

# IntervalZero

## RTX 2009 with Service Pack 2

### Frequently Asked Questions about Real-Time

#### What is Real-Time?

Real-time describes an application which requires a response to an event within some small upper bounded time frame. Typically, response times are in the millisecond or microsecond time frame.

#### What is the difference between “Hard” and “Soft” Real-Time?

Hard real-time requires that a response be logically correct and occur before a certain deadline or the result is incorrect, and holds no value.

Soft real-time requires that a response be logically correct and occur before a certain deadline or the result becomes increasingly inaccurate, meaning the result can still hold some value even though it occurred after the required deadline.

#### What does determinism mean in a real-time environment?

Determinism is defined as the ability to rationally predict, with a degree of precision, when an event will happen. Determinism, combined with a real-time environment, guarantees that an event will happen within a small response time and that the performance of this event is repeatable.

#### What is a real-time operating system, or RTOS?

A real-time operating system provides determinism and predictability when it responds to a given event through a specialized scheduler. IntervalZero ETST™ (Embedded Tool Suite) is an example of a real-time operating system.

#### Is Microsoft® Windows® a real-time operating system?

Windows is usually referred to as a general-purpose operating system because it does not allow applications or kernel-level drivers to completely mask interrupts and gain control over the operating system. Depending on hardware, interrupt latencies under Windows can exhibit very good values, averaging in the microseconds. However, worst-case interrupt latencies are unbounded and can exceed hundreds of milliseconds. Because of these unbound latencies,

deterministic response time is not guaranteed, making standard Windows Desktop and Server operating systems unacceptable for real-time use.

## **What is RTX?**

Even though there are a lot of similarities between an RTOS and RTX, RTX is not an RTOS. RTX is a real-time extension for Windows. RTX enhances Windows by providing hard real-time and control capabilities to a general purpose operating system that is familiar to both developers and end users. RTX consists of an extension to the Windows HAL and a separate real-time subsystem (RTSS) that schedules and controls all RTSS applications independently of Windows.

## **What is SMP?**

RTX supports Symmetrical Multiprocessor Systems (SMP). With this model, multiple processors can be configured for real-time activities. RTSS threads can be assigned to run on specific processors and they can run concurrently.

When running RTX on an SMP-enabled system, you configure how many of the processors are dedicated to Windows and how many are dedicated to the RTSS. RTX supports SMP systems that have as many as eight processors and of those eight, as many as seven can be dedicated to RTX.

## **Frequently Asked Questions about RTX**

### **How does RTX extend Windows to provide “hard” real time?**

The overall design of RTX affords developers the “best of both worlds” by providing the ability to use all the features, and technologies that Windows offers in addition to “hard” real-time behavior within an isolated and controlled subsystem. The RTX Runtime extends the Windows HAL and provides a real-time subsystem that has its own scheduler, which allows all RTSS applications to be prioritized ahead of all Windows applications or Windows operating system functionality. RTX controls system resources and is guaranteed to execute ahead of all Windows threads, Deferred Procedure Calls (DPCs), and interrupts. This means that RTX allows Windows to run only when all real-time processing is finished.

### **What are the benefits of RTX?**

The RTX Runtime enables general-purpose Windows processing and high performance real-time processing and control on commercial-off-the-shelf (COTS) machines. The RTX Runtime can be configured to take part in Windows minidumps or to take control and safely shutdown real-time processes if Windows encounters a failure.

The RTX SDK provides developers with a rich set of inter-process communication and synchronization capabilities, allowing RTSS applications to communicate with Windows applications and share data with them. Additionally, RTX provides developers with the ability to directly access I/O port address space, physical memory, or hardware without forcing any driver model on the user.

## What are the benefits of using RTX on an SMP system?

Using RTX on an SMP system provides significant benefits, including:

- **Performance Boost** - You can have multiple processors dedicated to critical, real-time tasks. You can concurrently run up to seven real-time threads on an eight-processor system.
- **Performance Scalability** - Performance scaling doesn't require code rewrites. You can adjust the real-time and non real-time performance balance by changing the number of RTSS processors and Windows processors.
- **High Availability** - Critical tasks can be scheduled to run on more than one RTSS processor.
- **IRQ Affinity** - You can specify a dedicated RTSS processor for processing the input and output of individual pieces of hardware.
- **System Fault Handling** - Real-time tasks survive over system crashes.

## Can the same application run on both SMP and non-SMP runtimes?

While SMP systems offer benefits to real-time applications, the RTX runtime is available in both SMP and non-SMP versions. The RTX API and executables are compatible across the platforms and between RTX SMP and non-SMP versions. This allows developers the freedom to develop application that can be scalable.

## How long has RTX been around?

The RTX product was released in 1995 to provide a real-time subsystem for Windows NT. RTX has continued to evolve, providing real-time support for all Microsoft professional operating systems. RTX 2009 with Service Pack 2 and RTX 2009 SMP with Service Pack 2, released in 2010, support both uni- and multi-processors running the following operating systems: Windows 7, Windows Vista, Windows XP Professional and Windows Server 2003.

## What is the current version of RTX?

The latest version of RTX is version RTX 2009 with Service Pack 2, released in June 2010.

## What types of industries or products typically use RTX?

RTX is used in a wide variety of products and vertical markets. Anyone designing an application that requires system control, determinism, or real-time performance on Windows can benefit from incorporating RTX into their product design.

RTX is used in some of the following markets:

- Industrial Automation
- Digital Audio

- Test & Measurement
- Medical
- Military Aerospace

## How is RTX packaged?

RTX offers SMP and non-SMP Runtime versions. Each Runtime version is packaged as a stand-alone Runtime or with a Software Development Kit (SDK).

The RTX Runtime comes in two varieties:

- An installation for full Microsoft Windows operating systems, such as Windows 7, Vista or Windows Server 2003 (licensed per machine or through an OEM/Site license.)
- A silent Runtime install that can be wrapped within your product
- A Source Level Definition (SLD) installation that supports the Windows XP Embedded, Windows Embedded Standard 2009 operating system through Microsoft Target Designer. The embedded runtime is licensed through a site license only.

The RTX SDK is licensed per machine and includes:

- The RTX Runtime subsystem (SMP or Non SMP), used to execute real-time applications and provide networking support to the RTX environment.
- A set of tools and utilities for diagnosing issues and monitoring the performance of your RTX environment.
- Headers and libraries needed to build RTSS applications

## What comes with the RTX Runtime?

The RTX Runtime comes with the following components:

- **RTX HAL Extension** - Extends the Windows HAL to support real-time control
- **RTX - RTSS Scheduler** - Schedules all RTSS threads ahead of Windows threads
- **RTX Win32 Support** - Allows Windows applications to communicate with RTSS applications
- **RTX Kernel Support** - Allows Windows kernel drivers to communicate with RTSS applications
- **RT-TCP/IP Stack** - TCP/IP stack for the RTSS
- **RTX Properties** - A control panel that configures the RTSS

- **RTX Server** - Displays RTSS application output
- **Command-line Utilities** – Control RTSS applications
- **Documentation** - RTX Subsystem Guide and RTX Utilities Guide

## What comes with the RTX SDK?

The RTX SDK comes with the RTX Runtime environment plus the following components:

- Header files and libraries
- Visual Studio Support
  - Supported Versions:
    - Visual Studio 2010
    - Visual Studio 2008
    - Visual Studio 2005
    - Visual Studio .NET 2003
  - Microsoft C Runtime support
  - Wizards
  - Debugger add-ins
- Debugger Data Extension for Microsoft WinDbg
- Development tools such as RTSS Object Viewer and TimeView
- Measurement tools such as PerformanceView and Platform Evaluator
- Comprehensive Help files and User Guides
- Source code sample to help explain more advance development topic

## Frequently Asked Questions about Platforms

### Are there any hardware or platform requirements for RTX?

The RTX Runtime runs on any Commercial Off The Shelf (COTS) platform that Windows supports. RTX supports uniprocessor, mobile processor, multiprocessor, hyper-threading-enabled, and multi-core platforms. However, because all systems are not the same, developers need to evaluate the latencies of any platform that they choose to ensure that the platform can support their real-time or control needs.

## How does RTX support multiprocessor systems?

RTX supports multiprocessor systems in either of the following configurations:

- Shared Mode, where RTX shares one processor with Windows.
- Dedicated Mode, where RTX takes exclusive control of one to seven processors. No Windows processing occurs on any of the dedicated RTX processors. The system must have an SMP Runtime version to support more than one dedicated RTX processor.

## Can RTX be used on a mobile processor system?

RTX can be used on mobile processor systems. However, since mobile processors use Intel's SpeedStep® technology to lower processor speed during Windows idle time to conserve energy, long latencies can occur when the processor becomes unavailable during the speed change. Through subsystem configuration, RTX can eliminate these possible latencies by not allowing the Windows idle thread to run, removing the possibility of the processor becoming unavailable because of a speed change.

## Does RTX support clustering?

RTX does not support cluster destination model enabled processors. This means that RTX will not run on systems with more than eight processors, or on systems with less than eight processors that force clustering. Clustering can be forced through the Advanced Configuration and Power Interface BIOS, by modifying the `FORCE_ APIC_CLUSTER_MODEL` bit in the Fixed ACPI Description Table (FADT) . To see if an APIC processor has forced clustering, search for the registry key `HKEY_LOCAL_MACHINE \HARDWARE\ACPI\FADT`. Under the FADT key there will be machine-specific keys that will contain a binary registry value of 00000000. Double-click on the binary value and scroll down to hex byte 0x70. Bit 18 (first bit in the field is bit 0) represents the flag `FORCE_ APIC_CLUSTER_MODEL`. A set bit (value of 1) indicates that all local APICs must be configured to use the cluster destination model when delivering interrupts in logical mode, and therefore the processor will not be supported by RTX. For more information on how the ACPI is configured, see <http://www.acpi.info/spec.htm>.

## Does RTX support hyper-threading?

RTX can be used on hyper-threading systems. RTX treats the logical processor created by Intel Hyper-Threading as a separate processor, so you can configure RTX for either shared or dedicated multiprocessor support. Because both logical processors share the same physical processor, one logical processor can adversely affect the performance of the other. It is recommended that you evaluate RTX performance to ensure that when hyper-threading is enabled, your real-time requirements are still achieved.

## Does RTX support Physical Address Extension (PAE)?

RTX supports PAE for all versions of Windows that support PAE. Prior to RTX 2009 SP1 PAE was only supported on shared configurations.

## **Does RTX support Data Execution Prevention (DEP)?**

Yes. Data Execution Prevention (DEP) is supported.

## **Does RTX have any tools to help determine which platform will work best for my real-time requirements?**

The RTX SDK includes a tool called Platform Evaluator that allows you to run a number of latency-measuring tests to determine how well platforms meet your real-time and control needs.

## **Frequently Asked Questions about Deployment**

### **What is needed for application deployment?**

To deploy your RTSS application, you must purchase an RTX SMP or non-SMP Runtime license for each system on which the application will run.

### **Can I run the RTX Runtime installation from within another installation?**

IntervalZero provides the option of a silent command line installation for the RTX Runtime, which allows an OEM to wrap the RTX installation and hide it within another installation. The silent installation requires an OEM license.

### **How do I configure my customer's real-time subsystem?**

IntervalZero provides a configuration interface that can be used to programmatically configure the RTSS. This allows customers to set up their software's subsystem requirements without requiring anything of their users.

### **How can I help debug my customer's issues?**

RTX can be configured to add active RTSS process information to the Windows minidump file, which can then be analyzed with Microsoft WinDbg. The RTX Debugger Data Extension for WinDbg helps you discover the state of the RTSS and all active RTSS processes at the point of crash, either with live kernel debugging or from a full kernel dump of a customer's system.

For developers who prefer to work in Visual Studio, RTX provides remote debugging capabilities for RTSS applications with Visual Studio 2010, Visual Studio 2008, Visual Studio 2005 and Visual Studio .NET 2003.

RTX also provides excellent flexibility for processing exceptions. You can configure RTX to handle exceptions with a structured exception handler; generate a debug break; or stop the process and dump memory.

A tracing API allows developers to create custom trace events that can be included in subsystem traces to help pinpoint problem areas.

# Frequently Asked Questions about Development

## How does RTX reduce development time?

Because RTX extends Windows, there is no need to spend time designing and developing an operating system before application development work even begins. RTX developers can create user interfaces and applications that take advantage of all the functionality that Windows offers; developers need to focus only on the real-time control pieces required to run an RTSS application. Even components that require real-time control can first be developed as a Windows application and then recompiled as an RTSS application with no code changes.

Since all real-time API (RTAPI) calls are Win32 compliant, developers use calls they already know and understand. There is no need to write driver code or follow a strict driver model to configure and use devices.

## Is RTX 2009 compatible with previously built RTSS applications?

RTX 2009 is source code compatible with previous versions of RTX. This means that any application built with an earlier version of RTX can be rebuilt with a later version of RTX without requiring any code changes. Real-time application executable code may also be run\_able on a previous version of RTX without rebuilding, but this is not guaranteed.

## Do I need special development and debugging tools to develop RTSS applications?

No. RTSS applications are developed with Microsoft Visual Studio. The RTX SDK provides wizards for easy project creation and templates to help you to get started. A Visual Studio Debugger add-in lets you debug RTSS applications in a familiar environment. Because RTSS applications run in kernel mode, they can also be debugged with a kernel-level debugger such as Microsoft WinDbg. The RTX SDK includes an RTX Debugger Data Extension for WinDbg that allows you to view active RTSS processes and objects.

## Can I use Win32 API calls or are all RTX calls proprietary?

RTX supports a subset of over 50 Win32 API calls that make sense in a real-time environment.

In addition, RTX provides a large selection of real-time API calls that developers use to access the RTSS and system resources. This real-time API (RTAPI) is composed of a set of unique API calls and Win32-based API calls.

The unique real-time API calls are Win32 modeled calls that provide essential programming capabilities required for real-time applications, along with access to the RTSS and system resources. These unique RTAPI calls have no equivalent Win32 calls.

The Win32-based API calls that are supported by RTX are unlike the RTAPI in that there are similar functions in the Win32 environment; however, these calls require a different implementation than their Win32 counterparts to support the real-time requirements of the RTSS. All the Win32-based calls are compatible with Win32 programming interface semantics, making it easy for developers already familiar with the Windows Win32 API.

## **How can I take advantage of user-mode memory protection during development?**

RTX was designed so that developers could design and develop applications as RTSS or Win32 applications. If built as Win32, developers can take advantage of features like user-mode memory protection and other third-party debugging tools specific to user mode applications. After an application is working as desired, recompile it as an RTSS application which runs in kernel mode with no code changes required.

## **Does RTX support Structured Exception Handling?**

Unlike other applications that run in kernel mode, RTSS applications support structured exception handling. RTX lets developers call structured exception handling functions such as try, catch, and throw within their RTSS application. For example, if an application references a NULL pointer, the application can handle the error by terminating or freezing the application that caused the fault without bringing down the system.

## **Can I use the Microsoft development libraries and technologies such as C Runtime in my RTX applications?**

RTX supports a subset of Microsoft C Runtime calls that you can call from your RTSS application. RTX provides C Runtime Support for Visual Studio 2010, Visual Studio 2008, Visual Studio 2005 and Visual Studio .NET 2003.

## **Do I need to use a kernel debugger to debug my RTSS application ?**

A kernel debugger is not necessary to debug an RTSS application. The RTX SDK includes Visual Studio debugger add-ins that let developers debug real-time or control applications that run in kernel mode from within the familiar Visual Studio development environment. The debugger add-ins for Visual Studio 2010, Visual Studio 2008, Visual Studio 2005 and Visual Studio .NET 2003 also support remote debugging so developers can remotely debug an RTSS application on a target system that has special hardware requirements.

However, if you prefer using Microsoft's WinDbg, the RTX SDK provides symbols to help with the debugging of your RTSS application, along with a Debugger Data Extension which allows users to view information about active RTSS processes, threads, and objects. Note that WinDbg cannot be used on dedicated configurations.

On systems that do not provide a COM port (RTX versions 8.1 and after) kernel debugging can be accomplished with:

- Null-modem cables
- USB 2.0 cables
- IEEE 1394 "FireWire" cables

## Are any tracing tools available?

The RTX SDK provides a tool called TimeView that lets you set up a system trace. These system traces time stamp and log a configurable set of system and process events, allowing developers to trace the behavior of their real-time application with minimal impact on their system's real-time performance. The RTAPI also provides instrument tracing functionality within RTSS application.

## Does RTX provide any sample code?

The RTX SDK includes a complete API reference guide for all supported functions in addition to small code segments that explain more complex concepts.

The RTX SDK also provides source code for a number of sample applications, some of which are the RTX measurement tools. These samples can be compiled and run, and show important concepts and application interaction.

## Are any measurement tools available?

RTX provides a number of tools and APIs that help developers measure system response and timer latency:

- **SRTM** (System Response Timer Measurement) – A command-line application that measures timer latencies and displays results in reports and histograms.
- **KSRTM** (Kernel System Response Timer Measurement) – A driver and a Win32 utility that measures HAL-level timer latencies and displays results in reports and histograms.
- **RTX Demo** – A graphic version of SRTM that displays timer latencies.
- **PerformanceView** – A graphical utility that shows CPU utilization by real-time applications, Windows processes, and system idle time. PerformanceView also displays the maximum amount of time RTX has held the CPU on shared systems.
- **ObjectViewer** – A graphical utility that displays information for all active objects in the RTSS environment, including thread creation date/time and duration.
- Several **APIs** for profiling across processors, including the following:
  - RtGetThreadTimes retrieves the execution time of a given thread.
  - QueryPerformanceCounter and QueryPerformanceFrequency provide accurate time tracing between multiple processors.

## **Frequently Asked Questions about how RTX interacts with Windows**

### **Are there limits on the number of threads or objects that can be created in RTX?**

Thread and object creation involves the allocation of several small RTSS structures in addition to the initial space allocated for the thread stack. There are no subsystem limitations on the number of threads. The only limit is the amount of available non-paged memory.

### **Can my real-time application communicate with a “regular” Windows application?**

RTX lets Win32 and RTSS applications communicate through a number of inter-process communication (IPC) objects. Use the RTAPIs to create objects that can be viewed and used by Windows processes. Similar to Windows inter-process communication, RTSS and Windows applications create or open handles to named objects or memory regions, allowing simple and standard communication and synchronization between real-time (RTSS) and non-real-time (Windows) applications. Shared memory regions allow Windows and RTSS applications to view the same physical memory without passing additional messages or data between environments.

### **Can a Windows driver communicate with my real-time application?**

RTX provides a set of real-time Kernel API calls (RTKAPI) that allow Windows drivers to access RTX inter-process communication objects from within a Windows kernel device driver. These RTKAPI calls are analogous to their RTAPI counterparts. For example, RtkOpenSemaphore is analogous to RtOpenSemaphore.

The RTKAPI functions and the RTAPI functions are used in the same way, but RTKAPI functions are used exclusively in the Windows kernel environment.

### **How does RTX support Plug and Play devices?**

RTX acquires the resources the device needs from the Windows Plug and Play Manager. To make this possible, a device’s driver must be updated to point to the RTX plug and play stub driver. After the device is controlled by RTX, the device’s resources must be updated to request a unique interrupt that is not already being used by Windows. (Sharing interrupts with Windows would cause determinism to be lost.) Once the device is set up, any RTSS application can access and use the device.

### **Does RTX support Message-based Interrupt (MSI/MSI-X) devices?**

RTX 8.1 and later support MSI and MSI-X capable devices, providing an alternative to line-based interrupts. This functionality is available on all RTX supported operating systems.

## **How do the RTX thread-based priorities relate to the Windows thread priorities?**

Windows has a set of 32 priority levels, ranging from 0 – 31. They are further defined into 4 priority classes, of which the “real-time” priority class is the highest priority class.

The "real-time" priority class, in turn, has 7 levels within the class. RTX uses a flat priority scheme of 127 priority levels. When there are any RTX tasks or RTX threads that are ready to run, RTX will gain total control of the system resources, regardless of what priority level a Windows thread may be granted.

All RTX priorities are higher than the highest Windows priorities.

The one case where the Windows priority scheme is relative to the RTX scheme is when an RTX application is compiled as a Win32 application and not as a RTSS application. In this case, RTX priority levels are mapped to Windows priority levels. These mappings are fixed and designed to preserve relative ordering among thread priorities.

## **Does RTX Support Priority Promotion?**

RTX provides the option to set one of three priority inversion protocols:

- Priority Promotion with Tiered Demotion elevates a low priority thread to the level of the highest priority thread that is waiting for the shared mutex until it has released the mutex requested by a higher priority thread.
- Priority Promotion with Limited Demotion - Elevates a low priority thread to the level of the highest priority thread that is waiting for the shared mutex until it has released all mutexes that it owns
- Disable - Does not elevate priorities in cases where a higher priority thread is waiting on a mutex held by a lower priority thread.

## **How does RTX ensure that Windows does not mask off real-time interrupts?**

RTX includes a real-time enabled Hardware Abstraction Layer (HAL) extension; this extension does not replace the existing Windows HAL. The extension maintains interrupt isolation between RTSS and Windows. Windows cannot mask (at the interrupt controller level) interrupts managed by RTSS. Windows interrupts are masked during RTSS processing. The real-time HAL extension supports high-resolution clocks and timers for RTSS, while it also supports non real-time clocks and timers for Windows. Other real-time HAL extension features include a software interrupt mechanism between RTSS and Windows, basic exception management, and enhancements for determinism. The HAL timer values can be changed via a predefined value table to as little as 1µs, or can be assigned a custom value using the SDK.

## **What about the Windows “Stop Conditions”?**

Windows STOPS, or Bug Checks, are the result of kernel level drivers or operating system components failing safety checks, bringing Windows to a controlled stop. These Windows STOPS are designed to keep data corruption to a minimum and help developers find out what went wrong.

Since the RTSS is able to continue running after Windows issues a STOP, developers can build safe shutdown handling into RTSS applications. RTX calls the shutdown handlers when Windows issues a STOP, allowing system real-time components to safely shut down. Once all shutdown handlers finish running, RTX lets the Windows shutdown process continue.

If the RTSS determines that Windows needs to be shut down, RTX issues a STOP, but instead of displaying the traditional Windows blue screen, RTX displays the RTX green screen, which contains information about the state of RTX at the time of the STOP.

## **Can I assign multiple IP addresses to a single NIC?**

RTX 8.1 and later support "virtual" IPv4 addresses. You can assign up to 32 virtual IP addresses to a single NIC.