Table of Contents
1 Introduction...............................................................................................................................................8
  1.1 Overview of eVM for Windows........................................................................................................8
2 Quick Start Guide.....................................................................................................................................10
3 Product Technical Description.............................................................................................................11
  3.1 What is Real-Time embedded virtualization? ..................................................................................11
  3.2 Theory of Operation of eVM............................................................................................................11
    3.2.1 What is eVM?............................................................................................................................11
    3.2.2 Introduction to virtualization.....................................................................................................11
    3.2.3 eVM and Virtualization technology............................................................................................12
    3.2.4 eVM system requirements (brief overview)..............................................................................12
    3.2.5 Performance: How eVM addresses Guest OS / Windows Application Needs .........................13
    3.2.6 Communications between Windows and the Guest OS: Virtual Device Interfaces...............13
    3.2.7 Communications between Windows and the Guest OS: Shared Memory.................................14
    3.2.8 eVM Architecture: Partitioning the PC between Windows and Guest OS.................................14
    3.2.9 Resources that are not isolated between Windows and the Guest OS.........................................16
    3.2.10 Device Management.................................................................................................................16
    3.2.11 What Happens when a Device is assigned to eVM...................................................................21
4 Installation ...............................................................................................................................................22
  4.1 Installation of eVM for Windows........................................................................................................22
  4.2 Virtual Ports....................................................................................................................................39
    4.2.1 Installation of Virtual Serial Ports...............................................................................................40
    4.2.2 Configuring the Virtual Ethernet Device.....................................................................................44
  4.3 Running the Sample Guest OS, iRMX.................................................................................................51
  4.4 Installation of a Guest OS...................................................................................................................53
    4.4.1 Configure the existing system....................................................................................................53
    4.4.2 Make a disk image of the system ...............................................................................................54
    4.4.3 Make a Guest OS configuration..................................................................................................54
    4.4.4 Installing the Guest OS from installation materials ..................................................................55
  4.5 Specific steps for certain Guest OSs. .................................................................................................55
    4.5.1 Installation and configuration of QNX 6.4 .................................................................................55
4.5.2 Installation and configuration of Windows CE ........................................ 58
4.6 Driver configuration and setup ................................................................. 59
  4.6.1 eVM Device Manager ........................................................................... 59
  4.6.2 Passing a Legacy IRQ Device to eVM ................................................. 64
  4.6.3 Selecting communication port modes ................................................. 73
  4.6.4 Conflict Example when Passing Legacy IRQ Device to eVM ............... 74
  4.6.5 Passing an MSI Device to eVM .......................................................... 76
  4.6.6 Bus Master Warning Example ........................................................... 79
  4.6.7 Passing a Device back to Windows ..................................................... 81
4.7 Shared Memory Interface ......................................................................... 85
  4.7.1 Description ......................................................................................... 85
  4.7.2 Configuration ..................................................................................... 85
  4.7.3 Shared Memory Usage ........................................................................ 87
  4.7.4 PCI configuration space definition ..................................................... 88
  4.7.5 Register definition .............................................................................. 88
  4.7.6 Typical operation ................................................................................. 89
A. Production installation ............................................................................... 90
  A.1 Saving the configuration with the eVM Configuration Panel .................. 91
  A.2 Saving the configuration with a command line .................................... 92
  A.3 Restoring or importing a known configuration with the eVM Configuration Panel ................................. 93
  A.4 Restoring or importing a known configuration with a command line ........ 94
  A.5 Assigning devices to eVM on the command line .................................. 94
    PCI devices ................................................................................................ 94
  A.6 Command Line Installation of Virtual Serial Ports ............................... 95
  A.7 Debug logging ....................................................................................... 95
B. eVM Virtual Machine ............................................................................... 96
  B.1 System Requirements .......................................................................... 96
  B.2 Base Virtual Machine Specification ..................................................... 96
    CPU ........................................................................................................ 96
    Memory .................................................................................................. 96
    Interrupt controller (8259A PIC) ............................................................ 96
    System Timer (8254 PIT) ....................................................................... 97
CMOS & Real Time Clock ................................................................. 97
Port 92 ........................................................................................................ 97
Serial ports (16550 UART) ......................................................... 97
DMA controller (8237 DMA controller) ........................................ 97
Floppy Disk Controller ............................................................... 98
IDE controller ....................................................................................... 98
PCI Host Bridge (440FX Host Bridge, PCI device 0) ......... 98
ISA Bridge (440FX ISA Bridge, PCI ID 8086/7000) ................. 98
EIDE Controller (PCI ID 8086/7010) ........................................... 98
ACPI Controller (PIIX4 ACPI Controller, PCI ID 8086/7113) .... 98

B.3 Optional Virtual Machine Components ........................................... 98

Ethernet controller .............................................................................. 98
Shared memory device ..................................................................... 99

B.4 BIOS ............................................................................................ 99

C. Enabled Guest OSs ......................................................................... 100

D. VtProbe .......................................................................................... 101

E. Modifying device drivers for bus-master devices ......................... 103
   E.1 Discovering the Guest OS physical base address ................. 103
   E.2 Using the Guest OS physical base offset value ..................... 103

F. eVM API calls .................................................................................. 104
   F.1 Include the API in your application ........................................ 104
   F.2 Types ............................................................................................ 104
   F.3 API Calls .................................................................................... 104
      VMM Management calls ........................................................... 104
      Command and Control calls ............................................... 104
      Configuration Management calls ........................................ 105
      Shared Memory calls ............................................................. 106
List of Figures

Figure 1.1 - eVM for Windows ........................................................................................................... 9
Figure 3.1 - Memory map for a 32 bit installation ........................................................................... 14
Figure 3.2 - Memory map for a 64 bit installation ........................................................................... 15
Figure 3.3 - Interrupt management .................................................................................................. 18
Figure 4.1 - VT-x not supported message ...................................................................................... 23
Figure 4.2 - Installation Wizard ..................................................................................................... 24
Figure 4.3 - License Agreement ....................................................................................................... 25
Figure 4.4 - Release Notes .............................................................................................................. 26
Figure 4.5 - Select eVM Installation Folder ..................................................................................... 27
Figure 4.6 - Choosing Standard Installation Type ............................................................................ 28
Figure 4.7 - eVM Activation ............................................................................................................. 29
Figure 4.8 - eVM Installation Verification ......................................................................................... 30
Figure 4.9 - eVM Installation Wizard Progress Dialog ................................................................. 31
Figure 4.10 - eVM Installation Wizard Successful Installation ...................................................... 32
Figure 4.11 - TenAsys Driver loading notification .......................................................................... 32
Figure 4.12 - eVM Installation Log ................................................................................................. 33
Figure 4.13 - Restart Prompt ........................................................................................................... 34
Figure 4.14 - eVM System Tray Icon ............................................................................................... 34
Figure 4.15 - Launching eVM Configuration Panel from the system tray ..................................... 34
Figure 4.16 - eVM Configuration Panel .......................................................................................... 35
Figure 4.17 - eVM Node Management Panel with iRMX Guest OS Installed ............................... 36
Figure 4.18 - Device Manager showing the Windows devices that can be passed to eVM .......... 37
Figure 4.19 - Device Manager showing all the devices, including ones that cannot be passed to eVM (the Guest OS) .................................................................................. 38
Figure 4.20 - Windows Device Manager ......................................................................................... 39
Figure 4.21 - eVM Node Management Panel Showing System Configuration with two Virtual Serial Ports .................................................................................................................. 40
Figure 4.22 - Virtual Serial Port COM Port Selections .................................................................. 41
Figure 4.23 - Saving eVM Node Management Panel Configuration ............................................. 42
Figure 4.24 - Confirmation to save configuration change ............................................................... 42
Figure 4.25 - Windows Device Manager Showing added Virtual Serial Port .............................. 43
Figure 4.26 - eVM Node Management Panel Showing Guest OS Serial Device Settings .......... 44
Figure 4.27 - Windows Control Panel, Select Network .................................................................. 45
Figure 4.28 - Windows Network and Sharing Center ...................................................................... 45
Figure 4.29 - Windows Network Connections ............................................................................... 46
Figure 4.30 - TenAsys Virtual Ethernet Adapter, Windows Local Area Connection Properties ..... 47
Figure 4.31 - Selecting TenAsys Virtual Ethernet Adapter TCP/IPv4 Properties ......................... 48
Figure 4.32 - Setting the Static IP Address for the TenAsys Virtual Ethernet Adapter .................. 49
Figure 4.33 - eVM Node Management Panel, Guest OS Virtual Ethernet Device Configuration 50
Figure 4.34 - eVM Node Management Panel, Selecting the Guest OS .......................... 51
Figure 4.35 - Guest OS, iRMX, Booting in the Virtual Machine .................................. 52
Figure 4.36 - eVM Node Management Panel, Stopping the Guest OS ......................... 53
Figure 4.37 - Launching the eVM Device Manager from the eVM Configuration Panel .... 60
Figure 4.38 - eVM Device Manager launch from Windows Vista or Window 7 ............... 60
Figure 4.39 - eVM Device Manager launch from Windows XP .................................. 61
Figure 4.40 - eVM Device Manager launch from Windows 10 ................................... 61
Figure 4.41 - eVM Device Manager showing all the Devices that can be passed .......... 62
Figure 4.42 - eVM Device Manager showing how to switch to show All devices ........... 63
Figure 4.43 - eVM Device Manager showing All devices and how to switch to show only Devices that can be passed ................................................................. 64
Figure 4.44 - COM1 properties in eVM Device Manager ................................................ 65
Figure 4.45 - Pass Device Using Pass to eVM Button .................................................... 66
Figure 4.46 - Pass Device Using Action Dropdown ........................................................ 67
Figure 4.47 - Device Pass Request Selected ................................................................. 68
Figure 4.48 - Using the Save Configuration Button ........................................................ 69
Figure 4.49 - Using Dropdown to Save the Configuration ............................................. 70
Figure 4.50 - Warning that VT-d support is needed if the device is a bus master with DMA...... 71
Figure 4.51 - Platform Restart Required to Apply New Device Configuration ............... 72
Figure 4.52 - Device Configuration Change Applied .................................................... 73
Figure 4.53 - Pass through and File configuration options ............................................. 74
Figure 4.54 - eVM Device Manager Showing Windows USB Controllers .................... 75
Figure 4.55 - eVM Device Manager Flagging a Conflict .............................................. 76
Figure 4.56 - Right-Click a Device to Pass it to eVM .................................................... 77
Figure 4.57 - MSI Capable Device Pass Request ......................................................... 78
Figure 4.58 - MSI Network Adapter Device Passed to eVM .......................................... 79
Figure 4.59 - VT-x Support Enabled but VT-d Support Disabled ................................... 80
Figure 4.60 - Bus Master Warning for no VT-d Support .............................................. 81
Figure 4.61 - Select the Network Adapter and Pass It back to Windows ....................... 82
Figure 4.62 - Request to Pass the Network Controller back to Windows ..................... 83
Figure 4.63 - Restart Request to Apply Device Configuration Changes ....................... 84
Figure 4.64 - Network Adapter Back Under Windows Control .................................... 85
Figure 4.65 - Setting size of Shared Memory ............................................................... 86
Figure 4.66 - Setting Shared Memory IRQ for Guest OS application ........................... 87
Figure A.1 - List of configured Guest OSs ................................................................. 90
Figure A.2 - Selecting a default Guest OS ................................................................. 91
Figure A.3 - Exporting INtime Configuration File from the eVM Configuration Panel .... 92
Figure A.4 - Warning when Importing an INtime Configuration File ......................... 93
Table B.1 - Virtual IRQ's ......................................................................................... 97
Figure D.1 - VtProbe, Hardware Platform Doesn't Support eVM ............................... 101
Figure D.2 - VtProbe: Hardware Supports Virtualization Technology ..................... 102
1 Introduction

1.1 Overview of eVM for Windows

eVM for Windows provides a virtual machine environment hosting an embedded or real-time operating system alongside Microsoft Windows. eVM for Windows runs in parallel to Windows on any Intel multi-core or multi-threaded PC platform that features Intel Virtualization Technology (VT). Both Windows and the embedded Guest OS run natively, as if they were running on their own dedicated platforms. Moving an embedded Guest OS plus the application running on it to an eVM platform is a simple process that provides instant access to modern multi-core computing platforms, as well as access to all of the applications, the development environment, and the rich graphic user interface capabilities that are available under Windows. The key features of eVM for Windows are:

- Memory partitioning such that the embedded Guest OS and Windows maintain their integrity – there is no chance for the one operating system to corrupt the memory space of the other OS.
- Memory re-mapping allows both Windows and the embedded Guest OS to run native code without any modification to the drivers and other memory-mapped utilities.
- Hardware access and interrupt mapping of specific I/O’s to the embedded OS, so that they can continue to be served with legacy drivers under the embedded Guest OS.
- Means of emulating legacy I/Os to minimize the rewriting of legacy code. Thus, a legacy ISA-based system can easily be migrated to a state of the art and less expensive single-board computer with the simple re-direction of the ISA peripheral to an equivalent on-board PCI device.
- Access to Windows I/O resources by the embedded OS via virtual I/O device drivers.
- Shared Memory block to enable Windows and the Guest OS to exchange information – supported by an API.
Figure 1.1 - eVM for Windows

1. Multi-core Intel Processor with VT-x.
2. Unvirtualized Windows 10, 8.1, 8, 7, Vista or XP (32- or 64-bit versions) – running on a dedicated processor Core.
3. Un-modified headless Guest OS – Real-time or other PC compatible embedded OS – running on dedicated processor Core.
4. eVM for Windows hypervisor runs on its own independent of Windows and the Guest OS.
5. Virtual I/O allows communication between the Guest OS and Windows.
6. I/O devices that are under the control of Windows.
7. I/O devices that are under the control of the Guest OS
8. Memory that is partitioned for the exclusive access by the Guest OS or Windows.
9. Shared memory option for messaging API to enable Guest OS and Windows applications to pass messages.
2 Quick Start Guide

It is important to understand the theory behind eVM virtualization before attempting to install and use this product. The next section provides the technical details and background of virtualization and eVM. If the technical specifics of eVM are already understood and you cannot wait to get started, sections 4.1 through 4.4 provide the installation and configuration instructions. Section 4.1 walks through the eVM installation process. Section 4.2 describes installing virtual serial ports. Section 4.2.2 describes configuring the virtual Ethernet device. Finally, Section 4.3 shows how to start the sample Guest OS, iRMX, provided with the installation.

Important: Verify the target platform for the appropriate VT-x and VT-d features required to support eVM for Windows. The installation runs a verification tool called VtProbe. The installation aborts if the target platform does not meet the virtualization criteria. Download the stand-alone VtProbe tool (VtProbe.exe) from www.tenasys.com. Use VtProbe to verify the platform’s compatibility with eVM before starting the product installation.
3 Product Technical Description

3.1 What is Real-Time embedded virtualization?
Embedded virtualization is a methodology used to partition the hardware resources of a platform between a host operating system and a guest operating system so critical hardware resources can be dedicated each operating system. Embedded virtualization optimizes and minimizes the virtualization overhead for the Guest OS. Real-time embedded virtualization ensures that critical hardware resources are expressly controlled by the Guest OS with minimum, deterministic device latency and minimal CPU overhead.

3.2 Theory of Operation of eVM
The eVM virtualization platform provides a Virtual Machine (VM) that hosts real-time and embedded operating systems running alongside Microsoft Windows. The Virtual Machine Manager (VMM) provides the hardware resources and a Guest OS environment whereby a Guest OS can run without modification.

3.2.1 What is eVM?
eVM is an embedded virtualization environment hosted on the Windows platform allowing consolidation of the real-time control capability of a Guest OS with the Windows capability on a common hardware platform.

eVM is a Virtual Machine Manager (VMM) that uses the Windows platform to create a Virtual Machine (VM) capable of supporting a Guest OS that can run as a real-time OS without degradation in real-time performance. eVM is designed to have minimal impact on the Windows platform. Some embedded products split processing between two computers: one for real-time and the other for Windows interface. Using the eVM virtualization environment allows you to take advantage of both the Windows services and the real-time Guest OS services on the same hardware platform. This provides a reduction in hardware cost by not having separate hardware systems, one to support Windows and the other to support the real-time OS. The tight connection of Windows and the Guest OS in the eVM environment provides some advantages for communications and interaction between Windows and the Guest OS. While the design of eVM has real-time performance in mind, it also supports non real-time Guest OSs for any application requiring a mixed OS environment.

3.2.2 Introduction to virtualization
Virtualization is the abstraction of computing resources. Abstraction separates the Guest OS from its physical resources. When supporting multiple operating systems on the same platform, some interfaces for a given operating system are emulated, and some are mapped directly to a given hardware interface. The emulation of the virtual interfaces is the key to successful virtualization.

Many of Intel’s new processor and chipset families include specific hardware capabilities that provide virtualization capability and improve the efficiency of the virtual environment
through hardware support. These hardware features are Intel Virtualization Technology. Two of the pieces of this Virtualization Technology used by eVM are VT-x, Intel Virtualization Technology for processor support; and VT-d, Intel Virtualization Technology for Directed I/O, which is chipset technology.

3.2.3 eVM and Virtualization technology

eVM is designed to take advantage of the Intel Virtualization Technology to run a Guest OS without having to modify the OS, drivers and interfaces. All that is required is to configure the Guest OS for the virtual machine (VM). The Guest OS and the applications running on the Guest OS handle important interfaces directly, and not through a virtualized layer. This allows the Guest OS to run in real-time without modification or customization to the virtual environment.

eVM partitions the platform. Windows drive some resources exclusively, and the Guest OS drives other resources. To maximize performance, eVM dedicates a whole CPU hardware thread to the Guest OS environment. Having no virtualization in the dedicated hardware thread, it is available to run the Guest OS software whenever the Guest OS is ready to run. Devices that are critical to the applications running on the Guest OS are dedicated to the Guest OS and are directly accessible to the applications running on the Guest OS without going through a virtualization layer. The VT-x technology assists the CPU to translate memory addresses in accessing the memory dedicated to the Guest OS.

There are also two virtual interfaces for communication between Windows and the Guest OS. There is a virtual serial interface and a virtual Ethernet interface that emulate wired links between the operating systems. A single virtual Ethernet link and up to four virtual serial links are supported.

Finally, some devices emulated on the virtual machine, because Windows has control of the real PC hardware. Devices such as RTC, access to the PCI bus, timers, and basic core PC functionality fall into this category.

3.2.4 eVM system requirements (brief overview)

The following lists the hardware requirements for a host platform that can run eVM:

- A multicore Intel processor or a single core processor with hyper-threading.
- Intel Virtualization Technology for x86 CPUs, (Intel VT-x), and a BIOS that supports enabling VT-x.
- By default, eVM dedicates 320 MB of RAM to the Guest OS and eVM from the Non-paged Windows pool. The minimum memory requirement for eVM is 64 MB plus the minimum memory requirement for the Guest OS.
- In the case where the Guest OS accesses a device using DMA, it is desirable to have Intel Virtualization Technology for Directed I-O, Intel VT-d, which eVM uses for DMA remapping. If the chipset supports VT-d, then the platform BIOS must also enable VT-d.
It can be difficult to determine which PC platforms and motherboards have VT-x and/or VT-d support. Documentation is poor on much of the hardware and BIOS available today for specific details about Intel Virtualization Technology support. It may take some discussions with technical support from the hardware manufacturer, when trying to choose appropriate hardware. It may even require obtaining an evaluation sample for actual testing. TenAsys provides a tool, VtProbe, which checks a given platform for VT-x and VT-d support. See Appendix D.

3.2.5 Performance: How eVM addresses Guest OS / Windows Application Needs
As stated earlier, the goal of eVM is to provide a virtual environment, hosted on a Windows platform that looks and performs like a dedicated PC for the Guest OS. The following discusses the performance issues.

eVM provides bounded, real-time performance for the Guest OS, with deterministic interrupt delivery. To do this, eVM makes sure that Windows does not handle real-time interrupts, and prevents Windows from ever masking a real-time interrupt. The other part of deterministic interrupt handling is the design of the interrupt forwarding mechanism, optimized to reduce interrupt latency and jitter and to provide a direct handoff to the real-time Guest OS.

A dedicated hardware thread provides maximum hardware performance of the Guest OS. The Guest OS does not have to share that core or hardware thread with any other operating system.

Since the Guest OS has a dedicated hardware thread, Windows has its own dedicated hardware threads. Thus, Windows does not share CPU resources, which would reduce performance. Windows directly handles Windows interrupts. Aside from removing one core and some memory from the Windows system, there is minimal impact to Windows normal operation.

3.2.6 Communications between Windows and the Guest OS: Virtual Device Interfaces
When discussing the performance aspects of eVM, emphasis is on the isolation between Windows and the Guest OS, but there will be times when both OSs will want to communicate with one another and share data. With both OSs on the same hardware, there are some unique opportunities to provide efficient communication mechanisms between the OSs. For example, there is a Virtual Ethernet connection that provides communication between the Guest OS and Windows. This has an NDIS driver on the Windows side and an NE2000 interface on the Guest OS side. This virtual interface uses shared memory to move packets between Windows and the Guest OS as if there were a wired Ethernet link between the two OSs. There are also virtual serial ports that provide data interchange between Windows and the Guest OS via a standard serial port interface.
3.2.7 Communications between Windows and the Guest OS: Shared Memory
Windows and the Guest OS can communicate via a block of shared memory. eVM manages shared memory allocated from the Windows non-paged pool. Part or all of the memory can be mapped to a Windows process while access from the Guest OS application is via a virtual PCI device which exposes the memory through a BAR (Base Address Register) in the PCI. See section 4.7 for further setup and usage details.

3.2.8 eVM Architecture: Partitioning the PC between Windows and Guest OS
This section looks at how the PC platform is partitioned using eVM to separate Windows and a Guest OS.

3.2.8.1 Hardware Threads
The eVM installation removes one hardware thread from Windows control and dedicates it to the Guest OS. In a multicore platform, the installation dedicates one core to eVM and the Guest OS. Standard Windows configuration mechanisms manage the system resources.

3.2.8.2 Memory
The eVM installation removes the memory required for eVM and the Guest OS from the Windows memory pool and dedicates it to the Guest OS using similar standard Windows configuration mechanisms, as was used for the thread control. This memory is non-paged, because it is no longer under Windows control. eVM allocates and dedicates the memory to eVM and the Guest OS before Windows boots.

Figure 3.1 - Memory map for a 32 bit installation
Translation of memory addresses present zero-based addressing to the virtual PC, so it looks like a real dedicated PC. VT-x supports memory remapping in hardware resulting in no modification of the Guest OS.

### 3.2.8.3 Interrupts

In order to access a device directly from the Guest OS, it is necessary to dedicate any interrupt resource associated with that device to eVM, so eVM can manage it and not Windows. eVM maintains the control of these resources and then passes control to the Guest OS. With eVM, not Windows managing these interrupt resources; the Guest OS maintains a deterministic system environment.

### 3.2.8.4 Devices

Virtual device is one class of device: disc controllers, RTC, PCI bus, host bridge on the PCI bus, timers, and interrupt controller. These software devices appear to the Guest OS as if they were real hardware devices. These virtual devices are all part of the virtual machine.

A second class of devices are real devices assigned to the Guest OS and taken away from Windows. The Guest OS has exclusive access to these devices and does not share them with Windows. A dedicated PCIe Ethernet card is an example of this. eVM
can dedicate PCI or PCIe devices to the Guest OS such that an application specific device is managed by the Guest OS and Windows is not aware of the device.

3.2.9 Resources that are not isolated between Windows and the Guest OS
Per the virtualization architecture, eVM provides isolation between Windows and the Guest OS, but software cannot manage some shared resources where the usage of these resources by one of the OSs can affect the other OS. The memory controller, internal graphics controller (also uses the memory controller), and the front side bus are examples of these kinds of resources. For example, heavy usage of the memory controller by one of the OSs is going to affect performance overall. The selection of processor, chipset, and clocking has an influence on the overall system performance. For the very highest performance applications, careful choice of hardware and evaluation with the appropriate measurements determines the optimum hardware platform.

3.2.10 Device Management
The eVM Device Manager utility manages devices dedicated to the Guest OS. Before using the eVM Device Manager to manage devices, it is important to know what devices can be dedicated to the Guest OS, what devices cannot, and why.

3.2.10.1 What is a device
The devices that eVM can dedicate to a Guest OS are PCI devices, PCI Express (PCIe) devices, and legacy devices (COM, LPT, etc.). These devices come in a variety of flavors; so it is important to determine if the device can, in fact, be dedicated to a Guest OS. The hardware features used to determine the device’s suitability for being a dedicated, Guest OS, device are:

1. Device Access: Does the device use memory mapped registers or I/O mapped registers?
2. Interrupt capability: Does the device generate an interrupt; and if so, what type of interrupt does it generate?
3. Bus Master capability: Is the device a bus master device, i.e. does the device generate its own memory accesses?

A USB memory disk is an example of a device that cannot be dedicated to the Guest OS. That is because the USB memory disk interfaces through the USB controller and not directly to the bus nor is it accessed directly by the device driver.

3.2.10.2 Identifying your device’s capabilities
The following describes the three classes of device capability that are of interest when configuring devices in eVM for dedicated access by the Guest OS.

1. Device Access:
   The method of device access to the device registers must be determined. The device can either use the I/O space or be memory mapped. The CPU Virtualization Technology, VT-x, can manage the mapping of direct memory or I/O space for the virtual
machine. The VT-x mapping also provides memory protection. eVM traps and blocks any access to an unmapped memory or I/O space. Thus errant driver software in the Guest OS could not corrupt the Windows platform.

Legacy devices need explicit configuration. Their I/O ranges must be determined and configured in eVM, so that eVM can allow access to these I/O ports by the Guest OS.

PCI and PCIe devices are discoverable. All the PCI/PCIe register and memory access information is contained in the PCI header. eVM reads the Windows configuration for the device and sets up the device’s registers and memory to be transparently accessible from the Guest OS. No interaction is necessary by the user to configure this device after assignment to eVM.

2. Interrupt Capability:
   The interrupt capability of the device needs to be determined. If the device has interrupt capability, then it needs to be determined if the interrupt is used by the Guest OS. eVM then needs to be appropriately configured.

3. Bus Master Capability
   Finally, it needs to be determined if the device is a bus master device, i.e. does the device contain a DMA engine and can it generate its own memory accesses.

3.2.10.3 Interrupts
   There are two classes of interrupts. The first class is the legacy interrupts, or IRQs that have the device interrupt lines connected to specific inputs of an interrupt controller. The interrupt controller manages the delivery of each interrupt IRQ to the CPU. The second class is the more modern Message Signaled Interrupt (MSI), where the device performs a special bus write to signal the chipset. The device has no interrupt lines. Instead, it has special parameters that the device writes to a memory space reserved for MSI interrupt activity. The interrupt goes directly to the CPU when the device writes to the MSI memory space.

   eVM handles both classes of interrupts for devices that are dedicated to the Guest OS. eVM catches all dedicated device interrupts. Then eVM forwards those interrupts to the Guest OS when it is ready to receive interrupts. The virtual machine provides virtual interrupt services for the Guest OS. The Guest OS may mask and unmask interrupts in the virtual machine. eVM monitors the interrupt masks and holds off interrupt forwarding for interrupts that are masked and forwards those interrupts to the virtual machine when they are unmasked.

3.2.10.4 Software Considerations
   To provide a PC-like virtual machine, eVM must manage both classes of interrupt and act as an interrupt controller for the virtual machine. The Guest OS interacts with the virtual interrupt controller of eVM as if it were a hardware interrupt controller. The eVM
virtual interrupt controller provides interrupt signaling to the Guest OS, interrupt masking capability, and interrupt enabling and disabling. Therefore, if the Guest OS has interrupts disabled, then eVM must wait and hold the interrupt before passing the interrupt to the Guest OS.

The virtual machine that supports the Guest OS has a different interrupt mapping for the virtual machine than the actual hardware platform. The virtual machine presents an 8259 legacy interrupt controller as the means for interfacing the dedicated device interrupts to the Guest OS. eVM maps both classes of interrupts, legacy and MSI, as IRQ's in the virtual machine. The Guest OS sees all interrupts as legacy IRQ interrupts in the virtual machine irrespective of whether the real hardware interrupt was a legacy interrupt or an MSI interrupt. eVM is acting like a software interrupt controller. The Virtual Machine specification describes specific mapping of devices to IRQs in the virtual machine. See Appendix B.

**Figure 3.3 - Interrupt management**

**Device A:** Is a PCI device and the driver supports MSI – *Note: Windows XP and earlier versions of Windows do not support MSI.*

**Device B:** Windows handles Device B normally.

**Device C and D:** Devices share the same IRQ line and the Guest OS needs to access Device D, so passing the device to the Guest OS does not pass the IRQ. Device C
remains a fully functional Windows device. The Guest OS may poll Device D, but
Device D cannot deliver an interrupt to the Guest OS.

**Device E:** Passes to the Guest OS. eVM services the IRQ E then delivers it to the
Guest OS as a virtual IRQ.

**Device F:** Is an MSI device. eVM handles the MSI and delivers it to the Guest OS as a
virtual IRQ.

*Note:* **eVM always handles the IRQ, meaning that eVM's interrupt handler runs
when the interrupt fires. The eVM interrupt handler converts the interrupt to an
input for the virtual Peripheral Interrupt Controller (PIC) and forwards the
translated vector to the Guest OS.**

**Example: Ethernet controller capture**

If an Ethernet controller that uses IRQ18 is dedicated to the virtual machine and the
Guest OS, eVM installs an interrupt handler for IRQ18 to manage the real interrupt on
the hardware. eVM maps that interrupt into one of the virtual machine’s IRQs, a different
IRQ from the actual hardware interrupt. In operation the eVM interrupt handler handles
the IRQ18. IRQ18 maps to an available IRQ on the virtual machine and the interrupt
forwards to the Guest OS when the Guest OS is ready to handle it.

There is some additional latency introduced by the forwarding and mapping of the real
hardware interrupt to the interrupt on the virtual machine, but this is a fixed and bounded
latency. Because this is a fixed and bounded latency, the Guest OS can still run in a
deterministic fashion. eVM has been optimized to minimize this latency, but this latency
is also determined by the hardware. Some CPU, chipset, and board designs are faster
than others are, but the hardware contribution to this latency is fixed. The virtual
machine is still deterministic.

**3.2.10.5 System hardware considerations – what eVM can or can’t do!**

There are some restrictions when mapping interrupts to the virtual machine and still
maintain a deterministic system. Legacy interrupts from PCI devices present a
restriction, because multiple PCI devices can share a single interrupt. In fact, the PCI
design shares an interrupt. This interrupt sharing presents a problem, if the devices
share an interrupt configured for different OSs. If one device is to be used as a
Windows device and another device is to be used as a Guest OS device and they both
share the same interrupt, then there is no way to provide a deterministic response for the
Guest OS device, since it is dependent on the running of a Windows interrupt handler.

When an interrupt occurs on a shared interrupt, the interrupt handlers must run for all
devices sharing that interrupt to determine which device caused the interrupt and to
clear the interrupt so it is available to handle a subsequent event. This would mean that
the Guest OS would be dependent on a Windows interrupt handler running to maintain a
deterministic system. eVM does not allow to assignment of a device to the Guest OS
and a device to Windows that share a common interrupt. The eVM Device Manager
flags an attempt to do this as a conflict. A conflict of this type is a property of the
motherboard and/or chipset design. It is not possible to configure around a shared interrupt, so it is important to choose a hardware design that allows interrupt isolation of devices dedicated to the Guest OS. One alternative for a device that has an interrupt conflict with another device used by Windows is to run the device in a polled mode. The device configuration should have interrupts turned off with register access dedicated to the Guest OS. MSI interrupts do not have any sharing problem.

3.2.10.6 DMA

A DMA device can generate its own memory accesses without CPU intervention. Such a device is a bus master device. For bus master devices, the Intel Virtualization Technology, VT-x, plays no part. VT-x manages address remapping for the CPU. Addresses that the DMA engine of a bus master device uses must remap for the virtual machine. The view of the memory assigned to the virtual machine starts at address zero, but the actual physical address offset of that memory from memory address zero is an amount dependent on the system configuration. Any addresses that the DMA engine would use must map to the actual physical address of the memory, adding the physical memory offset to the virtual memory addresses of the virtual machine.

There are two ways to handle DMA address mapping. One way is to modify the device driver so that it does the DMA address mapping. This requires preparing and installing a modified device driver on the Guest OS. The modified device driver must be able to determine the memory offset of the virtual machine’s physical memory and apply that offset to any DMA operation. See Appendix E for details. The other method for DMA address mapping is to use hardware specifically designed to provide address mapping of bus master devices. Intel Virtualization Technology, VT-d or Virtualization for Directed I/O, is the hardware mechanism that is available for DMA address mapping. The VT-d technology resides in specific chipsets that contain a DMA redirection unit that translates bus master address accesses on the fly.

3.2.10.7 Hardware consideration – VT-d

eVM defines an address domain that is assigned to the virtual machine for the Guest OS. Any device assigned to eVM also operates within that memory domain in hardware. eVM translates any address that is generated in that domain and maps it to the physical address of the virtual machine domain. If a device tries to access memory outside its domain, eVM, providing memory security and hardware protection, as well as memory mapping, blocks it.

If VT-d is not present, a standard bus master driver will not work and will compromise Windows, because a bus master device that is operating with a device driver that has not been modified for the virtual machine, will address the wrong area of memory, and will almost certainly corrupt Windows memory. If VT-d is not present, modification is required of the driver for the bus master to work with the virtual machine and do the appropriate address translations to function in the virtual machine. Therefore, it is highly
desirable to have VT-d supported by the hardware and enabled through the BIOS, if bus master devices are to be dedicated to the virtual machine, and hence the Guest OS.

3.2.11 What Happens when a Device is assigned to eVM

The following actions occur when the eVM Device Manager assigns a device to eVM from Windows:

- Remove the device driver from Windows to stop access.
- Device identity information passes from Windows to eVM, so eVM can recognize it. When eVM next starts up; eVM owns this device.
- Device identity passes to eVM with its IRQ or without its IRQ. If passing the device with its IRQ, an interrupt sharing violation is checked. If there is an interrupt sharing violation, the eVM Device Manager posts a warning.
- When eVM starts and sees that the device belongs to eVM, it obtains the device identifier and discovers the device. If the device has an IRQ, eVM assigns an interrupt handler for it. If it is a PCI device, eVM inserts the device into one of the virtual PCI slots provided by the virtual machine. The virtual machine created by eVM has five virtual PCI slots for dedicated PCI devices.
- If the device is a PCI device, eVM Scans the base address registers for that device and configures the virtual machine so any access to the base address registers go directly to the device and are not intercepted by eVM.
- If the device has an interrupt, eVM assigns a virtual IRQ in the interrupt map of the virtual machine.
- If the device does not have an IRQ, eVM disables the interrupt in the virtual PCI device header of the virtual machine.
- If this device is capable of being a bus master with VT-d enabled, then eVM assigns the device to the VT-d domain of the virtual machine for hardware address translation in the Guest OS.
- If this device is capable of being a bus master but VT-d is not present, the eVM Device Manager presents a warning. It is the responsibility of the developer to either provide the modified driver or disable bus master capability.
4 Installation

4.1 Installation of eVM for Windows

The following provides a systematic guide of the installation screens with comments as necessary, including Windows and eVM setup and License activation. The following description is for installation on Windows 10/8.1/8/7/Vista.

Run `evm..._installer.exe`, which begins the installation process by installing the Microsoft Visual C++ 2008 Redistributable Installer. It first extracts files to temporary folder, detects what applications are loaded on the system and, if not found, loads Microsoft Visual C++ Redistributable.
1. This message displays if and only if the computer does not support VT-x technology. See Appendix B1 – System Requirement for more information.

Figure 4.1 - VT-x not supported message
2. With the Visual C++ 2008 Redistributable installation complete, the eVM Installation Wizard launches automatically. See Figure 4.2. Click **Next** to continue.
3. Read and accept the TenAsys eVM Software License Agreement; and click **Next** to continue. See Figure 4.3.
4. Read the Release Notes as in Figure 4.4 and click **Next** to continue.

![Figure 4.4 - Release Notes](image-url)
5. Select the folder for the eVM installation. The default location is “C:\Program Files (x86)\eVM\”. See Figure 4.5. Click Next to continue.
6. Select the type of installation. For Complete Installation, as in Figure 4.6, eVM with virtual port support, all the utilities, and a sample Guest OS: iRMX is installed. A custom installation is the other choice. After selecting the installation type, click Next to continue.

![Select Installation Type](image)

**Figure 4.6 - Choosing Standard Installation Type**
7. The eVM Installation Wizard prompts for your name, organization, and the eVM activation code received with the eVM purchase and download. Leave the activation code blank for a trial installation. See Figure 4.7. Make sure to fill out all the fields properly and **be sure that the system has an Internet connection, for verification of the activation code with TenAsys over the Internet.** Click **Next** to continue.
8. When the activation code verified or skipped, the eVM Installation Wizard prompts to confirm to proceed with the installation. See Figure 4.8. To check or change anything at this point, click the Back button.

To proceed with the installation, click Next.
9. A progress dialog shows the status of the installation of the components and features. See Figure 4.9.

![Figure 4.9 - eVM Installation Wizard Progress Dialog](image)
10. After installation of all components and features of eVM, the Wizard indicates that the installation is completed successfully. See Figure 4.10. To view the Wizard installation log, check the **checkbox**. Note the saved log path. If you have trouble with the installation, TenAsys technical support may ask to see this log to help determine the problem. Now click **Finish**.

![Figure 4.10 - eVM Installation Wizard Successful Installation](image)

The first installation of eVM for Windows on the target system loads TenAsys drivers to support the virtual serial port and virtual network adapter. A prompt appears prior to the installation, as shown in Figure 4.11. Note: If you don’t check the box “Always trust software from "TenAsys Corporation”, this prompt appears each time additional virtual ports are installed.

![Figure 4.11 - TenAsys Driver loading notification](image)
11. If you chose to view the installation log, it is something similar to Figure 4.12. After viewing the log, you can close Notepad.

Figure 4.12 - eVM Installation Log
12. After clicking **Finish** in the dialog shown in Figure 4.10, the Wizard prompts to restart the system. See Figure 4.13. A restart is required, so eVM can take the memory that it needs for eVM and the Guest OS from Windows. In general, resizing Windows available memory requires a restart of the Windows OS. Click **Yes**, and wait for the system to reboot.

![System restart required](image)

**Figure 4.13 - Restart Prompt**

13. After the system has rebooted, wait for the eVM services to start, indicated by the ‘e’ icon in the system tray. See Figure 4.14.

![eVM System Tray Icon](image)

**Figure 4.14 - eVM System Tray Icon**

14. Click on the **eVM tray icon** and select **eVM Configuration** to launch the **eVM Configuration Panel**. See Figure 4.15.

![eVM Configuration Panel](image)

**Figure 4.15 - Launching eVM Configuration Panel from the system tray**
15. Launching the *eVM Configuration Panel* from the system tray brings up the *eVM Configuration Panel* with four icons providing setup and controls to run eVM for Windows.

![eVM Configuration Panel](image)

**Figure 4.16 - eVM Configuration Panel**

---

TenAsys Corporation 1400 NW Compton Drive Suite 301 Beaverton, OR 97006, USA.
Rev 4.0
16. Clicking on the **Node Management** icon in the **eVM Configuration Panel** brings up the **eVM Node Management Panel**. Click on the **Guests** tab in the right-hand side. If the **Complete eVM Installation** option was installed, the system iRMX is listed in the Guest selection line and the I/O setup for the selected guest below. The Guest OS starts running with the **Start guest** button. Clicking on the **System** tab displays the Windows configuration parameters. Clicking on the **Kernel** tab displays the eVM Configuration parameters. Clicking on the **Log** tab displays some of the install log.

![eVM Node Management Panel with iRMX Guest OS Installed](image)

**Figure 4.17 - eVM Node Management Panel with iRMX Guest OS Installed**

With the **Complete Installation**, eVM is installed with the two virtual serial ports and sample Guest OS, iRMX, complete with a sample configuration.
17. Going back to the eVM Configuration Panel and clicking on the eVM Device Manager lists the Windows devices that can be passed to the Guest OS running on eVM (Figure 4.18) and any device that has already been passed to the Guest OS. Selecting all devices under the View menu tab, all the devices including those that cannot be transferred to eVM are listed. See Figure 4.19. Note that this screen includes the TenAsys virtual Ethernet Port and virtual COM ports. The right hand-side of the window lists out installed devices and the legacy (hardware) interrupt they use.

**Figure 4.18 - Device Manager showing the Windows devices that can be passed to eVM**
Figure 4.19 - Device Manager showing all the devices, including ones that cannot be passed to eVM (the Guest OS)
18. Compare the eVM’s Device Manager to Window’s Device Manager by selecting the
Computer Management tool after a right click on the Windows icon, and then click on
Device Manager. See Figure 4.20. The list of devices allocated to Windows should
match. Devices allocated to eVM appear in the Windows Device Manager but with
the prefix “eVM”.

![Figure 4.20 - Windows Device Manager]

The list includes the TenAsys Virtual Ethernet Adapter, as well as the two emulated or
virtual serial ports. The property of each virtual serial port uses the next available COM
device, as shown in this system configuration, resulting in the virtual serial ports
becoming COM6 and COM7. Discussion of the configuration of the virtual Ethernet
Adapter follows in 4.2.2.

4.2 Virtual Ports

Using eVM allows two discrete hardware platforms, each with its own operating system
and applications, consolidated onto a single hardware platform. Not only does this
provide a reduction in hardware required to host the two systems, it also provides a
unique opportunity to simplify communications and connectivity between them. Instead
of having real hardware communications devices on each discrete hardware platform
that are hard-wired together, eVM can provide a virtual software connection between the
two systems, further reducing the hardware requirement to host the two interconnected systems.

### 4.2.1 Installation of Virtual Serial Ports

The eVM installation installs two virtual serial ports. If you want to add additional serial ports, eVM supports up to four virtual serial ports, you can add them through the eVM Configuration Panel, then selecting the eVM Node Management icon and selecting the Kernel tab. Look at a system with two (2) serial ports installed. See how to install and configure another virtual serial port, a virtual COM device. Each virtual serial port provides a virtual COM device for the virtual machine that the Guest OS can use for serial communications and a corresponding virtual COM device in Windows that looks like a standard COM device wired to the virtual COM device in the virtual machine. eVM manages this connection between the two COM devices using a shared memory interface.

1. Open the eVM Node Management Panel, and in the right panel click on the Kernel tab. See Figure 4.21.

![Figure 4.21 - eVM Node Management Panel Showing System Configuration with two Virtual Serial Ports](image)

The system configuration parameters are in the right pane of the eVM Node Management Panel. Note that there are two Virtual serial ports used at this time in the example.
2. Click on the dropdown on the box next to Virtual serial port 3 to specify a specific COM port number. See Figure 4.22. Select COM8 for this example.

![Figure 4.22 - Virtual Serial Port COM Port Selections](image-url)
3. To apply this configuration change, save the new configuration. From the eVM Node Management Panel, select the **Save** button at the bottom of the window. See Figure 4.23. Note: A screen prompt as shown in Figure 4.24 - Confirmation to save configuration change may appear.

![Figure 4.23 - Saving eVM Node Management Panel Configuration](image)

![Figure 4.24 - Confirmation to save configuration change](image)
4. Confirm the addition of the virtual serial port. Open Windows Explorer, right-click on Manage and then click on Device Manager. There are now three (3) virtual serial ports: TenAsys Com Emulator driver with the third associated with COM8. See Figure 4.25.

Figure 4.25 - Windows Device Manager Showing added Virtual Serial Port

Applications in Windows can now access this virtual COM port by connecting to COM5.
5. Connect the virtual COM ports in the Guest OS to the virtual COM ports of the Windows. Open eVM Node Management Panel, select the Guest OS in the right pane (in this example it is iRMX), and expand the serial section in the right panel. See Figure 4.26.

![Figure 4.26 - eVM Node Management Panel Showing Guest OS Serial Device Settings](image)

6. For each COM port connected to a virtual COM device in Windows, select the **COMn access**, and select **Windows**. In this example, iRMX has been set up for COM1 and COM1 to access Windows virtual COM devices.

7. To select the COMn device (where n = 1, 2, 3, ..) in Windows that the Guest OS COMm device will connect to, select **COMn Windows device**, and select the desired COMn device for Windows. For this example connect iRMX COM1 to Windows COM6; and connect iRMX COM2 to Windows COM7.

### 4.2.2 Configuring the Virtual Ethernet Device

In this example, consolidating two platforms interconnected via Ethernet, onto one platform, with the Ethernet connection managed by eVM via virtual Ethernet devices. The consolidated system consists of Windows as the host OS and Windows CE as the Guest OS.

There are two parts to setting up eVM to manage the Virtual Ethernet connection: the virtual Ethernet device on the Windows side, and the virtual Ethernet device on the Guest OS side.
Configuring the Windows side:
The following step-by-step guide is for Windows 10/8.1/8/7/Vista.

1. Open the Windows Control Panel and click on Network and Sharing Center. See Figure 4.27.

2. In the upper left of the Network and Sharing Center, click on Change adapter settings. See Figure 4.28.
3. The *Network Connections* appear with one of the network connections the *TenAsys Virtual Ethernet Adapter*. See Figure 4.29. Double-click on the *TenAsys Virtual Ethernet Adapter*.

![Network Connections](image)

*Figure 4.29 - Windows Network Connections*
4. See the *Local Area Connection Properties* for the TenAsys virtual Ethernet Adapter. This example uses TCP/IPv4, so uncheck the checkbox for **Internet Protocol 6 (TCP/IPv6)**. See Figure 4.30.

![Ethernet 3 Properties](image)

**Figure 4.30 - TenAsys Virtual Ethernet Adapter, Windows Local Area Connection Properties**
5. Select Internet Protocol Version 4 (TCP/IPv4), and click the Properties button. See Figure 4.31.

![Figure 4.31 - Selecting TenAsys Virtual Ethernet Adapter TCP/IPv4 Properties](image-url)
6. In the Internet Protocol Version 4 (TCP/IPv4) Properties window, select the **Use the following IP address:** radio button, and enter **10.0.0.1** in the IP address. Keep the default Subnet mask: of **255.0.0.0**. See Figure 4.32.

![Internet Protocol Version 4 (TCP/IPv4) Properties](image)

**Figure 4.32 - Setting the Static IP Address for the TenAsys Virtual Ethernet Adapter**

7. To accept this new IP address setting, click **OK**.

8. Close the Network Connections window. The Windows side configuration of the virtual Ethernet connection is complete.
Configuring the Guest OS side:

1. Open eVM Node Management Panel, select the Guest OS, and select the Network section, which has an item named Virtual Ethernet (NE2000). See Figure 4.33.

![Image of eVM Node Management Panel, Guest OS Virtual Ethernet Device Configuration]

Figure 4.33 - eVM Node Management Panel, Guest OS Virtual Ethernet Device Configuration

1. In the box next to Virtual Ethernet (NE2000), select Yes to enable the virtual Ethernet device.

2. The window shows the configuration parameters for the virtual Ethernet adapter for the virtual machine. Match the I/O address, and interrupt to what the Guest OS expects.

Note: we discuss more about the Guest OS device parameters in general, and with respect to QNX and Windows CE specifically in Section 4.5.

3. The Guest OS configuration needs to use an IP address on the same sub-net as the virtual Ethernet adapter on the Windows side. In this example with iRMX, we entered 10.0.0.2 as the IP address in the boot loader. The Guest OS now sees a virtual NE2000 Ethernet device at I/O 0x280, interrupt 12 and configures its NE2000 driver to use IP address 10.0.0.2 for that device.

The actual Guest OS needs to talk to the IP address of the Windows side (10.0.0.1); and when the Windows side responds, it needs to know to talk to the IP address of the Guest
OS side (10.0.0.2). In the Windows CE example, the Windows host expects to find a client at 10.0.0.2 and send it the image.

4.3 Running the Sample Guest OS, iRMX
Now with eVM installed, with a virtual COM device installed and configured the sample Guest OS: iRMX can run.

1. In Windows, open a terminal emulator program, and configure it to connect to the first virtual COM device. Recall from Figure 4.26 the first virtual COM device in our example was COM6.

2. Open eVM Node Management Panel and select iRMX under eVM Guest in the Guests tab. See Figure 4.34.

3. Click the Start guest button

![Figure 4.34 - eVM Node Management Panel, Selecting the Guest OS](image-url)
4. The Guest OS, iRMX, boots in the virtual machine. An eVM Console window opens, and the default debug output from iRMX is output to the virtual COM device and shows up in the terminal emulator window. See Figure 4.35.

Note: In this example the freeware PuTTY application is used, but any terminal emulation application can be used.
5. Once iRMX boots, type iRMX system commands in the terminal emulator and iRMX responds exactly as if it were running on its own dedicated hardware platform.

6. To stop iRMX select the Stop guest button in the eVM Node Management Panel started from the toolbar. The eVM Configuration Panel prompts to confirm stopping the Guest OS.

![eVM Node Management Panel, Stopping the Guest OS](image)

7. Click Yes, and iRMX is stopped.

### 4.4 Installation of a Guest OS

Installing a Guest OS on eVM consists of obtaining a disk or image of the target system, and creating a configuration for the Guest OS.

Copy the disk image from an existing system; or in some cases, installed it directly from the installation materials. The eVM Node Management Panel creates the configuration.

For an up-to-date listing of verified Guest OSs, check the following website: [www.tenasys.com](http://www.tenasys.com), and click on **Support-> Knowledge Base**.

#### 4.4.1 Configure the existing system

Referring to the Virtual Machine specification, Appendix B.2, configure the system to match the Virtual Machine hardware interface. The primary point to notice is the
absence of a video device and keyboard. Usually, any console I/O requires redirection to a serial port.

4.4.2 Make a disk image of the system
When configuring the system, make a disk image by using a suitable tool. Perhaps making a binary image of the disk with an imaging tool after attaching the disk to the Windows system. If the entire drive is not copied, make sure the Master Boot Record (MBR) of the disk is included, as well as the partition containing the application software. There are many Windows tools available to do this, such as the shareware utility: SelfImage.

If the image is from a Compact Flash (CF) drive, the CHS values in the configuration file may need adjustment to boot correctly. Some CF parts have unusual configurations that do not typically copy with the data from the drive.

Copy the disk image to the directory where eVM stores its images. This varies depending on the version of Windows, but the environment variable %EVMCFG% always contains the name of this directory. Copy the disk image file to this directory.

4.4.3 Make a Guest OS configuration
A system configuration needs creation. Assume that the system has one disk image, and the console is the first serial port (COM1).

1. Open the eVM Configuration Panel (use the eVM system tray icon or navigate to Control Panel->eVM Configuration Panel).
2. Select the eVM Node Management window.
3. Start a new configuration by clicking on New guest in the Guests tab.
4. In the pop-up window, name the new guest configuration. Configuration names may have up to 12 characters.
5. Assign the disk image. Expand section 2. ATA0 disk controller, and change “ATA0 enabled” to Yes.
6. In ATA0 master type, click the drop down arrow button at the right, select “Disk”. (The other option is for a CDrom image.)
7. In ATA0 master path, click the … button at the right, navigate to the disk image, and open it.
8. In section 4, “Serial”, change “COM1 access” to “Windows”. Note the Windows COM driver name. This is the other end of the virtual serial link from the Guest OS to Windows, and is the device opened by the terminal emulator.
9. In section 8, “BIOS settings”, change “Boot device 1” to Disk so the BIOS knows to look for a boot disk image. (Use CDrom for a CDrom image.)
10. Select Save to apply the change.

The configuration is now ready to test. Open a terminal emulator and use it to open the serial port from step 6 above.
Start the Guest OS by clicking on the **Start guest** button in the Guests tab.

### 4.4.4 Installing the Guest OS from installation materials

In some cases, the Guest OS can be installed using standard installation materials, so long as these do not depend on a video/keyboard console. If a floppy disk boot image is available you should copy this to the eVM Data Folder (%EVMCFG%) and assign the image to the floppy drive of the Guest OS configuration. To do this:

1. Open the **eVM Configuration Panel** and in the right panel, expand the Floppy device.
2. **For Floppy A enabled select Enabled.**
3. In **Floppy A path**, click on the ... button to the right, navigate to the floppy disk boot image file, and select it.
4. In section 8, “BIOS settings”, change “Boot device 1” to **Floppy** so the BIOS knows to look for a boot floppy image.
5. **Select Save** to apply the change.

### 4.5 Specific steps for certain Guest OSs.

Two example configurations are illustrated.

#### 4.5.1 Installation and configuration of QNX 6.4

This is an example of customizing a Guest OS and creating a disk image.

To install QNX 6.4 onto eVM for Windows create a configuration that boots without the use of a video card. A sample build file is included in this example:

1. Use mkifs to create a boot image from this build file, and then copy the boot image to `.boot`.

```bash
# # The build file for QNX Neutrino booting on a PC
#
[linker="ntox86-ld -T$QNX_TARGET/x86/lib/nto.link %h!=0, -Ttext 0x%t%)(d!=0, -Tdata 0x%d%) -o%o
 %i ![M-L%-i -uinit_%n -lmod_%n%]"
[virtual=x86,bios +compress] boot = {
    # Reserve 64k of video memory to handle multiple video cards
    startup-bios -s64k
    startup-script = {
        display_msg "QNX Neutrino 6.4 for eVM"
    }
}```
display_msg " "

# To save memory make everyone use the libc in the boot image!
# For speed (less symbolic lookups) we point to libc.so.3 instead of libc.so
procmgr_symlink ../../proc/boot/libc.so.3 /usr/lib/ldqnx.so.2

display_msg "----- Starting PCI services"
seedres
pci-bios
waitfor /dev/pci

display_msg "----- Starting console service"
devc-ser8250 -e -b57600 &
waitfor /dev/ser1
reopen /dev/ser1

# Default user programs to priority 10, other scheduler (pri=10o)
# Tell "diskboot" this is a hard disk boot (-b1)
# Tell "diskboot" to use DMA on IDE drives (-D1)
# Start 4 text consoles buy passing "-n4" to "devc-con" (-o)
# By adding "-e" linux ext2 filesystem will be mounted as well.
# [pri=10o] PATH=/proc/boot diskboot -b1 -D0 -odevc-con,-n4 -odevc-con-hid,-n4
display_msg "----- Starting EIDE driver"
devb-eide blk auto=partition dos exe=all cam quiet eide nobmstr
waitfor /dev/hd0

mount /dev/hd0t79 /
mount -tcd /dev/cd0 /fs/cd0

display_msg "----- Starting /dev/system/sysinit"
ksh -c /etc/system/sysinit

[type=link] /dev/console=/dev/ser1

# Include the current "libc.so". It will be created as a real file using
# its internal "SONAME", with "libc.so" being a symlink to it. The symlink
# will point to the last "libc.so.*" so if an earlier libc is needed
# (e.g. libc.so.1) add it before this line.
libc.so
libhiddi.so
libusbdi.so

# Include all the files for the default filesystems
libcam.so
io-blk.so
cam-disk.so
fs-qnx4.so
fs-qnx6.so
fs-dos.so
fs-ext2.so
cam-cdrom.so
fs-cd.so
fs-udf.so

# USB for console driver
devu-ehci.so
devu-ohci.so
devu-uhci.so
devh-usb.so
devh-ps2ser.so

[data=copy]
seedres
pci-bios
devb-eide
devb-umass
devb-ahci
devb-mvSata
devb-adpu320
devb-aha8
umass-enum
[search=${MKIFS_PATH}:${QNX_TARGET}/etc/umass-enum.cfg]
io-usb
io-hid
diskboot
logger
fesh
mount
devc-ser8250
#devc-con
#devc-con-hid

# These files will be unlinked after the system has started
# to release the memory back to the system. They are only
# needed during boot. If needed later on, they will be loaded
# from the boot device.
unlink_list={
    /proc/boot/devb-*
}

2. Now, configure the console device by editing /etc/config/ttys and add a line for /dev/ser1:

ser1 "/bin/login" qansi-m on
3. Configure the network parameters for the virtual Ethernet subnet between Windows and QNX. Define a subnet of 10.100.200.0, and give it two nodes, one for Windows and one for QNX. Edit `rc.local` to load the **ne2000 device** and configure it by adding the following lines:

```plaintext
io-pkt-v4 –dne2000 irq=12,ioport=0x280 –ptcpip
ifconfig en0 10.100.200.2 up
inetd &
qconn &
```

4. In Windows, edit the properties of the TenAsys Virtual Ethernet network connection and set the IP address to **10.100.200.1**.

5. Make an image of the disk. Use `dd` and write the image to another partition, or device (for example):

```plaintext
dd if=/dev/hd0 of=/mnt/tmp/qnximg.bin count=9900
```

*Note: the size (ifs) parameter must include both the boot sector and the partition included in the image.*

6. Copy the output file to the Windows `%EVMCFG%` directory and assign it as the first hard drive for the QNX configuration.

7. Make sure that there is at least one virtual serial terminal defined and add it to the configuration.

### 4.5.2 Installation and configuration of Windows CE

This is an example of installing a Guest OS using the materials provided by the vendor.

1. Create a DOS-bootable floppy diskette.

2. Copy the following files to it from the Windows CE 6.0 installation:
   `loadcepc.exe`
   `eboot.bin`

3. Create an autoexec.bat file including these commands:

   ```plaintext
   @echo off
   verify off
   loadcepc /v/e:280:12:192.168.99.2 EBOOT.BIN
   ```

*Note: this loads KITL using the NE2000 virtual Ethernet driver.*
4. Configure the Windows Virtual Ethernet driver to use IP address **192.168.99.1**.

5. Now, make a disk image of the floppy diskette, and copy it to the %EVMCFG% directory.

6. Add a floppy device to the configuration and assign this file to it.

   *Note: this loads the KITL components, so the Windows CE 6.0 installation can connect to the target.*

### 4.6 Driver configuration and setup

With eVM installed under Windows, the host OS, and with the Guest OS installed, devices need to be dedicated to the host OS, to the Guest OS, or shared between them. This section describes how to use the eVM Device Manager to dedicate devices.

#### 4.6.1 eVM Device Manager

Launch the eVM Device Manager from the eVM Configuration Panel by selecting **eVM Device Manager**. See Figure 4.37.

As an alternative, directly launch the **eVM Device Manager**:

- In Windows Vista or Windows 7 click the **Windows Start button**, enter “devconfig.exe” (or “devconfig64.exe” for 64-bit Windows versions) into Search programs and files and click on **devconfig[64].exe**. See Figure 4.38.
- In Windows XP select **Start->Run**, enter “devconfig.exe” (or “devconfig64.exe” for 64-bit Windows versions), and click **OK**. See Figure 4.39.
- In Windows 8, 8.1, or 10, use the search box for “devconfig.exe” (or “devconfig64.exe” for 64-bit Windows versions) or use the Run desktop app (Windows key + R), enter “devconfig.exe” (or “devconfig64.exe” for 64-bit Windows versions), and click **OK**. See Figure 4.40.
Figure 4.37 - Launching the eVM Device Manager from the eVM Configuration Panel

Figure 4.38 - eVM Device Manager launch from Windows Vista or Window 7
The eVM Device Manager displays two active panes. See Figure 4.41. The left pane shows two trees of device classes and devices: one controlled by Windows and the other managed by eVM. In either tree, the classes can expand and collapse.
Use the View menu to select between showing only devices that can be passed or all devices. Figure 4.41 shows the eVM Device Manager displaying only the devices that can be passed (Default). The View dropdown provides a choice for All devices.
Figure 4.42 - eVM Device Manager showing how to switch to show All devices

Figure 4.43 shows the eVM Device Manager displaying all devices. The View dropdown provides a choice for Devices that can be passed.
4.6.2 Passing a Legacy IRQ Device to eVM

For this example, choose COM1 as the legacy IRQ device. To confirm that this device fits the criteria, select **View->Devices that can be passed** from the dropdown and look at the right pane and see if COM1 is listed as a device with *Legacy interrupt usage*. Look at the left pane and expand the Windows devices **Ports (COM & LPT)** device node to see that COM1 is a Windows device that is a device that can be passed to eVM.
Figure 4.44 shows that COM1 is a Windows device that can be passed and is a legacy interrupt device.

In the left pane, click on the **Communications Port (COM1)**, and then either click on the **Pass to eVM** button. See Figure 4.45.
Or, from the dropdown, select **Action->Pass to eVM using legacy IRQ**, or right click on “Communication Port (COM1)”. See Figure 4.46.
When selecting the device to pass, the right pane shows COM1 selected, but at this point, the device configuration has not changed. See Figure 4.47.
The configuration must be saved to apply the requested change. Either click the **Save configuration** button, See Figure 4.48:
Figure 4.48 - Using the Save Configuration Button

Or, from the dropdown select **File->Save the configuration**. See Figure 4.49.
Prior to changing the configuration, if the passed device is a bus master, the eVM Device Manager shows a warning that the platform needs VT-d support to enable DMA transfers for the device. Without VT-d support, the device driver needs modification. See Figure 4.50.
Figure 4.50 - Warning that VT-d support is needed if the device is a bus master with DMA.

After clicking on **Yes** the device configuration is accepted, but not actually applied until the platform has rebooted. The eVM Device Manager prompts to restart the platform. See Figure 4.51.

Note: This is a Windows requirement and not all systems require a *Restart* for the configuration change to take effect. See *Windows Help* on Device Configuration for more information.
When the platform has restarted, launch the eVM Device Manager, and expand the **Ports (COM & LPT)** node under *Windows devices* and under *eVM devices*. See Figure 4.52. The Communications Port (COM1) is not on the Windows device list, but appears as an eVM device.
The legacy IRQ COM1 device has been passed to eVM for exclusive use by the Guest OS.

4.6.3 Selecting communication port modes

Virtual Serial Port configuration was covered in Section 4.2.2. Other serial port configuration options include Pass through and File. Select Pass through in the Guests tab of the eVM Node Management window after passing a physical communication port to eVM. Select File to send output from an Guest OS port to a file. The File option is handled by eVM using the Windows file drivers and does not involve any passed devices. See Figure 4.54.
4.6.4 Conflict Example when Passing Legacy IRQ Device to eVM

There is an issue with sharing an interrupt between Windows and the Guest OS. This would make the Guest OS dependent on a Windows interrupt handler running to maintain a deterministic system. eVM does not permit sharing a common interrupt between a device assignment to the Guest OS and a device assignment to Windows. The eVM Device Manager flags an attempt to do this as a conflict.

To illustrate this, open the eVM Device Manager, and under Windows devices, expand Universal Serial Bus controllers. See Figure 4.54.
In the left pane, select the first **USB Enhanced Host Controller** and, from the dropdown, select **Pass to eVM**.
In the right pane, see the conflict warning posted by the eVM Device Manager. See Figure 4.55.

### 4.6.5 Passing an MSI Device to eVM

In the example system, there are two Ethernet devices. One is an onboard controller that Windows is using, and the other is a PCI Express adapter with MSI capability that dedicated to eVM for the Guest OS to use. If the Windows device **Network adapters** node in the left pane of this system is expanded, the **Intel® Gigabit CT Desktop Adapter (MSI capable)** device is shown. To illustrate another way to pass a device to eVM, right-click on the **Intel® Gigabit CT Desktop Adapter (MSI capable)** device, and select **Pass to eVM using MSI**. See Figure 4.56.
Figure 4.56 - Right-Click a Device to Pass it to eVM
Again, the request to pass the device listed in the right pane. See Figure 4.57.

Save the device configuration, as before, and restart the platform. The change applies to the platform. After saving the configuration and restarting, launch the eVM Device Manager and expand the **Network adapters** device node under *Windows devices* and *eVM devices*. Notice the removal of the network device from Windows control and dedicated to eVM for use by the Guest OS. See Figure 4.58.
4.6.6 **Bus Master Warning Example**

To illustrate the bus master warning that the eVM Device Manager issues when passing an MSI device to a platform that does not support VT-d, turn off VT-d support in the BIOS of the test platform, and check with VtProbe that VT-d support is disabled. See Figure 4.59.
Repeating the exercise in Section 4.6.5, passing the MSI network device to eVM. When selecting **Pass to eVM**, the eVM Device Manager issues a warning for the network device. See Figure 4.60.
If the eVM Device Manager allowed this device to pass to eVM, the condition would require a modified bus master driver to manage the address offset for the virtual machine, or, if possible, disabling the bus master capability in the device.

4.6.7 Passing a Device back to Windows
Pass the network adapter, passed to eVM in Section 4.6.5, back to Windows. Select the Intel® Gigabit CT Desktop Adapter in the eVM Network adapters node in the left pane of the eVM Device Manager, and click on the Pass to Windows button. See Figure 4.61.
Again, eVM Device Manager shows the request in the right pane, except this time it shows the request to pass the device to Windows. See Figure 4.62.
Figure 4.62 - Request to Pass the Network Controller back to Windows

The eVM Device Manager prompts to restart the platform to apply this change. See Figure 4.63.
After restarting the platform, launching the eVM Device Manager, and expanding the device nodes in the left pane, the network adapter is back under Windows control. See Figure 4.64.
4.7 Shared Memory Interface

4.7.1 Description

eVM allocates and manages a single block of shared memory from the Windows non-paged pool to pass data between a Guest OS process and Windows process. Part or all of the shared memory may be mapped to a Windows process while the Guest OS application access it via a virtual PCI device which exposes the memory through a BAR (Base Address Register) in the PCI configuration header for the device.

eVM provides a synchronization mechanism to provide "doorbell" functionality that allows one side to signal the other when memory is changed. An API delivers this signal to a Windows application, and an interrupt from the virtual PCI device delivers this signal to the Guest OS application.

4.7.2 Configuration

Size of the shared memory is setup with the Node Management in the eVM Configuration Panel. Select the Kernel tab and go down to the Shared Memory size entry. Valid sizes include 0, 4, 8, 16, 32, 64, 128, 256, or 512 Mbytes. See Figure 4.65.
The eVM Guests tab of the eVM Node Management window sets the IRQ for Guest OS access. Go to the Shared Memory line and select yes next to the Enable shared memory line. This adds another line listing Memory write interrupt and an interrupt with an allocated number. See Figure 4.66.
4.7.3 Shared Memory Usage

4.7.3.1 Usage from the Windows application side
To access the shared memory buffer the Windows application must make use of the API described in Appendix F. First, the application must determine the configured size of the shared memory by calling `evmGetSharedMemoryInfo`, then mapping all or part of it by calling `evmMapSharedMemory`. Call `evmWaitForSharedMemory` to wait for a signal from the Guest OS application. Call `evmSignalSharedMemory` to send a signal to the Guest OS. Call `evmUnmapSharedMemory` to unmap the shared memory from the application.

4.7.3.2 Usage from the Guest OS application side
The Guest OS application gains access to the configured shared memory area by means of a virtual PCI device. First, the application must discover the device with PCI Vendor ID 0x10b5 and Device ID 0x9050. The interrupt line data found in the IntLine register in the PCI header sets up an interrupt handler, if required for the device. The shared memory is available via BAR4 in the device header, and the control and status registers are available via BAR3. When the Windows application calls `evmSignalSharedMemory`, this device generates an interrupt. When a value is written to the device’s control register, any Windows thread that has called `evmWaitForSharedMemory`, is awakened.
4.7.4 PCI configuration space definition
The virtual PCI device has the following minimal PCI configuration space definition:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size (bits)</th>
<th>Name</th>
<th>Default value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>16</td>
<td>VENDOR_ID</td>
<td>0x10b5</td>
<td>RO</td>
</tr>
<tr>
<td>0x02</td>
<td>16</td>
<td>DEVICE_ID</td>
<td>0x9050</td>
<td>RO</td>
</tr>
<tr>
<td>0x04</td>
<td>16</td>
<td>COMMAND</td>
<td>0</td>
<td>Only Bit 0, 1 and 8 implemented</td>
</tr>
<tr>
<td>0x06</td>
<td>16</td>
<td>STATUS</td>
<td>0x0280</td>
<td>RO</td>
</tr>
<tr>
<td>0x08</td>
<td>8</td>
<td>REVISION_ID</td>
<td>0x01</td>
<td>RO</td>
</tr>
<tr>
<td>0x09</td>
<td>24</td>
<td>CLASS_CODE</td>
<td>0x068000</td>
<td>“Other bridge device”</td>
</tr>
<tr>
<td>0x0e</td>
<td>8</td>
<td>HEADER_TYPE</td>
<td>0x00</td>
<td>Single-function PCI device</td>
</tr>
<tr>
<td>0x1c</td>
<td>32</td>
<td>BAR3</td>
<td>0x000000001</td>
<td>Register space I/O address</td>
</tr>
<tr>
<td>0x20</td>
<td>32</td>
<td>BAR4</td>
<td>0x000000000</td>
<td>Shared memory physical address</td>
</tr>
<tr>
<td>0x2c</td>
<td>16</td>
<td>SUBSYSTEM_VENDOR_ID</td>
<td>0x0000</td>
<td>RO</td>
</tr>
<tr>
<td>0x2e</td>
<td>16</td>
<td>SUBSYSTEM_DEVICE_ID</td>
<td>0x0002</td>
<td>RO</td>
</tr>
<tr>
<td>0x34</td>
<td>8</td>
<td>CAPABILITIES_PTR</td>
<td>0x00</td>
<td>No capabilities</td>
</tr>
<tr>
<td>0x3c</td>
<td>8</td>
<td>INTERRUPT_LINE</td>
<td>0x00</td>
<td>Programmed if the interrupt is enabled by configuration</td>
</tr>
<tr>
<td>0x3d</td>
<td>8</td>
<td>INTERRUPT_PIN</td>
<td>0x01</td>
<td>RO</td>
</tr>
</tbody>
</table>

4.7.5 Register definition
BAR3 describes the following registers defined within the I/O space.

4.7.5.1 Shared memory command register – offset 0 in BAR3
This register is a R/W, 32-bit register which is used to configure and use the device. Write command words to the register to perform specific actions. Valid command values are as follows:
### Value Name Description

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000001</td>
<td>ENABLE_INTR</td>
<td>Enables the device to generate interrupts. A doorbell signal sent to the device from Windows generates an interrupt.</td>
</tr>
<tr>
<td>0x00000002</td>
<td>DISABLE_INTR</td>
<td>Disables interrupt-generation.</td>
</tr>
<tr>
<td>0x00000003</td>
<td>RING_DOORBELL</td>
<td>Sends a doorbell signal to Windows waking any thread waiting in evmWaitForSharedMemory. The state is sticky; if no thread was waiting, the next thread to call evmWaitForSharedMemory returns immediately.</td>
</tr>
<tr>
<td>0x00000004</td>
<td>RESET_DOORBELL</td>
<td>Cancels any outstanding doorbell signals to Windows.</td>
</tr>
</tbody>
</table>

#### 4.7.5.2 Shared memory status register – offset 4 in BAR3

This register is a read-only register. Writing to it has no effect.

<table>
<thead>
<tr>
<th>31-5</th>
<th>4</th>
<th>3-2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(unused)</td>
<td>ASSERTED</td>
<td>(unused)</td>
<td>INTR_PEND</td>
<td>DOORBELL</td>
</tr>
</tbody>
</table>

Sending a doorbell signal to Windows sets the ASSERTED bit until a Windows application receives the signal.

Sending a doorbell signal from Windows sets the DOORBELL bit, and with interrupts enabled, the INTR_PEND bit is set until the next read of the status register.

#### 4.7.6 Typical operation

A typical sequence of operations by the Guest OS software is as follows:

1. Discover the device using normal PCI methods.
2. Enable I/O and memory access by writing 0x0003 to the PCI COMMAND register.
3. Using the value in the INT_LINE PCI register, install an interrupt handler.
4. Map the shared memory using the physical address in BAR4
5. Write ENABLE_INTR to the command register to enable interrupt on the device.

The Guest OS may write to and read from the memory, just like access from the Windows application. Synchronization of access to the memory uses the doorbell mechanism, or with spinlocks created in the shared memory for access from both sides.
A. Production installation

After eVM has been set up to meet the application requirements on the target development system, capture the configuration to replicate that setup in a production environment. Save the configuration from the target development system that is already set up and import that saved configuration to each of the production systems to avoid having to configure each installation of eVM in production. A configuration may contain multiple Guest OSs, but only one as the default. Figure A.1 shows a selection of multiple Guest OSs, each with its own boot image.

![Figure A.1 - List of configured Guest OSs](image)

Figure A.2 shows the selection of the default Guest OS from the list of guest OSs.
A.1 Saving the configuration with the eVM Configuration Panel

From the eVM Configuration Panel, select the two left icons: Node Management and Miscellaneous (hold the Ctrl key to select more than one icon), then click on Export Settings. In the Save As pop-up, specify the file name and location for the INtime Configuration File (.icf).
A.2 Saving the configuration with a command line

The `inconfig.exe` utility loaded as part of the `eVM for Windows` installation accepts command parameters as well as starting the `eVM Configuration Panel`. First, close any instance of the `eVM Configuration Panel`. The utility requires administration rights. To run this utility, open a `Command Prompt` window in administration mode by searching for “Command Prompt”, then right clicking on “Run as administrator”. In the `Command Prompt` window, you can enter the command line parameters described below. In Windows Vista or Windows 7 you can also enter the command line parameters described below in Windows `Start Search`.

**Note: for the “/n” command line switches, ‘n’ is case insensitive.**

Syntax usage:

```
inconfig /e:<export INtime Configuration File name and path>
```

Once eVM is configured on the target development system, the `/e` option of `inconfig.exe` exports the known working configuration to a file. Import the configuration and setup parameters into other systems using this configuration file. The eVM
configuration of the target development system can also be restored to a previously saved known state.

As an exercise, save that configuration in a file named MyeVMConf.icf by first closing the eVM Configuration Panel and then entering the command shown below in the Command Prompt window.

`inconfig /e:c:\MyeVMConf.icf`
This creates a file called `MYeVMConf.icf` in the root directory of the C: drive.

*Note: The eVM Configuration Panel only imports files with an “.icf” extension. The “.icf” extension is the default.*

### A.3 Restoring or importing a known configuration with the eVM Configuration Panel

With the first installation of eVM, there are no settings included in the configuration. From the eVM Configuration Panel, click on **Import Settings** and select the INtime Configuration File to import a known configuration from another system or to restore a saved configuration.

![Warning when Importing an INtime Configuration File](image)

**Figure A.4 - Warning when Importing an INtime Configuration File**
A.4 **Restoring or importing a known configuration with a command line**

With the first installation of eVM, there are no settings included in the configuration. The `/i` option of the `inconfig.exe` command allows the import of a known configuration from another system or the restoration of a saved configuration.

Syntax usage:

```
inconfig /i:<saved INtime Configuration File name and path>
```

To import the configuration from the target development system to a new production system, first close the eVM Configuration Panel.

Second, copy the INtime Configuration File `MyeVMConf.icf` to the root directory of the C: drive of each new production system.

Then import the configuration by entering the command shown below from the Command Prompt window.

```
inconfig /i:c:\MyeVMConf.icf
```

Opening the eVM Configuration Panel on the production system should display the configuration with the settings of the imported configuration. The eVM Configuration Panel should show that any modified system configuration settings have been restored.

A.5 **Assigning devices to eVM on the command line**

**PCI devices**

Pass the PCI device to the Guest OS when the Guest OS needs full control of the PCI device. Use the eVM Device Configuration utility, or the command line:

```
DevConfig[64] /option “device” [/instance n]
```

Where ‘option’ is either:

- `toevm` - pass a device to eVM and let the Guest OS use the interrupt as defined in the current configuration. There is no sharing of interrupts with another device controlled by Windows.

- `toevmnoirq` - pass a device to eVM and let the device use polling or use MSI (Message Signaled Interrupts) with a unique interrupt number assigned.

- `towin` - pass a device back to Windows. This option accepts “all” for “device”.


“device” - must be the full device name as shown in the eVM Device Configuration utility under the eVM tree. A single star may replace the trailing characters (*), if that still leaves a unique device name.

An instance number identifies each instance when more than one device matches the full device name: zero indicates the first instance.

A.6 Command Line Installation of Virtual Serial Ports
Use the eVM Configuration Panel or the command line to install more virtual serial ports when needed for the Guest OS:

\texttt{inconfig /s:<number>}

Where ‘number’ is the number of virtual serial ports to install (from 0 to 4).

The configuration result is the same as opening the eVM Configuration Panel; and in the System Configuration properties, setting Virtual Serial Ports to ‘number’, setting each virtual serial port to ‘next_free’, and then saving the system configuration. Using ‘0’ removes all virtual serial ports.

A.7 Debug logging
When inconfig is running, the bottom pane of the window presents debug information. The session’s logging can be saved to a file, inconfig.log, in the %INCONFIG% directory tree. Please note that the file only saves the current session, and a new session starts with a new blank file.

\texttt{inconfig /l}

Additional information can be logged to the file with the /v, verbose, switch.

\texttt{inconfig /l /v}
B. eVM Virtual Machine

The TenAsys eVM product presents a virtual machine to Guest OS software within a Windows environment that emulates an Intel x86-based PC with certain chipset and other characteristics. In order to run eVM an Intel dual-core platform that supports VT-x is required. Below is the description of the specification for the virtual machine created by eVM.

B.1 System Requirements

The minimum requirement to run eVM is a PC platform running Windows XP, Windows Vista or Windows 7 with an Intel dual-core processor or single-core processor with hyper-threading and VT-x supported by the CPU and enabled in the BIOS.

B.2 Base Virtual Machine Specification

The base platform consists of the following components:

CPU

The virtual machine uses one core of the platform processor. Windows uses the other core or cores. Most of the native characteristics are available but eVM hides some features, such as support for VT-x (there is no support for VT-x features in the Guest OS). eVM intercepts CPUID instructions and modifies some return values to hide features. The registers modified are the four returns by CPUID with EAX=1 and all these bits are configurable internally. By default, the following bits are disabled (always return 0) in ECX: VMX_CAPABLE (bit 5). eVM disables the following by default in EDX: HTT, MTRR, APIC, PAE.

Memory

eVM allocates memory as a single physically contiguous region to the Guest OS, and the VMM takes care of translating the physical addresses generated by the Guest OS software.

The VMM emulates paging using the VTLB algorithm defined by Intel. Systems that do not perform well under the VTLB algorithm are demand-paging systems that frequently reload the page directory pointer (such as Windows or Linux). VT-x2 hardware features are supported improving performance for the Guest OSs.

Hardware generated memory addresses (from a PCI bus master device for example) are automatically translated if the base platform supports and enables VT-d technology; otherwise a change is needed for any software, such as device drivers, which calculate a physical address. VMM provides an interface to the Guest OS software that generates the requested offset for this calculation.

Interrupt controller (8259A PIC)

The standard interrupt controller is the twin 8259-PIC architecture found in the legacy PC platform. VM emulates the interrupt controller. eVM supports all modes except non-8086 mode and Special Fully Nested Mode.
The default interrupt map for the VM is as follows:

<table>
<thead>
<tr>
<th>IRQ</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>System timer interrupt (from 8254)</td>
</tr>
<tr>
<td>1</td>
<td>(reserved)</td>
</tr>
<tr>
<td>3</td>
<td>Serial port 1 (COM2)</td>
</tr>
<tr>
<td>4</td>
<td>Serial port 0 (COM1)</td>
</tr>
<tr>
<td>5</td>
<td>PCI IRQ or</td>
</tr>
<tr>
<td>6</td>
<td>Floppy disk controller, or PCI IRQ</td>
</tr>
<tr>
<td>7</td>
<td>Spurious interrupt, or PCI IRQ</td>
</tr>
<tr>
<td>8</td>
<td>CMOS/RTC</td>
</tr>
<tr>
<td>9</td>
<td>PCI IRQ</td>
</tr>
<tr>
<td>10</td>
<td>PCI IRQ</td>
</tr>
<tr>
<td>11</td>
<td>PCI IRQ</td>
</tr>
<tr>
<td>12</td>
<td>NE2000</td>
</tr>
<tr>
<td>13</td>
<td>Legacy FPU error</td>
</tr>
<tr>
<td>14</td>
<td>Primary IDE controller</td>
</tr>
<tr>
<td>15</td>
<td>Secondary IDE controller</td>
</tr>
</tbody>
</table>

Table B.1 - Virtual IRQ's

The VM makes use of APIC virtualization if it is available in the CPU and the Guest OS supports APIC, which results in better interrupt performance.

**System Timer (8254 PIT)**

The system timer is the standard 8254 Programmable Interval Timer found in the legacy PC platform. The VM emulates the system timer. eVM does not currently support modes 1, 4 and 5, nor any BCD counting mode or 8-bit counting modes. Port 61 bits 1 and 4 are functional (speaker data and refresh timer bits).

**CMOS & Real Time Clock**

The VM fully supports CMOS access ports 70 and 71, and builds a CMOS RAM image at Guest OS boot time from the eVM configuration data. The real-time clock is accessible through the CMOS ports in the normal way, and it generates IRQ 8 as expected.

**Port 92**

Port 92 bits 0 ("high-speed reset") and 1 ("A20 control") are supported. eVM does not currently support other port 92 bits.

**Serial ports (16550 UART)**

eVM supports an emulation of the standard 16550 UART and up to four configurable serial ports. eVM supports all of the 16550 UART modes. Accesses to the virtual device may either pass through to a real UART or else the virtual UART transmits and
receives data to and from a Windows virtual serial driver, establishing a virtual serial link between Windows and the Guest OS software.

**DMA controller (8237 DMA controller)**

The legacy 8237 DMA controller pair is virtualized by eVM. Currently the only device supported by the virtual controller is the virtual floppy disk controller.

**Floppy Disk Controller**

This device simulates a standard floppy disk controller. Two devices are supported (drives A: and B:) and only 1.44 Mbyte diskettes are currently virtualized. The media itself is a disk image of floppy disk medium, served by a Windows process to the VMM.

**IDE controller**

The VMM virtualizes the standard IDE controller and presents two IDE controllers (primary and secondary) to the VM. Each virtual disk is actually a device image presented by the VMM from a Windows process. eVM supports virtual hard disk drives and CDROM devices. The device image takes the form of a binary file.

**PCI Host Bridge (440FX Host Bridge, PCI device 0)**

The VMM presents a virtualized 440FX host bridge (vendor ID 0x8086, device ID 0x1237), supporting a virtualized PCI local bus. The virtual host bridge allows the insertion of other virtual and real PCI devices into the virtual PCI local bus to intercept PCI configuration space accesses.

**ISA Bridge (440FX ISA Bridge, PCI ID 8086/7000)**

This virtual device primarily routes PCI interrupts to the interrupt controller, and emulates the APM registers (mostly non-functional since the VM does not emulate power management).

The VM provides four PCI IRQ links (PIRQA through PIRQD) routed via this device to inputs on the interrupt controller. The routing initializes according to the VM configuration at startup, and dynamically controlled through the IRQ routing registers in this device.

**EIDE Controller (PCI ID 8086/7010)**

This device emulates the EIDE controller, providing EIDE functionality for the disk controller. The VMM software provides the DMA functionality.

**ACPI Controller (PIIX4 ACPI Controller, PCI ID 8086/7113)**

This device emulates the ACPI hardware controller for the chipset. The VM does not emulate power management features but enables all other ACPI hardware features.

**B.3 Optional Virtual Machine Components**

**Ethernet controller**

An NE2000 virtual controller provides a shared-memory and non-interrupt driven Ethernet link to a Windows Ethernet driver.
Shared memory device

eVM provides a shared memory device for the development of a simple interface to a memory area shared between Windows and the Guest OS. The interface provides a simple signaling mechanism that generates interrupts from the virtual device. An API provided in eVM handles the Windows side of the interface.

B.4 BIOS

The VM BIOS provides the standard functionality of a PC BIOS, including all of the functions required to boot a Guest OS. The default action of the eVM is to execute the BIOS when first started, which then attempts to boot the Guest OS via the virtual IDE or floppy interface from a disk or CDROM image.
C. Enabled Guest OSs

For an up-to-date listing of enabled Guest OSs, check the following website: www.tenasys.com, and click on Support->TenAsys Knowledge Base.
D. VtProbe

VtProbe is one of the tools contained in the eVM installation. During installation VtProbe determines if the hardware platform has the required virtualization features to support eVM. Run VtProbe manually to check a hardware platform for eVM compatibility.

To run VtProbe, do a Start->Run and enter vtprobe or enter vtprobe in Search programs and files. If the platform does not support eVM, the tool shows no support for VT-x. In Figure D.1, the hardware platform supports neither VT-x nor VT-d.

![Figure D.1 - VtProbe, Hardware Platform Doesn't Support eVM](image)
In Figure D.2, the hardware platform supports both VT-x and VT-d.
E. Modifying device drivers for bus-master devices

A bus master is any device that generates bus cycles to transfer data. The bus master device programs with the physical memory addresses to read or write data. When a device driver running in the Guest OS programs a bus master device, either the platform must translate the physical addresses to direct the read/write operation to the correct physical memory address or a modified device driver must make this translation explicitly. This appendix describes how to make this modification.

E.1 Discovering the Guest OS physical base address

The Guest OS physical base address is the offset at which the Guest OS is located on the host. Any bus master device must add this offset to the programmed physical addresses. The VMCALL instruction in an interface to the VMM discovers the offset. The VMCALL instruction causes an exit to the VMM that then processes the request based on the contents of the CPU registers. To find the Guest OS physical address, execute the VMCALL instruction with EAX set to the value 0x80020001. If the call succeeds, clearing the carry flag, the EBX register returns the physical base address to the Guest OS. Register ECX returns the size of the memory allocated to the Guest OS. Intel documentation describes the VMCALL instruction in detail.

E.2 Using the Guest OS physical base offset value

The obtained base address offsets any physical address generated by the device. Modification is required of the device driver to include the offset in the physical address used by the driver.

Pseudocode of a function to implement the addition of the offset might look like this (Assume OsGetPhysicalAddress is the operating system’s regular call):

```c
Paddr_t LibGetRealPhysicalAddress(PVOID *logical_address)
{
    Paddr_t real_address = OsGetPhysicalAddress(logical_address);
    Uint32 sys_offset ;
    Uint32 regs[4];

    regs[0] = GETMEMPARAMS;
    if (do_vmcall(regs) == 0)
        sys_offset = regs[1];
    else
        sys_offset = 0;

    return real_address + sys_offset;
}
```
F. eVM API calls

The eVM API is a Windows library that allows the user to control or configure certain features from within a Windows application. Included in the API is also the means of accessing the shared memory area between Windows and the eVM Guest OS software.

F.1 Include the API in your application

To use the API, add the following settings to your Visual Studio C/C++ project:
- Compiler settings: Additional include directories: "$(EVM)\nt\include"
- Linker settings: General: Additional library directories: "$(EVM)\nt\lib"
- Linker settings: Input: Additional dependencies: evmapi.dll (or evmapi64.dll for 64-bit applications)

In the source code, add \#include <evmapi.h>

F.2 Types

EVMSTATUS

The eVM API status type.

EVMLOCATION

The eVM API location type. Identifies an eVM node.

F.3 API Calls

VMM Management calls

These calls are implemented using FindFirstFile, FindNextFile with filter *.cfg in the %EVMCFG% directory.

\textit{evmGetFirstVmm}

EMVLOCATION evmGetFirstVmm(void);

Returns a location value for the first VMM instance.

\textit{evmGetNextVmm}

EMVLOCATION evmGetNextVmm(EMVLOCATION vmmLoc);

Returns a location value for the next VMM instance found after the one specified by the parameter.

Command and Control calls

The following APIs are available for control of the eVM VMM, and the configured Guest OSs.

\textit{evmStartVirtualMachine}
EVMSTATUS evmStartVirtualMachine(EMVLOCATION vmmLoc);

Starts the VMM on a given node, where vmmLoc is the location of the node (returned from evmGetFirstVmm/evmGetNextVmm). This call loads and starts the VMM but does not start a Guest OS, unless configured for a default Guest OS.

**evmStopVirtualMachine**

EVMSTATUS evmStopVirtualMachine(EVMLOCATION vmmLoc);

Stops the VMM on a given node, where vmmLoc is the location of the node. This call first stops a Guest OS running on the VMM then stops the VMM.

**evmStartGuest**

EVMSTATUS evmStartGuest(EMVLOCATION vmmLoc, LPTSTR sGuestName);

Starts a Guest OS on the VMM indicated by the vmmLoc parameter, and sGuestName configuration name. The name corresponds to a configuration file name generated by the configuration utility. If sGuestName is NULL, then it starts the currently loaded, but not running, Guest OS.

**evmStopGuest**

EVMSTATUS evmStopGuest(EVMLOCATION vmmLoc);

Stops the Guest OS on the VMM indicated by the vmmLoc parameter.

**evmRestartGuest**

EVMSTATUS evmRestartGuest(EMVLOCATION vmmLoc);

Restarts a guest on the VMM indicated by the vmmLoc parameter. Starts the currently loaded configuration if a guest is not running.

**evmGetVMMStatus**

EVMSTATUS evmGetVMMStatus(EMVLOCATION vmmLoc, LPTSTR Buffer, DWORD BufferLength);

Returns the running state of the Guest OS indicated by the vmmLoc parameter. If a Guest is running, it returns 0 and also the name of the running configuration, if any. If no guest is running, returns -1; if an error was encountered, returns the error code.

**Configuration Management calls**

**evmGetFirstGuest**

EVMHANDLE evmGetFirstGuest(EMVLOCATION evmLoc, LPTSTR Buffer, DWORD BufferLength);
Returns a handle and the name of the first Guest OS configuration found.

**evmGetNextGuest**

```c
BOOL evmGetNextGuest(EVMHANDLE LastGuest, LPTSTR Buffer, DWORD BufferLength);
```

Returns the name of the next Guest OS configuration after the last one found.

**evmGetDefaultGuest**

```c
BOOL evmGetDefaultGuest(EMVLOCATION evmLoc, LPTSTR Buffer, DWORD BufferLength);
```

Returns the name of the default Guest OS configuration. Buffer length must be set before the call to the size of the available buffer. If the Guest OS name returned is too long for the buffer then the call fails and a suitable status returned.

**evmSetDefaultGuest**

```c
EVMHANDLE evmSetDefaultGuest(EMVLOCATION evmLoc, LPTSTR Buffer, DWORD BufferLength);
```

Sets the default Guest OS name. If Buffer is NULL then removes the default Guest OS name, so there is no default Guest OS. This overwrites any existing default Guest OS.

**Shared Memory calls**

Defines a single shared-memory buffer per node as a communications buffer between a Windows application and the Guest OS. The following calls manipulate access for the shared memory from Windows, using the standard NTX memory-mapping calls to map a segment catalogued in the VMM process.

**evmGetSharedMemoryInfo**

```c
EVMSTATUS evmGetSharedMemoryInfo(EVMLOCATION evmLoc, PEVMSHAREDMEMORYINFO evmInfo);
```

Returns basic information about the shared memory buffer. Includes at least the size of the buffer.

**evmMapSharedMemory**

```c
PVOID evmMapSharedMemory(EMVEVMLOCATION evmLoc);
```

This call maps all of the configured shared memory to the calling process.

**evmUnmapSharedMemory**

```c
EVMSTATUS evmUnmapSharedMemory(EVMLOCATION evmLoc, PVOID pMap);
```
Unmaps shared memory from an application previously mapped.

**evmSignalSharedMemory**

EVMSTATUS evmSignalSharedMemory(EVMLOCATION evmLoc);

Signal an internal semaphore associated with the shared memory area. Resulting in a signal sent to the Guest OS software on the node indicated by `evmLoc`.

**evmWaitForSharedMemory**

EVMSTATUS evmWaitForSharedMemory(EVMLOCATION evmLoc, DWORD dwMilliseconds);

Wait for a signal from the semaphore associated with the shared memory buffer, or time out after `dwMilliseconds`. 