Porting from RTX64 to MaxRT eRTOS

USER GUIDE

IntervalZero

MaxRT eRTOS

Porting from RTX64 to MaxRT eRTOS

IZ-DOC-MaxRT-eRTOS-0004

IntervalZero

200 Fifth Avenue, FL 6, STE 6020
Waltham, MA 02451
Phone: 781-996-4481
www.intervalzero.com

Contents

About this Porting Guide

This guide describes the steps to port an existing RTX64 application to MaxRT eRTOS. It also highlights system differences between the two products and issues to look for during migration.

This guide contains active links to the IntervalZero website and product documentation. Therefore, we recommend you have an Internet connection and an installation of MaxRT eRTOS.

> **Note:** This guide assumes you have installed the eRTOS Runtime. For more installation information, see the *MaxRT eRTOS Runtime Installation Guide*.

1

Product Comparison

eRTOS is a 64-bit standalone real-time operating system designed to run 64-bit binaries independently. In contrast, RTX64 is a 64-bit real-time extension for Windows, enabling real-time applications to run alongside Windows applications.

Unlike RTX64, eRTOS operates without Windows, providing benefits such as independence from periodic Windows updates, reduced hardware requirements (e.g., no need for a powerful GPU or large amounts of RAM), and full access to CPU cores without OS overhead. While some features available in RTX64 may not be present in eRTOS or have been re-architected, this design ensures precise timing and deterministic responsiveness without OS-related dependencies.

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

Available Product Packages

The RTX64 4.x and eRTOS 1.x product packages include some functionality differences:

RTX64 4.x	eRTOS 1.x

• Runtime
  ◦ Includes the Network Abstraction Layer (NAL)
  ◦ Includes the Treck RT-TCP/IP Stack (always installed; must be purchased separately)
• SDK
• Runtime Merge Modules
• RTX64 Vision (standalone add-on)
	• Runtime
	  ◦ Includes the Network Link Layer (NL2)
	  ◦ Includes the Treck TCP/IP Stack (must be purchased separately)
	  ◦ Includes the MCCI USB Stack
	• SDK

Resource Partitioning

Target Platform

eRTOS and RTX64 handle resource partition differently to ensure real-time processes receive the necessary resources while minimizing interference from non-real-time tasks.

- Processor Allocation – eRTOS uses all processors enumerated by UEFI/BIOS, while RTX64 requires a system with at least two logical processors/cores, with one core dedicated to RTSS.
- Memory and I/O Access – eRTOS uses all available system RAM and I/O devices.
- Storage Support – eRTOS supports only SATA drives operating in AHCI mode.
- File System Requirement – eRTOS requires the FAT32 file system.

Processor Enumerations

eRTOS and RTX64 handle processor enumeration differently. eRTOS utilizes all processors enumerated by UEFI/BIOS, while RTX64 follows a different enumeration method that may result in shared cache between Windows and RTX64 processors.

2

Activation, Licensing, and Configuration

eRTOS activation and configuration is done through the MaxRTActivationUtil.exe command line utility, which activates and locks your SDK products to a specific machine or IntervalZero-provided dongle. RTX64 activation and configuration is done through the Activation and Configuration UI utility and command line utility.

For more details about the MaxRT activation command line utility, please see Using MaxRTActivationUtil.exe in eRTOS Help.

Both RTX64 and eRTOS SDK components must be activated with a valid license before they can be used.

Activation

- Activation Key – A string used to generate a license that locks your product to a specific machine or an IntervalZero-provided dongle.
- License – A signed string generated by IntervalZero or from an activation key that allows you to access product features.
- License File – A file containing licenses for all activated product features.
- Customer ID (CID) – A unique identifier assigned to all IntervalZero customers. Under certain conditions, it is used to bind Runtime and SDK licenses.
- Node-locked – A license type that restricts usage to a specific system or with a single dongle

Below is a list of important differences between RTX64 and eRTOS related to component activation:

RTX64 4.x	eRTOS 1.x
• Activation and Configuration utility or command line utility Rtx64ActivationUtil.exe available in the RTX64 SDK and Runtime • Licenses are node-locked for Runtime and SDK. • License Server • Fingerprint File • IntervalZero Dongles • Standalone IntervalZero Dongle Activation Utility	• MaxRTActivationUtil.exe activation command utility available in the eRTOS SDK. • SDK licenses are node-locked. • Runtime licenses are not node-locked but associated with an SDK through a Customer ID (CID). • License Server (SDK only) • Fingerprint File (SDK only) • IntervalZero Dongles (SDK only) • Standalone IntervalZero Dongle Activation Utility (SDK only)

Licensing

eRTOS Runtime Licensing

To license the eRTOS Runtime, complete the following steps:

1. Log in to your IntervalZero Customer Center account and download your eRTOS Runtime License.
2. Place the downloaded license file, `eRTOS.lic`, in the eRTOS Runtime installation directory. By default, this path is `<eRTOS Volume Letter>:\MaxRT\eRTOS`.
3. For more information on eRTOS Runtime licensing, refer to the *eRTOS Help* and *eRTOS Runtime Installation Guide*.

eRTOS SDK Licensing

To license the eRTOS SDK, you can use the MaxRT activation command line utility, `MaxRTActivationUtil.exe`. This utility requires a valid activation key, which you can find in the email you received from IntervalZero Sales when you purchased eRTOS or by viewing your orders in the IntervalZero Customer Center. You can activate eRTOS SDK to a machine or IntervalZero-provided dongle. To activate with the eRTOS activation command line utility, do the following:

1. Launch a **Windows Command Prompt** as Administrator.
2. Type *MaxRTActivationUtil.exe -a <activation key>*.
3. Press **Enter**.

For more information on eRTOS SDK Licensing, refer to the *eRTOS Help* and *eRTOS SDK Installation Guide*.

Configuration

eRTOS does not use a Control Panel to configure the components like RTX64. eRTOS allows configuring its components, Real-time HAL, and Real-time Kernel through editing the following files:

- To configure and manage the network (NL2, and TCP/IP Stack), network interfaces, and USB stack, edit `RtConfig.rtreg` file.
- To configure the Real-time HAL, edit the `grub.cfg` file.
- To configure the Real-time Kernel, edit the `RtKrnlConfig.ini` file.

3

Tools

RTX64 provides a Control Panel to configure its Runtime components. eRTOS provides the following tools and utilities to configure different settings:

- eRTOS Console - Allows you to run commands and view application output using a USB keyboard. It functions similarly to the Windows Command Prompt and the RTX64 Console.
- eRTOS Visual Studio Remote Debug Monitor (`RtVSMon.ertos`) - Enables debugging of programs running on a different eRTOS target from a system with the eRTOS SDK and Visual Studio installed. In RTX64, debugging is handled by `VSMon.exe`, a Visual Studio executable.
- MaxRTActivationUtil.exe - A command-line utility used for activating the eRTOS SDK. .In RTX64, activation is handled through the Activation and Configuration Utility or via the command-line tool `Rtx64ActivationUtil.exe`.
- Network Utilities - eRTOS includes the same set of TCP/IP-based network utilities as RTX64 — **RtArp**, **RtRoute**, **RtIpConfig**, and **RtPing**, providing tools for diagnosing and configuring network connections.
- Run - Used to run an eRTOS application through the eRTOS Console or through the application batch file (`AutoStart.bat`). In RTX64, the RtssRun utility is used to run a RTSS application from the Windows command prompt.
- Kill - Used to view or terminate a specific eRTOS process. In RTX64, the RtssKill utility is used to forcibly terminate RTSS processes.
- System Response Time Measurement (SRTM) - A real-time API (RTAPI) timer latency measurement tool that measures timer latency observed by an application. This tool is available in both RTX64 and eRTOS.

Tool Sets

Below is a list of tool sets provided in RTX64 and eRTOS:

RTX64 4.x	eRTOS 1.x
• RtssRun	• Run
• RtssKill	• Kill
• RTX64 Activation and Configuration	• MaxRTActivationUtil.exe
• RTX64 MSpaces	• eRTOS MSpaces
• RTX64 Objects	• eRTOS Objects
• RTX64 Server	• eRTOS Console
• StampTool(SDK)	• eRTOS StampTool (SDK)
• System Response Timer Measurement	• System Response Time Measurement (SRTM)
• Network Utilities - **RtArp**, **RtIpConfig**, **RtPing**, and **RtRoute**	• Network Utilities - **RtArp**, **RtIpConfig**, **RtPing**, and **RtRoute**
• PCI Scan bus	• PCI Scan bus
• Microsoft Remote Debugging Monitor	• eRTOS Visual Studio Remote Debug Monitor (RtVSMon.ertos)
• RTX64 Control Panel	• Configuration files - `RtConfig.rtreg`, `grub.cfg`, and `RtKrnlConfig.ini`
• RTX64 Analyzer	
• RTX64 Latency View	
• RTX64 Monitor	
• RTX64 System Tray	
• RTX64 Task Manager	
• RTX64RemoteConfig(SDK)	
• KSRTM	

4

Subsystem

eRTOS is a standalone real-time operating system (RTOS), whereas RTX64 functions as a real-time extension for Windows. Although the RTX64 subsystem and the eRTOS kernel share the same underlying real-time logic, eRTOS offers additional low-level functionality that RTX64 instead delegates to Windows. Since eRTOS operates independently of Windows, several key differences exist, as outlined below.

No Windows Integration

Unlike RTX64, which runs real-time applications alongside Windows, eRTOS operates independently. As a result:

- No Windows Applications – Windows-based applications cannot be linked to eRTOS.
- No Windows Kernel Drivers – Kernel drivers designed for Windows cannot be linked or loaded in eRTOS.
- No Device Conversion to RTSS – In RTX64, devices must be converted for RTX64 use to prevent Windows from using them. In eRTOS, this is not necessary as the devices are accessible by default.
- No PnP INF/CAT and Driver Signing – Windows-style driver installation, including Plug and Play (PnP) INF files, catalog (CAT) signing, and certification, is not required or supported. Devices are readily accessible in eRTOS.

System Behavior Differences

- No Windows Shutdown Handling – eRTOS does not support the Windows-style shutdown process.
- No Blue Screen of Death (BSOD) – Since eRTOS is independent of Windows, system crashes do not result in a BSOD. Consequently, no crash dump files are generated, and debugging via WinDBG is not applicable.

HAL and Subsystem Differences

RTX64 relies on the Windows Hardware Abstraction Layer (HAL) for core system management, including CPU core allocation, cache control, interrupt handling, timer management, and other hardware operations. In contrast, eRTOS operates independently of any host OS, directly initializing and controlling hardware at the bare-metal level. eRTOS and RTX64 use different system components for HAL and subsystem management.

eRTOS and RTX64 use different system components for Hardware Abstraction Layer (HAL) and subsystem management:

Component	**RTX64 4.x**	**eRTOS 1.x**
File System	Uses native Windows File system (NTFS)	Implements lightweight FAT32 file system
Registry	Integrates with Windows Registry	Provides simplified custom registry structure
Memory	Allocates from Windows non-page memory pool	Manages memory independently with custom allocator
Hardware	Relies on Windows Hardware abstraction layer and drivers	Direct bare-metal hardware control

5

Application Organization

Executable and Library File Extensions

Real-time applications and dynamic link libraries have different file extensions in RTX64 and eRTOS:

	eRTOS	RTX64
Real-Time Application	`.ertos`	`.rtss`
Real-Time DLL	`.edll`	`.rtdll`

Thread Stack Size

The default thread stack size for eRTOS is 0x10000 which is bigger than RTX64 due to WolfSSL needs.

Memory Allocation

In RTX64, memory is allocated through multiple spaces (MSpaces), separating internal bookkeeping from process allocations. Each process has its own allocation space, and shared memory or IPC objects are handled by the Subsystem. In contrast, eRTOS uses a simpler local memory allocation approach.

6

Project Settings and Configurations

Like RTX64, eRTOS has application templates that can be used to create a real-time application executable or real-time DLL.

Configurations

The eRTOS application templates provide two default configurations that build 64-bit applications using the Microsoft 64-bit compiler or the Intel 16.0 x64 compiler:

- eRTOSDebug – eRTOS debug information (`.ertos` or `.edll`). The default stack size for this configuration with **C/C++ Runtime support** is 32 pages (131072 bytes).
- eRTOSRelease – eRTOS application or DLL(`.ertos` or `.edll`). The default Stack size for this configuration is 12 pages (65536 bytes).

Headers

Like RTX64, in eRTOS the RtApi header file has been divided into two header files:

- `RtApi.h` – this header file is needed by applications linked to eRTOS. In RTX64, it contains functionality available to both RTSS and RTX64-enabled Windows applications.
- `RtssApi.h` – this header file is only needed by the process applications. In RTX64, it contains functionality available for RTSS applications.

Because the NAL is now changed to NL2 in eRTOS, a few changes are also made for the header files:

- `Rtnl2Api.h` - this header file is to needed to use NL2. In RTX64, it uses `RtNalApi.h` for the NAL.
- `Rtnd.h` - this header file needs to be included by NL2 network driver sources.

To use NL2 and TCP/IP API calls in eRTOS, you need to include `Rtnl2Api.h` and `RttcpipApi.h`.

Libraries

In eRTOS, most of the libraries remain the same as in RTX64. Below highlights the changes in eRTOS:

- To use real-time processes (`.ertos`) within eRTOS, you will need to include `Rtkrnl.lib` instead of `Rtx_rtss.lib`, which is used in RTX64.
- To use TCP/IP you will need to include `Rttcpip.lib` as in RTX64.
- To use the eRTOS Network Link Layer (NL2), you need to include `Rtnl2Api.lib`. The NL2 is not available in RTX64, which utilizes the Network Abstraction Layer (NAL) using `RtNal.lib`.

7

Code Changes

This topic outlines the APIs included in RTX64 4.x SDKs that were enhanced, changed, or removed in eRTOS 1.x SDKs.

System affinity mask

- RTX64 – Processor numbering starts from the first RTSS processor to the last RTSS processor. The first RTSS processor number is equal to the last Windows processor number plus 1. The first RTSS processor number will never be 0.
- eRTOS – Processor numbering starts from the first processor to the last processor. The first processor number always being 0.

Removed / Deprecated APIs

This section lists RTX64 4.x APIs that were either removed from or are deprecated in eRTOS 1.x SDKs.

Removed / Deprecated Real-Time APIs

RTX64 4.x API	eRTOS 1.0 and later
RtGetCurrentHalTimerPeriod	Removed.
RtIsInRtss	Removed.
RtGetRuntimeVersionEx	Removed.
RtIsAppRunnableW	Removed.

RTX64 4.x API	eRTOS 1.0 and later
RtGetLicenses	Unsupported. Licensing APIs from RTX64/wRTOS are not supported in eRTOS.
RtIsRuntimeLicensed	Unsupported. Licensing APIs from RTX64/wRTOS are not supported in eRTOS,
RtIsTcpStackLicensed	Unsupported. Licensing APIs from RTX64/wRTOS are not supported in eRTOS,

Removed / Deprecated Windows-Supported APIs

eRTOS does not support Windows-managed code applications requiring interaction with the eRTOS Runtime. Additionally, it does not provide support for Windows driver inter-process communication (RTKAPI) functions.

RTX64 4.x API	eRTOS 1.0 and later
RegEnumKeyExA	Unsupported.
RegEnumKeyExW	Unsupported.
RegEnumKeyW	Unsupported.

8

Basic Networking

eRTOS utilizes the Network Link Layer (NL2) for processing and networking, while RTX64 utilizes the Network Abstraction Layer (NAL). Additionally, eRTOS manages networking, including NL2 and the TCP/IP stack, through the `RtConfig.rtreg` file, whereas RTX64 relies on its Control Panel for configuration.

Network Link Layer (NL2)

The RTX64 Runtime includes a Network Abstraction layer (NAL), while the eRTOS Runtime includes a completely redesigned Network Link Layer (NL2) 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 layer from the TCP/IP Stack. Using the NL2, you can easily take advantage of network functionality such as EtherCAT, TSN (Time Sensitive Networks), and PTP (Precision Time Protocol).

The NL2 supplies a simple API that 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.

Below are the major differences of NL2 from eRTOS:

- Improved support for hardware timestamping (simpler APIs, and possibility to get notified when an egress timestamp is available)
- Improved support for link status monitoring (no need to acquire a queue to monitor the link status)
- Improved support for MSI-X multi vector (users can now statically map queue interrupt sources to MSI-X Messages)
- Simplified driver development (a lot of code that drivers had to implement themselves is now put in common inside the NL2)

- Possibility to use a hardware queue in shared access mode (multiple applications can transmit/receive through it at the same time)
- Improved latency performances for transmit/receive operations on a queue used in exclusive access mode

Porting a NAL Driver to the NL2 Driver

This topic describes the steps required to port an RTX64 NAL NIC driver to an eRTOS 1.0 NL2 NIC driver that can be used by the eRTOS Network Link Layer (NL2).

For information on NL2 NIC drivers, see the Help topic *Creating a NIC Driver*.

TOPICS IN THIS SECTION:

Header and Library Requirements

To integrate your driver with the NL2, include the `Rtnd.h` header file. To prevent errors, do not use `Rtnl2Api.h`.

If your driver uses only a limited subset of the C standard library (e.g., **memset**, **memcpy**, **memcmp**, **strlen**, **strcmp**, **strcpy**, **strncpy**, **strcat**, **itoa**), you can replace the Microsoft C runtime libraries with the `startupDll.lib` and `RtCruntime.lib` libraries.

Additionally, remove the following header files and library from your driver if they were previously referenced:

- Headers: `rtNalApi.h`, `rtnapi.h`, `winsock2.h`, `ws2tcpip.h`
- Library: `RTX64Nal.lib`

Rework the Startup Logic

During its startup phase, the RTX64 NAL loads the NIC drivers and calls the following functions for each enabled NIC:

- RtndInitDriverializeInterface
- RtndConfigure
- RtndUpDown

In contrast, the eRTOS NL2 calls these functions:

- RtndInitDriver
- RtndManageInterface
- RtndQueryInterfaceCapability
- RtndSetInterface
- RtndStartInterface
- RtndQueryMacAddress
- RtndQueryInterfaceFeature

Rework the Shutdown Logic

During its shutdown phase, the RTX64 NAL calls this function:

- RtndUpDown

In contrast, the eRTOS NL2 calls these functions:

- RtndShutdownInterface
- RtndStopInterface
- RtndUnmanageInterface

Use NL2 Function Pointers instead of Direct Function Calls

RTX64 NAL NIC drivers link with `RTX64Nal.lib` and request NAL services by calling the following functions exported by the NAL process:

- RtnEnumPciCards
- RtnGetDeviceName
- RtnSetLinkAddress
- RtnGetMcastCount
- RtnIndicateStatus
- RtnNotifyTransmitQueue
- RtnTransmitCompleteCallback
- RtnNotifyRecvQueue
- RtnSetDataLong
- RtnGetDataLong
- RtnGetPacket
- RtnDecodePacket
- RtnIntializeCriticalLock
- RtnDeleteCriticalLock
- RtnEnterCriticalLock
- RtnLeaveCriticalLock

In contrast, eRTOS NL2 NIC drivers request NL2 services using the function pointers supplied by the NL2 in the call to RtndInitDriver:

- GetVerbose
- NotifyLinkStatusChange
- NotifyTxInterrupt
- NotifyRxInterrupt
- NotifyEgressTimestamp
- CreateTxBuffers
- DestroyTxBuffers
- CreateRxBuffers
- DestroyRxBuffers

Rework the Interrupts Handling

Interrupts Enabling

RTX64 NAL NIC drivers are expected to enable the following interrupts at startup and keep them enabled until the NAL shuts down:

- Link Status Change interrupt
- Transmit interrupt
- Receive interrupt

eRTOS NL2 NIC drivers are expected to enable the following interrupts at startup:

- Link Status Change interrupt
- Egress Timestamp interrupt (if the hardware supports Egress Timestamping)

The NL2 determines whether Transmit and Receive interrupts should be enabled by calling the following functions at runtime:

- RtndEnableTxInterruptSource
- RtndEnableRxInterruptSource

Interrupts Servicing

In RTX64 NAL NIC drivers, an IST typically does the following work:

- In case of a Link Status Change interrupt:
  - Set an event to wake up the Line Status thread
- In case of a Transmit interrupt:
  - Call **RtnNotifyTransmitQueue** (which wakes up the application's Transmit Complete thread) or set an event to wake up the driver's Transmit Complete thread
- In case of a Receive interrupt:
  - If the queue is attached to an application, update the count of received frames and call **RtnNotifyRecvQueue** (which signals the application's Receive event), otherwise, flush the queue (extract the filled buffers and re-submit them immediately)

Note: NAL NIC drivers ignore Egress Timestamp interrupts.

In eRTOS NL2 NIC drivers, an IST should typically do the following work:

- In case of a Link Status Change interrupt:
  - Call **RTND_CALLBACKS.NotifyLinkStatusChange**
- In case of a Transmit interrupt:
  - Call **RTND_CALLBACKS.NotifyTxInterrupt**
- In case of a Receive interrupt:
  - Call **RTND_CALLBACKS.NotifyRxInterrupt**
- In case of an Egress Timestamp interrupt:
  - Call **RTND_CALLBACKS.NotifyEgressTimestamp**

Note: NL2 drivers should not access the DMA rings or use a lock in their ISTs.

Rework the Link Status Monitoring Logic

RTX64 NAL NIC drivers have their Line Status Monitoring thread, which awakes every time a Link Status Change interrupt occurs. The thread performs the following tasks:

- Reads the new link status from the NIC registers
- Reconfigures some of the NIC registers, if required
- Calls **RtnIndicateStatus**

eRTOS NL2 NIC drivers don't need to have their Line Status Monitoring thread because the NL2 itself implements the monitoring logic. NL2 NIC drivers only need to provide the RtndQueryLinkStatus function, which should do the following:

- Read the new link status from the NIC registers
- Reconfigure some of the NIC registers, if required
- Return the new link status to the NL2

Rework the Frame Transmission Logic

About Frame Transmission

In the RTX64 NAL, the general rule is that frame transmission is serviced by a dedicated Transmit Complete thread. Depending on the application's request, this thread can run within the context of the application process or the context of the driver (i.e. the context of the NAL process). When running in the application context, the Transmit Complete thread calls a callback provided by the application every time a buffer in the queue is consumed. When running in the context of the driver, the Transmit Complete thread simply extracts the consumed buffer and makes it available for future transmission.

In the eRTOS NL2, frame transmission is always serviced at the initiative of the NL2, which calls RtndExtractTxBuffer.

Attach Transmit Queues

When an application acquires a Transmit Queue and enables the Transmit Complete callback functionality, the RTX64 NAL calls the following function of the NIC driver:

- RtndAttachToTransmitQueue

This function notifies the driver that Transmit interrupts for this queue should be serviced by the application's Transmit Complete thread rather than the driver's Transmit Complete thread.

For eRTOS NL2 NIC drivers, the equivalent function is RtndAttachTxQueue, which is called from the context of any process that plans to use this queue.

Submit Buffers

When the application provides a buffer containing a frame to transmit, the RTX64 NAL calls the following functions of the NIC driver:

- RtndTransmit
- RtndTransmitEx

In contrast, the eRTOS NL2 calls these functions:

- RtndSubmitTxBuffer
- RtndApplyTxBuffers

Extract Buffers

When a Transmit Queue is acquired by an application that enabled the Transmit Complete Callback functionality, the RTX64 NAL calls the following function from the application's Transmit Complete thread to extract the consumed buffers:

- RtndServiceTransmitQueue

In other cases, the NAL driver is responsible for automatically extracting the consumed buffers using its Transmit Complete thread.

In contrast, the extraction of consumed buffers by the eRTOS NL2 NIC drivers is always done upon request from the NL2 using the following function:

- RtndExtractTxBuffer

Detach Transmit Queues

When an application releases a Transmit Queue that has the Transmit Complete callback functionality enabled, the RTX64 NAL calls the following function of the NIC driver:

- RtndDetachTransmitQueue

This function notifies the driver that Transmit interrupts for this queue should now be serviced by the driver's Transmit Complete thread rather than the application's Transmit Complete thread.

For eRTOS NL2 NIC drivers, the equivalent function is RtndDetachTxQueue, which is called from the context of any process that previously called RtndAttachTxQueue.

Rework the Frame Reception Logic

About Frame Reception

In the RTX64 NAL, frame reception is serviced either by the application that acquired the queue or by the driver's IST when no application has acquired the queue.

In the eRTOS NL2, frame reception is always serviced at the initiative of the NL2, which calls RtndExtractRxBuffer.

Start/Stop the Queue

This is an optional feature that doesn't exist in RTX64 NAL NIC drivers.

If a NIC supports dynamically starting/stopping a Receive Queue independently of the other Receive Queues, the NL2 NIC driver should implement the following functions:

- RtndStartRxQueue
- RtndStopRxQueue

The eRTOS NL2 will call these functions to ensure that a Receive Queue is in the same initial state each time a new application acquires it.

Attach Receive Queues

When an application acquires a Receive Queue, the RTX64 NAL calls the following function of the NIC driver:

- RtndAttachToReceiveQueue

This function notifies the driver that Receive interrupts for this queue must be serviced by the application rather than the driver's IST.

For eRTOS NL2 NIC drivers, the equivalent function is RtndAttachRxQueue, which is called from the context of any process that plans to use this queue.

Extract Buffers

When an application acquires a Receive Queue, the rTX64 NAL calls the following functions from the application's process to extract the filled buffers:

- RtndReceive
- RtndReceiveWithCallback

In other cases, the NAL driver is responsible for automatically extracting the filled buffers in the IST and re-submitting them immediately to the queue.

For eRTOS NL2 NIC drivers, buffer extraction is always done upon request from the NL2 using the following function:

- RtndExtractRxBuffer

Submit Buffers

RTX64 NAL NIC drivers automatically re-submit the extracted buffers after they have been processed (either after they have been copied to the application's buffer by `RtndReceive` or after they have been supplied to the application through the Receive Callback function).

In contrast, eRTOS NL2 NIC drivers do not re-submit automatically; they wait for the NL2 to call the following functions:

- RtndSubmitRxBuffer
- RtndApplyRxBuffers

Detach Receive Queues

When an application releases a Receive Queue, the RTX64 NAL calls the following function of the NIC driver:

- RtndDetachReceiveQueue

This function notifies the driver that the driver's IST should now service Receive interrupts for this queue rather than the application.

For eRTOS NL2 NIC drivers, the equivalent function is RtndDetachRxQueue, which is called from the context of any process that previously called RtndAttachRxQueue.

Remove Unnecessary Locks

RTX64 NAL NIC drivers generally need to use their locks for at least the following purposes:

- One lock per Transmit Queue, used by the driver’s Transmit Complete thread and by the following functions to protect concurrent access to the queue registers: **RtndAttachToTransmitQueue**, **RtndDetachTransmitQueue**, **RtndTransmit**, **RtndTransmitEx**, **RtndServiceTransmitQueue**.
- One lock per Receive Queue, used by the IST and by the following functions to protect concurrent access to the queue registers: **RtndAttachToReceiveQueue**, **RtndDetachReceiveQueue**, **RtndReceive**, **RtndReceiveWithCallback**, **RtndIoctl**.
- One general lock, used by the following function to protect concurrent access to the general NIC registers: **RtndIoctl**.

The eRTOS NL2 implements its locks and ensures that specific functions are called from within a single thread only. Therefore, NL2 NIC drivers don't require additional locks.

Replace the RtndIoctl Function

The RTX64 NAL calls the **RtndIoctl** function for different purposes specified by the command code. The table below lists the equivalent functions of the eRTOS NL2 NIC driver that provide the same services:

RtndIoctl command code	**NL2 NIC driver equivalent function**
ENIOCPROMISC	RtndSetPromiscuousMode
ENIOCALL, ENIOCNORMAL	RtndSetPassBadFramesMode
ENIOADDMULTI, ENIODELMULTI	RtndSetMulticastFilter
ENIOLINKSTATUS	RtndQueryLinkStatus
RTNAL_IOCTL_RX_MONITOR	There is no equivalent; this functionality doesn't exist in the NL2.
RTNAL_IOCTL_TX_MONITOR	There is no equivalent; this functionality doesn't exist in the NL2.
RTNAL_IOCTL_READ_TX_HW_TIMESTAMP	RtndExtractLastTxTimestamp
RTNAL_IOCTL_READ_RX_HW_TIMESTAMP	There is no equivalent; this functionality doesn't exist in the NL2.
RTNAL_IOCTL_READ_HW_TIMER	There is no equivalent; this functionality doesn't exist in the NL2.
RTNAL_IOCTL_READ32_REG	There is no equivalent; this functionality doesn't exist in the NL2.
RTNAL_IOCTL_WRITE32_REG	There is no equivalent; this functionality doesn't exist in the NL2.

RtndIoctl command code	NL2 NIC driver equivalent function
RTNAL_IOCTL_READ_RX_PACKET_COUNT	There is no equivalent; this functionality doesn't exist in the NL2.
RTNAL_IOCTL_READ_TX_PACKET_COUNT	There is no equivalent; this functionality doesn't exist in the NL2.
RTNAL_IOCTL_USE_RX_NOTIFICATIONS	There is no equivalent; this functionality doesn't exist in the NL2.
RTNAL_IOCTL_SET_RX_POLLING	There is no equivalent; this functionality doesn't exist in the NL2.
RTNAL_IOCTL_CLEAR_RX_POLLING	There is no equivalent; this functionality doesn't exist in the NL2.
RTNAL_IOCTL_GET_RX_ETHERTYPE_FILTER	RtndGetDispatcherEtherTypeEntry
RTNAL_IOCTL_SET_RX_ETHERTYPE_FILTER, RTNAL_IOCTL_CLEAR_RX_ETHERTYPE_FILTER	RtndSetDispatcherEtherTypeEntry
RTNAL_IOCTL_SET_RX_MESSAGE_TYPE_TO_TIMESTAMP	There is no equivalent; this functionality doesn't exist in the NL2.
RTNAL_IOCTL_SET_TX_MESSAGE_TYPE_TO_TIMESTAMP	There is no equivalent; this functionality doesn't exist in the NL2.
RTNAL_IOCTL_GET_SYSTEM_TIMER	There is no equivalent; this functionality doesn't exist in the NL2.
RTNAL_IOCTL_SET_INTERRUPT_MODERATION	RtndSetInterruptModeration
RTNAL_IOCTL_GET_PCI_BUS_LOCATION	There is no equivalent; this functionality doesn't exist in the NL2.

Replace the RtndRequest function

The RTX64 NAL calls the `RtndRequest` function for different purposes as specified by the OID. The table below lists the equivalent functions of the eRTOS NL2 NIC driver that provide the same services:

RtndRequest OID	**NL2 NIC driver equivalent function**
RTND_OID_GEN_MAC_ADDRESS	RtndQueryMacAddress
RTND_OID_GEN_MEDIA_CONNECT_STATUS, RTND_OID_GEN_MEDIA_DUPLEX_STATE, RTND_OID_GEN_LINK_SPEED	RtndQueryLinkStatus
All other OIDs	There is no equivalent; these functionalities don’t exist in the NL2.

Implement Frame Buffer Allocation Functions

The following functions are specific to the NL2 and must be implemented only in NL2 NIC drivers. They allow the NL2 to allocate or release a batch of buffers used to hold Ethernet frames for a specific Transmit or Receive Queue. These functions are necessary because each NIC may have special requirements for allocating frame buffers, making it the driver's responsibility to handle these allocations.

The functions to implement are as follows:

- RtndAllocateTxFrameDataBuffers
- RtndFreeTxFrameDataBuffers
- RtndAllocateRxFrameDataBuffers
- RtndFreeRxFrameDataBuffers

TCP/IP Stack

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

The major difference of TCP/IP stack in eRTOS from RTX64 is the addition of APIs that allow changing the IP address at runtime.

Configuring Multiple IP Addresses in eRTOS

Unlike RTX64, eRTOS does not support pre-configuring multiple IP Addresses for a NIC in the registry file — only one can be pre-configured. To add additional IP Addresses, users must write an application that calls **RttcpipAddInterfaceAddress**. IP addresses can also be removed using **RttcpipDeleteInterfaceAddress**. These additional IP addresses must be reconfigured each time the eRTOS system starts with the TCP/IP stack enabled.

Support

For help with eRTOS, contact IntervalZero Technical Support by phone or access the online support resources available at https://www.intervalzero.com/en-support/en-customer-service/

Contacting Technical Support by Phone

Note: If you purchased an IntervalZero product through a third-party reseller, please contact the reseller for support.

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 this information ready when you contact IntervalZero Technical Support:

- Your Support ID

  Customers who purchase direct support receive an e-mail address and password for accessing the IntervalZero Customer Support Portal.

- Your eRTOS version number

Note: You must have a valid maintenance contract to receive product support.

Online Resources

Visit https://www.intervalzero.com/en-support/en-customer-service/ to log in to the Customer Support Portal (requires valid credentials), access online product Help, and view Support and Lifecycle policies and Product Release Notices.

Index