
UINT32 RtosSocketConnect (
    RTOSLIB_HANDLE      hObject,
    const PRTOSOCKET_ADDR pAddr )
```

##### Parameter

*hObject*

[in] Object handle returned from  
RtosSocketCreate()

*pAddr*

[in] Pointer to a RTOSSOCKET\_ADDR structure containing the address to connect to.  
Its member 'bySize' must be initialized with sizeof(RTOSSOCKET\_ADDR).

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

Before calling RtosSocketConnect pAddr→bySize must be initialized with  
sizeof(RTOSSOCKET\_ADDR ).

See RtosSocketAccept for details of RTOSSOCKET\_ADDR.

---

#### 11.1.19.1 RtosSocketShutdown

This function does shutdown a connection.

```
UINT32 RtosSocketShutdown (
    RTOSLIB_HANDLE      hObject )
```

##### Parameter

*hObject*

[in] Object handle returned from  
RtosSocketCreate()

##### Return

RTE\_SUCCESS on success and an error-code on failure.

##### Comment

-

---

### 11.1.19.1 RtosSocketRecv

This is the (TCP) function to receive data into a buffer.

```
UINT32 RtosSocketRecv (
    RTOSLIB_HANDLE    hObject,
    UINT8             *pBuffer,
    UINT32            dwBufferSize,
    UINT32            *pdwBytesRead )
```

#### Parameter

*hObject*  
[in] Object handle returned from RtosSocketCreate()

*pBuffer*  
[out] Buffer to receive the data read

*dwBufferSize*  
[in] Length of string buffer in bytes

*pdwBytesRead*  
[out] Number of bytes read into the buffer

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

If there is no data to be read the function will return after timeout elapsed.  
If data was read the function will return as soon as the read buffer is full or there is no more data to be read.  
The timeout can be configured - see RtosObjectOptionSet for details.

---

### 11.1.19.1 RtosSocketSend

This is the (TCP) function to send data from a buffer.

```
UINT32 RtosSocketSend (
    RTOSLIB_HANDLE    hObject,
    const UINT8       *pBuffer,
    UINT32            dwBufferSize,
    UINT32            *pdwBytesWritten )
```

#### Parameter

*hObject*  
[in] Object handle returned from RtosSocketCreate()

*pBuffer*  
[out] Buffer to receive the data read

*dwBufferSize*  
[in] Length of string buffer in bytes

*pdwBytesWritten*  
[out] Number of bytes written

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The function will return after all data was written or the timeout elapsed.  
If the timeout elapsed the number of bytes written will be returned in *pdwBytesWritten*.  
The timeout can be configured - see RtosObjectOptionSet for details.

---

### 11.1.19.2 RtosSocketRecvFrom

This is the (UDP) function to receive data into a buffer and optionally get the source address.

```
UINT32 RtosSocketRecvFrom (  
    RTOSLIB_HANDLE      hObject,  
    UINT8               *pBuffer,  
    UINT32              dwBufferSize,  
    UINT32              *pdwBytesRead,  
    PRTOSSOCKET_ADDR   pAddr )
```

#### Parameter

*hObject*

[in] Object handle returned from RtosSocketCreate()

*pBuffer*

[out] Buffer to receive the data read

*dwBufferSize*

[in] Length of string buffer in bytes

*pdwBytesRead*

[out] Number of bytes read into the buffer

*pAddr*

[out] Pointer to a RTOSSOCKET\_ADDR structure to return the source address. Its member 'bySize' must be initialized with sizeof(RTOSSOCKET\_ADDR).

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

Before calling RtosSocketRecvFrom *pAddr*→*bySize* must be initialized with sizeof(RTOSSOCKET\_ADDR).

If there is no data to be read the function will return after timeout elapsed.

If data was read the function will return as soon as the read buffer is full or there is no more data to be read.

The timeout can be configured - see RtosObjectOptionSet for details.

---

### 11.1.19.3 RtosSocketSendTo

This is the (UDP) function to send data from a buffer to a specific address.

```
UINT32 RtosSocketSendTo (  
    RTOSLIB_HANDLE      hObject,  
    const UINT8         *pBuffer,  
    UINT32              dwBufferSize,  
    UINT32              *pdwBytesWritten,  
    const PRTOSSOCKET_ADDR pAddr )
```

#### Parameter

*hObject*

[in] Object handle returned from RtosSocketCreate()

*pBuffer*

[out] Buffer to receive the data read

*dwBufferSize*

[in] Length of string buffer in bytes

*pdwBytesWritten*

[out] Number of bytes written

*pAddr*

[in] Pointer to a RTOSSOCKET\_ADDR structure containing the destination address. Its member 'bySize' must be initialized with sizeof(RTOSSOCKET\_ADDR).

## Return

RTE\_SUCCESS on success and an error-code on failure.

## Comment

The function will return after all data was written or the timeout elapsed.

If the timeout elapsed the number of bytes written will be returned in `pdwBytesWritten`.

The timeout can be configured - see `RtosObjectOptionSet` for details.

---

### 11.1.19.4 RtosSocketReadFrom

This is the (UDP) function to receive data into a buffer and optionally get the source address.

```
UINT32 RtosSocketReadFrom (
    RTOSLIB_HANDLE      hObject,
    UINT8               *pBuffer,
    UINT32               dwBufferSize,
    UINT32               *pdwBytesRead,
    RTOSSOCKET_ADDR     pAddr,
    UINT32               dwTimeoutMs )
```

#### Parameter

*hObject*

[in] Object handle returned from  
`RtosSocketCreate()`

*pBuffer*

[out] Buffer to receive the data read

*dwBufferSize*

[in] Length of string buffer in bytes

*pdwBytesRead*

[out] Number of bytes read into the buffer

*pAddr*

[out] Pointer to a `RTOSSOCKET_ADDR` structure to return the source address.  
Its member 'bySize' must be initialized with `sizeof(RTOSSOCKET_ADDR)`.

*dwTimeoutMs*

[in] Timeout in milliseconds

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

Before calling `RtosSocketReadFrom` `pAddr->bySize` must be initialized with `sizeof(RTOSSOCKET_ADDR)`.

If there is no data to be read the function will return after timeout elapsed.

If data was read the function will return as soon as the read buffer is full or there is no more data to be read.

---

### 11.1.19.5 RtosSocketWriteTo

This is the (UDP) function to send data from a buffer to a specific address.

```
UINT32 RtosSocketWriteTo (
    RTOSLIB_HANDLE      hObject,
    const UINT8         *pBuffer,
    UINT32              dwBufferSize,
    UINT32              *pdwBytesWritten,
    const PRTOSSOCKET_ADDR pAddr,
    UINT32              dwTimeoutMs )
```

#### Parameter

*hObject*  
[in] Object handle returned from RtosSocketCreate()

*pBuffer*  
[out] Buffer to receive the data read

*dwBufferSize*  
[in] Length of string buffer in bytes

*pdwBytesWritten*  
[out] Number of bytes written

*pAddr*  
[in] Pointer to a RTOSSOCKET\_ADDR structure containing the destination address. Its member 'bySize' must be initialized with sizeof(RTOSSOCKET\_ADDR).

*dwTimeoutMs*  
[in] Timeout in milliseconds

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The function will return after all data was written or the timeout elapsed. If the timeout elapsed the number of bytes written will be returned in pdwBytesWritten.

---

### 11.1.19.6 RtosSocketRead

(TCP) This is a synonym for RtosObjectRead. See RtosObjectRead for details.

```
UINT32 RtosSocketRead (
    RTOSLIB_HANDLE      hObject,
    UINT8              *pBuffer,
    UINT32              dwBufferSize,
    UINT32              *pdwBytesRead,
    UINT32              dwTimeoutMs )
```

---

### 11.1.19.7 RtosSocketWrite

(TCP) This is a synonym for RtosObjectWrite. See RtosObjectWrite for details.

```
UINT32 RtosSocketWrite (
    RTOSLIB_HANDLE      hObject,
    const UINT8         *pBuffer,
    UINT32              dwBufferSize,
    UINT32              *pdwBytesWritten,
    UINT32              dwTimeoutMs )
```

#### 11.1.19.8 RtosSocketWait

This is a synonym for RtosObjectWait. See RtosObjectWait for details.

**UINT32 RtosSocketWait (**  
    **RTOSLIB\_HANDLE           hObject,**  
    **UINT32                   dwTimeoutMs )**

---

#### 11.1.19.9 RtosSocketBreakWait

This is a synonym for RtosObjectBreakWait. See RtosObjectBreakWait for details.

**UINT32 RtosSocketBreakWait (**  
    **RTOSLIB\_HANDLE           hObject )**

---

#### 11.1.19.10 RtosSocketClose

This is a synonym for RtosObjectClose. See RtosObjectClose for details.

**UINT32 RtosSocketClose (**  
    **RTOSLIB\_HANDLE           hObject )**

---

#### 11.1.19.11 RtosSocketOptionSet

This is a synonym for RtosObjectOptionSet. See RtosObjectOptionSet for details.

**UINT32 RtosSocketOptionSet (**  
    **RTOSLIB\_HANDLE           hObject,**  
    **UINT32                   dwOptionId,**  
    **const UINT8             \*pBuffer,**  
    **UINT32                   dwBufferSize )**

---

#### 11.1.19.12 RtosSocketOptionGet

This is a synonym for RtosObjectOptionGet. See RtosObjectOptionGet for details.

**UINT32 RtosSocketOptionGet (**  
    **RTOSLIB\_HANDLE           hObject,**  
    **UINT32                   dwOptionId,**  
    **const UINT8             \*pBuffer,**  
    **UINT32                   dwBufferSize )**

## 11.1.20 RTOS Library – device functions

RtosLib contains device functionality for configuring host (Windows) devices.  
This API is only available on host-side.

---

### 11.1.20.1 RtosDeviceA

This function allows device configuration.

**UINT32 RtosDeviceA (**  
    **const CHAR\*     szParams )**

#### Parameter

*szParams*  
    [in]     Configuration parameters – see Comment for details

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The function does not require RtosLibInit to be called. In fact it must be called without RtosLibInit when installing RtosDrv.

Usage: RtosDevice(“[<opt>...] <command> [<arg>...]”)

<opt>...       Zero or more options (see list below)  
<command>     Specifies a command (see list below)  
<arg>...       Zero or more arguments that modify a command.

#### Options:

-force         Force operation  
-noui         Do not display user interface  
-all          Also include non-present devices (default is only present devices)

#### Commands:

rte\_install -inf:<inf>  
    Install RtE device support. This will update and register the Class(Co)Installers.  
    -inf:<inf>     Can specify a single inf file or an directory containing inf files

rte\_uninstall [-deldp]  
    Remove RtE device support including all assigned devices (present and non-present),  
    registry entries, services, inf's, class(co)installer, ...  
    -deldp       Deletes installed driver packages for all RtE devices

rte\_add [-slot] [-inf:<inf>] [<n>] <id> [<id>...]  
    Add RtE device(s) assignment.  
    A registry entry will be written identifying this device as RtE device – and so only RtE  
    drivers will be accepted.  
    Be careful specifying ‘-noui’ because this might block an unsigned driver dialog and so  
    cause assignment to fail.  
    -slot        On <pciid> the PCI slot will be assigned and not the device.  
    -inf:<inf>    Can specify a single inf file or an directory containing inf files  
    <n>          Specifies the n-th occurrence of a matching device  
    <id>         Specifies the id of the devices.

```

rte_remove [-deldp] [-slot] [<n>] <id> [<id>...]
rte_remove [-deldp] [-slot] [<n>] =<class> [<id>...]
    Remove RtE device(s) assignment.
    The registry entry created by 'rte_add' will be removed.
    -deldp          Deletes installed driver packages for all RtE devices
    -slot           On <pciid> the PCI slot will be assigned and not the device.
    <n>             Specifies the n-th occurrence of a matching device
    <id>            Specifies the id of the devices.
    =<class>        Specifies a device setup class.

rte_update -inf:<inf> <id> [<id>...]
    Update existing drivers. This can be used to update an existing driver.
    It is also possible to update a Windows device to become an RtE device.
    It is not possible to update a RtE device assigned using "rte_add" to become a
    Windows device again - this requires "rte_remove".
    -inf:<inf>      Can specify a single inf file or an directory containing inf files
    <id>           Specifies the id of the devices.

rte_configure -<prop:val> [-<prop:val>...] [<n>] <id> [<id>...]
rte_configure -<prop:val> [-<prop:val>...] [<n>] =<class> [<id>...]
    Configure RtE device(s) properties
    <prop:val>      Property and value to be set
    -int_mode:0    Interrupt mode default (line interrupt)
    -int_mode:1    Interrupt mode polling (requires specialized rt-driver)
    -int_mode:2    Interrupt mode MSI (only available on PCIe devices)
    -osid:x        Assign device to OS x (0=Rtos, 1=Rtos1, ..., 0xFFFFFFFFE=Auto)
    -name:x        Assign display name x to device. #2, #3, #4 will be appended case
multiple devices.
    <n>             Specifies the n-th occurrence of a matching device
    <id>            Specifies the id of the devices.
    =<class>        Specifies a device setup class.

root_addorupdate -inf:<inf> <hwid>
    Updates a root enumerated driver or installs it when not found.
    -inf:<inf>      Specify a single inf file
    <hwid>         Specifies a hardware ID for the device.

root_add <inf> <hwid>
    Install root enumerated driver
    -inf:<inf>      Specifies an INF file with installation information for the device.
    <hwid>         Specifies a hardware ID for the device.

root_remove [-deldp] [<n>] <id> [<id>...]
    Remove root enumerated driver
    -deldp          Deletes installed driver packages for all RtE devices
    <n>             Specifies the n-th occurrence of a matching device
    <id>            Specifies the id of the devices.

find [<n>] <id> [<id>...]
find [<n>] =<class> [<id>...]
    List found devices
    <n>             Specifies the n-th occurrence of a matching device
    <id>            Specifies the id of the devices.
    =<class>        Specifies a device setup class.

enable [<n>] <id> [<id>...]
enable [<n>] =<class> [<id>...]
    Enable found devices
    <n>             Specifies the n-th occurrence of a matching device
    <id>            Specifies the id of the devices.
    =<class>        Specifies a device setup class.

```

disable [<n>] <id> [<id>...]  
 disable [<n>] =<class> [<id>...]  
     Disable found devices  
     <n>          Specifies the n-th occurrence of a matching device  
     <id>         Specifies the id of the devices.  
     =<class>     Specifies a device setup class.

dp\_add <inf>  
     Install driver package  
     -inf:<inf>     Specifies an INF file.

dp\_delete <inf>  
     Remove driver package  
     -inf:<inf>     Specifies an Oem\*.inf file.

#### ID Examples:

<pciid>	\$5:2:0	PCI device with address bus 5 device 2 function 0
<hwid>	ISAPNP\PNP0501	Hardware ID
	*PNP0501	Hardware ID with apostrophe ( ' prefixes literal match - matches exactly as typed, including the asterisk.)
<id>	*	All devices
	ISAPNP\PNP0501	Hardware ID
	*PNP*	Hardware ID with wildcards ( * matches anything)
	@ISAPNP\*	Instance ID with wildcards ( @ prefixes instance ID)
	*PNP0501	Hardware ID with apostrophe ( ' prefixes literal match - matches exactly as typed, including the asterisk.)
	=SYSTEM	All devices of class 'system'
	\$5:2:0	PCI device with address bus 5 device 2 function 0
<n> <id>	:2 *PNP0501	2nd occurrence of Hardware ID with apostrophe

#### Calling RtosDevice() function:

- RtosDevice is provided by RtosLib32.dll or RtosLib64.dll. Please pay attention that on a 64bit system it is required to use RtosLib64.lib to succeed with device installation functions.
- Alternatively the Uploader (RtosUpload.exe) parameter 'device' can be used to forward a single string to RtosDevice(). To put multiple parameters into that string it has to be enclosed by quotation marks.

This example will listing all COM devices:

```
>>C:\>RtosUpload.exe /nosleep /nowait /device "-all find *PNP0501"<<
```

#### RtE Setup:

- During installation Setup will call  
 >>RtosUpload.exe /nosleep /nowait /device "-noui rte\_install -inf:\"\$PATH\_TO\_RTE\_ROOT\$Bin\Windows\Drivers\RTOS\_Installer.inf\""<<  
 to prepare for RtE device assignment.
- During uninstall Setup will call  
 >>RtosUpload.exe /nosleep /nowait /device "-noui rte\_uninstall -deldp"<<  
 to assign all RtE devices back to Windows, Remove registrations and optionally delete all installed RtE driver packages, if "-deldp" was specified.

#### Assigning a Pro1000 to RtE

- Assigning all found instances:  
 >>C:\>RtosUpload.exe /nosleep /nowait /device "rte\_add -inf:\"\$PATH\_TO\_RTE\_ROOT\$Bin\Windows\Drivers\" PCI\VEN\_8086&DEV\_10A7"<<
- Assign first occurrence only :  
 >>C:\>RtosUpload.exe /nosleep /nowait /device "rte\_add -inf:\"\$PATH\_TO\_RTE\_ROOT\$Bin\Windows\Drivers\" :1 PCI\VEN\_8086&DEV\_10A7"<<
- Assing by PCI address (Bus 3 Device 2 Function 0)  
 >>C:\>RtosUpload.exe /nosleep /nowait /device "rte\_add -inf:\"\$PATH\_TO\_RTE\_ROOT\$Bin\Windows\Drivers\" \$3:2:0"<<
- Remove by PCI address (Bus 3 Device 2 Function 0)  
 >>C:\>RtosUpload.exe /nosleep /nowait /device "-noui rte\_remove -deldp \$3:2:0"<<  
 This will also delete the driver package for this device.

## Configuring a Pro1000

- Change interrupt to polling mode:  
This requires a specialized driver supporting polling mode. EC-Master LinkLayer for example are supporting this mode. Regular OS drivers will require an interrupt. If your driver requests an interrupt but the device is configured for polling mode probably an "unexpected interrupt id" message will appear when the driver starts.  
>>C:\>RtosUpload.exe /nosleep /nowait /device "-noui rte\_configure -int\_mode:1 \$3:2:0"<<
- Change interrupt to MSI mode:  
MSI is supported by most PCIe cards. Its useful to solve interrupt problems and can be configured calling:  
>>C:\>RtosUpload.exe /nosleep /nowait /device "-noui rte\_configure -int\_mode:2 \$3:2:0"<<
- Assigning a device to another OS (Multi-RTOS):  
>>C:\>RtosUpload.exe /nosleep /nowait /device "-noui rte\_configure -osid:1 \$3:2:0"<<
- Renaming a device:  
>>C:\>RtosUpload.exe /nosleep /nowait /device "-noui rte\_configure -name:\\"My New Device Name Pro1000\" \$3:2:0"<<

## Finding Pro1000

- Find all present Pro1000 on a system using Hardware ID:  
>>C:\>RtosUpload.exe /nosleep /nowait /device "-all find PCI\VEN\_8086&DEV\_10A7"<<
- Find all present Pro1000 on a system using Hardware ID and PCI address B3-D0-F0:  
>>C:\>RtosUpload.exe /nosleep /nowait /device "-all find \$3:0:0"<<

## Assigning a PCI slot

This is usefull if you have a dedicated PCI slot being assigned to the RtE but different cards to be plugged in. It might also help if you're using an OS imag on slightly different mainboards where device instance might be different.

As consequence of assigning a slot the device can be exchanged but will still be assigned to RtE automatically (in case the required drivers have been provided using for example 'dp\_add' command).

- Assign PCI slot (Bus 3 Device 2 Function 0)  
>>C:\>RtosUpload.exe /nosleep /nowait /device "rte\_add -slot -inf:\"  
\$PATH\_TO\_RTE\_ROOT\$\Bin\Windows\Drivers\" \$3:2:0"<<
- Remove PCI slot (Bus 3 Device 2 Function 0)  
>>C:\>RtosUpload.exe /nosleep /nowait /device "rte\_remove -noui -slot \$3:2:0"<<

---

### 11.1.20.1 RtosDeviceW

This function allows device configuration. See RtosDeviceA for details.

**UINT32 RtosDeviceW (**  
**const WCHAR\* wszParams )**

## 11.1.21 RTOS Library – memory reservation functions

RtosLib contains memory reservation functionality for reserving memory without starting RTOS. A setup can use this function to minimize the number of reboots required after installation to start a RTOS.

---

### 11.1.21.1 RtosSetMemoryConfigurationA

This function does install or remove memory reservation.

```
UINT32 RtosSetMemoryConfigurationA (  
    const CHAR*    szParams  
    const CHAR*    szConfigFileName )
```

#### Parameter

*szParams*

[in] Configuration parameter:  
“-a” for install or update reservations (requires config file)  
“-u” for uninstall reservations (will never ask for reboot)  
“-ur” for uninstall reservations (will ask for reboot when required)  
“-V” for validate reservations (requires config file)

*szConfigFileName*

[in] Config file – see comment for details

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The function does not require RtosLibInit to be called.

---

### 11.1.21.2 RtosSetMemoryConfigurationW

This function does install or remove memory reservation. See RtosSetMemoryConfigurationA for details.

```
UINT32 RtosSetMemoryConfigurationW (  
    const WCHAR*   wszParams  
    const WCHAR*   wszConfigFileName )
```

## 11.1.22 RTOS Library – virtual I/O (VIO) functions

RtosLib VIO functions allow reading and, if supported by the counterpart, writing to a virtual channel. Such a virtual channel is used by each OS for message output and (if supported) command input. The PuTTY based debug console is using this interface.

---

### 11.1.22.1 RtosVioCreate61

This function creates a handle to read or write to a VIO channel.

```
UINT32 RtosVioCreate61 (
    UINT32          dwChannelIndex
    BOOL           blsPrimary
    RTOSLIB_HANDLE* phObject )
```

#### Parameter

##### *dwChannelIndex*

[in] Channel index (0, 1, ...) – default OS channel equals the OS id.

##### *blsPrimary*

[in] Virtual I/O data are transferred between the primary OS, which usually is Windows, and the secondary OS (SOS), which usually is the RTOS.

##### *phObject*

[in] Pointer to receive a RTOSLIB\_HANDLE object (VIO).

#### Return

RTE\_SUCCESS on success and an error-code on failure.

#### Comment

The function does not require RtosLibInit to be called.

---

### 11.1.22.2 RtosVioClose

This is a synonym for RtosObjectClose. See RtosObjectClose for details.

```
UINT32 RtosVioClose (
    RTOSLIB_HANDLE hObject )
```

---

### 11.1.22.3 RtosVioRead

This is a synonym for RtosObjectRead. See RtosObjectRead for details.

```
UINT32 RtosVioRead(
    RTOSLIB_HANDLE hObject,
    UINT8          *pBuffer,
    UINT32         dwBufferSize,
    UINT32         *pdwBytesRead,
    UINT32         dwTimeoutMs )
```

---

### 11.1.22.4 RtosVioWrite

This is a synonym for RtosObjectWrite. See RtosObjectWrite for details.

```
UINT32 RtosVioWrite(
    RTOSLIB_HANDLE hObject,
    const UINT8    *pBuffer,
    UINT32         dwBufferSize,
    UINT32         *pdwBytesWritten,
    UINT32         dwTimeoutMs )
```

## **11.2 RTOS Library example applications**

Shipped with VmfWin are some example applications for VxWorks and Windows CE which show how to use the RTOS library in a user application. The example applications may serve as a starting point.

The following examples are extracted from the ACONTIS CeWin product (Windows CE + Windows XP)

- ...\Examples\RtosLib\CeWin\WinCE
  - Windows CE example applications
- ...\Examples\RtosLib\CeWin\Windows
  - Windows XP example applications

The following examples are extracted from the ACONTIS VxWin product (VxWorks + Windows XP)

- ...\Examples\RtosLib\VxWin\VxWorks
  - VxWorks example applications
- ...\Examples\RtosLib\VxWin\Windows
  - Windows XP example applications

## 11.3 RTOS Library – compatibility issues for VxWin and CeWin 3.5

Most of the RTOS Library functions are compatible with VxWin 3.5 and CeWin 3.5.

### 11.3.1 Compatibility mode

To enable compatibility mode within existing applications some macros have to be defined prior to including RtosLib.h:

- a) VxWin 3.5, CeWin 3.5 – Windows XP side of the application

```
#define RTOSLIB_API_VERSION 35
#define RTOSLIB_WIN
#include <RtosLib.h>
```

- b) VxWin 3.5 – VxWorks side of the application

```
#define RTOSLIB_API_VERSION 35
#define RTOSLIB_VXWORKS
#include <RtosLib.h>
```

- c) VxWin 3.5 – Windows CE side of the application

```
#define RTOSLIB_API_VERSION 35
#define RTOSLIB_WINCE
#include <RtosLib.h>
```

### 11.3.2 Initialization

Prior to using the RTOS Library it has to be initialized. See section 11.1.3 for more information. Existing applications have to be adjusted appropriately.

### 11.3.3 Time/date and timezone synchronization

- RtosSetClockMaster()  
Dynamically setting the clock master (time and date) is deprecated. Setting a clock master for an OS has to be done statically by configuration settings.
- RtosStartTimeSync()  
This function has been replaced by RtosTimeSyncStart().  
The priority of the started task can be set using system functions. The default is lowest priority.
- RtosStopTimeSync()  
This function has been replaced by RtosTimeSyncStop ().

### 11.3.4 Function SetOutputStream

The function SetOutputStream does not exist in the new RTOS Library. Instead the new function RtosSetOutputPrintf (see chapter 11.1.10.2) should be used.

## 12 EC-Master (MAC-ID) – Licensing

### 12.1 General

This product requires a valid license to be run. If your project also requires a MAC-ID licensed EC-Master it's possible to use the MAC-ID based license of EC-Master to get this product licensed. Finally one MAC-ID based licence is required to get both products (RTOS VM and EC-Master) licensed.

### 12.2 Required steps

Add the following function call of the EC-Master library o your project and recompile it.

```
- ecatSetLicenseKey( "__LICENSE_KEY__" );
```

No further actions are needed.

For additional information please check-out the specific EC-Master manual.

**Hint:** It's recommended that the call of `ecatSetLicenseKey` is done somewhere at start-up of the application.

## 13 CodeMeter – Licensing

### 13.1 Development vs. Runtime

This product requires a valid license to be run. The development version requires an USB dongle containing a valid development license. Together with your ordered runtime licenses you will get a software based solution.

### 13.2 License Software “CodeMeter”

Both licenses, the hardware (USB-Dongle) based development license and the software based runtime license, require the “CodeMeter Runtime-Kit” to be recognized by software. This Kit will be automatically installed by our product setup.

It can alternatively be downloaded from “[http://wibu.com/download\\_user.php](http://wibu.com/download_user.php)”.

### 13.3 Install a license

The “CodeMeter Runtime-Kit” was installed by the product setup. Additionally a license must be installed.

#### 13.3.1 License received on USB dongle

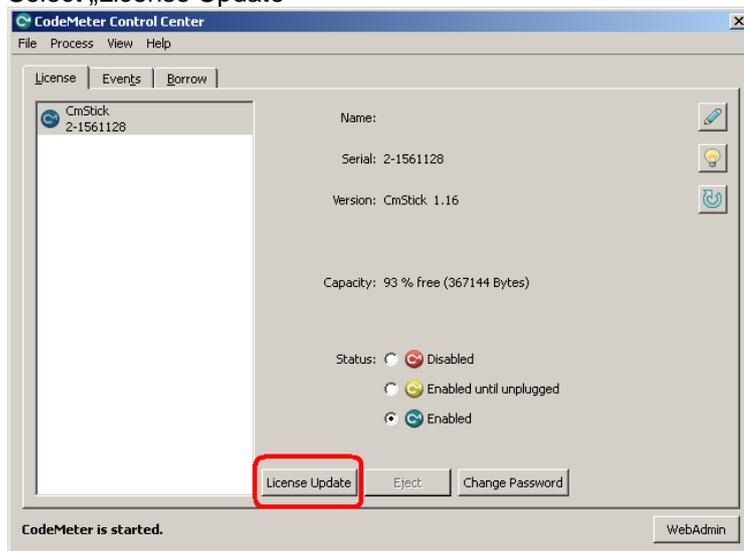
If you received an USB-Dongle including the license you can just plug the dongle into the PC and as soon as it is recognized the „CodeMeter Control Center“ will change its icon from grey to green.

#### 13.3.2 License required for existing USB dongle

In case you already own a USB dongle and want to add a license you need to create a license request and send it to your support contact to receive a license update.

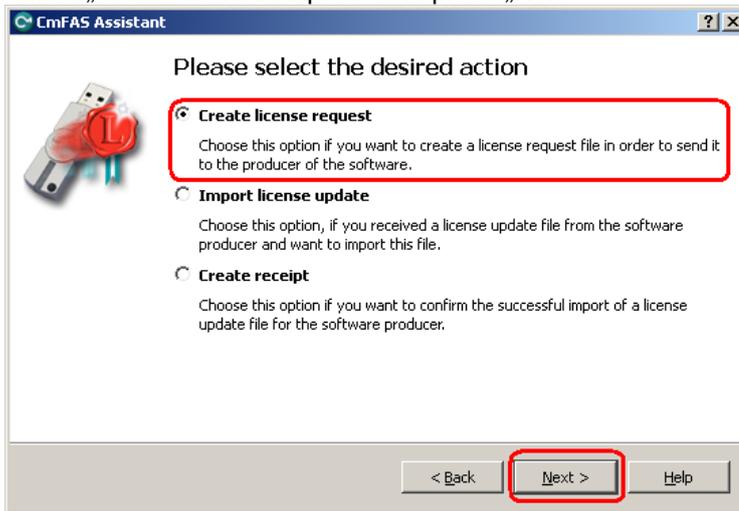
##### 13.3.2.1 Generate license request

- Plug in your dongle
- Open „CodeMeter Control Center“  and select the dongle you want to receive a new license or a license update for.
- Select „License Update“

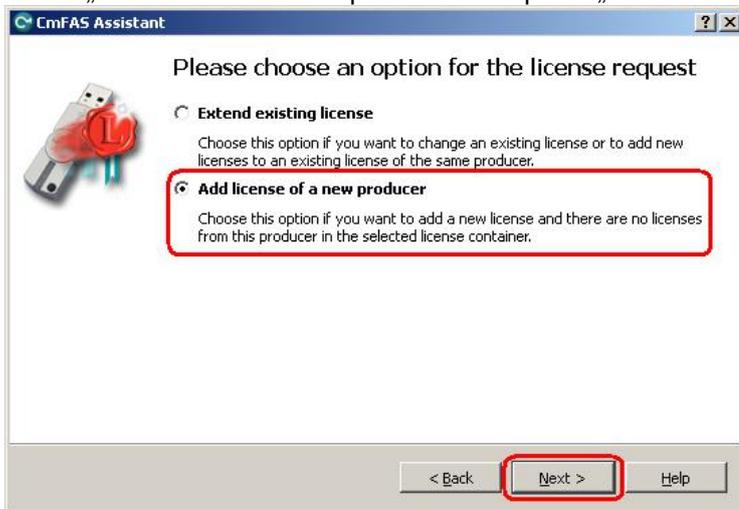


- Press „Next“

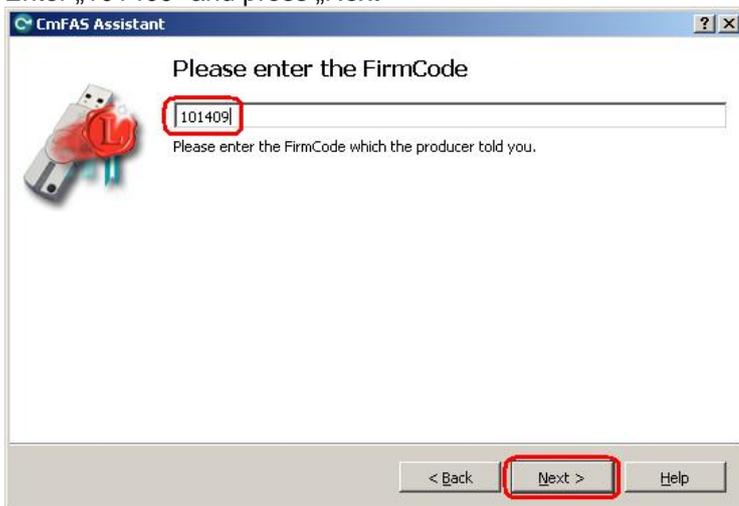
- Select „Create license request“ and press „Next“



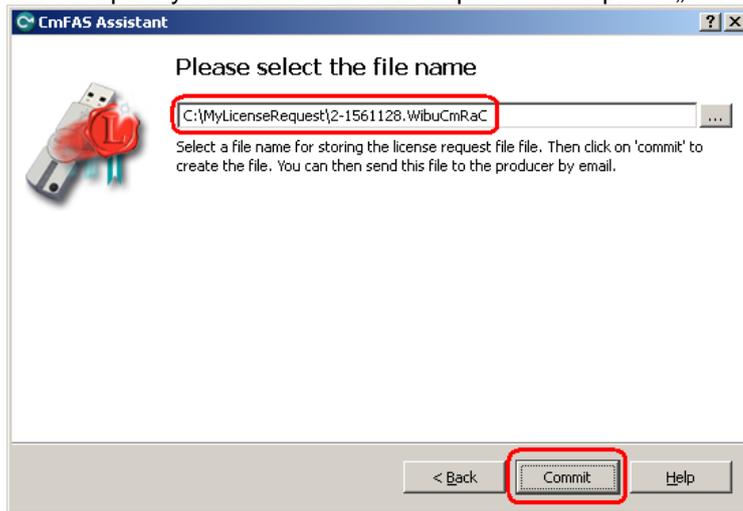
- Select „Add license of a new producer“ and press „Next“



- Enter „101409“ and press „Next“



- Select a path you want to save the request to and press „Commit“



Please send the created file to your support contact requesting a new license.

### 13.3.2.2 Import a new license or a license update

If your support contact sent you a license update file (“YourFilename.WibuCmRaU”) you may install it using the “CodeMeter Control Center”.

- Plug in your dongle
- Open „CodeMeter Control Center“ 
- Drop the file “YourFilename.WibuCmRaU” from the File-Explorer into the “CodeMeter Control Center”



### 13.3.3 License for software based solution

#### 13.3.3.1 Replace files for software solution

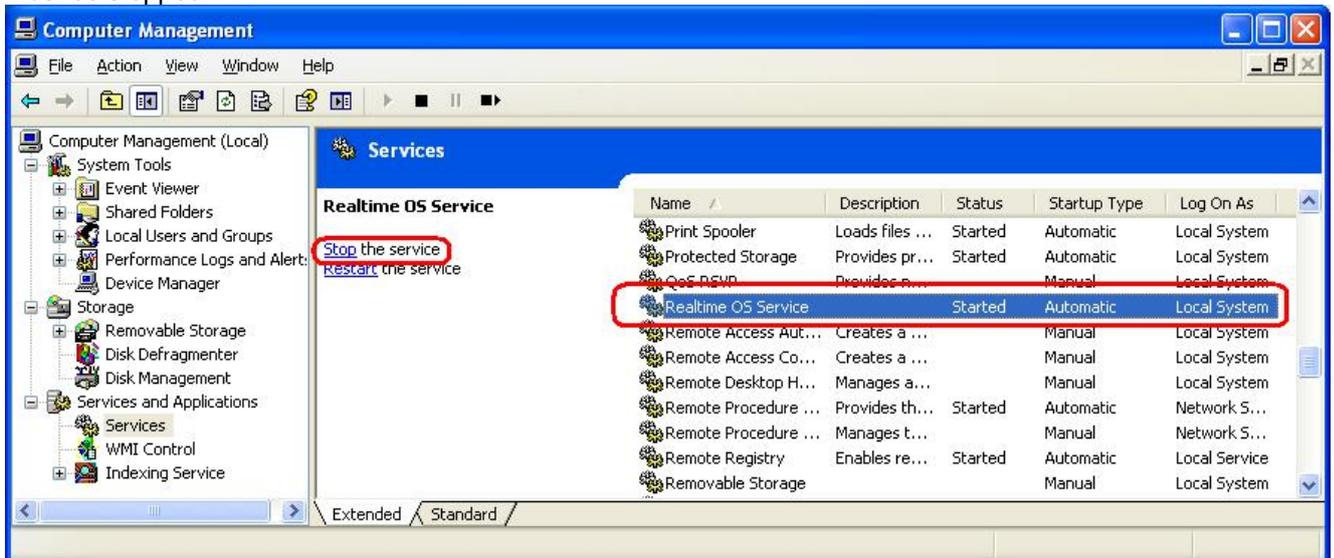
For the software based solution you will receive several files from your support

- a \*.wbb file
- file "RtosLib32.dll" and "RtosLib64.dll"

To replace the existing .dll files in your product installation directory with the new files

- any running RTOS and
- the "Realtime OS Service"

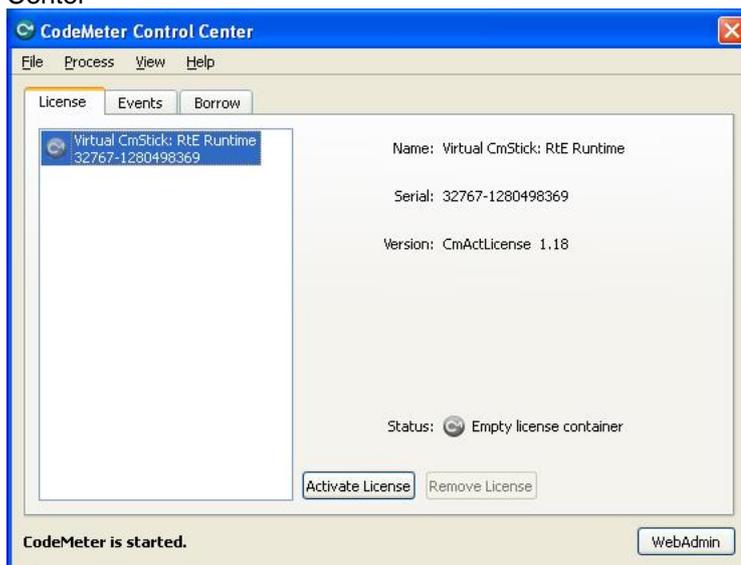
must be stopped.



After stopping any running RTOS and the "Realtime OS Service" you can replace the existing file(s) with the new file(s). Afterwards "Realtime OS Service" should be started again.

#### 13.3.3.2 Import empty license container

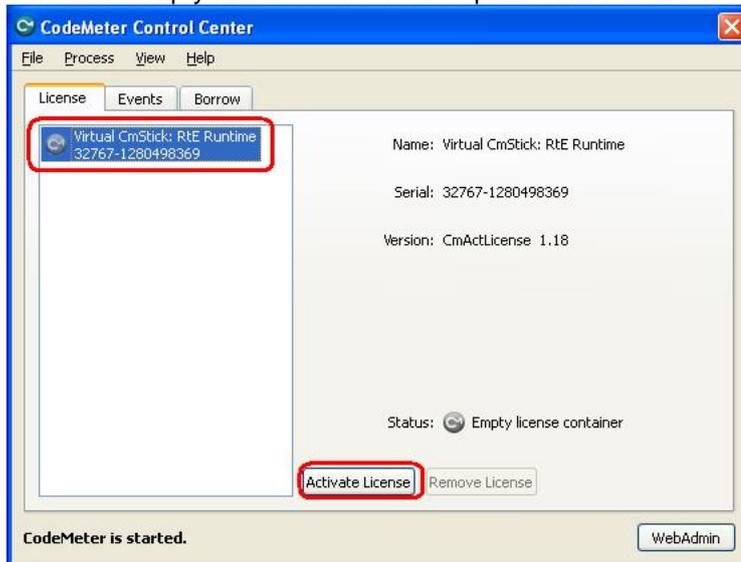
- Open „CodeMeter Control Center“
- Drop the file "FileFromSupport.wbb" from the File-Explorer into the "CodeMeter Control Center"



#### 13.3.3.3 Activate License

- Open „CodeMeter Control Center“

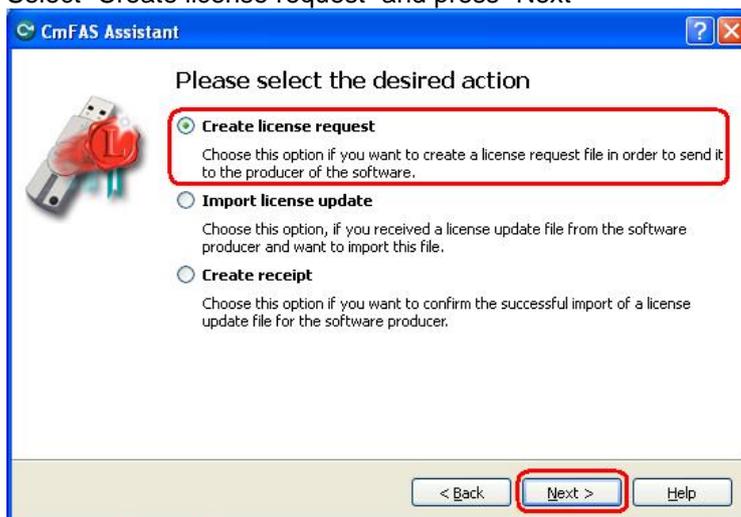
- Select the empty license container and press "Activate License"



- press "Next"



- Select "Create license request" and press "Next"



- Select a path you want to save the request to and press „Commit“

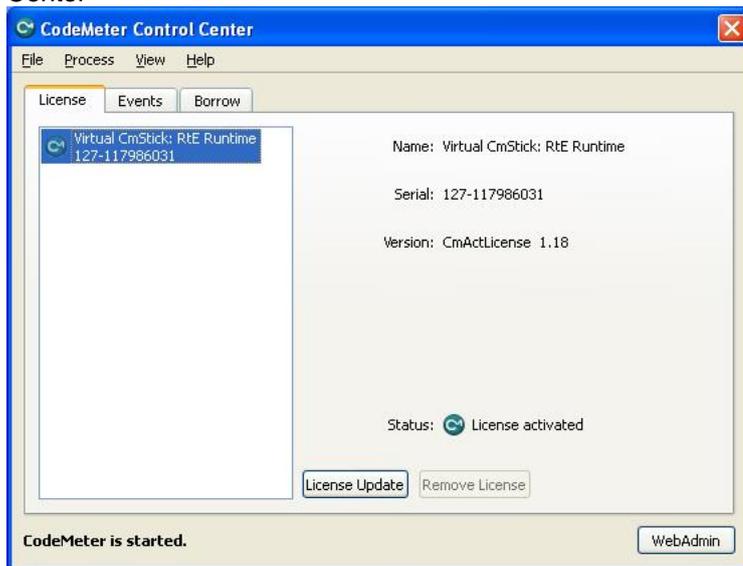


Please send the created file to your support contact requesting a new license.

#### 13.3.3.4 Import a new license or a license update

If your support contact sent you a license update file (“YourFilename.WibuCmRaU”) you may install it using the “CodeMeter Control Center”.

- Open „CodeMeter Control Center“ 
- Drop the file “YourFilename.WibuCmRaU” from the File-Explorer into the “CodeMeter Control Center”



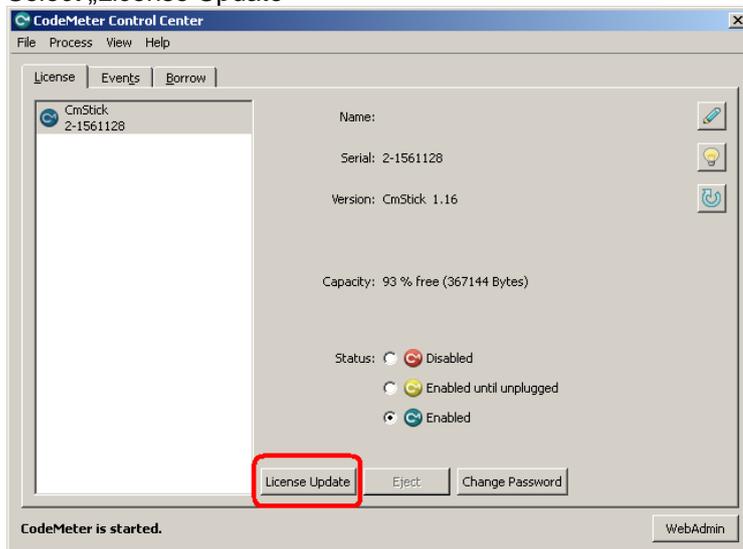
Now you should have a valid license.

## 13.4 Update a license

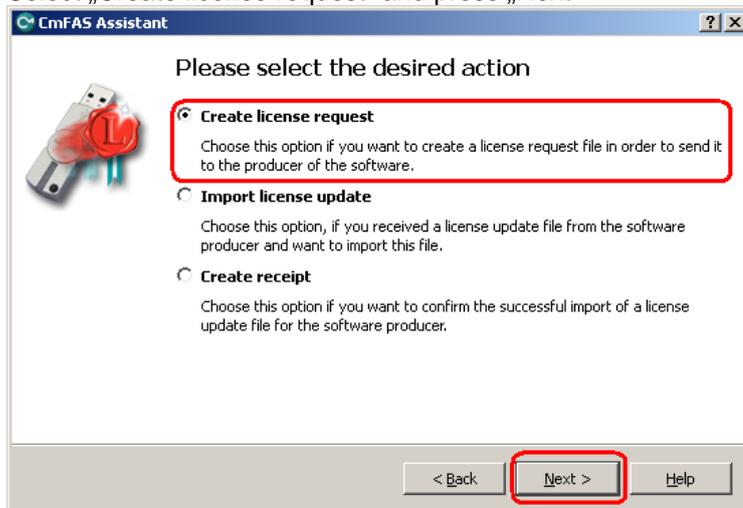
In this case you need to create a license request and send it to your support contact to receive a license update.

### 13.4.1 Generate license request

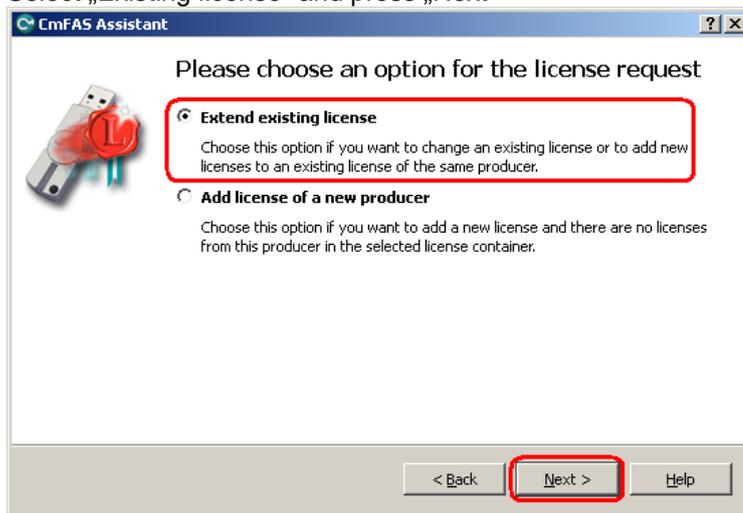
- Open „CodeMeter Control Center“  and select the dongle you want to receive a license update for.
- Select „License Update“



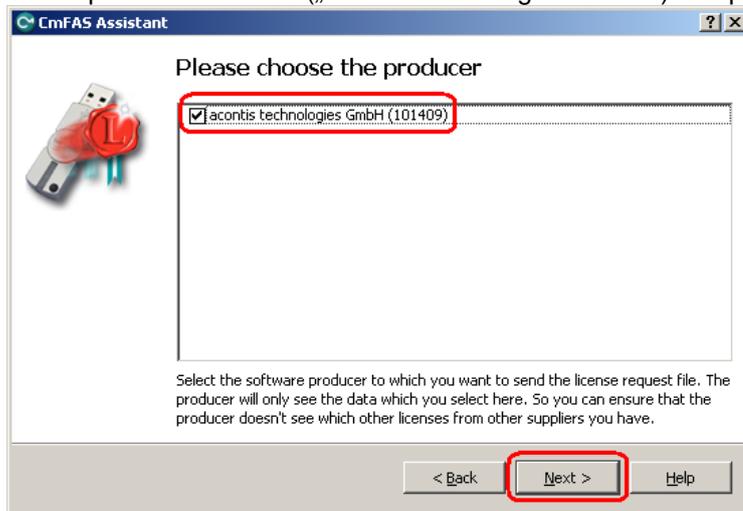
- Press „Next“
- Select „Create license request“ and press „Next“



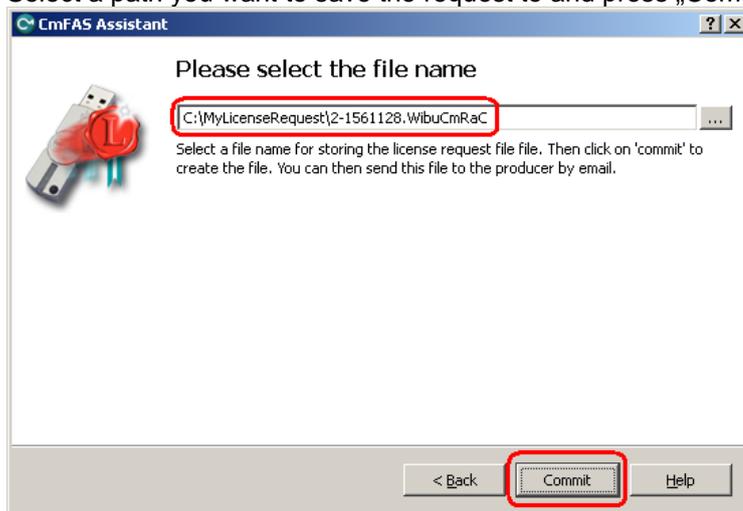
- Select „Existing license“ and press „Next“



- Select producer 101409 („acontis technologies GmbH“) and press „Next“



- Select a path you want to save the request to and press „Commit“



Please send the created file to your support contact requesting a new license.

### 13.4.2 Import a license update

If your support contact sent you a license update file ("YourFilename.WibuCmRaU") you may install it using the "CodeMeter Control Center".

- Open „CodeMeter Control Center“ 
- Drop the file "YourFilename.WibuCmRaU" from the File-Explorer into the "CodeMeter Control Center"



### 13.5 Sharing a License

While a runtime license is normally configured to be used only “local” a development license can be used over the network. This allows to plug the dongle into a license server and use its license(s) from another PC over the network.

A license which can only be used locally:



The screenshot shows the CodeMeter WebAdmin interface. The 'Licenses' tab is selected, and the 'CmContainer' is set to '127-117986031'. Below this, a table lists licenses for '5010 | acontis technologies GmbH'. The table has columns for Product Code, Name, Unit Counter, Expiration Time, Activation Time, and License Quantity. One license is listed with Product Code '4294901760', Name 'RtE Runtime', and License Quantity 'local', which is circled in red.

Product Code	Name	Unit Counter	Expiration Time	Activation Time	License Quantity
<a href="#">4294901760</a>	RtE Runtime	n/a	n/a	n/a	local

Licenses which can be used over the network:



The screenshot shows the CodeMeter WebAdmin interface. The 'Licenses' tab is selected, and the 'CmContainer' is set to '2-1561128'. Below this, a table lists licenses for '101409 | acontis technologies GmbH'. The table has columns for Product Code, Name, Unit Counter, Expiration Time, Activation Time, and License Quantity. Two licenses are listed: '16975104' for 'VxWin 3.5 Development' and '17039616' for 'VxWin 4.1 Development', both with a License Quantity of '1', which are circled in red.

Product Code	Name	Unit Counter	Expiration Time	Activation Time	License Quantity
<a href="#">16975104</a>	VxWin 3.5 Development	n/a	n/a	n/a	1
<a href="#">17039616</a>	VxWin 4.1 Development	n/a	n/a	n/a	1

#### 13.5.1.1 Single PC

The “CodeMeter Runtime-Kit” was installed by the product setup. You can just plug the dongle into the PC and as soon as it is recognized the „CodeMeter Control Center” will change its icon from grey to green.

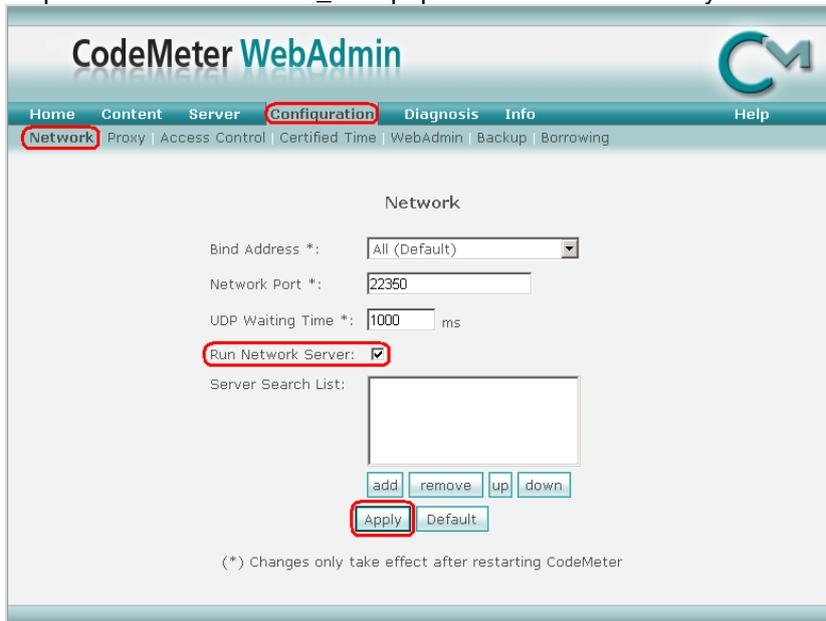
#### 13.5.1.2 Two PCs with dongle on target

If you have a PC with the runtime environment (target) where you want to plug the dongle in and a development PC (host) you need the “CodeMeter Runtime-Kit” being installed on the target PC. This should already be done by the product setup. You can just plug the dongle into the target PC and as soon as it is recognized the „CodeMeter Control Center” will change its icon from grey to green.

### 13.5.1.3 Two PCs with dongle on host

If you have a PC with the runtime environment (target) and a development PC (host) where you want to plug the dongle in you need the “CodeMeter Runtime-Kit” being installed on both PC. This should already be done by the product setup if you used it to install runtime components on the target and development components on the host.

Alternatively the “CodeMeter Runtime-Kit” can be downloaded from “[http://wibu.com/download\\_user.php](http://wibu.com/download_user.php)” and installed manually.



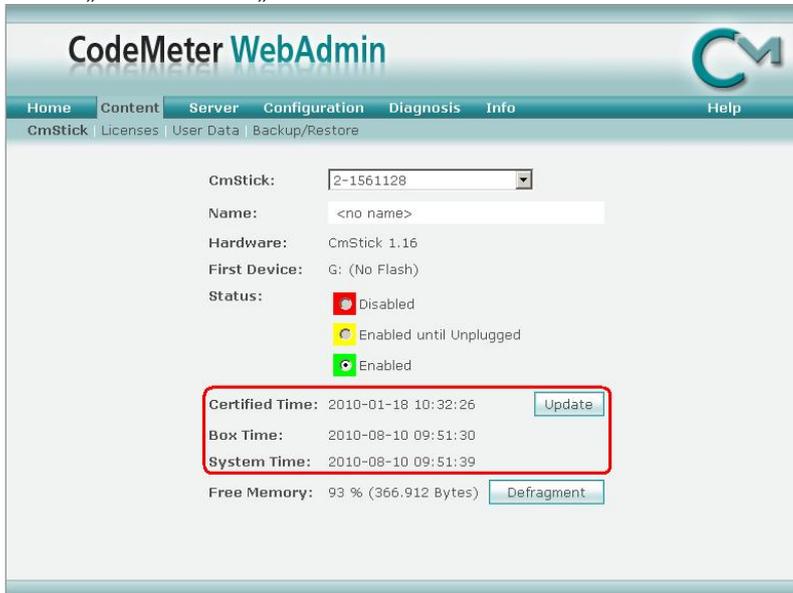
On your host PC you must configure the “CodeMeter Control Center” to act as license server:

- Open “CodeMeter Control Center”
- Click “WebAdmin”
- Select “Configuration” – “Network”
- Enable “Run Network Server”
- Press “Apply”

## 13.6 Troubleshooting

If a license error occurs in spite of a valid license possibly the Dongle has an incorrect time. In this case the time has to be updated:

- Open „CodeMeter Control Center“ 
- Select „WebAdmin“
- Select „Content“ and „CmStick“



- Press "Update" to update the time. A connection to the internet will be required.

## 14 RTOSWin OEM Branding

### 14.1 General

By default, the ACONTIS RTOS-VM runtime modules show up with ACONTIS product branding. The RTOSWin solution provider or RTOSWin OEM customer may want to use his own brand labels instead of the ACONTIS branding.

The ACONTIS RTOS-VM runtime modules provide mechanisms to customize manufacturer specific product information. This can be achieved by some RTOS-VM specific functions and also by common Microsoft Windows configuration possibilities.

### 14.2 Module specific Branding

#### 14.2.1 RtosDrv.sys

The driver RtosDrv.sys hosts several product information like company name, registry base path, product support internet address and product name. These informations are configured in the file "RtosDrv.inf" which is used to install the driver. After the installation these values are part of the driver registry values and provided by the RtosDrv to several RTOS-VM applications.

These values can either be modified in the "RtosDrv.inf" file or later in the registry.

Because the registry path ( HKLM\SYSTEM\CurrentControlSet\Enum\Root\SYSTEM\xxxx\Device Parameters\Product , where xxxx is a system specific number ) contains an computer specific element this way is only applicable on identical configured systems.

Within RtosDrv.inf the product specific information can be found in the last block at the end of the file. Additional information about the driver containing the company name (the second last block) might also be adapted.

Changing a file will invalidate the given driver signature and Windows will warn about an unsigned driver at installation time. This can be solved with a new signature. How to sign a driver and the limitations is not part of this document. Please contact ACONTIS support in case a new signature shall be generated.

#### 14.2.2 RtosVnet.sys and RTOS\_xxx.inf

Information about the driver manufacturer can be found in the last block of the file and might be adapted. Changing a file will invalidate the given driver signature and Windows will warn about an unsigned driver at installation time. This can be solved with a new signature. How to sign a driver and the limitations is not part of this document.

#### 14.2.3 RtosService.exe

Name and descriptions can be changed using the Windows Service Controller API to manually register / un-register the service instead of calling "RtosService.exe INSTALL" and "RtosService.exe UNINSTALL". Additionally, most Setup programs, like for example InstallShield, are providing functionalities to register and un-register services using custom name and description.

The application embedded icon (displayed in the Explorer) can be replaced using a binary resource editor tool.

#### 14.2.4 RtosControl.exe

The application's embedded icon (displayed by the Windows Explorer) can be replaced using a binary resource editor tool. Additionally, RtosControl.exe supports replacement of images shown in dialogs by adding a file to the applications directory:

Dialog	Description	Required filename
Taskbar / About dialog	Application icon	RTOSControl.ico
Evaluation dialog	Company logo	RTOSControl02.bmp
Evaluation dialog	Product logo	RTOSControl03.bmp

#### 14.2.5 UploadRtos.exe (RTE <=4.x) or RtosUpload.exe (RTE >=5.x)

The application's embedded icon (displayed by the Windows Explorer) can be replaced using a binary resource editor tool. Most error message texts can be changed in

- UploadRtos.dll (RTE <=4.x)
- RtosLib32.dll and RtosLib64.dll (RTE >=5.x)

using a resource editor. As an alternative a customer written Uploader.exe could be used.

#### 14.2.6 RtosPnp.sys

The Windows Device Manager shows an ACONTIS specific icon for the Rtos device class, which can be changed. This icon is part of the RtosPnpInstaller.dll which can be found in windows\system32 directory and the Rtos-Inf directory. There are several possibilities to change the icon:

1. Use an resource editor to edit the dll and change the icon
2. Use an resource editor to edit the dll and change add an additional icon + change the value of "Icon" which can be found twice in each RTOS\_xxx.inf.  
A value of "0" represents the first icon of the dll, the value "1" is reserved.
3. Change the value of "Icon" which can be found twice in each RTOS\_xxx.inf to a negative value.  
Negative values represent the system predefined icons. "-18" for example represents the "Unknown" icon (18).

Changing a file will invalidate the given driver signature and Windows will warn about an unsigned driver at installation time. This can be solved with a new signature. How to sign a driver and the limitations is not part of this document.

To update a system where RtosPnpInstaller.dll is already installed with a new icon number you must change the value of "Icon" in at least one of the RTOS\_xxx.inf files, then right-click on the modified file and choose "install". After that the RTOS device class should use the new icon (perhaps a reboot might additionally be required).

## 15 Appendix A – Platforms and performance

### 15.1 Real Time behaviour and the RTOS-VM

When running on top of the RTOS-VM a RTOS can interrupt every Windows XP application and device driver at any time when running in shared mode. In exclusive mode the RTOS runs completely independent from and fully parallel to Windows.

This guarantees real-time performance and deterministic behaviour for the RTOS.

However, one must nonetheless be careful about using PC cards that make long DMA (direct memory access) transfers, for DMA transfers can considerably increase interrupt latency time.

For example, consider a graphics card that wants to copy graphics data from PC memory into its own graphics memory. Normally, the CPU has access to the PC memory, but if the graphics card adapter requests a DMA transfer, the CPU gives up control of the bus. While the graphics card monopolizes the bus (to copy data into its own RAM), the CPU must wait until the DMA transfer has completed. While the CPU is waiting to reacquire the bus, it can neither respond to interrupts nor execute code in the usual fashion.

This problem is so tightly bound to an inflexible aspect of the hardware that the RTOS-VM can do nothing to effectively relieve the situation.

### 15.2 Platform Evaluation

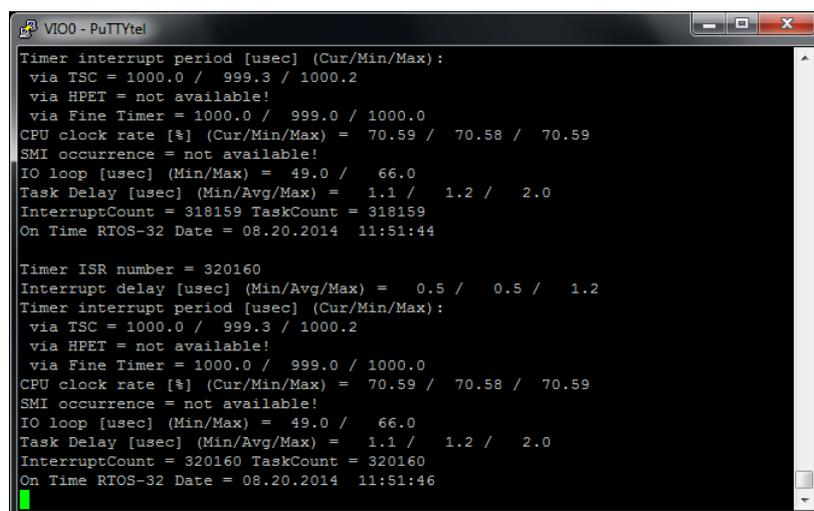
Prior to using the RTOS-VM you have to verify that the PC platform where the RTOS is supposed to run will fulfil your real-time requirements. While keeping your real-time criteria in mind, you must determine that the interrupt latency times in your system are satisfactory. If it so happens that the latency is greater than you expect, you should investigate to determine if this is being caused by DMA operations. And if DMA is indeed the problem, you should then try to discover which hardware device is causing it, for in many cases the DMA-problem can be solved.

To evaluate the real-time behaviour of your PC platform you need to use two types of tools: one to generate load on Windows XP and a second to measure real-time response in the RTOS.

Of the readily available tools for creating loads in Windows XP, a good one is PassMark's "BurnIn Test," a shareware tool. Refer to <http://www.passmark.com/products/bit.htm>. With this tool you can, for example, specify which hardware device to test, how long a test should last, and so on.

To measure real-time latency, you will have to use a test application running on the RTOS. Most RTOS-VM based solution will ship such a tool called "Realtimedemo":

- RTOS-32: Start image "Realtimedemo.bin"
- CE: Call "Realtimedemo" from shell
- VxWorks: Call "demoStart" from shell



```
VIO0 - PuTTYtel
Timer interrupt period [usec] (Cur/Min/Max):
  via TSC = 1000.0 / 999.3 / 1000.2
  via HPET = not available!
  via Fine Timer = 1000.0 / 999.0 / 1000.0
CPU clock rate [%] (Cur/Min/Max) = 70.59 / 70.58 / 70.59
SMI occurrence = not available!
IO loop [usec] (Min/Max) = 49.0 / 66.0
Task Delay [usec] (Min/Avg/Max) = 1.1 / 1.2 / 2.0
InterruptCount = 318159 TaskCount = 318159
On Time RTOS-32 Date = 08.20.2014 11:51:44

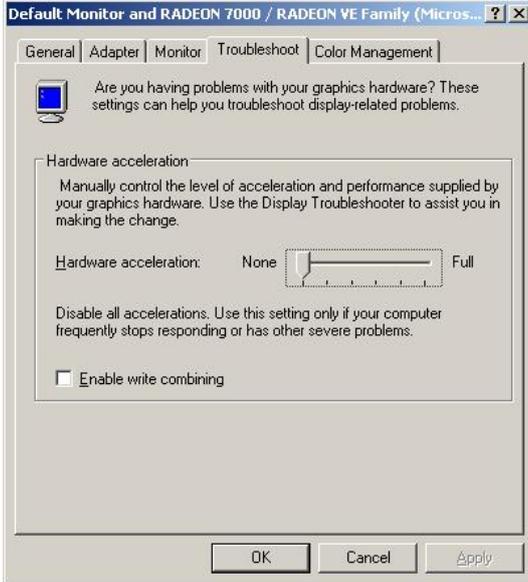
Timer ISR number = 320160
Interrupt delay [usec] (Min/Avg/Max) = 0.5 / 0.5 / 1.2
Timer interrupt period [usec] (Cur/Min/Max):
  via TSC = 1000.0 / 999.3 / 1000.2
  via HPET = not available!
  via Fine Timer = 1000.0 / 999.0 / 1000.0
CPU clock rate [%] (Cur/Min/Max) = 70.59 / 70.58 / 70.59
SMI occurrence = not available!
IO loop [usec] (Min/Max) = 49.0 / 66.0
Task Delay [usec] (Min/Avg/Max) = 1.1 / 1.2 / 2.0
InterruptCount = 320160 TaskCount = 320160
On Time RTOS-32 Date = 08.20.2014 11:51:46
```

Even on slow Intel Atom or Celeron systems that run with a nominal of a 500 MHz CPU, the interrupt latency should never exceed approximately 30 to 40 microseconds. Using such a test application together with a tool like "BurnIn Test", you can evaluate your platform, disclosing any devices which may be detracting from your system's real-time capabilities.

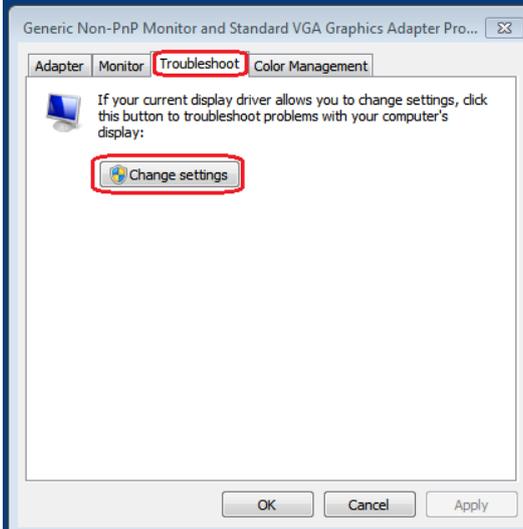
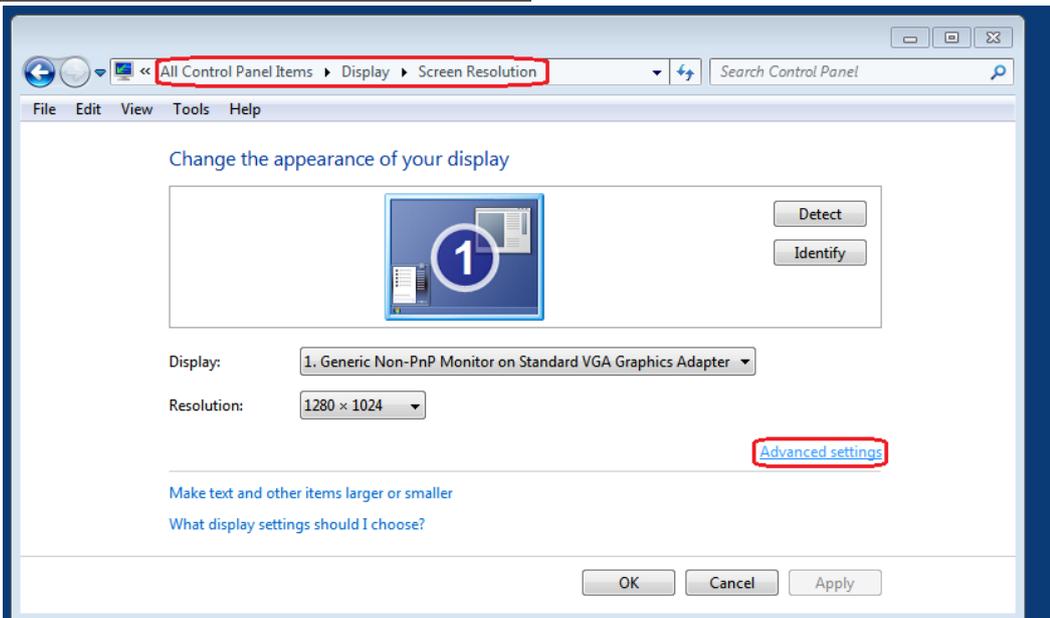
### 15.3 Reducing DMA latency problems

If you have determined that DMA is undermining your system's ability to serve as a real-time platform, the following list suggests a number of things you can do:

- Disable 3D hardware acceleration in your graphic settings.

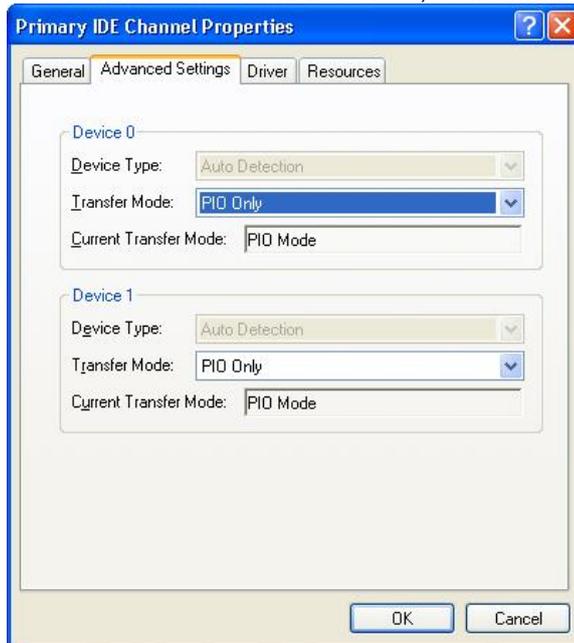


XP:

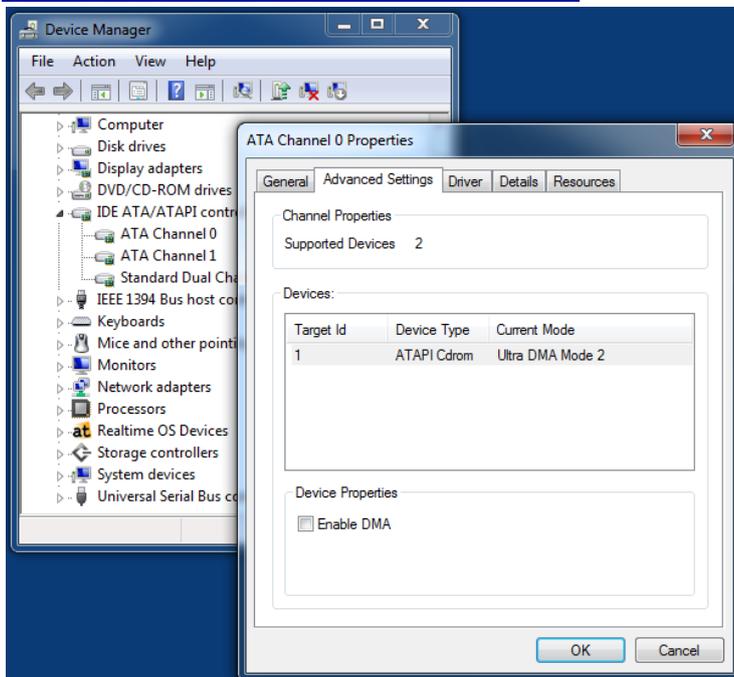


W7:

- Change other settings of your graphics card, i.e., reduce the resolution and/or the number of displayed colours. In some cases, you may want to try a different graphics card or a different graphics card driver. You might try using a resolution of 640x480 and 8 bit colours, the least - demanding settings, to determine if a latency problem is truly related to the graphics card adapter.
- Try adjusting the advanced BIOS settings for the graphics card. Experiment with limiting the size or duration of DMA accesses.
- Try changing the settings of your network card adapter. If it uses DMA, try another network adapter that does not.
- Eliminate USB or IEEE 1394 devices that use extensive DMA transfers (e.g. USB hard disks or CDROM drives), or try changing their device driver settings.
- Use PIO transfer mode for IDE devices, as illustrated below:



XP:



W7:

## 15.4 CPU throttling

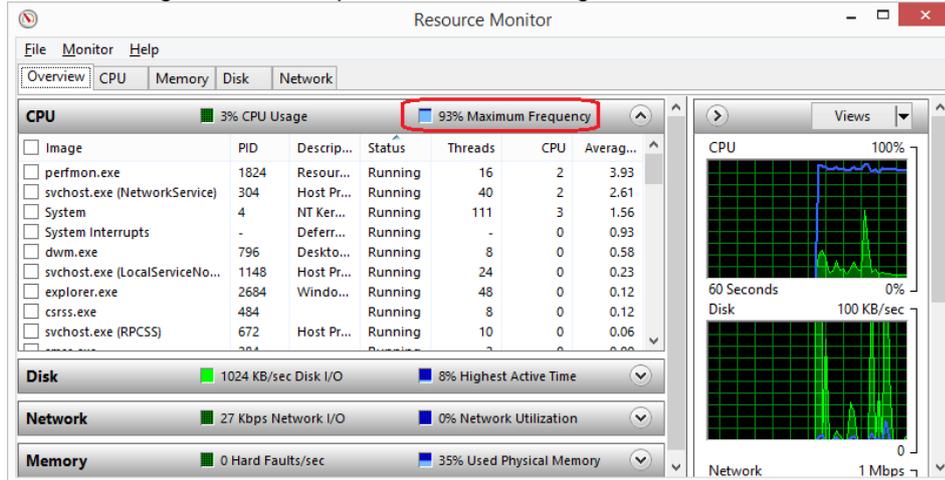
There are several CPU throttling mechanisms being used in modern CPUs for energy savings getting in conflict with real-time requirements:

- C1E (Enhanced Halt State)  
reduces the clock rate when the CPU goes idles.
- Intel® SpeedStep™ / “EIST” (“Enhanced Intel SpeedStep Technology”)  
changes the clock rate depending on the current configuration and workload.

### 15.4.1 Detection

If throttling becomes active on a PC depends on its CPU, BIOS, OS and RTOS configuration.

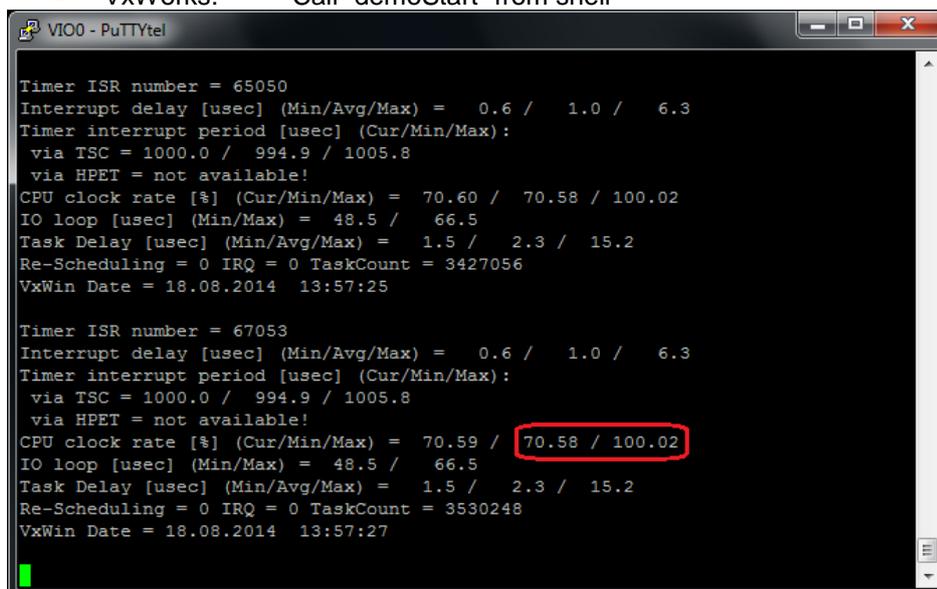
CPU throttling can for example be detected using Windows “Resource Monitor”:



When throttling is enabled the maximum frequency is fluctuating.

Throttling settings may be processor specific so the check should be done on the real-time assigned processors. Therefor “Realtimedemo” can be used which is part of RtE:

- RTOS-32: Start image “Realtimedemo.bin”
- CE: Call “Realtimedemo” from shell
- VxWorks: Call “demoStart” from shell



When throttling is enabled the CPU clock rate is fluctuating.

```

VIO0 - PuTTYtel
Timer ISR number = 123094
Interrupt delay [usec] (Min/Avg/Max) = 1.2 / 8.9 / 53.3
Timer interrupt period [usec] (Cur/Min/Max):
  via TSC = 499.6 / 485.6 / 1043.1
  via HPET = 999.2 / 954.7 / 1045.4
CPU clock rate [%] (Cur/Min/Max) = not available!
IO loop [usec] (Min/Max) = 16.6 / 127.7
Task Delay [usec] (Min/Avg/Max) = 4.6 / 18.2 / 98.2
Re-Scheduling = 0 IRQ = 0 TaskCount = 3857574
VxWin Date = 18.08.2014 14:03:01

Timer ISR number = 125095
Interrupt delay [usec] (Min/Avg/Max) = 1.2 / 8.9 / 53.3
Timer interrupt period [usec] (Cur/Min/Max):
  via TSC = 500.7 / 485.6 / 1043.1
  via HPET = 1001.4 / 954.7 / 1045.4
CPU clock rate [%] (Cur/Min/Max) = not available!
IO loop [usec] (Min/Max) = 16.6 / 127.7
Task Delay [usec] (Min/Avg/Max) = 4.6 / 18.2 / 98.2
Re-Scheduling = 0 IRQ = 0 TaskCount = 3910716
VxWin Date = 18.08.2014 14:03:03

```

Another indicator for throttling is when TSC differs from HPET period up to 50%.

### 15.4.2 How to disable

Windows XP

requires a CPU driver to add an optimized idle routine using “hlt” statement for throttling.

This means one of the following actions is sufficient to disable it:

- Disable Speed/Step and C1E by BIOS
- OR Disable processor driver
- OR Disable Speed/Step and C1E by VMF

Windows 7 and newer

includes a standard idle routine already using “hlt” - the optimized idle routine of the CPU driver uses the advanced “mwait” for throttling.

This means “Disable processor driver” is required and additionally one of the following actions:

- Disable Speed/Step and C1E by BIOS
- OR Disable Speed/Step and C1E by VMF

Please run RealtimeDemo again after changes to ensure throttling was successfully disabled.

#### 15.4.2.1 PC BIOS

The BIOS may contain CPU configuration options called

- “C1E Support”
- “Intel® SpeedStep™” or “EIST” (“Enhanced Intel SpeedStep Technology”)
- Something containing “throttling”

These options should be disabled.

#### 15.4.2.2 Processor Driver

The processor driver can be disabled in the Windows registry.

- Start “regedit”
- Navigate to “\HKLM\SYSTEM\CurrentControSet\services\Xxx” where “xxx” can be
  - o “Processor” (processr.sys)
  - o “intelppm” (intelppm.sys)
  - o “AmdK8” (amdk8.sys)
  - o “AmdPPM” (amdppm.sys)
 depending on the processor.
- Change the value of “Start” from to “4” to disable the driver.

A reboot is required to use the new configuration.

### 15.4.2.3 VMF

VMF can disable CPU throttling by reconfiguring each CPU used by RTOS. This will be done during RTOS startup. This feature is disabled on default but can be enabled using the following config file entry:

[Optimizations]

```
"ValueC1"=dword:1 ; omitted = 0 = disable, 1 = enable,  
; Other value = enable + force model (Exmpl: 0x617=CPU Family 6 Model 23)  
; Do no force a model without being prompted to do!
```

## 15.5 System Management Interrupt (SMI)

Another source of latency problems up to milliseconds are system management interrupts. They are hardly to detect without an external clock reference.

```
VIO0 - PuTTYtel  
Timer ISR number = 54485354  
Interrupt delay [usec] (Min/Avg/Max) = 0.6 / 1.6 / 28.7  
Timer interrupt period [usec] (Cur/Min/Max):  
  via TSC = 999.8 / 974.2 / 1027.0  
  via HPET = 999.8 / 966.3 / 1034.3  
CPU clock rate [%] (Cur/Min/Max) = 56.05 / 53.49 / 172.55  
SMI occurrence = 0  
IO loop [usec] (Min/Max) = 31.7 / 153.7  
Task Delay [usec] (Min/Avg/Max) = 2.5 / 5.7 / 54.7  
Re-Scheduling = 0 IRQ = 0 TaskCount = -1905191088  
VxWin Date = 20.08.2014 11:47:13
```

Newer CPUs contain a SMI counter being evaluated by the RealtimeDemo.

Possible reasons for SMI's are:

- "USB Legacy support"  
→ Can be disabled in most BIOS.
- Some Video driver use SMI.  
→ Use another driver for example Standard VGA.

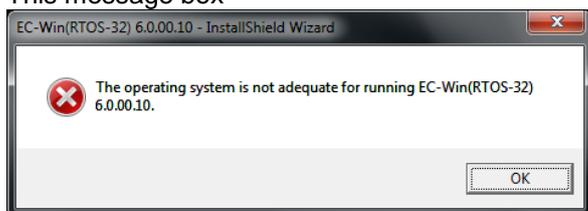
## 16 Appendix B – Troubleshooting

A variety of common errors can occur when starting up the RTOS for the first time. A few are mentioned below:

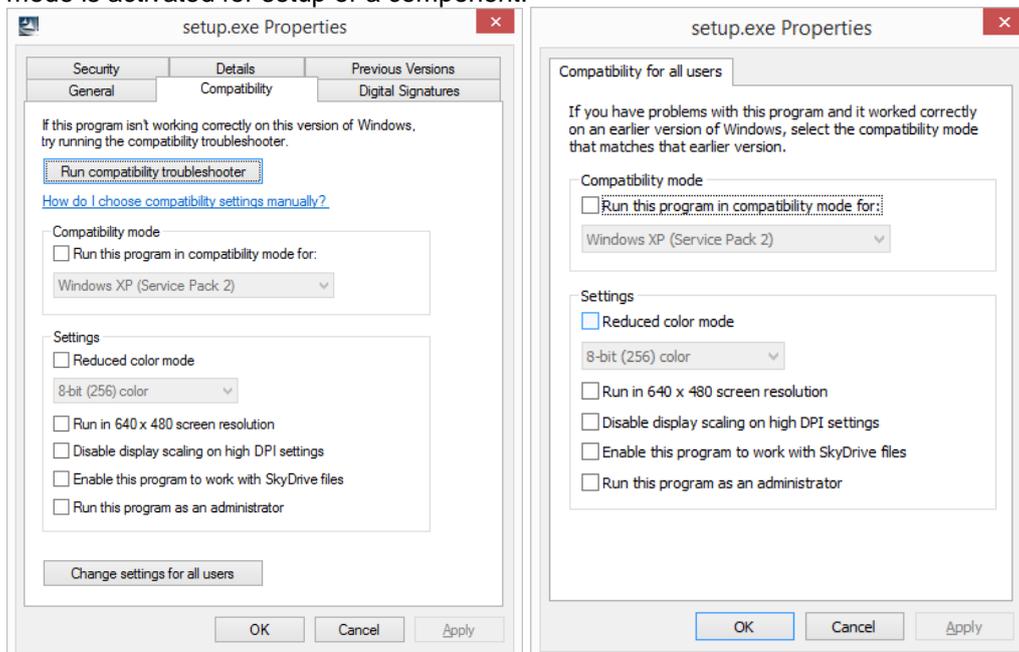
### 16.1 Setup fails

Setup creates a log-file in the Windows. Its name depends on the product and version. For example EC-Win(RTOS-32) might create the logfile “C:\Windows\EC-Win(RTOS-32) 6.0.00.07.LOG”. Please check this file for additional information.

- Check– especially on a new platform – if you’re using the latest RtE version.
- If installing from network or inside VMWare, first copy all files locally and run setup locally.
- On InstallScript engine errors sometimes it needs to be updated. RtE Setup requires version 10:  
<http://support.installshield.com/kb/files/Q108158/IsScript101.zip>  
<http://consumerdocs.installshield.com/selfservice/viewContent.do?externalId=Q108158&sliceId=1>
- Another InstallShield Scripting Runtime error might be soled by:  
<http://consumerdocs.installshield.com/selfservice/viewContent.do?externalId=Q108340&sliceId=1>
- This message box



can appear when running Setup on an OS not supported or when application compatibility mode is activated for setup or a component.



Please ensure compatibility mode is disabled for

- Setup.exe
- C:\Program Files (x86)\Common Files\InstallShield\Driver\10\Intel 32\IDriver.exe (64bit OS)
- C:\Program Files\Common Files\InstallShield\Driver\10\Intel 32\IDriver.exe (32 bit OS)

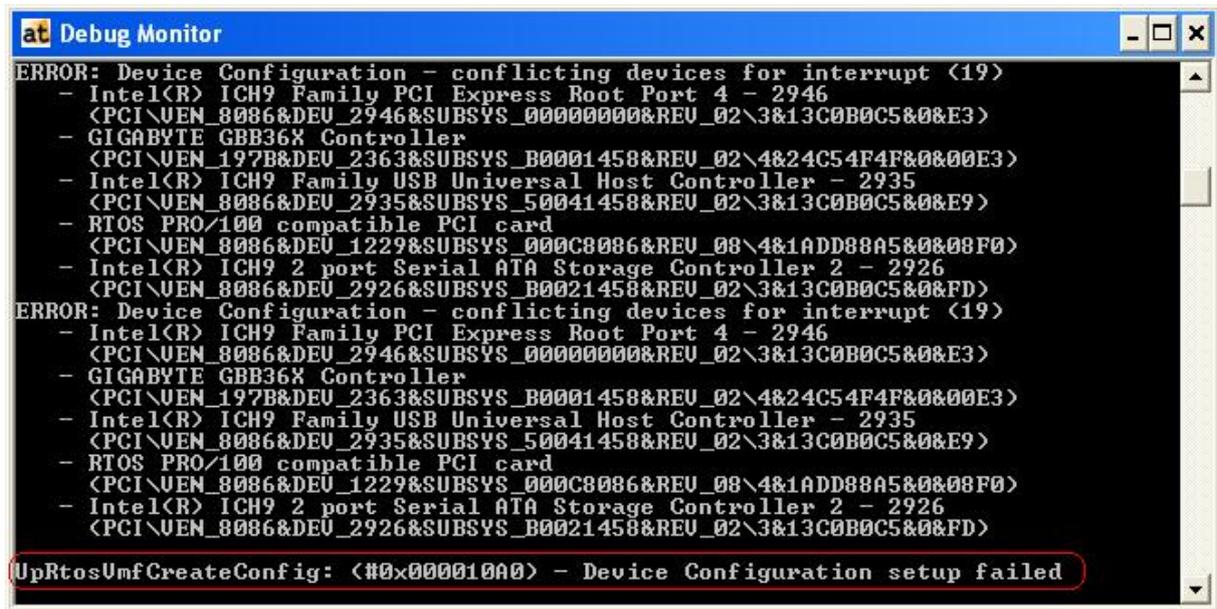
## 16.2 System does not boot

- 1) First it should be checked if system can start in Safe Mode.  
If this is possible probably a configuration change caused the problem.  
For example when multiple large shared memories are configured it is possible the system has not enough memory remaining to boot. Such a situation can be solved this way:
  - Start the system in Safe Mode
  - Change the SharedMemory configuration
  - Start the RTOS to update with the new configuration
  - Reboot the systemThe same can be done in case a memory configuration change caused the problem.
- 2) If Safe Mode also can't boot the Windows "repair console" should be used to rename the driver RtosDrv to prevent it from being loaded:  
`"ren c:\Windows\System32\drivers\RtosDrv.sys RtosDrv_org.sys"`  
The repair console can started from the repair boot menu or when booting from the installation disk. After renaming the driver the console can be left by calling "exit" to reboot the system.  
Please contact support in case this solved the problem.
- 3) If renaming the driver did not help call "bcdedit /enum {default}" from the repair console:  
When it shows a huge "badmemorylist" over multiple pages then please call  
`"bcdedit /deletevalue {default} badmemorylist"`.  
When it contains "firstmegabytepolicy = UseAll" then please call  
`"bcdedit /deletevalue {default} firstmegabytepolicy"`.  
After the call(s) succeeded the console can be left by calling "exit" to reboot the system.
- 4) In case nothing was helpful a startup repair should be done.  
On Windows XP this can be done booting from the installation disk and run "repair installation".  
On Windows 7 or newer run "Startup Repair". Sometimes this can be selected from the repair boot menu and sometimes it is required to boot from the Installation DVD.

## 16.3 Common startup problems

A variety of common errors can occur when starting up the RTOS for the first time. A few are mentioned below:

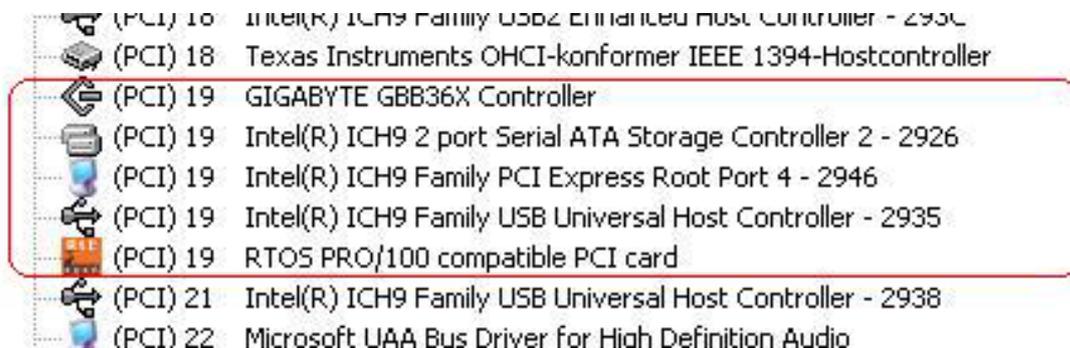
### 16.3.1 IRQ sharing with Windows (10A0)



```
at Debug Monitor
ERROR: Device Configuration - conflicting devices for interrupt (19)
- Intel(R) ICH9 Family PCI Express Root Port 4 - 2946
  (PCI\VEN_8086&DEV_2946&SUBSYS_00000000&REV_02\3&13C0B0C5&0&E3)
- GIGABYTE GBB36X Controller
  (PCI\VEN_197B&DEV_2363&SUBSYS_B0001458&REV_02\4&24C54F4F&0&0E3)
- Intel(R) ICH9 Family USB Universal Host Controller - 2935
  (PCI\VEN_8086&DEV_2935&SUBSYS_50041458&REV_02\3&13C0B0C5&0&E9)
- RTOS PRO/100 compatible PCI card
  (PCI\VEN_8086&DEV_1229&SUBSYS_000C8086&REV_08\4&1ADD88A5&0&08F0)
- Intel(R) ICH9 2 port Serial ATA Storage Controller 2 - 2926
  (PCI\VEN_8086&DEV_2926&SUBSYS_B0021458&REV_02\3&13C0B0C5&0&FD)
ERROR: Device Configuration - conflicting devices for interrupt (19)
- Intel(R) ICH9 Family PCI Express Root Port 4 - 2946
  (PCI\VEN_8086&DEV_2946&SUBSYS_00000000&REV_02\3&13C0B0C5&0&E3)
- GIGABYTE GBB36X Controller
  (PCI\VEN_197B&DEV_2363&SUBSYS_B0001458&REV_02\4&24C54F4F&0&0E3)
- Intel(R) ICH9 Family USB Universal Host Controller - 2935
  (PCI\VEN_8086&DEV_2935&SUBSYS_50041458&REV_02\3&13C0B0C5&0&E9)
- RTOS PRO/100 compatible PCI card
  (PCI\VEN_8086&DEV_1229&SUBSYS_000C8086&REV_08\4&1ADD88A5&0&08F0)
- Intel(R) ICH9 2 port Serial ATA Storage Controller 2 - 2926
  (PCI\VEN_8086&DEV_2926&SUBSYS_B0021458&REV_02\3&13C0B0C5&0&FD)
UpRtosUmfCreateConfig: (<#0x000010A0>) - Device Configuration setup failed
```

#### Causes:

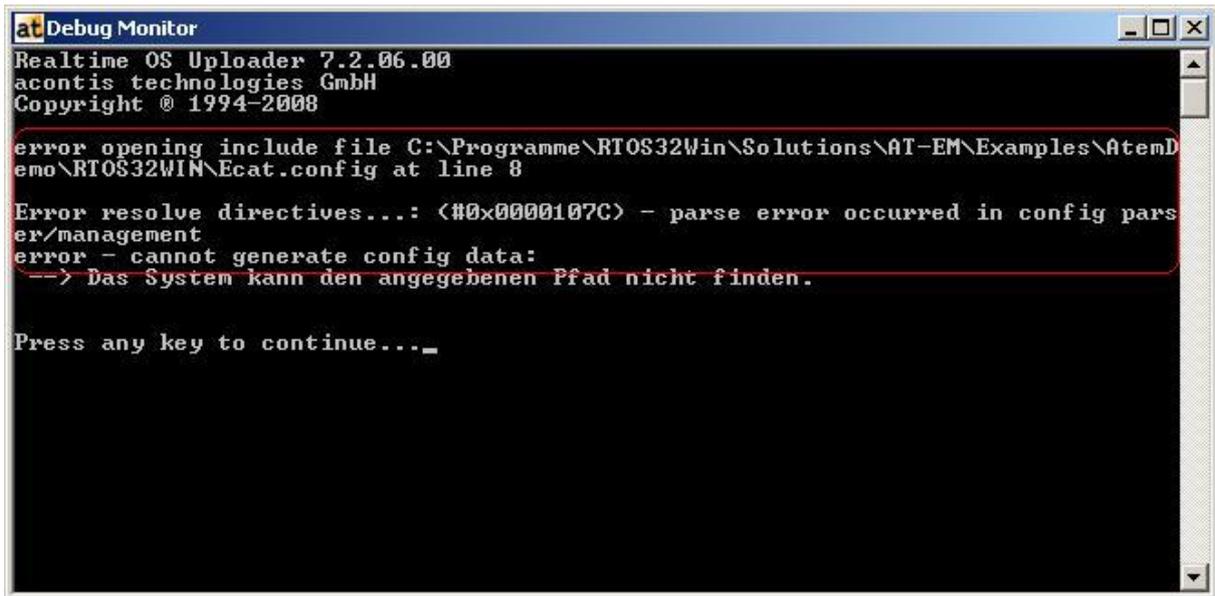
The IRQ of the Rtos-Device is shared with a Windows-Device. Rtos-Devices need an exclusive IRQ.



There are several alternatives to solve the problem:

- Put the PCI card into another slot to use another interrupt line, which hopefully doesn't conflict.
- A PCI express card supporting MSI's can be configured to use MSI.
- If the RTOS device will be polled the interrupt can be disabled.  
Attention: Every normal RTOS driver will require an interrupt. This is only an option in combination with driver software known not to require the interrupt.
- When the conflicting Windows device is not required it can be disabled by BIOS or Windows device manager.
- When it is known that the Windows device will never use its interrupt in can be added to the interrupt ignore list.  
Attention: This will lead to unpredictable behaviour case the device anyway uses the interrupt.

### 16.3.2 Error opening include file (107C)



```
Debug Monitor
Realtime OS Uploader 7.2.06.00
acontis technologies GmbH
Copyright © 1994-2008

error opening include file C:\Programme\RTOS32Win\Solutions\AT-EM\Examples\AtemD
emo\RTOS32WIN\Ecat.config at line 8

Error resolve directives...: <#0x0000107C> - parse error occurred in config pars
er/management
error - cannot generate config data:
--> Das System kann den angegebenen Pfad nicht finden.

Press any key to continue..._
```

#### Causes:

The path or filename is not correct. The include filename is invalid or it is not located in the selected path.

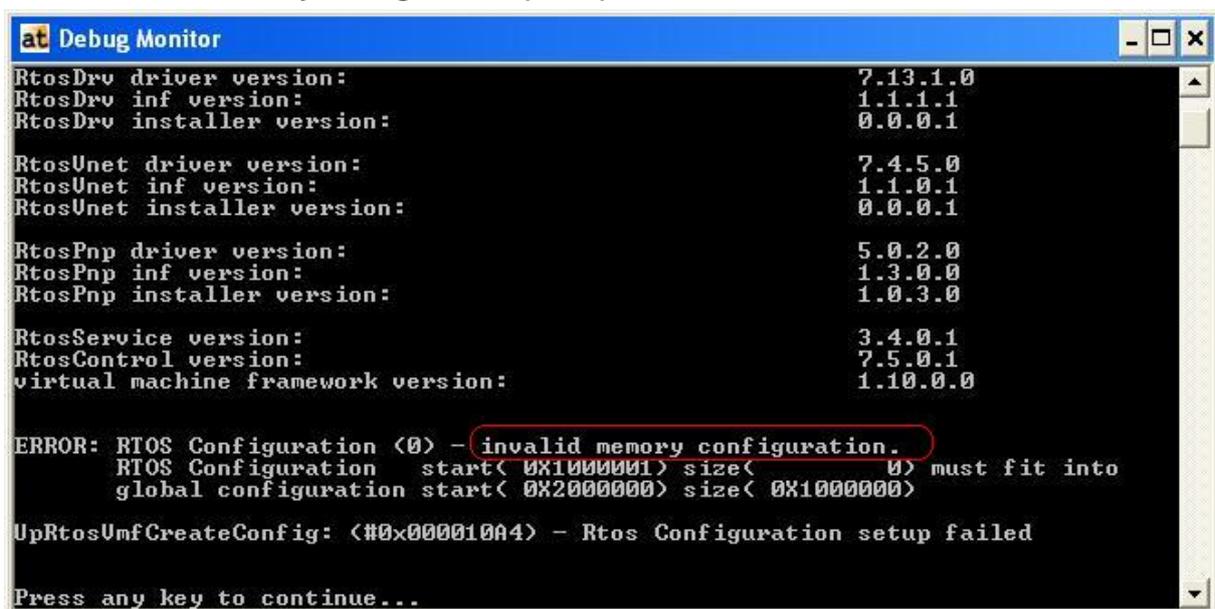
### 16.3.3 Configured RTE memory range not available

When updating from an older version it is possible the currently configured memory range will not be detected correctly. In this case please

- Remove the current configuration by calling "RtosUpload.exe /memcfg /u" from a DOS box.
- Reboot the PC
- Start RTOS again (should tell about required reboot)
- Reboot
- Start RTOS (should work now)

Before manually (means not using Setup) updating from one version to another the memory reservation should be removed first.

### 16.3.4 Invalid memory configuration (10A4)



```
Debug Monitor

RtosDrv driver version:          7.13.1.0
RtosDrv inf version:             1.1.1.1
RtosDrv installer version:       0.0.0.1

RtosUnet driver version:         7.4.5.0
RtosUnet inf version:            1.1.0.1
RtosUnet installer version:      0.0.0.1

RtosPnp driver version:          5.0.2.0
RtosPnp inf version:             1.3.0.0
RtosPnp installer version:       1.0.3.0

RtosService version:             3.4.0.1
RtosControl version:             7.5.0.1
virtual machine framework version: 1.10.0.0

ERROR: RTOS Configuration <0> - invalid memory configuration.
      RTOS Configuration start< 0X10000001> size< 0> must fit into
      global configuration start< 0X20000000> size< 0X10000000>

UpRtosUmfCreateConfig: <#0x000010A4> - Rtos Configuration setup failed

Press any key to continue...
```

#### Causes:

### MemoryStartAddress isn't in the range of the RteMemory.

(RteMemoryStartAddress >= **MemoryStartAddress** <= (RteMemorySize+RteMemoryStartAddress))

### MemorySize is too high

MemorySize > RteMemorySize

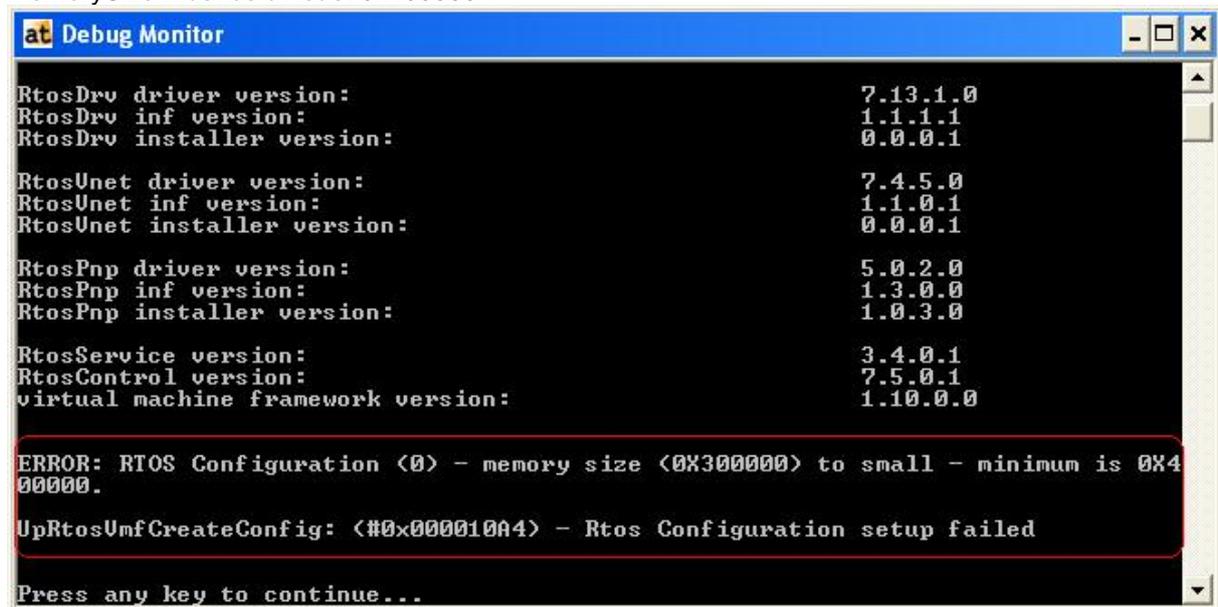
Or MemorySize+Offset > RteMemorySize

### Image is too big

MemorySize < Image Size

### Memory size is too small

MemorySize must be at least 0x400000



```
at Debug Monitor
RtosDrv driver version: 7.13.1.0
RtosDrv inf version: 1.1.1.1
RtosDrv installer version: 0.0.0.1
RtosUnet driver version: 7.4.5.0
RtosUnet inf version: 1.1.0.1
RtosUnet installer version: 0.0.0.1
RtosPnp driver version: 5.0.2.0
RtosPnp inf version: 1.3.0.0
RtosPnp installer version: 1.0.3.0
RtosService version: 3.4.0.1
RtosControl version: 7.5.0.1
virtual machine framework version: 1.10.0.0

ERROR: RTOS Configuration <0> - memory size <0X300000> to small - minimum is 0X400000.
UpRtosUmfCreateConfig: <#0x000010A4> - Rtos Configuration setup failed

Press any key to continue...
```

## 16.4 Windows Clock Delay

In case the Windows clock seems to be delayed

- Using SharedCore check if the clock correction is enabled
- Using ExclusiveCore check if disabling the clock correction solves the problem.

See chapter "9.1Clock Correction" for configuration details.

## 16.5 Windows Network Stack

In some circumstances it is possible the Windows network IP stack becomes corrupt.

This can be identified for example

- If opening TCP/IP parameters of a network connection results in an error message
- If "ipconfig -all" shows another IP address than TCP/IP configuration dialog

When the stack is corrupted it can be reset by calling

"netsh int ip reset c:\resetlog.txt"

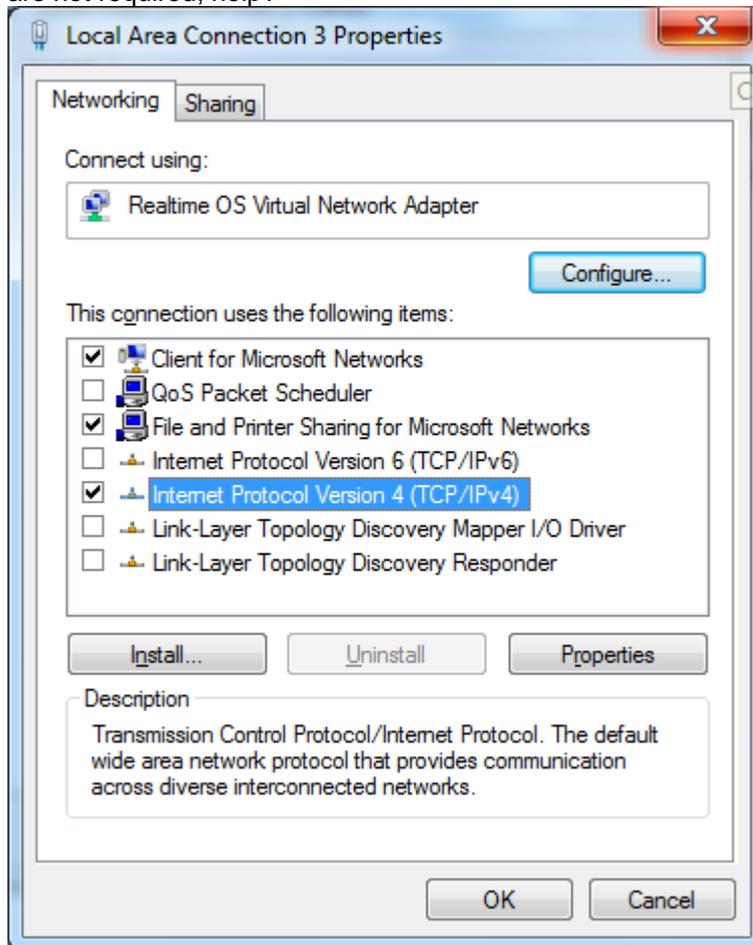
from an administrator command line.

After a reboot all IP settings will be reset. All manual settings have to be configured again.

## 16.6 Network Share access

In case the network share can't be accessed by RTOS please check the following:

- Can the network folder be accessed from the same and also from another PC?
- Can RTOS ping the PC using the computer name (not the IP)?
- Is the config file entry for network share and password correct?
- Does disabling of all protocols and services of "Realtime OS Virtual Network Adapter", which are not required, help?



## 16.7 Timer frequency

All VMF timer are depending on the Local APIC timer. Its input frequency is normally measured by VMF itself during start-up.

### 16.7.1 Setting a dedicated frequency

Under rare circumstances, when a very exact timer interrupt period is required, it might be required to manually configure the Local APIC input frequency on a PC.

The currently used processor input frequency can be displayed calling:  
"RtosUpload.exe /idshow 11,0,1"

The frequency can be set for each processor using a config file entry:

[Processors]

"LocalApicTimerInputFrequency0"=hex:...

"LocalApicTimerInputFrequency1"=hex:...

The values are 64 Bit values and have to be provided in "Little Endian" format.

e.g.: hex:00,CA,9A,3B,00,00,00,00 for 1 GHz

e.g.: hex:B4,BE,F9,0B,00,00,00,00 for 200916660 Hz

### 16.7.2 Frequency measure error

While VMF determines the Local APIC frequency it checks the quality of this measurement.

If measure time exceeds a limit the error code 0x270B =

RTE\_ERROR\_TIMER\_MEASUREFREQUENCY\_DELAYLIMIT will be returned.

This error indicates that the measured frequency will probably not be very accurate and reliable.

This might be acceptable in case no Realtime is required.

The limit can be configured using config-file entry

[Vmf]

"TimerMeasureDelayLimit"=dword:X

See chapter "6.5

Section [Vmf]" for details.

## **16.8 Interrupt / Timer Latency**

Please check chapter “15 Appendix A – Platforms and performance” for possible reasons and solutions.

## **16.9 Virtual Network Adapter Performance**

On default the Windows Virtual Network adapter is configured to use polling mode. In case the performance is not sufficient the system can be configured for using interrupt mode. See chapter “10.1.1.1 Polling or Interrupt Mode” for details.

## **16.10 RtosLib Event / MsgQueue / Pipe / Socket Performance**

On default the events are configured using polling mode. In case the performance is not sufficient the system can be configured for using interrupt mode. See chapter “6.8.2 Enable / Disable Comm Interrupt” for details.

## **16.11 Using RTOSVM inside a Hypervisor**

It is not officially supported to run an RTOSVM inside a Hypervisor because realtime behaviour is undetermined.

Nevertheless it might be useful during development.

Because such a system runs very slow Local APIC timer frequency measure will fail - see chapter “16.7.2 Frequency measure error” for details.

## 17 Appendix C – Manual Installation

There are several elements required or optional to be installed.

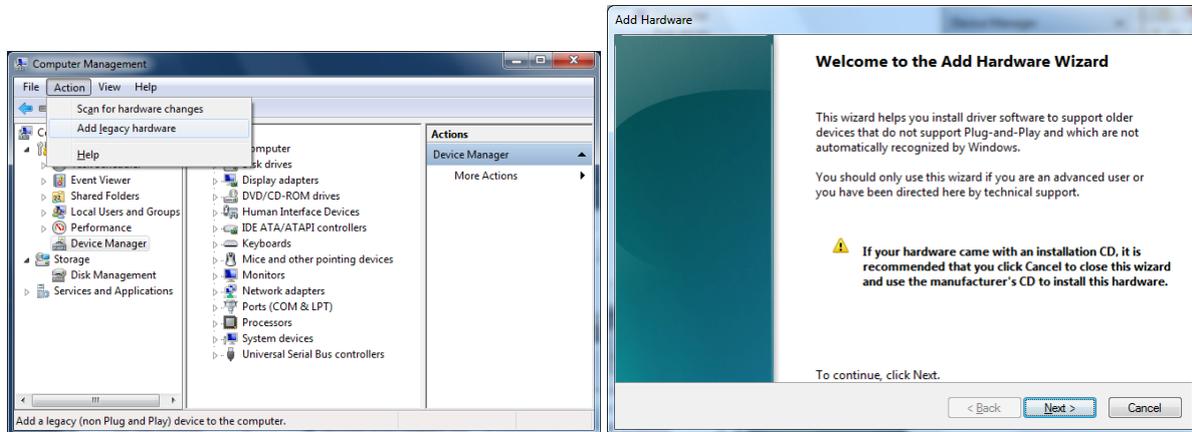
The automatic equivalent of Setup.exe the component "Runtime Files" installs all of them.

- "Realtime OS Driver"  
Required driver for loading and accessing VMF and OS
- "Realtime OS Uploader"  
Required application to communicate with "Realtime OS Driver"
- "Virtual Machine Framework"  
Required file containing the code shared between all OS.
- "Configuration Files"  
Configuration files are required to control the behaviour of the Uploader and VMF.
- "Realtime OS Service"  
Optional Service for handling RtosLib functionality.  
It is required if at least of this features should be usable by Windows:  
SharedMemory, Events, TimeSync, TimeZoneSync  
If not installed the config file entry "WaitForRtosCommSubsystems" must be set to "0"
- "Realtime OS Control"  
Optional task bar application for showing message boxes in case of errors.  
If not installed the config file entry "LaunchRtosControl" must be set to "0"
- "Realtime OS Virtual Network Driver"  
Optional network driver for accessing virtual network
- "Debug Console"  
Optional application to show information, debug messages and – depending on the product – gain access to a console for accessing an OS.

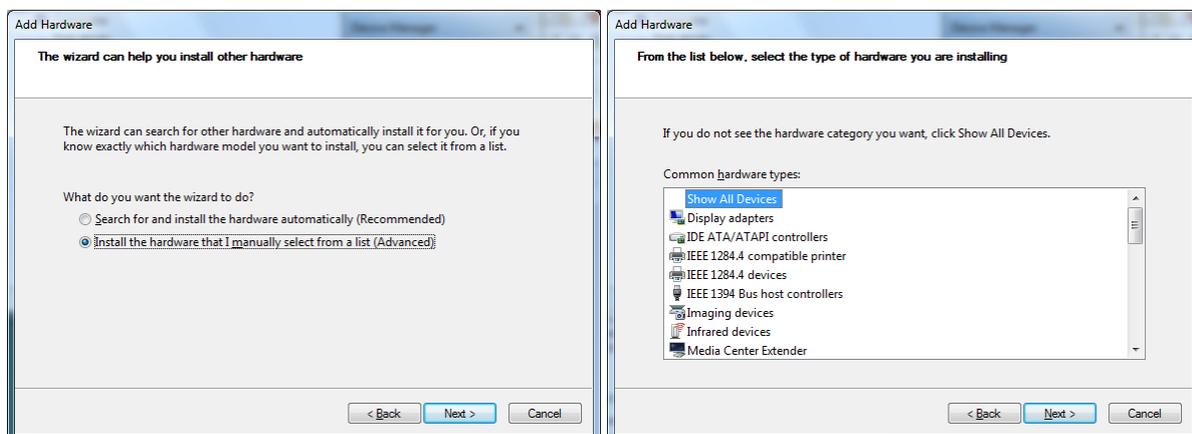
The following steps are described for Windows 7 but the procedure is similar for previous versions.

## 17.1 Realtime OS Driver

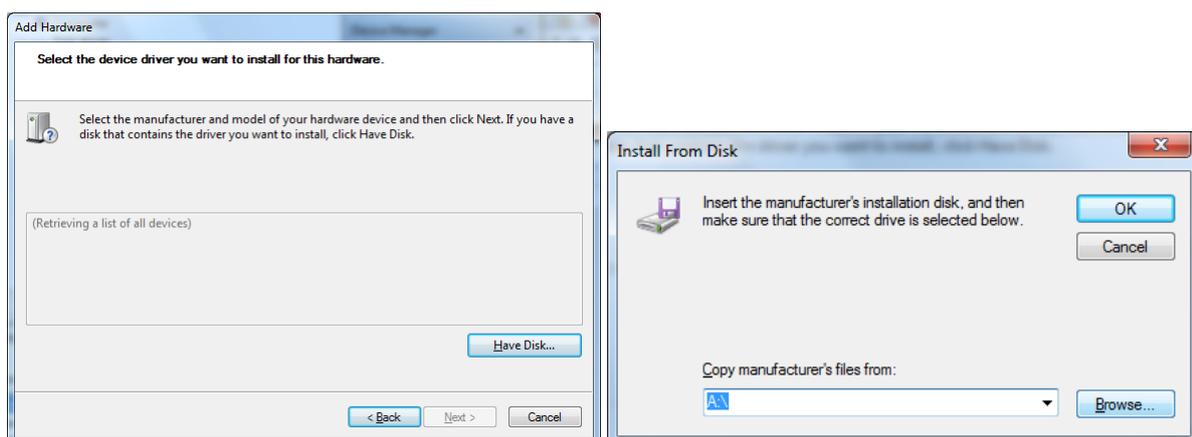
The driver has to be installed using the “Add Hardware Wizard”.  
On XP it can be found at “Control Panel” → “Add Hardware”  
On Windows 7 it can be accessed through the “Device Manager”.



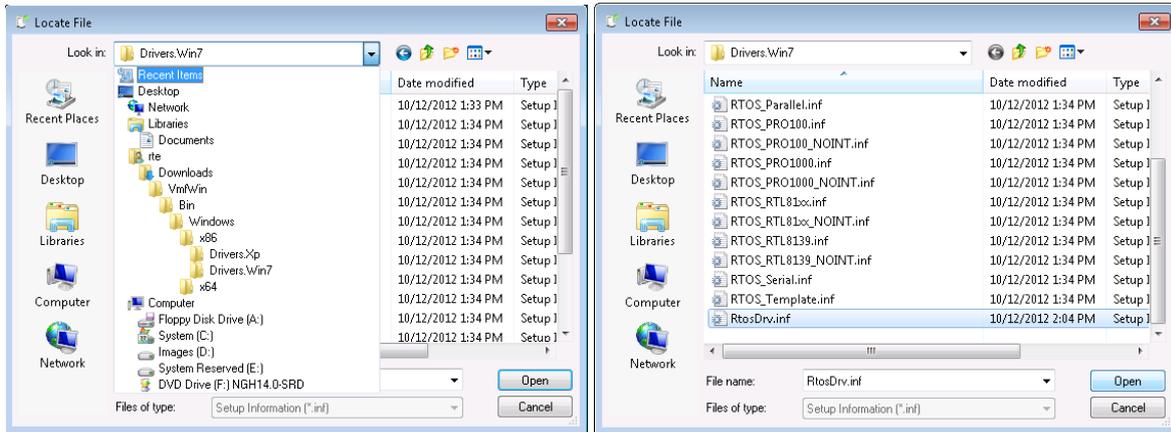
First start the “Add Hardware Wizard”.



Select manual selection of the hardware to be added.



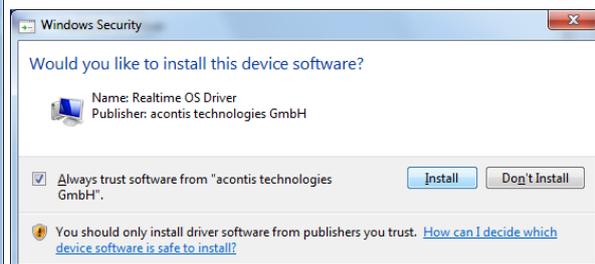
Add a new device driver from...



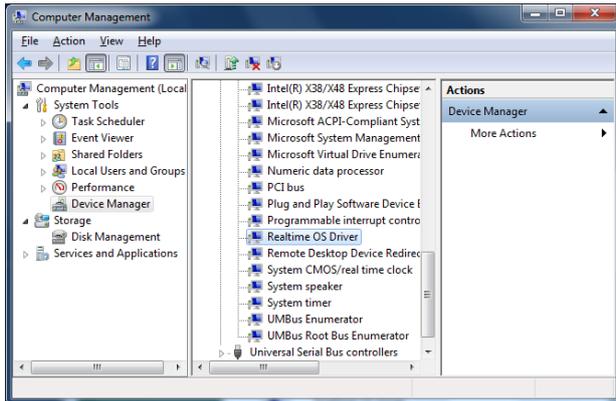
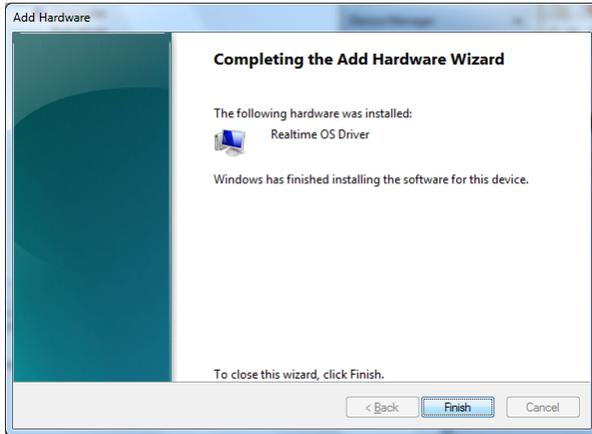
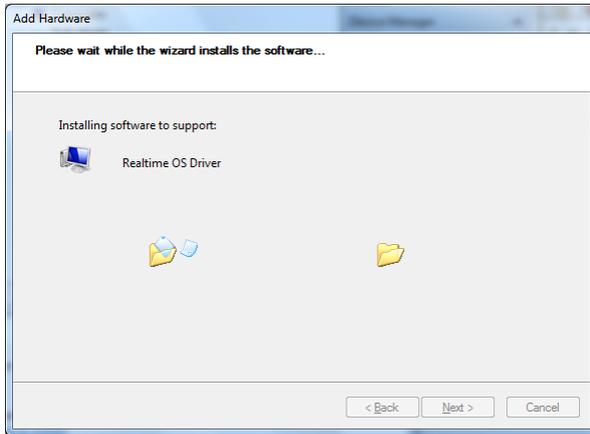
... the appropriate directory. Use

- "x86\Drivers.XP" for Windows XP, 32 Bit
- "x86\Drivers.Win7" for Windows 7, 32 Bit
- "x64\Drivers" for Windows 7, 64 Bit

The supported Windows versions are depending on your RTE version. Support for 64 Bit for example is starting with RTE 5.0



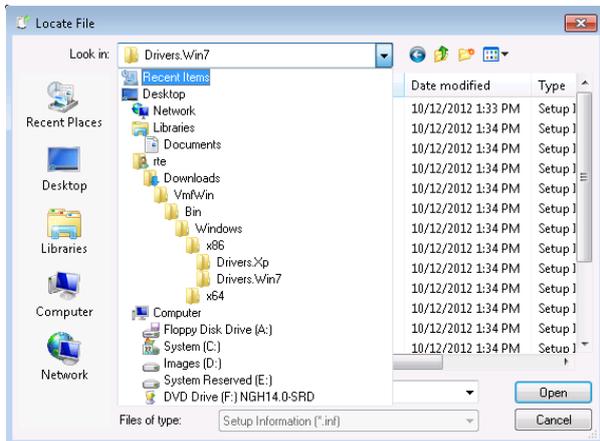
Depending on your Windows version and driver signing policies you may see different dialogs.



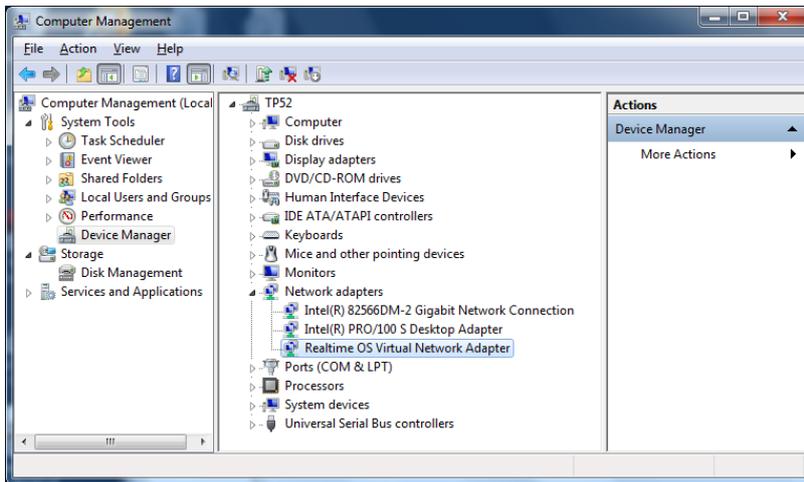
After successful installation you should be able to see “Realtime OS Driver” in the list of “System Devices” in the Windows Device Manager.

## 17.2 Realtime OS Virtual Network Driver

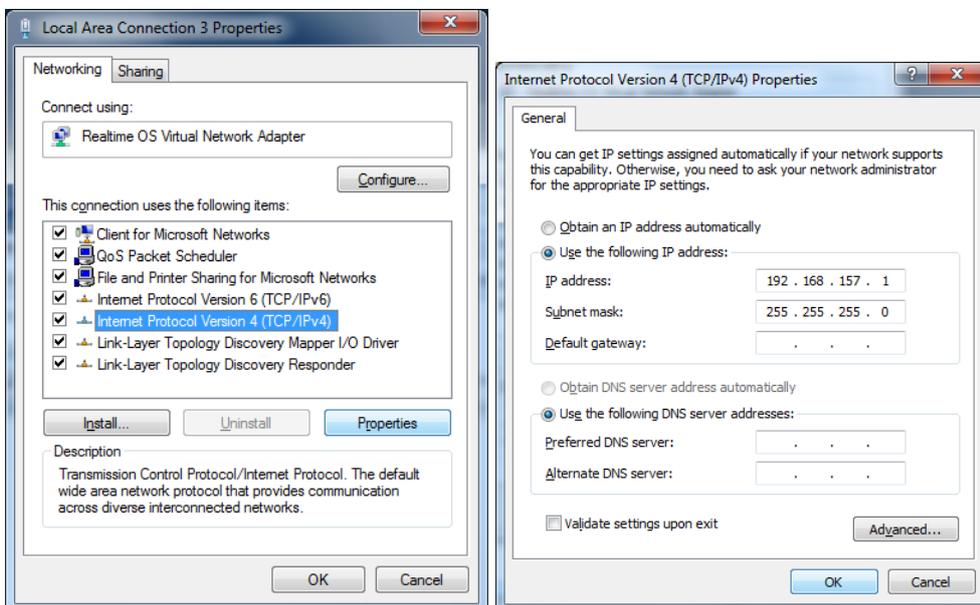
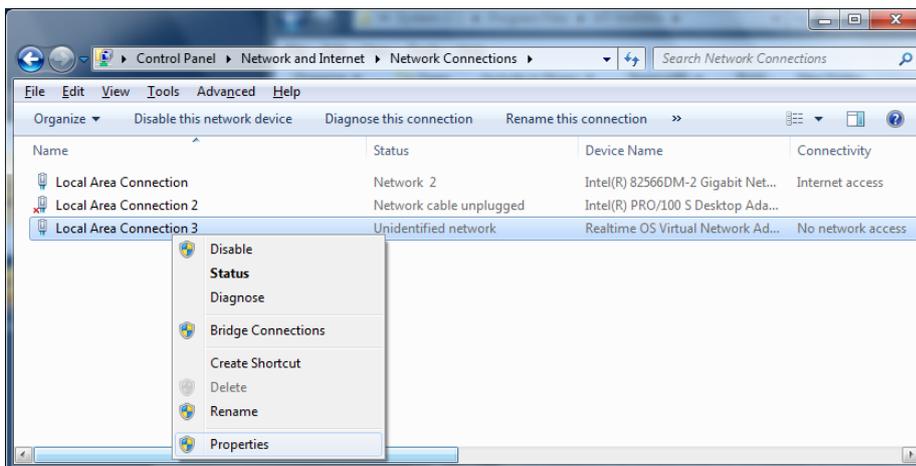
The “Realtime OS Virtual Network Driver” installation is similar to the previous “Realtime OS Driver” installation.



The main difference is that you have to select the “Realtime OS Virtual Network Driver”. All other steps are equal.

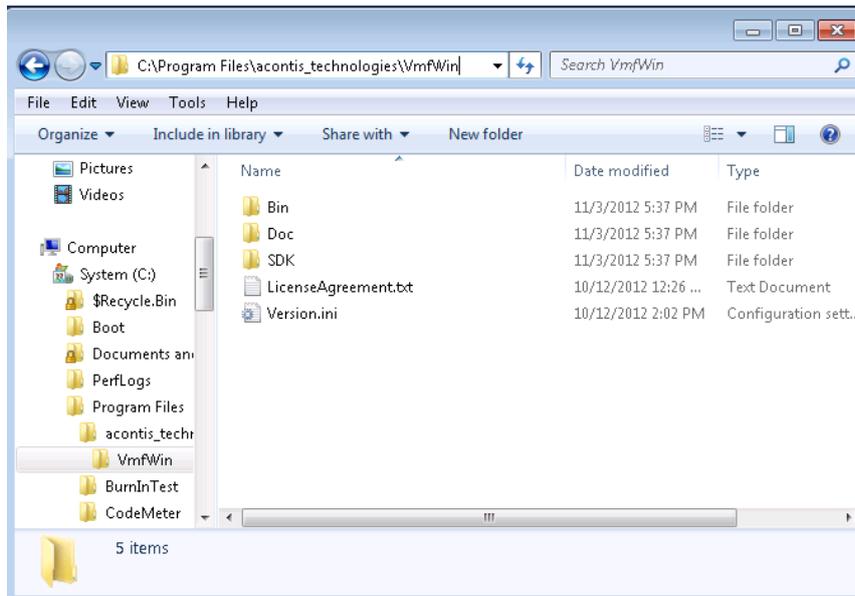


After the installation finished you the Device Manager should show the new network device.



Finally the IP address has to be configured.

## 17.3 Product Files

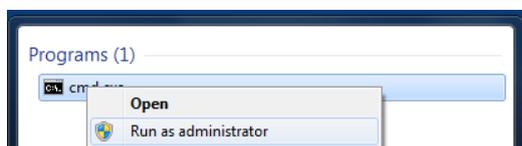


Create a new folder where the product should be installed to and copy all files from your installation directory into that folder.(without “Setup.exe”, “ProductIcon.ico” and “autorun.inf”) The number of files will depend on your product.

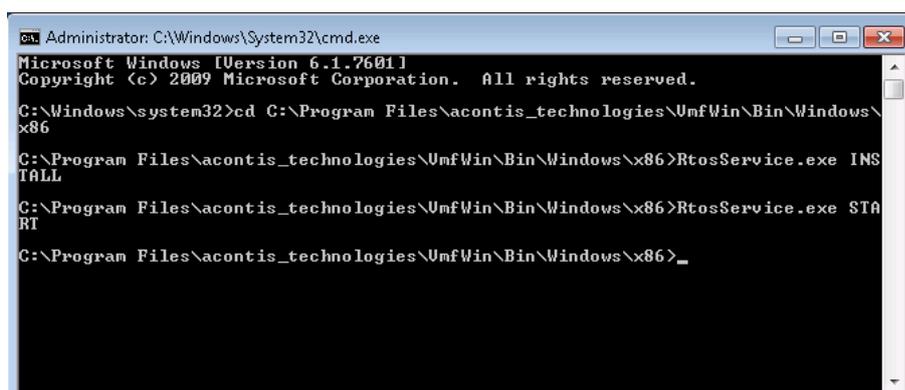
### 17.3.1 Virtual Machine Framework

The VMF file “vmf.bin” requires no special installation.

### 17.3.2 Realtime OS Service



Start a command shell with administrator rights.



32-Bit Windows:

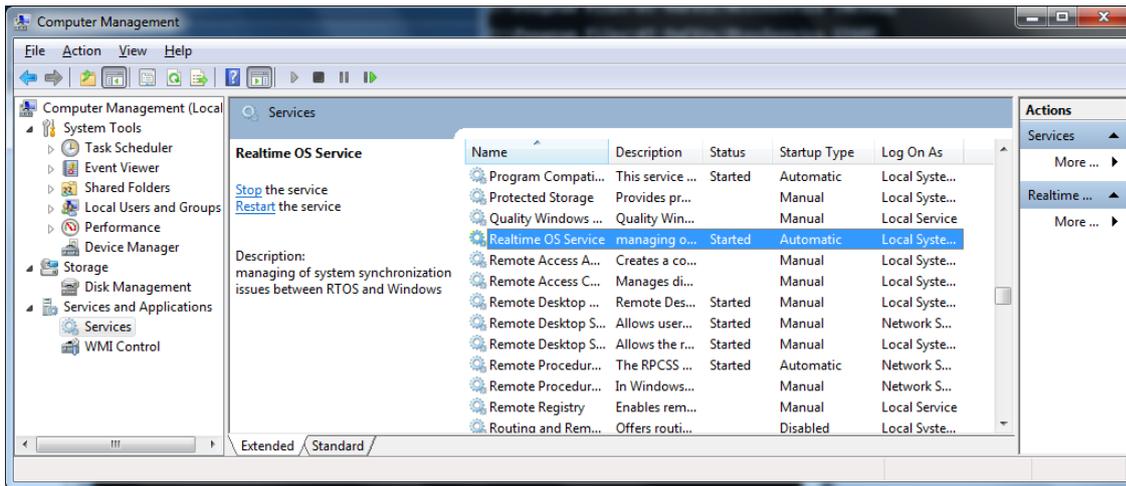
Change to de directory “C:\Program Files\acontis\_technologies\VmfWin\Bin\Windows\x86” ...

64-Bit Windows:

Change to de directory “C:\Program Files\acontis\_technologies\VmfWin\Bin\Windows\x64” ...

...and then type in “RtosService.exe INSTALL” followed by “RtosService.exe START”

The service can be removed using the commands “STOP” and “UNINSTALL”.



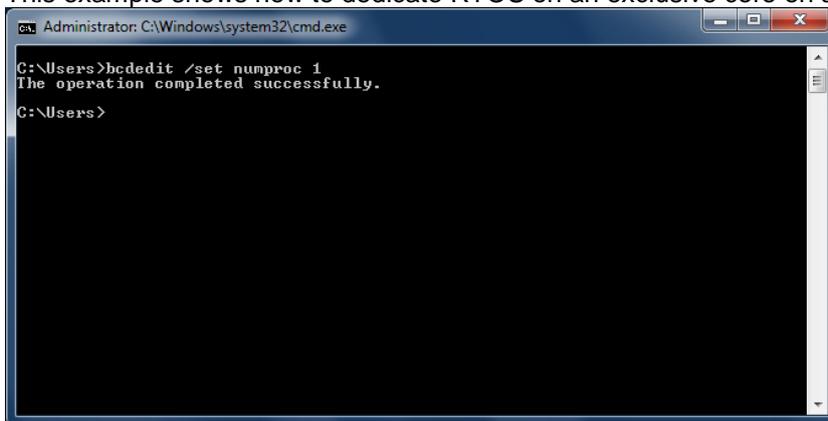
After the installation the Windows Services should show the new “Realtime OS Service”.

### 17.3.3 Realtime OS Control

RtosControl.exe requires no special installation.

### 17.3.4 Configuration Files

If SharedCore is not required or technically not possible ExclusiveCore can be used. This example shows how to dedicate RTOS on an exclusive core on a dual core PC.



Windows must be limited in its processor usage.

- XP: edit c:\boot.ini and add “numproc=X”
- Win7: call “bcdedit /set numproc X” from the command shell – started with administrator rights.  
(remove: “bcdedit /deletevalue numproc”)

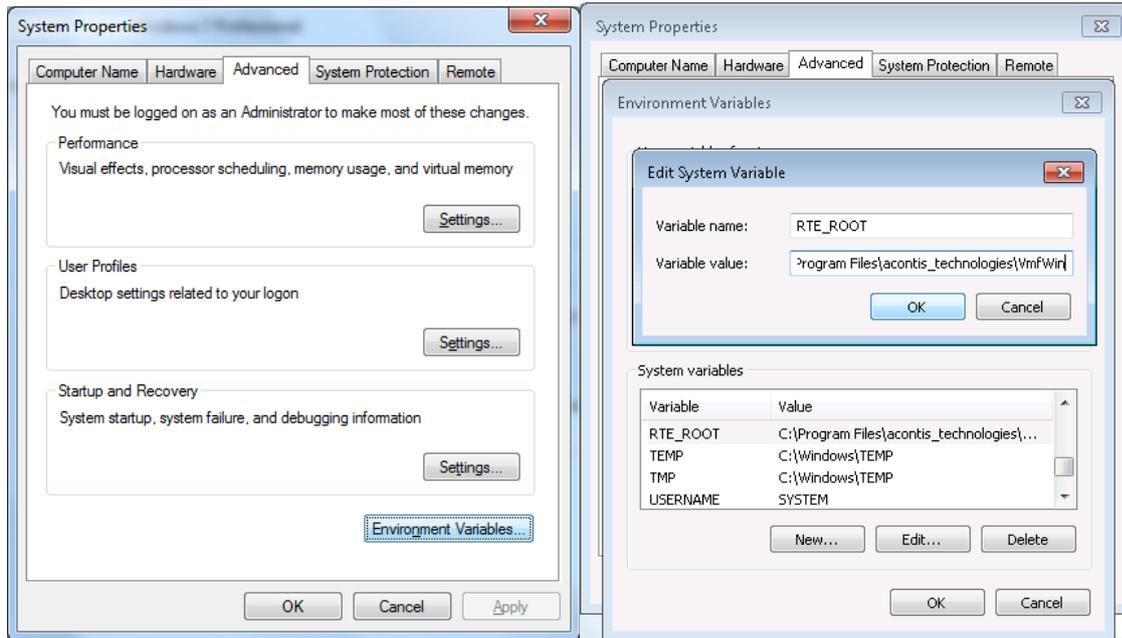
A reboot is required to activate the new setting.

In this example Windows is limited to one processor so the second is available.

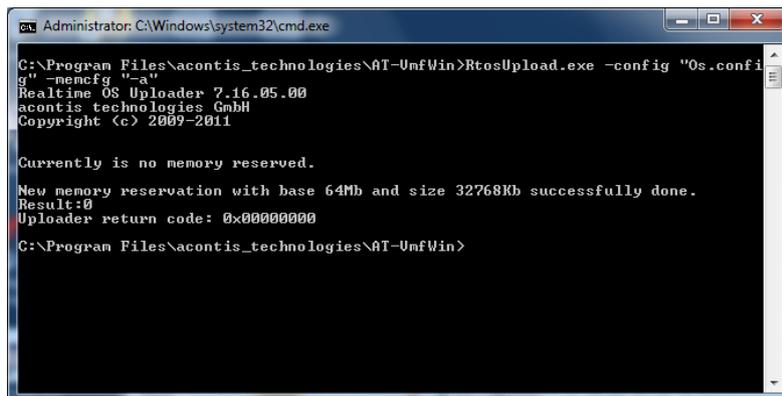
The config file “general.config” should contain the section “[Rtos]” with the value “ProcessorMask”.

“ProcessorMask” must be set to the value 2 to use the second processor.

## 17.3.5 Realtime OS Uploader



The environment variable “RTE\_ROOT” has to be set in the Windows system properties. “Realtime OS Uploader” requires the variable “RTE\_ROOT” to have the value of the product installation directory – for example “C:\Program Files\acontis\_technologies\VmfwWin”.



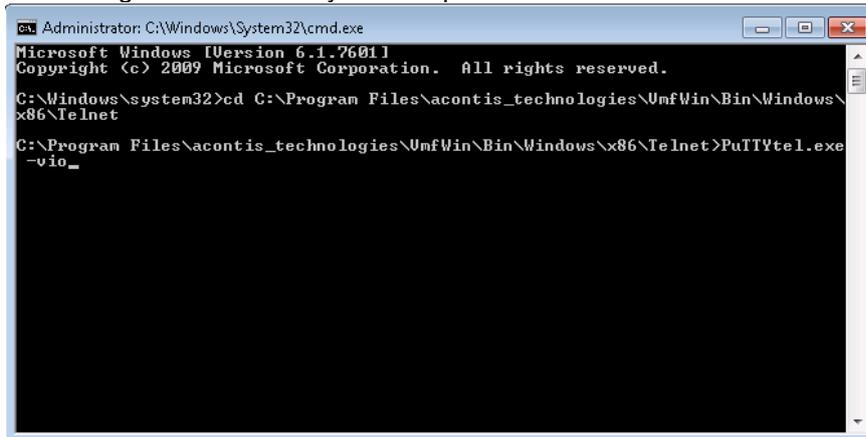
Memory has to be reserved from Windows for loading a RTOS. The screenshot shows how memory reservation can be updated (values are taken from the given config file) without loading an OS image. Depending on the product the name of the config file may vary.

Additionally the memory reservation is checked each time VMF is started and reconfigured automatically if the settings in the config file had changed.

A reboot is required after the memory configuration has been updated.

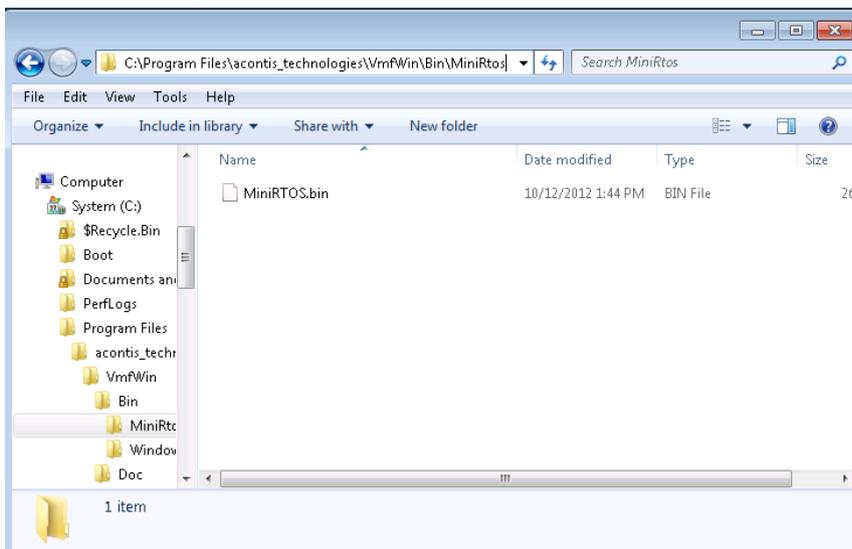
## 17.4 Debug Console

The debug console has only to be copied. There is no other installation required.



To start the debug console the parameter "-vio" is required.

## 17.5 Os Image



Copy the OS image to the desired directory.

The name and directory of the OS image file are product depending. This example shows VmfWin so please adapt this step to the product you're using.

## 17.6 CodeMeter Runtime Environment

If your version is bound to a Software or USB license the WIBU "CodeMeter Runtime for Windows" has to be installed to verify the license.

If no license was found – because there is no license or then "CodeMeter Runtime for Windows" is missing – a "license not found" error message will occur during start of an OS.

The latest version of "CodeMeter Runtime for Windows" can be downloaded from <http://www.wibu.com> (Support & Downloads → User → User Software).

## 17.7 First Start of an OS

```
Administrator: C:\Windows\system32\cmd.exe - RtosUpload.exe -config "Os.config" "MiniRtos\Mi...
C:\Program Files\acontis_technologies\UmfWin\Bin\Windows\x86>"C:\Program Files\acontis_technologies\UmfWin\Bin\Windows\x86\RtosUpload.exe" -config "..\config\Os.config" -vmf "..\vmf.bin" "..\..\MiniRtos\x86\MiniRtos.bin"
Realtime OS Uploader 7.17.00.00
acontis technologies GmbH
Copyright (c) 2009-2012

Product:           UmfWin U5.0.00 (5.0.0.1)
RtosWin version:   5.0
Hosting OS:       Windows 7

UploadRTOS.dll version:           7.18.8.0

RtosDrv - Realtime OS Driver
  Driver version           9.1.1.0
  Inf file version         1.2.2.0
  Installer version        0.0.0.1
  Required UMF version     1.27.5.0

RtosUnet - Realtime OS Virtual Network Adapter
  Driver version           7.7.0.0
  Inf file version         1.1.3.0
  Installer version        0.0.0.1
  Required UMF version     1.20.0.0

RteDevice - Realtime OS Driver
  Driver version           ignored
  Inf file version         ignored
  Installer version        ignored
  Required UMF version     1.27.5.0

RteDevice - Realtime OS Virtual Network Adapter
  Driver version           ignored
  Inf file version         ignored
  Installer version        ignored
  Required UMF version     1.20.0.0

RtosService version:           3.5.1.0
RtosControl version:           7.6.2.1

Virtual machine framework version: 1.27.5.0
Virtual machine framework version required by Uploader: 1.27.2.0

Loading UMF...Error: Reboot is required to update Memory-Configuration
Uploader return code: 0x00001082

Press any key to continue..._
```

To start an OS the config file and OS image must be passed as parameters to the Uploader. In this case the complete parameter is:

32-Bit Windows:

```
"C:\Program Files\acontis_technologies\UmfWin\Bin\Windows\x86\RtosUpload.exe" -config
"..\config\Os.config" -vmf "..\vmf.bin" "..\..\MiniRtos\x86\MiniRtos.bin"
```

64-Bit Windows:

```
"C:\Program Files\acontis_technologies\UmfWin\Bin\Windows\x64\RtosUpload.exe" -config
"..\config\Os.config" -vmf "..\vmf.bin" "..\..\MiniRtos\x86\MiniRtos.bin"
```

After first time installation the Memory-Configuration might require a second reboot.

The name of the config file and OS image file are product depending. This example uses VmfWin so please adapt this step to the product you're using.

```

Administrator: C:\Windows\System32\cmd.exe

C:\Program Files\acontis_technologies\UmfWin\Bin\Windows\x86>"C:\Program Files\acontis_technologies\UmfWin\Bin\Windows\x86\RtosUpload.exe" -config "..\config\0s.config" -umf "..\umf_bin" ".\..\MiniRtos\x86\MiniRtos.bin"
Realtime OS Uploader 7.17.00.00
acontis technologies GmbH
Copyright (c) 2009-2012

Product:           RTOS32Win 05.1.00 (5.1.0.99)
Base:              05.1.0.99
RtosWin version:   5.1
Hosting OS:        Windows 7 Service Pack 1.0 (32 bit)

UploadRTOS.dll version:           7.19.2.0

RtosDrv - Realtime OS Driver
  Driver version           9.3.2.0
  Inf file version         1.3.0.0
  Installer version        0.0.0.1
  Required UMF version     1.28.0.0

RtosUnet - Realtime OS Virtual Network Adapter
  Driver version           7.8.0.0
  Inf file version         1.2.0.0
  Installer version        0.0.0.1
  Required UMF version     1.28.0.0

RteDevice - RTOS PRO/100 compatible PCI card
  Driver version           5.4.0.0
  Inf file version         1.5.0.0
  Installer version        1.1.0.0
  Required UMF version     1.28.0.0

RteDevice - Realtime OS Driver
  Driver version           ignored
  Inf file version         ignored
  Installer version        ignored
  Required UMF version     1.28.0.0

RteDevice - Realtime OS Virtual Network Adapter
  Driver version           ignored
  Inf file version         ignored
  Installer version        ignored
  Required UMF version     1.28.0.0

RtosService version:           3.6.0.0
RtosControl version:           7.7.1.0

Virtual machine framework version: 1.28.2.0
Virtual machine framework version required by Uploader: 1.28.0.0

Reservation active   : base 64Mb and size 32768Kb.

Loading UMF...Ok
Virtual machine framework version required by OS: 1.28.x.x

Start OS - Id:0... Ok
Uploader return code: 0x00000000

C:\Program Files\acontis_technologies\UmfWin\Bin\Windows\x86>

```

Finally it should start successfully.

```

Administrator: C:\Windows\System32\cmd.exe

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd C:\Program Files\acontis_technologies\UmfWin\Bin\Windows\x86\Telnet
C:\Program Files\acontis_technologies\UmfWin\Bin\Windows\x86\Telnet>PuTTYtel.exe -vio_

```

To start the debug console the parameter "-vio" is required.

```
=====  
=====  
VMF anchor signature : 0x414E4348.  
OS ID                 : 0.  
VMF version          : 1.27.5.0  
VMF descriptor       : @ 0x2A000000.  
VMF data descriptor  : @ 0x4021B00.  
=====  
Mini RTOS running on exclusive core  
Top of memory @0x5000000  
=====  
Mini RTOS booting...  
+bspHwInit  
-bspHwInit  
MMU[0]: VA=0x2A000000, PA=0x2A000000, Size=4096kByte, Flags=0x7  
MMU[1]: VA=0x3BD01000, PA=0x3BD01000, Size=4096kByte, Flags=0x7  
MMU[2]: VA=0xFED00000, PA=0xFED00000, Size=4kByte, Flags=0x1F  
MMU[3]: VA=0xFEC00000, PA=0xFEC00000, Size=4kByte, Flags=0x1F  
MMU[4]: VA=0xFEE00000, PA=0xFEE00000, Size=4kByte, Flags=0x1F  
MMU[5]: VA=0xB000, PA=0xB000, Size=4kByte, Flags=0x7  
MMU[6]: VA=0x40000000, PA=0x40000000, Size=16384kByte, Flags=0x7  
MMU[7]: VA=0x2A0D9000, PA=0x2A0D9000, Size=196kByte, Flags=0x7  
MMU[8]: VA=0x3EF87000, PA=0x3EF87000, Size=100kByte, Flags=0x7  
MMU enabled!  
Enter rtosInitTask()  
+bspHwInit2  
TimerHwInterruptId = 1  
Timer0InterruptId = 4  
Timer1InterruptId = 5  
TimerHwInterruptVector = 0xE0  
Timer0InterruptVector = 0xE2  
Timer1InterruptVector = 0xE3  
-bspHwInit2  
System clock rate = 1000 Hz  
RTOS IPI CPC: interrupt id = 7, vector = 0xFC  
RTOS IPI Debug: interrupt id = 8, vector = 0xFB  
RTOS IPI TLB Flush: interrupt id = 9, vector = 0xFA  
RTOS IPI TSC Reset: interrupt id = 10, vector = 0xF9  
RTOS IPI Shutdown: interrupt id = 11, vector = 0xF8  
RTOS processor mask: 0x2  
RTOS Boot Processor mask: 0x2  
Boot Processor: CPU 0: Index 1: APIC ID = 1  
-----  
Enter appInitTask()  
Virtual Network Polling period = 1 msec  
COM1 not available for RTOS  
COM2 not available for RTOS  
Press 'c' to continue!  
█
```

The debug console output depends on the product. In some products debug console implements an entire shell to access the OS.

### **17.8 Product specific additional steps**

If a product requires additional installation steps they will be described in the “Manual Installation” section of the product specific manual.

## **18 Version History**

A general version history containing information about new features, migration hints and improvements can be found in the release notes file "ReleaseHistory.txt".