Porting Applications from RTX64 3.x to 4.0

USER GUIDE

IntervalZero

RTX64

Porting Applications from RTX64 3.x to RTX64 4.0 IZ-DOC-X64-0252

IntervalZero

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

www.intervalzero.com

Contents

About this Porting Guide

RTX64 4.0 contains new functionality and significant changes and improvements to existing functionality from RTX64 3.x versions. This document describes the steps required to port an existing RTX64 3.x application to RTX64 4.0. 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.

NOTE: For information on porting from RTX to RTX64, see the *RTX64 Porting Guide*.

1

Product Comparison

Functionality that existed in RTX64 3.x may not currently exist in RTX64 4.0, or it may have been re-architected. Some shared functionality may behave differently in RTX64 3.x and RTX64 4.0.

This section outlines major differences in product packages and system requirements.

Available Product Packages

The RTX64 3.x and RTX64 4.0 product packages include some functionality differences:

RTX64 3.x	RTX64 4.0

• Runtime (can be installed silently)
  ◦ includes the RT-TCP/IP Stack (optional; must be purchased separately)
• SDK
• Runtime Merge Modules
• Network Abstraction Layer (NAL; standalone add-on)
• RTX64 Vision (standalone add-on; RTX64 3.7 only)
	• Runtime (can be installed silently)
	  ◦ includes the Network Abstraction Layer (NAL)
	  ◦ includes the RT-TCP/IP Stack (always installed; must be purchased separately)
	• SDK
	• Runtime Merge Modules

Software Requirements

Operating System Support

RTX64 3.x	RTX64 4.0
• Windows 10 (build support depends on 3.x version) • Windows 8.1 with Update • Windows 7 SP1 • Windows Embedded 8 • Windows Embedded 7	• Windows 10 (Semi-Annual Channel 1909, 1903, 1809, and LTSC 2019 and 2016)

Microsoft Visual Studio Support

RTX64 3.x	RTX64 4.0
• Visual Studio 2019 (RTX64 3.7 only) • Visual Studio 2017 • Visual Studio 2015 • Visual Studio 2013 • Visual Studio 2012	• Visual Studio 2019 • Visual Studio 2017 • Visual Studio 2015 (deprecated)

2

Subsystem Configuration and Concepts

Configuration

Dongle Support

RTX64 3.x	RTX64 4.0
• Small form factor • Standard	• Small form factor

Control Panel

The RTX64 control panel underwent several changes between RTX64 3.x and RTX64 4.0. The most significant of these changes are described below.

Home Page

The Home page of the control panel was redesigned for RTX64 4.0. The right side of this page includes an information panel that provides information on the Runtime components you have installed, as well as their activation status. The information panel also displays the status of installed components, such as whether a component is running, whether Subsystem Monitoring is enabled, the current Subsystem memory allocation setting, and the start behavior of network components. You can also start and stop the RTSS Subsystem and associated components directly from the information panel.

Managing Memory

In RTX64 3.x, you could customize default behavior for how the RTX64 Subsystem interacts with Windows during memory allocation from the *Configure Memory Allocation Behavior* control panel page.

In RTX64 4.0, this control panel page is called *Manage memory*. This page now memory allocation values for the Subsystem and other Runtime components on a single page. This includes memory settings for Networking (NAL and Stack) and Monitoring.

The *Manage memory* page in RTX64 4.0 now provides a *Memory profile*, where you can configure the approximate amount of system memory available to RTX64 and view memory usage information for RTX64 components. When changes are made to allocation values on the page, the usage values update accordingly. This provides a helpful approximation of how memory will be allocated initially, and whether there is enough remaining memory to run the number of RTSS processes you require.

Zeroing Memory on Allocation

The memory allocated by **malloc** or similar C-Runtime functions is not initialized to zero according to the C99 specification, which Windows follows. To maintain backwards compatibility, RTX64 4.0 provides a **Zero memory on allocation** setting.

This setting is enabled by default.

NOTE: Zeroing memory may cause a performance lag. If this lag is not acceptable, disable Zero memory on allocation. When disabled, memory requests from some C-Runtime functions or RTAPIs will not be initialized to zero at allocation. However, by definition, memory requests from **calloc**/**_recalloc**, **VirtualAlloc**, and **HeapAlloc/HeapReAlloc** with the HEAP_ZERO_MEMORY flag will be initialized to zero at allocation.

See the RTX64 Help for more information.

Local Memory Architecture

Local memory allocation was re-architected in RTX64 4.0. Local memory allocation is now per-process, not Subsystem wide. In RTX64 4.0, the local memory configuration contains multiple allocation spaces (MSpaces) that make up a system: the RTX64 Subsystem, NAL, and RT-TCP/IP Stack along with each RTSS process and Windows proxy process. Memory allocation requests from RTAPIs, including from C-Runtime libraries, are allocated within these memory allocation spaces. For more information, see the Help topic *Allocation Spaces in Local Memory*.

Memory is allocated from one of the process MSpaces unless the memory is required to remain over process exit, such as memory for IPC objects and cross-process shared memory. Such allocations are made from the Subsystem's MSpace.

When local memory is used, the Subsystem's MSpace is created when the first request for memory is made, unless commit is selected in which case the memory is allocated as soon as the Subsystem starts. Individual process MSpaces behave similarly. Memory is allocated on the first memory request unless RtssRun with option */i initial_size* is used. It provides deterministic behavior for normally non-deterministic functions, as well as greater flexibility and functionality after a system crash (blue screen).

You can specify the initial size of the local memory allocation spaces (MSpaces) in kilobytes. If RTSS applications exhaust the memory you initially allocate, you can configure each MSpace to automatically expand.

Default Memory Allocation

The default memory allocation selection is different in RTX64 3.x and RTX64 4.0.

RTX64 3.x	**RTX64 4.0**
Use Windows memory (non-deterministic)	*Use local memory (deterministic)* Default memory allocation cannot be overridden. For example, when *Use local memory* is selected, the RTX64 Subsystem, NAL, and RT-TCP/IP Stack along with each RTSS process and Windows proxy process use local memory.

RtssRun Memory Parameters

RtssRun provides different memory parameters in RTX64 3.x and RTX64 4.0.

RTX64 3.x	RTX64 4.0
/l If *Request from Windows* is enabled in the RTX64 control panel, the /l switch overrides this setting and causes RTSS applications to request memory from the RTX64 local memory pool instead whenever memory is allocated. */n* If *Request from local memory pool* is enabled in the RTX64 control panel, the /n switch overrides this setting and causes RTSS applications to request memory from the Windows non-paged pool instead whenever memory is allocated.	You cannot override the default memory allocation selection (see above) on a per-process basis. Accordingly, parameters */l* and */n* were removed. New local memory parameters were added: */e expand_size* The amount of local memory, in kilobytes, by which to expand the process MSpace. This applies only when *Use local memory* is selected in the control panel. */i initial_size* The initial amount of local memory (allocated at process startup), in kilobytes, within the process MSpace. This applies only when *Use local memory* is selected in the control panel.

MSpaces Tool

RTX64 4.0 introduces a new RTX64 MSpaces utility, which displays local memory allocation spaces (MSpaces) of all RTSS processes (optionally including internal system processes and proxy processes). For each process's external and internal MSpace, this utility displays summary and fragmentation information within the MSpace as a whole, or separately within Local Pool and Pool Cache of the MSpace. The memory ranges can be sorted in ascending order by range start address.

See the RTX64 Help for more information.

3

Project Settings and Configurations

RTX64 SDK Environment Variable

The RTX64 SDK environment variable is different between RTX64 3.x and RTX64 4.0. This needs to be replaced in all existing projects built with an RTX64 3.x SDK.

RTX64 3.x SDK Environment Variable	RTX64 4.0 SDK Environment Variable
`$(RTX64SDKDir3)`	`$(RTX64SDKDir4)`

Headers

The following header files were renamed between RTX64 3.x and RTX64 4.0:

RTX64 3.x	RTX64 4.0
ErrorCodes.h	RtErrorCodes.h
Licensing.h	RtLicensing.h

The Network Abstraction Layer (NAL) APIs are installed with the RTX64 4.0 SDK. To use Network Abstraction Layer (NAL) API calls, you need to include `RtNalApi.h`.

Libraries

The Network Abstraction Layer (NAL) is installed with the RTX64 4.0 Runtime. To use the NAL, you will need to include `RtNal.lib` when developing and building your application.

4

Networking

Network Components

The RTX64 real-time network consists of two primary components: a Network Abstraction Layer (NAL) and an RT-TCP/IP protocol stack.

Real-Time Network Abstraction Layer (NAL)

The NAL 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 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).

RTX64 3.x	RTX64 4.0
Available as a separate standalone component.	Installed by the RTX64 Runtime.

RT-TCP/IP Stack

The RT-TCP/IP Stack is an optional purchasable protocol stack that provides deterministic processing and networking capability.

RTX64 3.x	RTX64 4.0
Optionally installed by the RTX64 Runtime. Requires a separate purchasable license.	Always installed by the RTX64 Runtime. Requires a separate purchasable license. In RTX64 4.0, the TCP/IP Stack is dependent on the NAL.

Configuring and Controlling the Network

RTX64 4.0provides several options for configuring network (NAL and TCP/IP Stack) behavior and performance through the *Configure and control the network* control panel page. You can also manually start, stop, and restart the NAL and/or Stack directly from this page.

You can add, delete, set properties for, and associate filters with RTX64 interfaces from the *Manage interfaces* control panel page.

Real-time NIC Device Drivers

As of RTX64 4.0, the Runtime installer no longer installs RT-TCP/IP Stack NIC drivers. Only Network Abstraction Layer (NAL) drivers are installed. These drivers were previously available in the standalone version of the NAL. See the *RTX64 Supported NICs* document for a complete list.

See the Help topic *Porting a TCP/IP Driver to the NAL* for the steps required to port an existing (pre-RTX64 4.x) TCP/IP Stack driver for use with the new network architecture in RTX64 4.0 and later, where the RT-TCP/IP Stack is layered on the Network Abstraction Layer (NAL).

Including Network Support in Applications

In RTX64 4.0, the RTX64 Application and RTDLL project templates in Visual Studio include a new setting to optionally include support for the Network Abstraction Layer (NAL) component.

5

Coding Changes

The following sections list the APIs in the RTX64 3.x SDK that fall under these classifications in RTX64 4.0:

- Were enhanced with non-breaking changes
- Underwent breaking changes
- Were removed or deprecated

Real-Time APIs

This section outlines the changes to Real-Time APIs in RTX64 4.0.

Enhanced Real-Time APIs

This section lists the APIs included in the RTX64 3.x SDK that were enhanced with non-breaking changes in RTX64 4.0.

RTX64 3.x API	RTX64 4.0
IP_ADAPTER_ADDRESSES	Added support for these members: • *ReceiveLinkSpeed* specifies the current speed in bits per second of the receive link for the adapter. • *TransmitLinkSpeed* specifies the current speed in bits per second of the transmit link for the adapter.
MF_EVENT_KIND	Updated event MF_EVENT_KIND_PROCESS_ CREATE to include information on Local Memory MSpaces.
RtUpdateProcThreadAttribute	Added new attributes: • RT_PROC_THREAD_ATTRIBUTE_MSPACE_ MINIMUM_INITIAL_SIZE • RT_PROC_THREAD_ATTRIBUTE_MSPACE_ EXPAND _SIZE

Breaking Changes to Real-Time APIs

This section lists the APIs included in the RTX64 3.x SDK that underwent breaking changes in RTX64 4.0.

IMPORTANT! Existing real-time applications that call these APIs may need to be updated to ensure binary compatibility with RTX64 4.0.

RTX64 3.x API	RTX64 4.0
RtAllocateLocalMemory	
Allocates memory from a pre-allocated RTSS local memory pool to avoid SRI activity if allocating memory from the Windows memory pool.	Allocates memory from the external MSpace of the current process.
RtAllocateLocalMemoryEx	
Allocates memory from a pre-allocated RTSS local memory pool to avoid SRI activity if allocating memory from the Windows memory pool. Using the memory allocation option RTALLOC_NOT_ZERO_MEMORY, RtAllocateLocalMemoryEx allows you to allocate memory without zero-initializing the memory region.	Allocates memory from the external memory allocation space (MSpace) of the current process. RtAllocateLocalMemoryEx allows you to allocate memory without zero-initializing the memory region.
RtUpdateProcThreadAttribute	Deprecated attribute RT_PROC_THREAD_ATTRIBUTE_USE_LOCAL_MEMORY

RtGetProcessAffinityMask
`BOOL RtGetProcessAffinityMask(`
`  HANDLE hProcess,`
`  PDWORD_PTR pProcessAffinityMask,`
`  PDWORD_PTR pSystemAffinityMask`
`);`
	`BOOL RtGetProcessAffinityMask(`
	`  HANDLE hProcess,`
	`  PUINT64 pProcessAffinityMask,`
	`  PUINT64 pSystemAffinityMask`
	`);`

RtSetProcessAffinityMask
`BOOL RtSetProcessAffinityMask(`
`  HANDLE hProcess,`
`  DWORD_PTR ProcessAffinityMask`
`);`
	`BOOL RtSetProcessAffinityMask(`
	`  HANDLE hProcess,`
	`  UINT64ProcessAffinityMask`
	`);`

RTX64 3.x API	RTX64 4.0

RtGetProcessorInfo
`BOOL RtGetProcessorInfo(`
`  PDWORD_PTR lpLowestSystemProcessor,`
`  PDWORD_PTR lpHighestSystemProcessor`
`);`
	`BOOL RtGetProcessorInfo(`
	`  PDWORD lpLowestSystemProcessor,`
	`  PDWORD lpHighestSystemProcessor`
	`);`

RtSleepFt	Removed Windows support. This function can now only be called from an RTSS process or RTDLL.
RtSleepFtEx	Removed Windows support. This function can now only be called from an RTSS process or RTDLL.

Removed/Deprecated Real-Time APIs

This section lists the RTX64 3.x APIs that were either removed from or are deprecated in RTX64 4.0.

RTX64 3.x API	RTX64 4.0
RtExpandLocalMemory	Removed. Use **RtExpandMSpace**, which expands the specified MSpace by the size specified.
RtGetLicenseFeatureStatus	Removed. Use **RtGetLicenses**, which retrieves information about all licenses available on the system.
RtGetLicenseFeatureStatusEx	Removed. Use **RtGetLicenses**, which retrieves information about all licenses available on the system.
RtQueryLocalMemory	Removed. Use **RtQueryProcessMSpace**, which queries memory allocation space information in the specified process MSpace, including various summary statistics of its local pool and pool cache.
RtQueryRtssInformation	Removed.

RTX64 3.x API	RTX64 4.0
RtShrinkLocalMemory	Removed. Use **RtShrinkMSpace**, which shrinks the specified MSpace by the size specified.

Real-Time Network APIs

RTX64 3.x API	RTX64 4.0
RtndFrameAllocate	Removed. Use **RtnFrameAllocate**, which can be used by a filter driver, and an RTSS application with RT-TCP/IP support enabled, to allocate a block of memory to store an Ethernet frame.
RtndFrameFree	Removed. Use **RtnFrameFree**, which returns an allocated frame back to the TCP/IP Stack.
RtndFrameTransmit	Removed. Use **RtnFrameTransmit**, which can be called when a filter or RTSS application wants to transmit a frame pointed to by the parameter EthernetFrame.
RtndInitialize	Deprecated. This function will be removed from the SDK in a future version. As a replacement, you can use **RtndInitializeInterface**, which is called for each instance of the driver when the driver should configure the NIC hardware.

Native Framework APIs

This section outlines the changes to Native Framework APIs in RTX64 4.0.

Enhanced Native Framework APIs

This section lists the APIs included in the RTX64 3.x SDK that were enhanced with non-breaking changes in RTX64 4.0.

RTX64 3.x API	RTX64 4.0
RTFW_SCHEDULED_PROCESS	Added new members: • *MSpaceInitialSize* specifies the size, in kilobytes, by which to allocate Non-Paged memory from Windows into the Local Pool of the MSpace at process startup. • *MSpaceExpandSize* specifies the size, in kilobytes, by which to expand an RTSS processes' internal/external MSpaces.

Breaking Changes to Native Framework APIs

This section lists the APIs included in the RTX64 3.x SDK that underwent breaking changes in RTX64 4.0.

RTX64 3.x API	RTX64 4.0

RtfwGetSubsystemStatus
`bool RtfwGetSubsystemStatus(`
`  RTFW_SUBSYSTEM_STATUS * pStatus`
`);`
	`bool RtfwGetSubsystemStatus(`
	`  RTFW_COMPONENT_STATUS * pStatus`
	`);`

RtfwStartNetwork	Renamed **RtfwStartTCPIPStack**
RtfwStopNetwork	Renamed **RtfwStopTCPIPStack**

RTX64 3.x API	RTX64 4.0
RTFW_FIRST_RTSS_PROCESSOR A constant used by RTFW_NETWORK_ CONFIGURATION and RTFW_NETWORK_INTERFACE which represents the first RTSS processor, regardless of its actual processor number.	Renamed **RTFW_DEFAULT_RTSS_PROCESSOR** When configuring the Network Abstraction Layer (NAL) or RT-TCP/IP stack, this constant represents the RTSS processor on which the NAL or Stack processes are running. When configuring other components of the RTX64 Subsystem, this constant represents the first RTSS processor.

RTFW_LOCAL_MEMORY_CONFIGURATION
 	
`typedef struct _RTFW_LOCAL_MEMORY_CONFIGURATION(`
`  size_t Size;`
`  bool UseLocalMemory;`
`  bool AutoExpandPool;`
`  bool AutoShrinkPool;`
`  unsigned int InitialSize;`
`  unsigned int ExpansionSize;`
`} RTFW_LOCAL_MEMORY_CONFIGURATION, * PRTFW_LOCAL_ MEMORY_CONFIGURATION;`
	`typedef struct _RTFW_LOCAL_MEMORY_CONFIGURATION(`
	`  size_t  Size;`
	`  bool  EnableLocalMemory;`
	`  unsigned int SystemExtMSpacePoolMinimumSize;`
	`  bool  SystemExtMSpacePoolCommit;`
	`  bool  SystemMSpacesPoolExpandable;`
	`  unsigned int  SystemMSpacesPoolExpandSize;`
	`  bool  SystemMSpacesPoolShrinkable;`
	`  unsigned int ProcessExtMSpacePoolMinimumSize;`
	`  bool  ProcessMSpacesPoolExpandable;`
	`  unsigned int  ProcessMSpacesPoolExpandSize;`
	`  bool  ProcessMSpacesPoolShrinkable;`
	`  bool  ZeroMemoryAtAllocation;`
	`} RTFW_LOCAL_MEMORY_CONFIGURATION, * PRTFW_LOCAL_ MEMORY_CONFIGURATION;`

RTX64 3.x API	RTX64 4.0

RTFW_NETWORK_INTERFACE

```
typedef struct _RTFW_NETWORK_INTERFACE(
    size_t Size;
    TCHAR FriendlyName[RTFW_MAX_INTERFACE_NAME_
CHARS];
    bool Enabled;
    TCHAR Driver[MAX_PATH];
    TCHAR DeviceInstanceID[RTFW_MAX_INTERFACE_
INSTANCEID_CHARS];
    TCHAR PCIBusLocation[RTFW_MAX_INTERFACE_
PCIBUSLOCATION_CHARS];
    size_t CountIPv4Addresses;
    RTFW_IPV4_CONFIGURATION IPv4Configurations
[RTFW_MAX_IPS_PER_INTERFACE];
    TCHAR Gateway[RTFW_IPV4ADDRESS_LENGTH];
    TCHAR IPv6Address[RTFW_MAX_INTERFACE_
IPV6ADDRESS_CHARS];
    unsigned int IPv6Prefix;
    unsigned int ReceivePriority;
    DWORD ReceiveIdealProcessor;
    unsigned int MTU;
    unsigned int NumberReceiveBuffers;
    unsigned int NumberTransmitBuffers;
    RTFW_INTERRUPT_TYPE InterruptType;
    unsigned int InterruptPriority;
    DWORD InterruptIdealProcessor;
    bool LinkStatus;
    unsigned int LinkStatusPriority;
    DWORD LinkStatusIdealProcessor;
    bool FilterDriverEnabled;
    TCHAR FilterDriver[MAX_PATH];
} RTFW_NETWORK_INTERFACE, * PRTFW_NETWORK_
INTERFACE;
```

```
typedef struct _RTFW_NETWORK_INTERFACE(
    size_t Size;

    // These fields are required for TCP/IP and
NAL interfaces.
    TCHAR Name[RTFW_MAX_INTERFACE_NAME_CHARS];
    bool Enabled;
    TCHAR Driver[MAX_PATH];
    unsigned int MaxPacketSize;
    bool LinkStatus;
    TCHAR PCIBusLocation[RTFW_MAX_INTERFACE_
PCIBUSLOCATION_CHARS];
    RTFW_INTERRUPT_TYPE InterruptType;
    unsigned int TransmitCompletePriority;
    unsigned int TransmitCompleteIdealProcessor;
    unsigned int NumberReceiveBuffers;
    unsigned int NumberTransmitBuffers;
    unsigned int NumberReceiveQueues;
    unsigned int NumberTransmitQueues;
    unsigned int DefaultReceiveQueue;
    DWORD InterruptIdealProcessor;
    unsigned int InterruptPriority;
    TCHAR DeviceInstanceID[RTFW_MAX_INTERFACE_
INSTANCEID_CHARS];
    bool SupportTCPIP;

    // These fields are required only if
SupportTCPIP is set to TRUE.
    unsigned int CountIPv4Configurations;
    RTFW_IPV4_CONFIGURATIONW IPv4Configurations
[RTFW_MAX_IPS_PER_INTERFACE];
    TCHAR Gateway[RTFW_IPV4ADDRESS_LENGTH];
    unsigned int MTU;
    bool FilterEnabled;
    TCHAR FilterDriver[MAX_PATH];
    TCHAR IPv6Address[RTFW_MAX_INTERFACE_
IPV6ADDRESS_CHARS];
    unsigned int IPv6Prefix;
    unsigned int ReceivePriority;
    DWORD ReceiveIdealProcessor;
} RTFW_NETWORK_INTERFACE, * PRTFW_NETWORK_
INTERFACE;
```

RTX64 3.x API	RTX64 4.0
RTFW_RESTART_TYPE	
typedef enum _RTFW_RESTART_TYPE(RTRT_NONE, RTRT_NETWORK = 5 RTRT_SUBSYSTEM = 10, RTRT_MACHINE = 20, } RTFW_RESTART_TYPE;	typedef enum _RTFW_RESTART_TYPE(RTRT_NONE, RTRT_TCPIP = 5 RTRT_NETWORK = 7 RTRT_SUBSYSTEM = 10, RTRT_MACHINE = 20, } RTFW_RESTART_TYPE;
RTFW_SCHEDULED_PROCESS	
typedef struct _RTFW_SCHEDULED_PROCESS(size_t Size CHAR Filename[MAX_PATH], CHAR Parameters[MAX_PATH], bool UseLocalMemory, unsigned int IdealProcessor, uint64_t AffinityMask, RTFW_SCHEDULED_ID ScheduledID, } RTFW_SCHEDULED_PROCESS, * PRTFW_SCHEDULED_ PROCESS;	typedef struct _RTFW_SCHEDULED_PROCESS(size_t Size CHAR Filename[MAX_PATH], CHAR Parameters[MAX_PATH], bool Reserved, unsigned int IdealProcessor, uint64_t AffinityMask, RTFW_SCHEDULED_ID ScheduledID, uint32_t MSpaceInitialSize, uint32_t MSpaceExpandSize } RTFW_SCHEDULED_PROCESS, * PRTFW_SCHEDULED_ PROCESS;

Removed Native Framework APIs

This section lists the RTX64 3.x APIs that were removed from RTX64 4.0.

RTX64 3.x API	RTX64 4.0
RtfwGetNetworkConfiguration	Removed. Use these new APIs: • **RtfwGetNALConfiguration** returns the configuration of the Network Abstraction Layer (NAL). • **RtfwGetTCPIPConfiguration** returns the configuration of the RT-TCP/IP Stack.
RtfwSetNetworkConfiguration	Removed. Use these new APIs: • **RtfwSetNALConfiguration** configures the Network Abstraction Layer (NAL). • **RtfwSetTCPIPConfiguration** configures the RT-TCP/IP Stack.
RTFW_NETWORK_CONFIGURATION	Removed. Use these new APIs: • **RTFW_NAL_CONFIGURATION** represents the configuration of the Network Abstraction Layer (NAL). • **RTFW_TCPIP_CONFIGURATION** represents the configuration of the RT-TCP/IP Stack.
RTFW_SUBSYSTEM_STATUS	Removed. Use **RTFW_COMPONENT_STATUS**, which represents the status of an RTX64 component.

Managed Framework APIs

This section outlines the changes to Managed Framework APIs in RTX64 4.0.

Enhanced Managed Framework APIs

This section lists the APIs included in the RTX64 3.x SDK that were enhanced with non-breaking changes in RTX64 4.0.

RTX64 3.x API	RTX64 4.0
IntervalZero.RTX64.Config.ScheduledProcess class	Added new properties: • *MSpaceInitialSize* gets/sets the size, in kilobytes, by which to allocate Non-Paged memory from Windows into the Local Pool of the MSpace at process startup. • *MSpaceExpandSize* gets/sets the size, in kilobytes, by which to expand an RTSS processes' internal/external MSpaces.

RTX64 3.x API	RTX64 4.0
IntervalZero.RTX64.Config.Subsystem class	Added new local memory configuration properties: • *EnableLocalMemory* • *NALExtMSpacePoolMinimumSize* • *NALMSpacesPoolExpandable* • *NALMSpacesPoolExpandSize* • *ProcessExtMSpacePoolMinimumSize* • *ProcessMSpacesPoolExpandable* • *ProcessMSpacesPoolExpandSize* • *ProcessMSpacesPoolShrinkable* • *SystemExtMSpacePoolMinimumSize* • *SystemExtMSpacePoolCommit* • *SystemMSpacesPoolExpandable* • *SystemMSpacesPoolExpandSize* • *SystemMSpacesPoolShrinkable* • *TCPIPExtMSpacePoolMinimumSize* • *TCPIPMSpacesPoolExpandable* • *TCPIPMSpacesPoolExpandSize* • *ZeroMemoryAtAllocation* Added a new method: • *SetHALTimerPeriod*

RTX64 3.x API	RTX64 4.0
IntervalZero.RTX64.RTAPI.RtssEnvironment class	Added new methods: • Method *GetProcessMSpace* retrieves a descriptor of the external and internal memory allocation space (MSpace) for the specified RTSS process or system process. • Method *QueryProcessMSpace* queries memory allocation space information for the specified process MSpace, including various summary statistics of its local pool and pool cache. • Structure MSPACE_INFO contains information for a single memory allocation space (MSpace).

Breaking Changes to Managed Framework APIs

This section lists the APIs included in the RTX64 3.x SDK that underwent breaking changes in RTX64 4.0.

RTX64 3.x API	RTX64 4.0

Subsystem.AddScheduledProcess Method (IntervalZero.RTX64.Config Namespace)
`public void AddScheduledProcess(`
`  string fileName,`
`  string parameters,`
`  uint idealProcessor,`
`  ulong affinityMask,`
`  bool useLocalPool,`
`  int order = -1`
`);`
	`void AddScheduledProcess(`
	`  string fileName,`
	`  string parameters,`
	`  uint idealProcessor,`
	`  ulong affinityMask,`
	`  uint MSpaceInitialSize,`
	`  uint MSpaceExpandSize,`
	`  int order = -1`
	`);`

Removed Managed Framework APIs

This section lists the RTX64 3.x APIs that were either removed from or are deprecated in RTX64 4.0.

RTX64 3.x API	RTX64 4.0
IntervalZero.RTX64.RTAPI.IO.RTInterruptHandler class	Removed.
IntervalZero.RTX64.RTAPI.RTShutdownHandler class	Removed.
IntervalZero.RTX64.RTAPI.Threading.RTTimer class	Removed.
IntervalZero.RTX64.Config.IConfigNetworkInterface interface	These properties were removed: • *LinkStatusPriority* • *LinkStatusIdealProcessor* • *UseLocalMemory*
IntervalZero.RTX64.Config.IConfigSubsystem interface	These properties were removed: • *DisableLocalMemoryExpansion* (use new properties *ProcessMSpacesPoolExpandable* and *SystemMSpacePoolExpandable*) • *DisableLocalMemoryShrink* (use new properties *ProcessMSpacesPoolShrinkable* and *SystemMSpacePoolShrinkable*) • *LocalMemoryPoolExpansionSize* (use new properties *ProcessMSpacesPoolExpandSize* and *SystemMSpacePoolExpandSize*) • *LocalMemoryPoolInitialSize* (use new property *SystemExtMSpacePoolCommit*) • *UseLocalMemory* (use new property *EnableLocalMemory*)

RTX64 3.x API	RTX64 4.0
IntervalZero.RTX64.Config.Subsystem class	These properties were removed: • *DisableLocalMemoryExpansion* (use new properties *ProcessMSpacesPoolExpandable* and *SystemMSpacePoolExpandable*) • *DisableLocalMemoryShrink* (use new properties *ProcessMSpacesPoolShrinkable* and *SystemMSpacePoolShrinkable*) • *LocalMemoryPoolExpansionSize* (use new properties *ProcessMSpacesPoolExpandSize* and *SystemMSpacePoolExpandSize*) • *LocalMemoryPoolInitialSize* (use new property *SystemExtMSpacePoolCommit*) • *UseLocalMemory* (use new property *EnableLocalMemory*)
IntervalZero.RTX64.Monitor.MonitorEventTLSFree class	These properties were removed: • *Status* • *Error*

RTX64 3.x API	RTX64 4.0
IntervalZero.RTX64.RTAPI.RtssEnvironment class	These properties were removed: • *TotalMemory* • *AvailableMemory* • *ContiguousMemory* These methods were removed: • *EnableInterrupts* • *DisableInterrupts* • *AddShutdownHandler* • *RemoveShutdownHandler*
IntervalZero.RTX64.RTAPI.IO namespace	These enumerations were removed: • *InterruptDisposition* • *InterruptMode*
IntervalZero.RTX64.RTAPI.Runtime.InteropServices namespace	These methods were removed: • *RTHandle.Alloc* • *RTMarshal.PhysicalAddressFromVirtualAddress* • *RTMarshal.AllocHLocal* • *RTMarshal.AllocHContiguous*

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 4.0 Runtime > Control Panel.
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