Porting Applications from RTX to RTX64

USER GUIDE

IntervalZero

RTX RTX64

Porting Applications from RTX to RTX64 IZ-DOC-X64-0038-R10

IntervalZero

400 Fifth Avenue
Fourth Floor
Waltham, MA 02451
Phone: 781-996-4481

www.intervalzero.com

Contents

About this Porting Guide

This document describes the steps required to port an existing RTX application to RTX64. The objective of this document is to not only walk you through the steps, but to highlight system differences between the two products and point out commonly-encountered issues that might arise during the migration.

This document contains active links to the IntervalZero website and product documentation. Therefore, an Internet connection and an installation of RTX64 are required in order to fully take advantage of this guide.

1

Product Comparison

The RTX Subsystem is built as a set of 32-bit binaries. As a result, it cannot be used with a 64-bit Windows operating system. The RTX64 Subsystem has been architected and rebuilt as a set of 64-bit binaries, which allows for the RTX64 Subsystem to run on a 64-bit Windows operating system; taking full advantage of 64-bit optimizations and features. Because of this, functionality that existed in RTX may not currently exist in RTX64, or it may have been re-architected. Some shared functionality may behave differently in RTX and RTX64.

You can view a comparison document online at http://www.intervalzero.com/technical-support/guides-and-minitutorials/ that provides a comparison of RTX and RTX64. It should help you to determine whether the functionality you use in your RTX application(s) is available in RTX64.

Available Product Packages

RTX and RTX64 are available in different product packages/installs:

RTX	RTX64
• Runtime and SDK (combined install; the Runtime feature can be installed silently) • Merge Modules	• Runtime (can be installed silently; includes the RT-TCP/IP Stack which must be purchased separately) • SDK (can be installed silently) • Runtime Merge Modules • Network Abstraction Layer (NAL; add-on)

2

Resource Partitioning

Target Platform

RTX64 and the latest version of RTX do not support the concept of a shared configuration. RTX64 and RTX 2016 require systems with at least two logical processors/cores, having one core dedicated to RTSS. RTX versions prior to RTX 2016 support shared configurations and can be installed on uni-processor systems.

The total number of processors supported on a system:

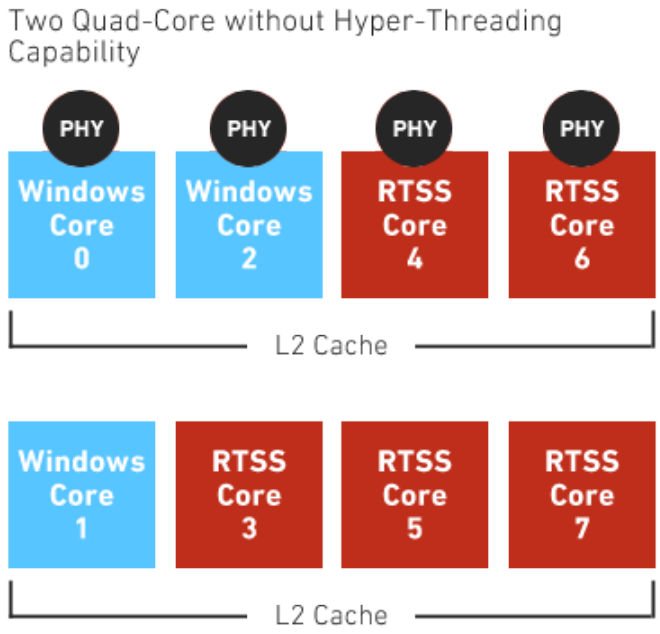
- RTX – 32 (up to 31 for RTSS)
- RTX64 – 64 (up to 63 for RTSS)

RTX64 does not have the limitation regarding clustered systems. In the RTX product, a system with more than eight cores is considered to be in clustered mode. During processor partitioning, at most four processors can be assigned to Windows. This limitation does not exist in RTX64, where you can assign as many processors as you want to Windows.

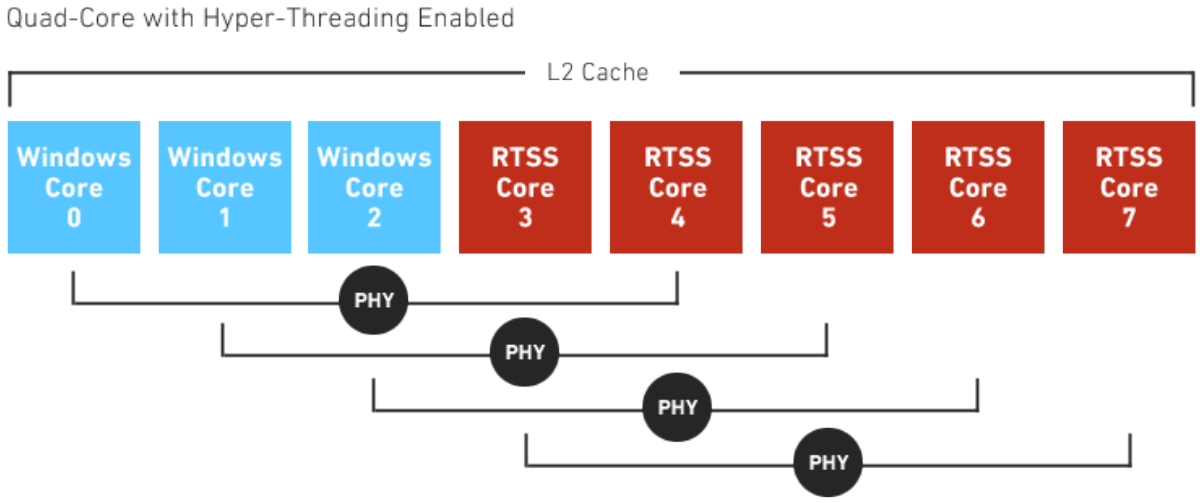
Processor Enumeration

The enumeration of processors is different between 32-bit and 64-bit Windows operating systems. This is important: if you share cache between Windows and RTSS processors, you can introduce latency into your RTSS application.

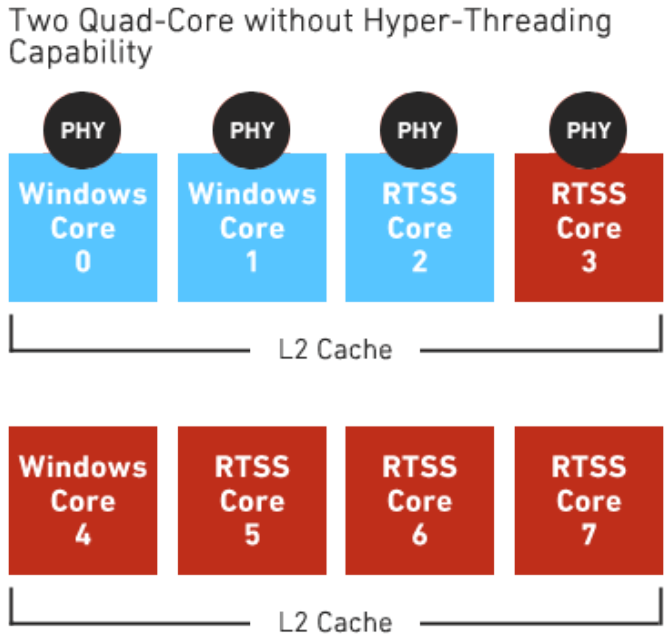
On a 32-bit Windows OS with multiple cores, the cores are enumerated beginning with the first core from each physical processor. For example, if you have 8 logical cores—3 cores dedicated to Windows and 5 to RTSS:



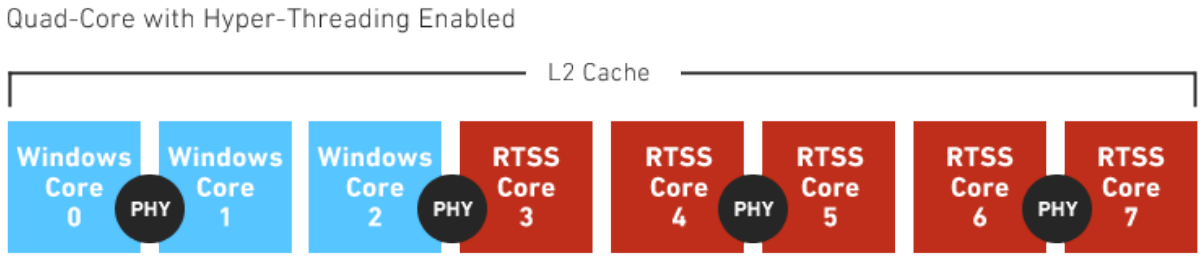
If hyper-threading is enabled, processors are enumerated as follows:



On a 64-bit Windows OS with multiple cores, the cores are enumerated beginning with the first physical processor then the next. For example if you have 8 logical cores—3 cores dedicated to windows and 5 to RTSS.



If hyper-threading is enabled processors are enumerated as follows:



Subsystem Configuration and Concepts

RTX64 provides a control panel for configuring Subsystem behavior and performance. The underlying Properties API that exists in RTX is available in RTX64 as a Managed Code interface (`IntervalZero.RTX64.dll`) and a Native Code interface (`FrameworkNativeLib.dll`). These interfaces allow for configuration of the RTX64 Subsystem.

- **NOTE:** See the **RTX64Config** sample project, included with the RTX64 SDK, for a demonstration of how to use the RTX64 Managed Code Framework (`IntervalZero.RTX64.dll`) to configure the RTX64 Subsystem.
- **NOTE:** See the **Native Framework Client** sample project, included with the RTX64 SDK, for a demonstration of how to use the RTX64 Native Framework library in a Visual Studio C/C++ project.

These are concepts from RTX that no longer exist or are different in RTX64:

- RTX64 no longer uses the Windows image loader. Because of this, there is no longer the concept of:
    - process slots
    - registering of RTDLLs
    - registering of boot time processes (in RTX64 scheduled processes can be scheduled to start on Subsystem startup through the RTX64 Task Manager)
- Starvation Timeout functionality in RTX has been modified with RTX64. It is now referred to as a Watchdog Timer. Also, timeout calculation has been improved to better support SMP architecture with dedicated RTSS cores.
- Priority Inversion Prevention still exists in RTX64 but the concept of limited demotion does not. Within RTX64 you can either:
    - turn priority prevention off
    - turn priority prevention on with tiered demotions - with this protocol, a lower-priority thread that owns a mutex needed by a higher-priority thread will have its priority temporarily promoted to that of the high priority waiting thread until it has released the requested mutex
- Similar to RTX, the RTX64 thread stack cannot grow. Due to architecture changes the default thread Stack size in RTX64 has increased from two pages to eight pages. The default Stack size for **RTSSDebug** configurations with **C/C++ Runtime support** enabled is 32 pages (131072 bytes).
- The current implementation of the local memory allocation is the same between RTX and RTX64. However, the default pool size in RTX64 has increased from 65536 bytes to 1048576 bytes.

- Shutdown handling has been re-architected for RTX64. As a result, the causes for a shutdown handler to be called have changed. Also, the RTX64 shutdown handler is not called as early in the shutdown process as it is when using RTX.
- Time Quantum can be set in microsecond granularity in RTX64. In RTX it is milliseconds.
- Exceptions cannot be configured individually in RTX64, where they can in RTX.
- RTX64 uses a concept called monitoring and provides a graphical user interface through Percepio. RTX utilizes a real-time event tracing tool, RTX Time View, that allows you to efficiently capture and display the execution sequence of threads within RTX. In RTX64, this is done through the Monitor utility, which allows you to trace the behavior of your real-time applications by recording significant events that occur during execution of those applications. You can then view saved session data in Tracealyzer.
- RTX64 supports search paths for loading RTSS applications and RTDLLs.

RTX	RTX64	Description
RT_SHUTDOWN_NT_SYSTEM_SHUTDOWN	SHTDN_REASON_WINDOWS_SYSTEM_SHUTDOWN	The system is starting a normal shutdown. Shortly after all shutdown handlers have been executed, Windows will stop.
RT_SHUTDOWN_NT_STOP	SHTDN_REASON_WINDOWS_STOP	Windows has stopped (i.e., blue screen or stop screen). RTSS will continue to operate with service restrictions.
	SHTDN_REASON_RTX_SYSTEM_SHUTDOWN	RTX64 stop has been requested. Shortly after all shutdown handlers have been executed, RTX will stop.

Signing INF Files for Custom Hardware in RTX64

The RTX64 PNP file contains a list of basic devices along with Network Interface Cards (NICs) that are supported by RTX64. To convert a device that RTX64 does not provide out-of-the-box support for, you must create an INF file for Windows to allow that device to be converted to RTX64. The INF file is used to associate custom devices with the RTX64 Plug and Play drivers that request and obtain resources, such as IRQs. Network devices should reference the Network class and be associated with **RTX64pnpNet**. Other devices should be associated with the **RTX64pnp** driver.

64-bit operating systems require drivers to be signed. Once you have created your INF file, and you need assistance with signing it, please contact IntervalZero Support. Support will sign it for you. See the RTX64 TechNote *Creating a Custom INF File* for information and instructions.

Windows 10 uses attestation signing and supports Secure Boot on 64-bit systems.

3

Application Organization

RTSS Dynamic Link Libraries

RTX utilizes the concept of RTSS dynamic link libraries (RTSS DLL). An RTSS DLL is not truly a DLL but an RTSS process (with the .RTSS extension) that exports functions for use by other RTSS processes. Since RTSS processes run in kernel space, other RTSS applications can access and use the RTSS DLL's exported functions.

NOTE: The concept of an RTSS DLL is no longer supported in RTX64; it is recommended that all RTSS DLLs be changed to Real-time dynamic link libraries (RTDLL).

Real-time Dynamic Link Libraries

With RTX Real-time dynamic link libraries (RTDLLs), you could only use explicit loading. This means that you had to call LoadLibrary and GetProcAddress within your application to use functions exported from an RTDLL. With RTX64, you can still use explicit loading but you can also use implicit loading. This means you can include the library in your linker setting and then use exported functions directly.

There are a few additional changes with RTDLL functionality in RTX64:

- LoadLibrary no longer accepts the `.dll` extension when called within an RTSS application. You must use `.rtdll`.
- You are no longer required to register an RTDLL with the subsystem. The RTX64 image loader will look in the current directory where the RTSS application loading the RTDLL is located, it will then use the Subsystem search path. You can also use LoadLibrary and give a full path to the RTDLL if you do not wish to place it in the same directory as the RTSS process.

- In RTX64, data within an RTDLL is private. RTDLLs are loaded within the process space of the loading RTSS process. If multiple processes load the same RTDLL, it will be loaded into each process' space. This behavior is different from RTX, where a RTDLL would be marked as shared when registered and loaded under the subsystem and data was shared between all processes.
- In RTX64, RTDLLs can be loaded within the context of the loading process and can be loaded implicitly or explicitly.

RTAPI

The RTAPI libraries for RTX64-enabled Windows applications have been completely rewritten to allow for a common interface for 32-bit and 64-bit windows applications. Because of this there are some key differences between RTX and RTX64.

Priority Class

In RTX, all RTX-enabled windows processes begin execution using the normal Win32 priority class (THREAD_PRIORITY_NORMAL), but are placed in the Win32 real-time priority class (THREAD_PRIORITY_TIME_CRITICAL) after a call to RtGetThreadPriority or RtSetThreadPriority.

Priority of Proxy Threads

When using RTX-enabled Windows processes, proxy threads are created within the RTX subsystem to mirror their Windows counterparts. By default, these proxy threads are set to the RTX priority of RT_PRIORITY_MIN but are mapped to an equivalent RTSS priority if RtSetThreadPriority is called from your Windows application for a given thread. As for RtCreateTimer, its timer handler thread is set to THREAD_PRIORITY_TIME_CRITICAL.

This is not true for RTX64. Priorities are no longer mapped between Windows and RTSS. **RtSetThreadPriority** and **RtGetThreadPriority** are deprecated from the Windows-supported RTAPI calls. Developers must now set the priority of Windows threads using the Windows functions **SetThreadPriority** and **GetThreadPriority**, and set the priority of Proxy threads using **RtSetProxyThreadPriority** and **RtGetProxyThreadPriority**.

4

Project Settings and Configurations

Like RTX, RTX64 has an application wizard that can be used to create an RTSS executable or RTDLL.

Configurations

The RTX64 application wizard provides four default configurations that build 64-bit applications using the Microsoft 64-bit compiler or the Intel 17.0 x64 compiler:

- Debug – windows (.exe or .dll) with debug information
- Release – windows (.exe or .dll)
- RTSSDebug – RTX64 (.rtss or .rtdll) with debug information. Note that the default Stack size for this configuration with **C/C++ Runtime support** enabled is 32 pages (131072 bytes), depending on the Visual Studio version.
- RTSSRelease – RTX64 (.rtss or .rtdll)

RTX64 also supports 32-bit Windows processes. To create a 32-bit configuration, follow the directions located here http://www.intervalzero.com/library/RTX64/HTML5/RTX64_Help.htm#PROJECTS/Application%20Development/32BitUserSpaceRTX64Application.htm

Headers

RTX contains a header file, RtApi.h, that contains the prototypes for all RTX exported functions. In RTX64 the RtApi header file has been divided into two header files:

- RtApi.h – contains functionality available to both RTSS and RTX64-enabled Windows applications
- RtssApi.h – contains functionality available for RTSS applications

To build code as a Windows .EXE or an RTX64 .RTSS application, we recommend the following:

```
#include "rtapi.h"
#ifdef UNDER_RTSS
#include "rtssapi.h"
#endif //UNDER_RTSS
```

To use RT-TCP/IP API calls in RTX you needed to include `Drvutil.h` and `Rtnapi.h`. In RTX64, you just need to include `Rtnapi.h`.

Libraries

For a Real-time process (RTSS) within RTX64, you will need to include `Rtx_rtss.lib` and `StartupCRT.lib` if you are including the Microsoft C Runtime. In RTX, these libraries are provided as object (.obj) files. To use RT-TCP/IP you will need to include `RtTcpip.lib`. In RTX this library was called `Rtxtcpip.lib`.

For a Windows process (EXE) linked to RTX64 you will need to include `Rtapi.lib`. There are two versions of this library: one if you are using the x86 compiler and another for x64.

RTX64 linker inputs when the C Runtime library is included:

RTSSRelease/x64	RTSSDebug/x64
• startupCRT.lib	• startupCRTd.lib
• libcmt.lib	• libcmtd.lib
• libcpmt.lib	• libcpmtd.lib
• libucrt.lib	• libucrtd.lib
• libvcruntime.lib	• libvcruntimed.lib
• rtx_rtss.lib	• rtx_rtss.lib

RTX64 linker inputs when no C Runtime library is included:

RTSSRelease/x64	RTSSDebug/x64
• rtapi_rtss.lib	• rtapi_rtss.lib
• rtx_rtss.lib	• rtx_rtss.lib

5

Coding Changes

Real-Time API Calls

The table below lists the Real-Time APIs that were available in RTX but are not available in RTX64.

RTX 2016	RTX64 3.7

`PVOID RtAllocateLockedMemory(`
`  UINT nNumberOfBytes`
`);`
	Not supported, as RTX64 no longer supports soft Real-time features in a Win32 process.

`HANDLE RtAttachInterruptVector(`
`  PSECURITY_ATTRIBUTES pThreadAttributes,`
`  ULONG StackSize,`
`  VOID (RTFCNDCL *pRoutineIST)(PVOID ContextIST),`
`  PVOID ContextIST,`
`  ULONG Priority,`
`  INTERFACE_TYPE InterfaceType,`
`  ULONG BusNumber,`
`  ULONG BusInterruptLevel,`
`  ULONG BusInterruptVector`
`);`
	Use RtAttachInterrupt and RtReleaseInterrupt

RTX 2016	RTX64 3.7
HANDLE RtAttachInterruptVectorEx(PSECURITY_ATTRIBUTES *pThreadAttributes*, ULONG *StackSize*, BOOLEAN (*RTFCNDCL *pRoutineIST*) (*PVOID Context*), PVOID *Context*, ULONG *Priority*, INTERFACE_TYPE *InterfaceType*, ULONG *BusNumber*, ULONG *BusInterruptLevel*, ULONG *BusInterruptVector*, BOOLEAN *ShareVector*, KINTERRUPT_MODE *InterruptMode*, INTERRUPT_DISPOSITION (*RTFCNDCL *pRoutineISR*)(*PVOID Context*));	Use RtAttachInterrupt and RtReleaseInterrupt
BOOL RtCommitLockHeap(HANDLE hHeap, ULONG nNumberOfBytes, VOID (RTFCNDCL *pExceptionRoutine) (HANDLE));	Not supported, as RTX64 no longer supports soft Real-time features in a Win32 process.
BOOL RtCommitLockProcessHeap(ULONG nNumberOfBytes, VOID (RTFCNDCL *pExceptionRoutine) (HANDLE));	Not supported, as RTX64 no longer supports soft Real-time features in a Win32 process.
BOOL RtCommitLockStack(ULONG nNumberOfBytes);	Not supported, as RTX64 no longer supports soft Real-time features in a Win32 process.

RTX 2016	RTX64 3.7

`BOOL RtFreeLockedMemory(`
`  PVOID pVirtualAddress`
`);`
	Not supported, as RTX64 no longer supports soft Real-time features in a Win32 process.

`INT RtGetThreadPriority(`
`  HANDLE hThread`
`);`
	Deprecated on the Windows side. Use `RtGetProxyThreadPriority`.

`BOOL RtLockKernel(`
`  ULONG Section`
`);`
	Not supported, as RTX64 no longer supports soft Real-time features in a Win32 process.

`BOOL RtLockProcess(`
`  ULONG Section`
`);`
	Not supported, as RTX64 no longer supports soft Real-time features in a Win32 process.

`BOOL RtReleaseInterruptVector(`
`  HANDLE hInterrupt`
`);`
	Not supported

`BOOL RtSetThreadPriority(`
`  HANDLE hThread,`
`  int RtssPriority`
`);`
	Deprecated on the Windows side. Use `RtSetProxyThreadPriority`.

`ULONG RtStartPerfMeasure(`
`  ULONG perfMeasureType,`
`  ULONG *ioBuffer,`
`  ULONG ioBufferSize,`
`  ULONG *ioBufferWrite,`
`  LONGLONG *lPerfCyclesPerSecond,`
`  ULONG *platformConfig,`
`  ULONG *rtssProcessorNumber`
`);`
	Not provided for developer use

RTX 2016	RTX64 3.7

`ULONG RtStopPerfMeasure(`
`  ULONG perfMeasureType,`
`  ULONG *ioBuffer`
`);`
	Not provided for developer use

`BOOL RtTraceEvent(`
`  ULONG TraceEventID,`
`  PVOID arg1,`
`  PVOID arg2`
`);`
	`HANDLE RtGenerateEvent(`
	`  UNSIGNED INT kind,`
	`  VOID * buffer,`
	`  SIZE_T size`
	`);`

`BOOL RtUnlockKernel(`
`  ULONG Section`
`);`
	Not supported, as RTX64 no longer supports soft Real-time features in a Win32 process.

`BOOL RtUnlockProcess(`
`  ULONG Section`
`);`
	Not supported, as RTX64 no longer supports soft Real-time features in a Win32 process.

Windows Supported APIs

The table below lists the Windows supported APIs that are available in RTX but are not available in RTX64.

RTX 2016	RTX64 3.7

`INT GetThreadPriority(`
`  HANDLE hThread`
`);`
	Deprecated. This API can no longer be used by Windows processes linked to RTX64. Instead, users must use the Windows API for setting the priority of Windows threads. To get the priority of Windows Proxy threads that interact with the RTSS Subsystem, users must use the new Proxy API `RtGetProxyThreadPriority`.

`BOOL SetThreadPriority(`
`  HANDLE hThread,`
`  int nPriority`
`);`
	Deprecated. This API can no longer be used by Windows processes linked to RTX64. Instead, users must use the Windows API for setting the priority of Windows threads. To set the priority of Windows Proxy threads that interact with the RTSS Subsystem, users must use the new Proxy API `RtSetProxyThreadPriority`.

Inline Assembly

One of the constraints for the Microsoft and Intel x64 compilers are to have no inline assembler support. This means that functions that cannot be written in C or C++ will either have to be written as subroutines or as intrinsic functions supported by the compiler. Certain functions are performance-sensitive while others are not. Performance-sensitive functions should be implemented as intrinsic functions. For more information on intrinsic, go here http://msdn.microsoft.com/en-us/library/26td21ds%28v=vs.100%29.aspx

6

RT-TCP/IP Stack

The Real-time TCP/IP Stack (RT-TCP/IP) is a separate purchasable feature of RTX64, whereas with RTX the RT-TCP/IP stack was part of the Subsystem by default. The RTX64 Stack is a deterministic, high-performance stack based on the Treck TCP/IP version 6.0 Release. The RTX RT-TCP/IP Stack is based on the Fusion 8.x version.

The RTX64 RT-TCP/IP Stack was designed with SMP in mind; it has the capability of running internal threads (such as receive and transmit) across multiple cores. The Treck stack is also capable of handling process threads running on multiple cores. This was a limitation of the Fusion stack.

Configuration

Configuration of the RTX64 RT-TCP/IP Stack is no longer done through an INI file. All stack configurations are done through the RTX64 control panel or programmatically through the managed code framework.

Below are other differences between RTX and RTX64:

- RAW sockets
- WSAStartup can be called multiple times in RTX64 (3.0 and above)
- RTX64 supports running components on different cores

Network Driver Development

The RTX64 Subsystem comes with a number of real-time network drivers. For more information on what is available, see the *RTX64 Supported NICs* document available online at http://www.intervalzero.com/technical-support/guides-and-minitutorials/.

There is a defined interface for those who want to build their own NIC driver that can interact with the RT-TCP/IP Stack. This interface is different from the RTX interface. Below are RTX functions that have been modified or removed.

Real-time NIC Device Drivers

RTX 2016	RTX64 3.7

`int RtndInitialize (`
`  short* stackIniFile,`
`  char* pszSectionName,`
`  int fDisplayErrorMessages,`
`  char *deviceName,`
`  RtndConfigStruct *rtCfg`
`);`
	`int RtndInitialize (`
	`  int fDisplayErrorMessages,`
	`  char *deviceName,`
	`  PRTNINTERFACE pNetInterface`
	`);`

Real-time Network Functions

RTX 2016	RTX64 3.7

`BOOL RtnAddArp(`
`  char * pDevName,`
`  unsigned char * pLinkAddr,`
`  unsigned char * pProtoAddr,`
`  unsigned short TTL`
`);`
	`BOOL RtnAddArp(`
	`  char * pDevName,`
	`  unsigned char * pLinkAddr,`
	`  unsigned char * pProtoAddr,`
	`  unsigned short TTL`
	`);`

`void * RtnAllocateFrame(`
`  char *DevName,`
`  int DataSize`
`);`
	Functionality provided by `RtndFrameAllocate`

`void RtnAllocRecvBuffers(`
`  void *ndp`
`);`	

RTX 2016	RTX64 3.7
int RtnEnumPciCards(int (*fnCheckPciCard) (PPCI_COMMON_CONFIG pPciInfo, ULONG bus, PCI_SLOT_NUMBER SlotNumber, PVOID pCardInfo), PVOID pUserInfo);	BOOL RtnEnumPciCards(int (*fnCheckPciCard) (PPCI_COMMON_CONFIG pPciInfo, ULONG bus, PCI_SLOT_NUMBER SlotNumber, PVOID pCardInfo, int fMacAddressPresent), PVOID pUserInfo int fMacAddressPresent);
void * RtnFreeFrame(void *Frame);	
void RtnFreeRecvBuffers(void *ndp);	
FILTER_STATE RtnGetFilterState(u32 ipaddr, FILTER_ID filter);	
int RtnGetPrivateProfileString(const WCHAR* fileName, char* pszSection, char* pszEntry, char* pszDefault, char* pszBuffer, int cbBuffer);	

RTX 2016	RTX64 3.7
unsigned long RtnHtoi(char* pszHex);	
BOOL RtnIsDeviceOnline(Cahr* devName);	
BOOL RtnIsStackOnline();	
void RtnProcessRecvQueue(void *ndp);	
void RtnQueueRecvFrame(void *ndp, void *mp, unsigned long framesize);	
void RtnQueueRecvPacket(void *ndp, void *mp, unsigned long packetsize);	
FILTER_STATE RtnSetFilterState(u32 ipaddr, FILTER_ID filter, FILTER_STATE State);	
BOOL RtnTransmitFrame(void *mp);	Functionality provided by `RtndFrameTransmit`

Real-time NIC Filter Functions

RTX 2016	RTX64 3.7

void RtndUpDownFilter (
    void *ndptr,
    unsigned short flags,
    char *options
);	

void RtndIOCTLFilter(
    void *ndp,
    int cmd,
    char *addr
);	

FRAME_STATE RtndReceiveFilter(
    void *ndptr,
    void *mptr,
    unsigned long framesize
);
	BOOL RtndReceiveFilter(
	    EthernetHeader *pEthernetHeader,
	    void *pData,
	    unsigned long ulEthernetDataSize
	);

FRAME_STATE RtndTransmitFilter(
    void *mp
);
	BOOL RtndTransmitFilter(
	    EthernetHeader *pEthernetHeader,
	    void *pData,
	    unsigned long ulEthernetDataSize
	);

7

Real-Time Network Abstraction Layer (NAL)

The RTX64 Network Abstraction Layer (NAL) add-on is a network layer that abstracts the network hardware and driver functions from the upper-level protocol stacks and provides management interfaces for those upper layers to easily query for and use available network assets. It is a separate protocol layer from the RT-TCP/IP Stack. Using the NAL, you can more easily take advantage of network functionality such as EtherCAT, TSN (Time Sensitive Networks), and PTP (Precision Time Protocol).

The NAL supplies a simplified API which abstracts the caller from the various register configurations which vary from adapter to adapter. It also supplies methods to allow direct layer 2 transmit and receive calls within the driver, thus eliminating the latencies found in a TCP/IP stack.

For example, the user can call a transmit function in the driver that allows the caller to pass multiple packets at once. This greatly improves performance and allows for transmission of small packets at near line speed. The driver will direct the packets through whichever priority queue it was instructed and can call the caller back with extended information like the actual transmission time of the packet.

Getting Support

IntervalZero offers a number of support options for RTX64 users, including technical support and the IntervalZero Website.

Third-Party Support

If you are a customer who purchased an IntervalZero product through a third-party reseller, contact the reseller for support.

Contacting Technical Support by Phone

Location	Number	Hours
United States	1-781-996-4481 At the prompt, press 3 for Support.	Monday - Friday, 8:30 a.m. – 5:30 p.m. US Eastern Time (GMT-500), excluding holidays.
R.O.C. Taiwan	+ 886-2-2556-8117	Monday - Friday, 9:00 a.m. – 5:00 p.m. Taipei Standard Time (GMT+8), excluding holidays.

Before Calling Technical Support

Please have the following information ready before calling IntervalZero Technical Support:

- **Your Support ID**: customers who purchase direct support receive an e-mail address and password for use when accessing the IntervalZero support web site.
- **The version number of your RTX64 software**: determine the version of RTX64 installed on your system.
- **Your maintenance status**: check to make sure you have a valid maintenance contract.

TO FIND YOUR VERSION OF RTX64:

Windows 10

1. Navigate to Start > RTX64 3.7 Runtime > Control Panel.
2. Record the RTX64 version that is shown in the RTX64 control panel.

Windows 8.1

1. Click the **RTX64** metro tile. The RTX64 Control Panel appears.
2. Record the RTX64 version that is shown in the RTX64 control panel.

Windows 7

1. Navigate to **Start > Control Panel > System and Security category > RTX64**.
2. Record the RTX64 version that is shown in the RTX64 control panel.

IntervalZero Website

The IntervalZero Customer Support Web page is located at:

http://www.intervalzero.com/technical-support/

The IntervalZero support web pages provide electronic access to the latest product releases, documentation, and release notes. With a valid e-mail address and password, you can access the online problem report database to submit new issues or to obtain the status of previously reported issues.

Index