



INtime 7 Quick Start Guide

July, 2024

TenAsys Corporation
1400 NE Compton Drive, #301
Hillsboro, OR 97006 USA
+1 503 748-4720
fax +1 503 748-4730
info@tenasys.com
www.tenasys.com



This document is protected by US and international copyright laws.

TENASYS, INTIME and IRMX are registered trademarks of the TenAsys Corporation.

† Other companies, products, and brands mentioned herein may be trademarks of other owners.

Information regarding products other than those from TenAsys has been compiled from available manufacturers' material. TenAsys cannot be held responsible for inaccuracies in such material.

TenAsys makes no warranty for the correctness or for the use of this information and assumes no liability for direct or indirect damages of any kind arising from the information contained herewith, technical interpretation or technical explanations, for typographical or printing errors, or for any subsequent changes in this article.

TenAsys reserves the right to make changes to specifications and product descriptions at any time, without notice, and without incurring any liability. Contact your local TenAsys sales office or distributor to obtain the latest specifications and product descriptions.

Copyright © 2005–2024, TenAsys Corporation, All Rights Reserved

No part of this guide may be copied, duplicated, reprinted, and stored in a retrieval system by any means, mechanical or electronic, without the written permission of the copyright owner.

July, 2024 Edition

INtime SDK v7 Quick Start Guide

Contents

Welcome! – Before You Begin	5
Notational Conventions	6
Terms	7
Requirements	7
INtime SDK and INtime for Windows Requirements	7
INtime Distributed RTOS Deployment Host Requirements	8
SDK Installation	8
INtime Distributed RTOS Deployment Host Installation	12
Example #1: The INtime Application Wizard – HelloWorld	16
Using the INtime Application Wizard	16
Stop and start the application from Visual Studio	21
Introducing the INtime Explorer	23
Debugging HelloWorld with Visual Studio	25
Example #2: Working Together – Windows and Real-time	26
Two processes – one application	26
Creating the Real-Time Process	27
Creating the Windows Process	31
Create the Project and Setup the Environment	31
Creating a Graphical User Interface	33
Edit the Code	34
Running the Complete Solution	38
EXAMPLE #3 – Working with multiple INtime Nodes	40
Creating the RTSend application	41
Edit the code	41
Running the solution	43
Adding a second node	44
INtime for Windows: - Setting up a second Node	44
INtime Distributed RTOS: - Setting up a second Node	45

Modifying RTSend application for a second node	46
Running the complete solution with a second node	47
Example #4: The INscope Performance Analyzer	48
How Fast is Deterministic?	48
Fast Does Not Equal Deterministic	49
A Multi-threaded Example	49
Trace the Threads With INscope.....	53
6).... In a few moments the <i>View Trace</i> button appears, indicating that the trace buffer is full. Click <i>View Trace</i> . The event trace for <i>MultiThread</i> appears in the INscope upper-right pane.	54
Next Steps	58
A. Configuring the <i>INtime for Windows Kernel</i> (local Node).....	59
<i>INtime for Windows Node Management</i>	59
<i>INtime for Windows Device Manager</i>	62
B. <i>INtime for Windows Sample Applications</i>	63

Figures

Figure 1: INtime for Windows configuration:.....	6
Figure 2: INtime Distributed RTOS configuration:	6
Figure 3: Entering Network License FQDN or IP	10
Figure 4: Enter USB Drive letter.....	12
Figure 5: Create USB installer.	12
Figure 6: Boot status	15
Figure 7: DRTOS Web Configuration	15
Figure 8: Creating a new project.	16
Figure 9: Selecting template	17
Figure 10: Selecting name and location for solution	17
Figure 11: Creating a New INtime Project	18
Figure 12: Selecting Process Elements	18
Figure 13: Specifying Polling Thread Parameters	19
Figure 14: Wizard Summary Screen	19
Figure 15: Files Generated by the wizard.	20
Figure 16: Selecting INtime Project	21
Figure 17: Selecting an INtime Node from Visual Studio.....	22
Figure 18: HelloWorld Console Window	22
Figure 19: Configuring INtime Explorer Options	23
Figure 20: HelloWorld Console Window	24
Figure 21: Terminating the HelloWorld Process	24
Figure 22: Setting a Breakpoint.....	25
Figure 23: Basic INtime Solution Architecture	26
Figure 24: Data-flow.....	27
Figure 25: Selecting the MFC Application Template	31
Figure 26: Specifying the Project Name and Location.....	31
Figure 27: MFC Application Type Selections	32
Figure 28: Specifying Additional Include Directories	33
Figure 29: Dialog Editor in the Toolbox.....	33
Figure 30: NTXData Dialog Box.....	34
Figure 31: Running the Complete Solution	39
Figure 32: RTData process console output.	43
Figure 33: RTSend process console output.....	44
Figure 34: INtime Configuration Panel	44
Figure 35: INtime Node Management applet	45
Figure 36: NodeA and NodeB are shown as local nodes.	45
Figure 37: Configure Distributed RTOS	46
Figure 38: Distributed RTOS configuration interface	46
Figure 39: Selecting a Node within Visual Studio	47
Figure 40: Comparison of Real-time Systems	48
Figure 41: Modifying Thread Parameters	50
Figure 42: Modifying Thread Parameters	50

Figure 43: MultiThread Project Summary 51
Figure 44: MultiThread Application Output 53
Figure 45: INscope Event Trace 54
Figure 46: Zoomed INscope Trace 55
Figure 47: Event Based detail 56
Figure 48: INtEX View of the Multithread App 57
Figure 49: INtime Control Panel..... 59
Figure 50: Node Management Kernel Tab..... 60
Figure 51: Node Management System Tab 61
Figure 52: Device Configuration applet..... 62

Welcome! — Before You Begin

Thank you for your interest in our INtime® 7 SDK (Software Development Kit) supporting the INtime RTOS. The INtime 7 SDK supports the following usage configurations:

- *INtime for Windows*, where the INtime RTOS runs simultaneously alongside the Microsoft® Windows® operating system.
- *INtime Distributed RTOS*, where INtime runs as a stand-alone RTOS. Configurations run the same binary application and support multi-core implementations with one or more instance of the INtime RTOS running on the same host.

INtime for Windows offers a unique solution for developing embedded real-time applications for the Microsoft Windows platform. Your real-time application can run in conjunction with Windows applications or as two independent applications, running alongside each other.

Both configurations use Microsoft Visual Studio and the same TenAsys tools to create and debug INtime applications. With *INtime for Windows* the INtime SDK typically resides on the same host as the real-time application being developed, while in *INtime Distributed RTOS*, the INtime applications and the Windows-based SDK run on separate hosts connected by Ethernet, as shown in the next figures.

Figure 1: **INtime for Windows** configuration:

With Windows running the SDK and the INtime OS (local node) on the Same host

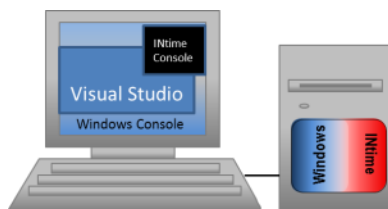
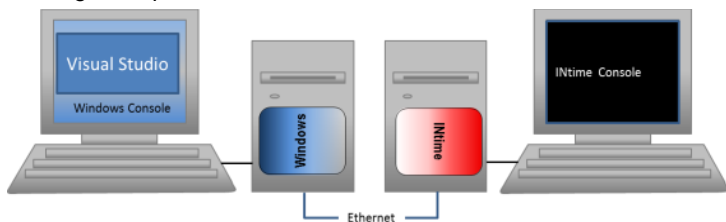


Figure 2: **INtime Distributed RTOS** configuration:

With Windows running the SDK and the INtime OS (deployment node) running on separate hosts



This guide familiarizes you with the INtime development tools. You can find detailed technical information about the INtime software architecture, kernel, and APIs in the online help files and user manuals.

When running the examples in the *INtime for Windows* configuration setup; Examples 1, 2 and 4 can be performed with Windows and INtime each running on a dedicated HW thread¹ on a host that supports two HW threads. Example 3 requires a host with three (3) or more HW threads.

When running the Examples 1, 2, 3 and 4 in the *INtime Distributed RTOS* configuration setup, a second x-86 host is needed on which to install INtime. Example 3 requires the second host with two (2) or more HW threads.

Notational Conventions

This guide uses the following conventions:

- All numbers are decimal unless otherwise stated.

¹ A HW thread is defined as a single processor core or one side of a single core Hyper threaded processor. For example, a Dual-Core processor and a Single-core Hyper- threaded processor supports two (2) HW threads. A Dual core hyper-threaded processor supports four HW threads.

- Bit 0 is the low-order bit. If a bit is set to 1, the associated description is true unless stated otherwise.
- Data structures and syntax strings appear in this font.

Notes indicate important information about the product.

Tips indicate alternate techniques or procedures to use to save time or better understand the product.

Terms

- **System:** Total interactive collection of hosts used in a networked solution.
- **Host:** computer that shares memory, IO bus, and a single BIOS. A host may have multiple cores and sockets.
- **Core:** one of many cores in a multi-core processor.
- **Hardware Thread:** One hardware core or hyperthreaded core.
- **Node:** one OS instance (a Windows node can include multiple cores). INtime for Windows has at least two Nodes on one Host: one for Windows and at least one for INtime.

Requirements

- INtime can be installed on hosts with up to 64 hardware threads.
- INtime can be assigned to a maximum of 32 hardware threads.
- Note that Hyperthreading provides 2 hardware threads for each hardware core when enabled.
- Each INtime process can address a maximum of 4GB memory. Multiple processes can be loaded on each INtime instance (thread) enabling access to the entire 64-bit memory address space.

INtime SDK and INtime for Windows Requirements

The tutorial applications in this guide will be built and executed directly on the development host. The development host needs to meet the following minimum requirements:

- Pentium class (or better) processor
(See note above listing the kind of processor required to support each example application.)
- 40MB of free RAM for INtime and your real-time applications
- 250MB hard disk space for tools, examples, and documentation
- Windows 11, Windows 10, Windows Server 2022, 2019, or 2016
(See the Knowledge Base at tenasys.com/my-tenasys/knowledge-base/ for updated Windows version support information). Both 32-bit and 64-bit versions of Windows are supported.
- Windows must be loaded using the Windows bootloader. Third-party bootloaders will not load INtime successfully.

- Visual Studio (2022, 2019, 2017, or 2015). (See the Knowledge Base at tenasys.com/my-tenasys/knowledge-base/ for updated Visual Studio version support information.)

INtime for Windows applications run with Windows 11, Windows 10, Windows Server 2022, Windows Server 2019, and Windows Server 2016. The examples in this guide focus on the Windows 10 environment and with Visual Studio 2019. Check the installer readme file for any amendments to these requirements.

INtime Distributed RTOS Deployment Host Requirements

The deployment host must be a PC platform with the following requirements:

- Pentium class (or better) processor with APIC enabled.
(See note above listing the kind of processor required to support each example application.)
- At least 64 MB of RAM per hardware thread plus additional memory for your applications
- For initial installation, the host must be capable of booting from USB media, with IDE, or SATA, NVMe, eMMC interface and storage media attached.
- A keyboard is required for installation; both USB and PS/2 types are supported.
- A supported network controller interface is required for connection to the Windows development host – onboard or on a card. (See the Knowledge Base at tenasys.com/my-tenasys/knowledge-base/ for updated list of supported network cards.)

SDK Installation

The following describes how to install the INtime development tools and kernel on the development host.

Before you begin

- Make sure that the development host meets the requirements listed in the Platform Requirements section above.
- Install Visual Studio

Note: If you install Visual Studio after installing INtime, use the INtime Configuration Manager to add the INtime development tools to Visual Studio.

- Make sure you are logged on with Administrator privileges.

Note: If the user account that will use INtime on this host is not an administrator, you must add the user to the system's INtime Users Group.

- If you plan to install a network-licensed product, get the IP address or fully qualified domain name of the license server. Contact your company's IT personnel for this information.
- Insert the CID/USB license key, if your product includes one.

Install the software

Download and execute the file "intime<version>full_installer.exe" from the My Downloads page of the TenAsys.com website.

An alternative is to insert the INtime CD-ROM. A welcome dialog appears.

If the welcome dialog does not appear, double-click *readme.htm*, located in the root directory of the INtime CD-ROM. The file appears in your default browser. Click the "SDK Installation" link at the bottom of the page. The installation starts.

The installation procedure is like that of most standard Windows applications. You are prompted to accept the INtime software license agreement to complete the installation procedure.

Files are installed in the following locations:

Files	Location
INtime development tools and sample files	Environment variable: %INTIME% ¹
Sample projects	My Documents\INtime\Projects (for the user who installed INtime).
Configuration files	Environment variable: %INTIMECFG% ²

- 1 Typically C:\Program Files\INtime. On 64-bit versions of Windows this is C:\Program Files (x86)\INtime. Make note of this directory so you can locate it again if you wish to inspect header files and other INtime files.
- 2 Typically C:\Program Data\TenAsys\INtime. Make note of this directory so you can locate it again if you wish to inspect configuration or license files.

Three install options are provided:

- **Development Tools Only**
This option installs only the SDK; it does not install INtime for Windows runtime software. Select this option to set up a development host for use with an INtime Distributed RTOS

deployment host. The INtime Distributed RTOS components are not installed.

- **Development Tools & INtime for Windows.**

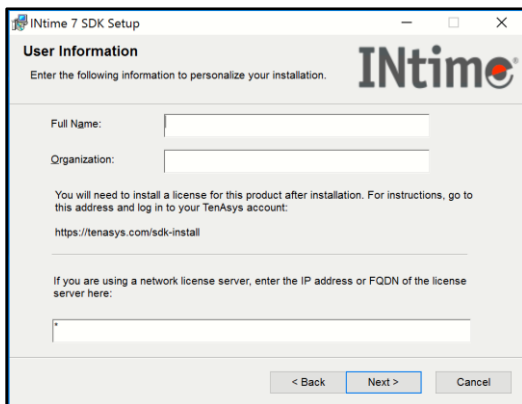
This option installs the SDK and the components required to configure and run local INtime nodes. Select this option to develop applications for INtime for Windows.

You can also select this option to develop INtime Distributed RTOS applications. You will, however, need to select the deployment node to develop and run the application on the deployment host.

- **Custom**

The installer prompts for a Name and Organization. If a network license has been purchased (for 6 or more SDK licenses) enter the IP address or the Fully Qualified Domain Name of the license server. Leave this box empty for an evaluation license or CID Key license. Contact your company's IT personnel or your INtime site coordinator for the license server information needed to complete this licensing step.

Figure 3: Entering Network License FQDN or IP.



After installation is completed, the installation program prompts to reboot the host. After the host reboots, the INtime runtime environment and services can be configured. For the purposes of this document the default configuration will suffice. From the INtime Status Application icon in the task bar, open the INtime Configuration Panel, then the License Manager to enter the License String for an evaluation license or for a CID Key license. Skip this step for a network license.

IMPORTANT: Once the installation is complete, the development host checks for the appropriate license to run. To use the host you must load and activate the license. For details, see the document [INtime SDK Evaluation License Procedure](#) for the license installation procedure at tenasys.com/sdk-install or tenasys.com/policies

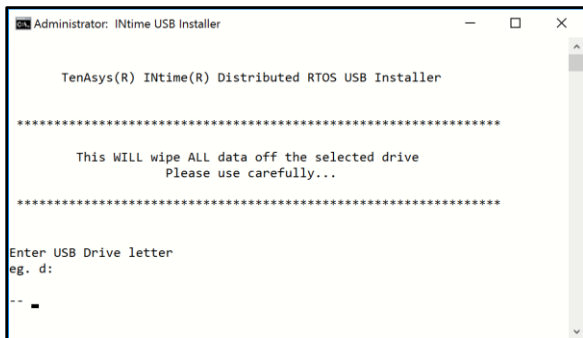
Also see [INtime Deployment Trial License Procedure](#) for trial licenses on either INtime for Windows or INtime Distributed RTOS hosts (where INtime is installed on a host that is not the SDK host).

INtime Distributed RTOS Deployment Host Installation

The steps below describe the INtime Distributed RTOS deployment host installation process. The installation is done by booting from a USB flash drive.

Create the USB installer using the shortcut **Create an INtime Distributed RTOS USB installer** in the INtime programs group.

Figure 4: Enter USB Drive letter



Insert the USB flash drive and enter the drive letter at the prompt.

Figure 5: Create USB installer.



Change the BIOS settings to boot from the USB stick once. Both Legacy and UEFI boot mode are supported.

Note: If the partition type of the USB stick is MBR (Master Boot Record) it cannot be used to install INtime Distributed RTOS on hosts with a UEFI only BIOS. Repartition the USB stick to GPT (GUID Partition Table) with the Windows utility: diskpart.

On boot, an automatic installation script runs that prompts for responses to several configuration questions. This configuration must be completed within 10 minutes.

Note: At each step, the script prompts to continue. If you choose not to continue, the previous step generally repeats.

Configuration questions include:

Prompt	Description
Keyboard Configuration	Select a keyboard from the menu by entering its number, and then press Enter.
Storage Device Partition and Format	A list of storage devices with drive names appears. Select the drive for the INtime install.
Repartitioning the device	<ol style="list-style-type: none">1. Do you wish to repartition the device (all data will be lost)? [y/n]. Enter 'y' to repartition, 'n' to keep the existing partitions.2. Format device ... Proceed [y/n]. Enter 'y' to format, 'n' to keep the existing format.
Install the files	Type 'y' to unpack the INtime software files into the partition.
Software Installation	Enter 'y' to proceed.
Time Zone Configuration	If the default does not apply, follow the prompts to select your time zone.
Set Time and Date	Adjusts the PC real-time clock (battery clock), depending on whether you want time kept in UTC or your local time zone. Most PCs keep their clock in local time, but you can adjust the battery clock to keep time in UTC (Universal Coordinated Time, or GMT).

Prompt	Description
Network Configuration	<p>Interfaces detected by the installer appear. Choose the default host interface, used to connect to your development host.</p> <ul style="list-style-type: none"> • DHCP Enter 'y' if you wish the network address to be assigned by the local DHCP server. • IP Address (if DHCP is not selected). Enter an IPv4 address appropriate for the local network. • Netmask Enter an appropriate netmask for the local network. • Gateway May be left blank, otherwise the address of the forwarding gateway for the local subnet. • Enter Host Name Should be a unique name recognizable among the devices on the network. Host name may contain only the ASCII letters 'a' through 'z' (in a case-insensitive manner), the digits '0' through '9', and the characters ('-', '_'). • Enter Domain Name. May be left blank or choose the local internet domain name. For example, <code>mydomain.com</code>. The Domain Name has the same character restrictions as hostname. Use '.' to separate subdomain names.
Set Administrator Password	Use this password to gain access to the web-based configuration utility on the deployment host. Re-enter the password at the prompt for confirmation.
Reboot the host	Remove the installation media and allow the host to reboot.

After rebooting, the INtime RTOS load process appears, as shown below.

IMPORTANT: Once the installation is complete, the deployment host checks for the appropriate license to run. After 10 minutes without a license the host will shut down. To use the host, you must load and activate the deployment license. For details, see the document *INtime Deployment Trial License Procedure* for trial licenses on either INtime for Windows or INtime Distributed RTOS hosts at tenasys.com/sdk-install or tenasys.com/policies

Figure 6: Boot status

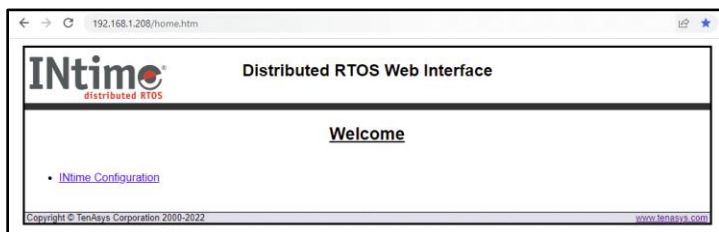
```
Loading init process (NodeID 0)
iosys: root file system is clean, skipping check
init: INtime Distributed RTOS version 7.0.22346.1
init: INtime Kernel Version 7.0.22345
init: Loading syslogd
init: Loading USB stack
** /dev/ada0p1
** Phase 1 - Read and Compare FATs
** Phase 2 - Check Cluster Chains
** Phase 3 - Checking Directories
** Phase 4 - Checking for Lost Files
14 files, 4179520 free (130610 clusters)

init: License check...
init: Got a multicore license
init: Loading node manager
init: Loading network
dhclient: rtl1g0 no link ...
dhclient: rtl1g0 no link ...
dhclient: rtl1g0 got link
dhclient: rtl1g0 bound to 192.168.1.208 - renewal in 43200 seconds.
init: Loading network apps
init: Loading FTP server
init: Loading web server
init: Loading SMTP client
init: Loading GOBS service
DrtosCIExtension.rsl loaded
Welcome!
DRTOS
User name:
```

To verify that the INtime Distributed RTOS host is up, check if its webserver responds by opening a browser and entering its IP address: 192.168.1.208 (displayed in the screen above “dhclient: New IP Address (rtl1g0): 192.168.1.208”)

When the following screen appears, click “INtime Configuration”.

Figure 7: DRTOS Web Configuration



A login screen follows. Use the password entered under “Set Administrator Password:” during the DRTOS installation.

To use INtime Distributed RTOS for more than 10 minutes, a license is required. Obtain a Distribution Trial license from the [My Account](#) page on the [tenasys.com](#) website. Use the “Activate manually: install license file.” Or “Activate manually: install license string.” from the License tab in the INtime Distributed RTOS web interface.

Example #1: The INtime Application Wizard – HelloWorld

This exercise introduces the INtime Application Wizard, which you use to create a simple real-time process. The Wizard adds template code for elements to a new project, such as semaphores, threads, shared memory allocation, interrupt handling, and client threads. The Wizard creates the foundation for the *HelloWorld* example project.

In the *HelloWorld* project you will create a thread that executes in an infinite loop. The thread will sleep approximately 1000 milliseconds and print the phrase “HelloWorld” ten times in an INtime console window, per each loop iteration.

Note: For the sake of brevity, only screenshots of significant value are shown within the tutorials of this guide.

Tip: Open the electronic (PDF) version of this guide and use the Adobe Acrobat “Text Tool” to copy and paste code fragments from the documentation into your Visual Studio project. These projects are also included in the SDK installation sample projects directory: My Documents\INtime\Projects (for the user who installed INtime).

Example #1 uses the “Hello World Sample” shortcut for the “HelloWorld” project.

Using the INtime Application Wizard

- 1) Create a directory on your development machine called *INtimeApps* (suggested to store the examples from this Guide).
- 2) Start Visual Studio.
- 3) Create a new project using the INtime Application Wizard template near the bottom of the list.

Figure 8: Creating a new project.

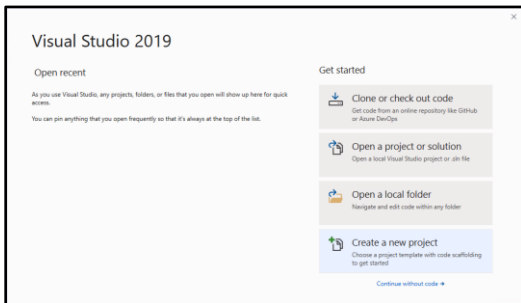


Figure 9: Selecting template

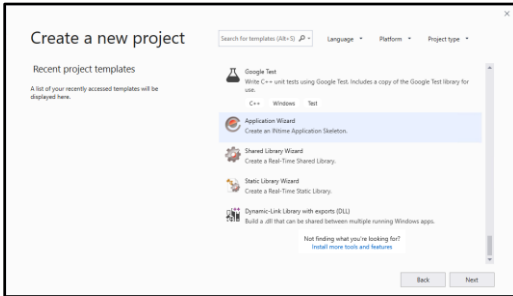
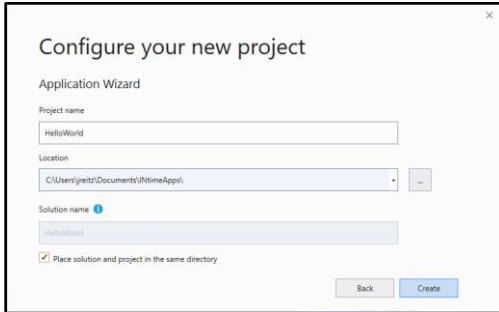


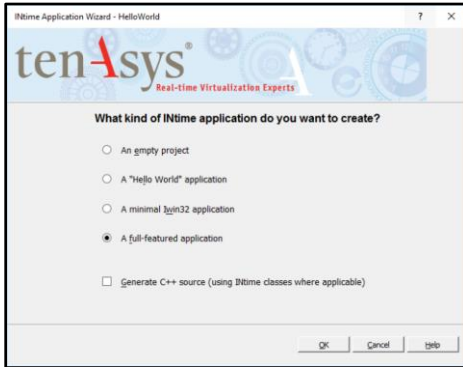
Figure 10: Selecting name and location for solution



4) Under Installed Templates:

- a. Select *INtime Projects*.
- b. Enter *HelloWorld* as the project name.
- c. Set the location (path) to the *INtimeApps* directory that you created above.
- d. Select *Application Wizard*. Click OK. The wizard dialog appears.

Figure 11: Creating a New INtime Project



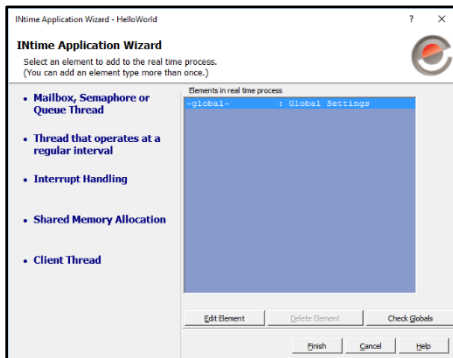
- 5) Select *A full-featured application* and leave the C++ box unchecked.

Note: This tutorial does not use the INtime wizard's *HelloWorld* application because the features of this sample project will be more interesting.

- 6) Click *OK* to continue.

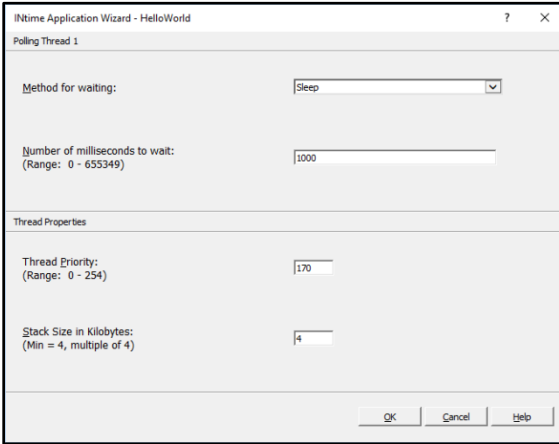
The Add Elements dialog appears. This is where you add elements to the real-time process, such as mailboxes, semaphores, and threads. You can create these elements manually, but, using the INtime Wizard saves time and minimizes errors.

Figure 12: Selecting Process Elements



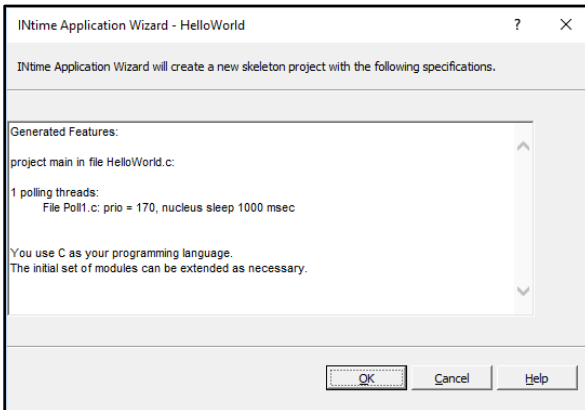
- 7) Select *Thread* which operates at a regular interval from the list of real-time process elements. The element detail dialog appears.

Figure 13: Specifying Polling Thread Parameters



- 8) Keep the default settings for the polling thread, so the thread will wake up every 1000 milliseconds. Click *OK* to return to the *Add Elements* dialog.
- 9) Highlight `-global-` in the elements list on the right of the dialog and click *Edit Element*. In the dialog box that appears, you can modify real-time process parameters. The default parameters are fine for this example.
- 10) Click *OK* and then *Finish*. The final wizard summary screen appears.

Figure 14: Wizard Summary Screen

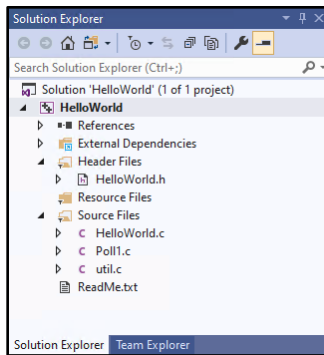


11) Click *OK*. The wizard generates project files.

The Visual Studio solutions explorer displays the following .C files generated by the wizard:

- *HelloWorld.c*: the `main()` function which contains initialization and cleanup code. The file name is derived from the project name.
- *Poll1.c*: the polling thread code generated by the *add real-time elements* section of the wizard.
- *Util.c*: contains general-purpose utility routines.

Figure 15: Files Generated by the wizard.



12) Edit *Poll1.c*:

- Open *Poll1.c*.
- Add an integer named 'x' at the start of the polling thread.
- Add a `for` loop and `printf()` statements after the `TODO` comment. The resulting code should look like the following (additions are shown in bold):

```
void Poll1(void* param)
{
    int x;
    #ifdef _DEBUG
        fprintf(stderr, "Poll1 started\n");
    #endif

    ...intervening lines removed for brevity...

    while (!gInit.bShutdown)
    {
        RtsSleep(1000);

        #ifdef _DEBUG
            fprintf(stderr, "Poll1 waking up\n");
        #endif

        // TODO: do what has to be done every 1000
        milliseconds
        For (x = 0; x < 10; x++)
```



```

        printf("Hello World!\n");
    }
    // tell that this thread is dead
    gInit.htPoll1 = NULL_RTHANDLE;
}

```

Make sure the build type is set to *Debug* (go to the **Build|Configuration Manager...** menu or select *Debug* on the menu bar and build the solution (**Build|Build Solution** or type **Ctrl+Shift+B**). The *HelloWorld* program compiles and links.

Stop and start the application from Visual Studio

- 1) INtime for Windows:
 - Start the NodeA application:
 - a. Start the Node by clicking the hidden icon in the Windows Toolbar.
 - b. Click the INtime (e icon).



- c. Click on **Start NodeA**.

INtime Distributed RTOS:

Make sure that the target node is booted and running.

- 2) Select the target node in INtime Properties:
 - a. In Visual Studio select the INtime project icon in the Solution Explorer window (as shown in the Figure below) and right-click it.
 - b. Select Properties from the window.

Figure 16: Selecting INtime Project

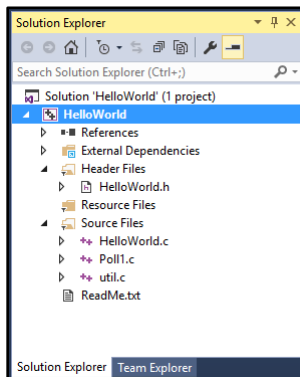
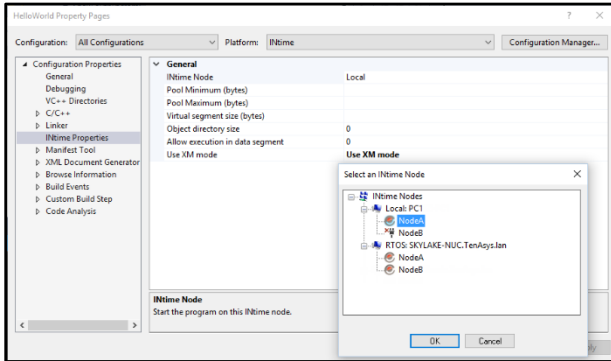



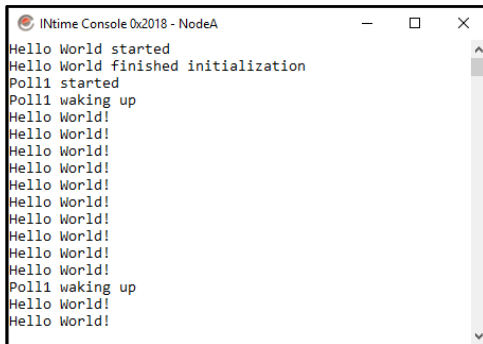
Figure 17: Selecting an INtime Node from Visual Studio.



- 3) Select the node on which you wish to run your program.
The figure above shows “NodeA” running on PC1, the host on which the Windows and INtime SDK is running (also known as a local Node). This represents an *INtime for Windows* configuration setup. NodeB has a different icon because it is not running. “NodeA” running on DRTOS.TenAsys.Ian represents an *INtime Distributed RTOS* configuration setup.
For this tutorial, select the local “NodeA”.
- 4) To run the application with Debug, do one of the following:
 - Select **Debug|Start Debugging**
 - Press **F5**
 - Click the green arrow  on the tool bar.

An INtime console window appears and the message *Hello World!* appears ten times each second inside the console window.

Figure 18: HelloWorld Console Window



- 5) To stop the application, click the Stop icon (square) on the Toolbar



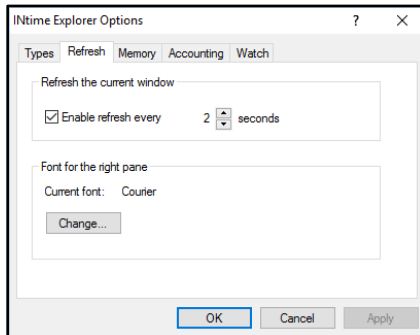
or type **Shift+F5**.

Introducing the INtime Explorer

In this Example, the INtime Explorer (aka INTex) will load and run the just-built *HelloWorld* application. INTex displays the real-time objects present on an INtime node (an INtime real-time kernel).

- 1) Start the INtime kernel, if it is not already running, from the INtime Status Monitor in the Windows system tray. Select **Start NodeA**. Note that *NodeA* is the default name of the INtime kernel, you can create other nodes and assign different names.
- 2) Start INtime Explorer using its shortcut in the INtime program group.
- 3) Select your node from the dialog box and click *OK*.
- 4) Turn on the INtime Explorer automatic refresh:
This feature is useful when debugging a *local* INtime node.
 - a. Select **View|Options...** on the INTex menu.
 - b. Select the Refresh tab.
 - c. Check the *Enable refresh every* box and set the interval for two seconds.
 - d. Click *OK*.

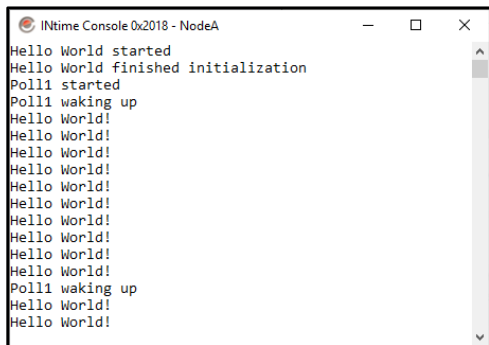
Figure 19: Configuring INtime Explorer Options



- 5) Load and run *HelloWorld* using one of these methods:
 - Click the second button on the INTex toolbar.
 - Select **File|Load RT app**.
- 6) Navigate to the debug folder in your *HelloWorld* project directory and select the real-time executable file *HelloWorld.rta*.
- 7) Click *Open* to load and start the real-time process on the INtime kernel.

INtime for Windows: A console window and the message *Hello World!* appears ten times each second inside the console window.

Figure 20: HelloWorld Console Window



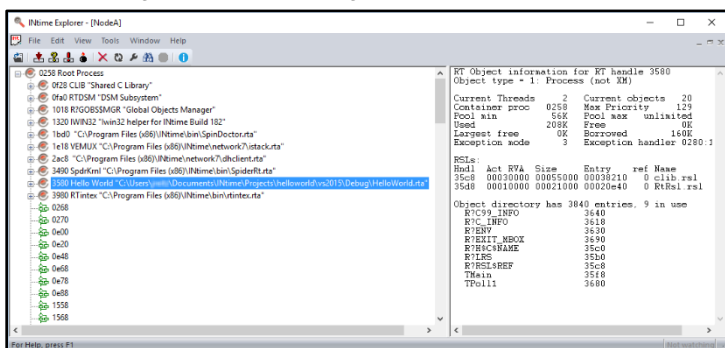
```
INtime Console 0x2018 - NodeA
Hello World started
Hello World finished initialization
Poll1 started
Poll1 waking up
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Poll1 waking up
Hello World!
Hello World!
```

INtime Distributed RTOS: the output appears on the target node console.

Terminate the *HelloWorld* process by doing the following:

- 1) Go to the INtime Explorer main window.
- 2) Find the *HelloWorld* real-time process in the left window of INtime Explorer (each INtime icon represents one real-time process).
- 3) Click the line to select the HelloWorld process.
- 4) Click the red 'X' button in the toolbar to delete the process.

Figure 21: Terminating the HelloWorld Process



Answering *Yes* to the deletion warning pop-up terminates the real-time process. The *HelloWorld* process icon disappears from the INtime Explorer process list. Notice that the *HelloWorld* console window remains on your desktop, but the console window's title bar displays *Finished*.

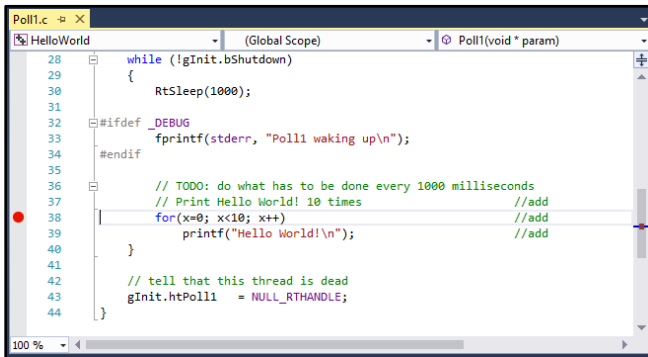
Debugging HelloWorld with Visual Studio

With INtime, you can debug real-time processes directly from within Visual Studio (from Visual Studio 2008 onwards). Using the just-created *HelloWorld* project, you can step through the code and perform basic debugging tasks.

Note: If you are continuing directly from the previous section, steps 1 to 4 are not necessary.

- 1) If the INtime kernel is not already running, start it using **INtime Status Monitor|Start NodeA**. (In the Windows Toolbar)
- 2) Start the Visual Studio development environment.
- 3) Open the *HelloWorld* project.
- 4) Open *Poll.c* within the *HelloWorld* solution.
- 5) Set a breakpoint on the `for` loop, using one of these methods:
 - Click the vertical bar to the left of the source window.
 - Place the cursor on the line and press the *F9* key.

Figure 22: Setting a Breakpoint



- 6) Start the debugger using one of these methods:
 - Press the *F5* key.
 - Click Start on the Visual Studio tool bar.

The *HelloWorld.RTA* process launches.

Note: If you are not running the default configuration, you may need to select the target INtime node in the INtime project settings.

The *HelloWorld* process runs to the breakpoint. Following the break, you can step through the code and watch variables change (e.g., '*x*') as you step through the loop. Debugging an INtime real-time thread in this way is virtually identical to debugging a Windows thread.

Example #2: Working Together – Windows and Real-time

The typical *INtime for Windows* solution consists of these executables:

- A standard Windows process that provides access to the Windows user interface, database functions, and other Windows-specific functions.
- A real-time *INtime* process containing time-critical threads.

The *INtime* NTX library manages communication between the two parts.

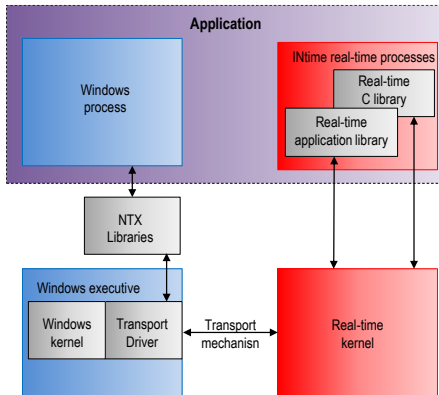
This example uses *INtime data mailbox* objects to demonstrate how a simple Windows MFC dialog process can exchange data with real-time threads running on the *INtime* kernel.

Tip: Example #2 uses “RT Data Project” shortcut for the “RTData” project and the “NTX Data Project” shortcut for the “NXTData” project in the sample projects directory.

Two processes – one application

Three data mailboxes, *MY_MBOX_1*, *MY_MBOX_2*, and a third mailbox, *MBOX_Signal*, which will be used to send data between two processes: *NTXData.exe* (a Windows process) and *RTData.rta* (a real-time process) and to signal between two real-time threads. Together these two processes comprise a single *INtime* software application.

Figure 23: Basic *INtime* Solution Architecture



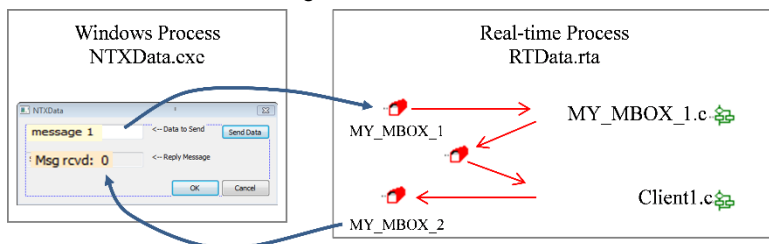
This data mailbox example is only one possible solution for sharing data between an *INtime* real-time application and a Windows application; other solutions might incorporate shared memory or exchanging semaphores between Windows processes and real-time processes.

Tip: To learn more, locate the topic “*INtime* System Description” in the *INtime* Help.

Creating the Real-Time Process

First, we will create *RTData.rta*, the real-time process launched by *NTXData.exe*. The real-time process sets up the mailboxes and waits for the Windows process to send a data message through the first mailbox. After a message is received, the real-time process sends data back to the Windows process using the second mailbox. The third mailbox is used for internal communication between real-time threads within the real-time process.

Figure 24: Data-flow



- 1) Open Visual Studio, create a real-time project called *RTData*, and place it in the *INtimeApps* directory you created for the *HelloWorld* example.

Note: It is important to name this project “*RTData*,” exactly as shown above. The name you specify is used as a real-time process object identifier and is referenced in later code. INtime object names are case-sensitive.

- 2) Choose *A full-featured application* from the INtime Application Wizard and click *OK* (leave *C++* unchecked for this example).
- 3) Add a *Data mailbox* element by selecting *Mailbox, Semaphore or Queue Thread* from the list of available elements. (Set the *Type of object this thread waits at* to *Data mailbox* in the options dialog.)
- 4) Type *MY_MBOX_1* for the *Catalog the object with this name* field.
- 5) Click *OK* to return to the elements setup dialog.
- 6) Repeat the preceding three steps to add a second data mailbox, but this time name it *MY_MBOX_2*.
- 7) Click *OK* to return to the elements setup dialog.
- 8) From the elements setup dialog add a *Client Thread* (last element in the list).
- 9) Check the *Send to data mailbox* item (upper left), leave all other items unchecked, then click *OK* to return to the elements setup dialog.
- 10) Click *Finish* followed by *OK*. The wizard automatically generates real-time code templates.

The code generated by the above steps is only a starting point. Modifications are required to turn this project into a running program. Data mailbox `MY_MBOX_1` receives messages from the Windows process, and data mailbox `MY_MBOX_2` sends messages to the Windows process.

The client thread in `Client1.c` sends messages to the Windows process via `MY_MBOX_2`. The code in `MY_MBOX_2.c` is only used to create that data mailbox. In addition, we will manually add a third data mailbox for inter-thread communication.

Note: This file and function structure is not necessarily the most efficient or elegant solution; it is being used to quickly demonstrate the INtime architecture and the use of INtime wizards to generate template code.

Make the modifications shown below in bold to `RTData.c`. This file contains the real-time process' `main()` function. These modifications add a third data mailbox to coordinate receiving data from `MY_MBOX_1` and sending data via `MY_MBOX_2`. The last lines added take control of the region object and release that control after thread initialization is complete.

Tip: Open the electronic (PDF) version of this Guide and use the Adobe Acrobat "Text Tool" to copy and paste these code fragments directly from the documentation into your Visual Studio project.

Changes to RTData.c

```
// global variables
RTHANDLE    hRootProcess;
DWORD       dwKtickInUsecs;
RTHANDLE    hMBOX_Signal;

...intervening lines removed for brevity...

// create mailbox, semaphore and message queue threads
hMBOX_Signal = CreateRtMailbox(DATA_MAILBOX | FIFO_QUEUING);
if (hMBOX_Signal == BAD_RTHANDLE)
    Fail("Cannot create signaling data mailbox");
```

Do not forget to include a global declaration for the region object, `hMBOX_Signal`, at the end of `RTData.h`.

Changes to RTData.h

```
extern RTHANDLE    hRootProcess;
extern DWORD       dwKtickInUsecs;
extern RTHANDLE    hMBOX_Signal;
```


Open *MY_MBOX_1.c*. The Wizard generated code to create, initialize, and retrieve data from the mailbox. We are adding code to print received data to a console window and signal to *Client1* that a reply message can be sent.

Changes to MY_MBOX_1.c

```
// TODO: operate on byMmessage and dwActual
//Print the message received from the mail box
printf("This is the message: %s\n", byMessage);

//Indicate that the message was received
SendRtData(hMBOX_Signal, "go", 3);
}
}
```

Next, open *MY_MBOX_2.c* and remove the lines in the `while` loop that wait for data to be received from the data mailbox; in the code fragment below, they are commented out. In this example we use this thread only to initialize the data mailbox. To eliminate compile warnings, remove the declarations as well.

The Windows process receives from this data mailbox and the real-time process sends through this data mailbox. Add a line at the end of the `while` loop to suspend the thread.

Changes to MY_MBOX_2.c

```
// WORD wActual;
// BYTE byMessage[128];
// RTHANDLE hMY_MBOX_2;

...intervening lines removed for brevity...

// wActual = ReceiveRtData(hMY_MBOX_2, byMessage, WAIT_FOREVER);
// if (0 == wActual)
// {
//     Fail("Receive from data mailbox MY_MBOX_2 failed");
//     break;
// }

// TODO: operate on byMmessage and dwActual
SuspendRtThread(GetRtThreadHandles(THIS_THREAD));
```

Finally, open *Client1.c* and add the `retMessage[]` array that will build return messages. Remove the lines used to look up the process handle, since the data mailbox we will reference in this thread was created in this process. Modify the parameters accordingly in the line that gets the handle to the *MY_MBOX_2* data mailbox.

Changes to Client1.c

```
void Client1(void *param)
{
    RTHANDLE    hProcess = NULL_RTHANDLE;
    RTHANDLE    hDmbx;
    char        retMessage[128];
    int         x = 0;
    int         y;
    // TODO: adjust process and mailbox name
    // TODO: remove the next lines if the data mailbox
    // was created in this process
// hProcess = LookupRtHandle(hRootProcess, "DMBX_OWNER",
WAIT_FOREVER);
// if (BAD_RTHANDLE == hProcess)
// {
//     Fail("Cannot find data mailbox process");
//     return;
// }

    // TODO: replace hProcess by NULL_RTHANDLE
    // if the data mailbox was created in this process
// hDmbx = LookupRtHandle(hProcess, "DMBX_NAME", WAIT_FOREVER);
hDmbx = LookupRtHandle(NULL_RTHANDLE, "MY_MBOX_2", 5000);
if (BAD_RTHANDLE == hDmbx)
{
    Fail("Cannot find data mailbox");
    return;
}
}
```

Finally, add code in the `while` loop to wait for the signal indicating that we should send a message to the Windows process. We will assemble the message sent by including an incremented count value so each response message is unique.

```
while (!gInit.bShutdown)
{
    // TODO: put client code that must be repeated here
    // the RtSleep call is just an example
// RtSleep(1000);
ReceiveRtData(hMBOX_Signal, retMessage, WAIT_FOREVER);
// TODO: fill a message and its size
// if (!SendRtData(hDmbx, "test", 5))
y = sprintf(retMessage, "%s %i", "Msg rcvd: ", x++);
if (!SendRtData(hDmbx, retMessage, ++y))
{
    Fail("Cannot send to data mailbox");
    break;
}
}
```

We are ready to build the application. Choose **Build|Build Solution** or type **Ctrl+Shift+B** from the Visual Studio menu to compile and link. Check the *Debug* folder in your *RTData* project directory and you should find an *RTData.rta* file, among others. This is the real-time process' executable (equivalent to a Windows EXE file).

Creating the Windows Process

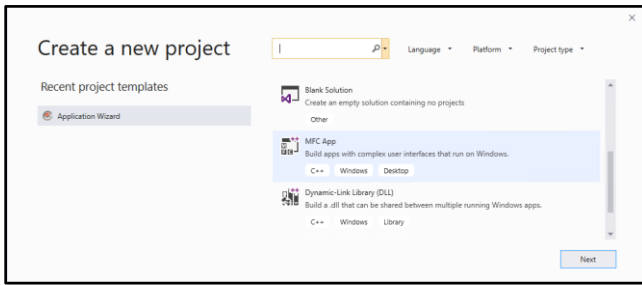
When we create the Windows part of our application, we need to setup the project environment to include NTX support. Creating the Windows application takes a few steps.

Create the Project and Setup the Environment

Note: These instructions are specific to Visual Studio 2019, but should be similar for other versions.

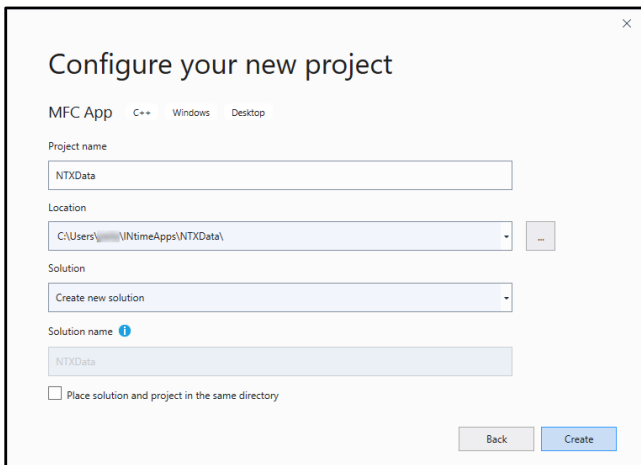
- 1) Start Visual Studio.
- 2) From the menu, select **File|New|Project** or type **Ctrl+Shift+N**.
- 3) Scroll down and select the *MFC App* template. Click Next.

Figure 25: Selecting the MFC Application Template



- 4) Specify *NTXData* as the project name, set the Location to the *INtimeApps* folder. Click Create.

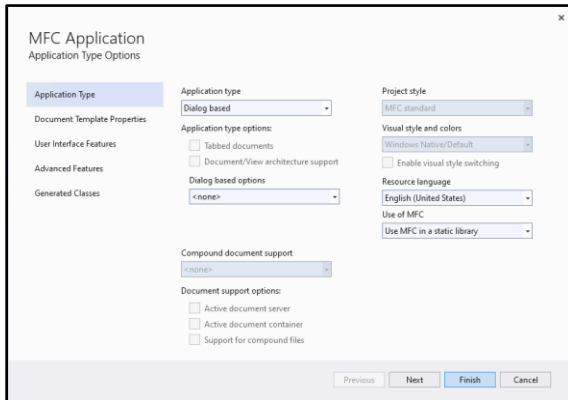
Figure 26: Specifying the Project Name and Location



- 5) Within the *Application Type* options, select *Dialog based* under *Application type*, and *Use MFC in a static library* under *Use of MFC*.

Note: When using Unicode libraries do not use the `_T()` function when passing text into functions requiring `LPTSTR`. See the conversion at the end of `NTXDataDlg.cpp`.

Figure 27: MFC Application Type Selections



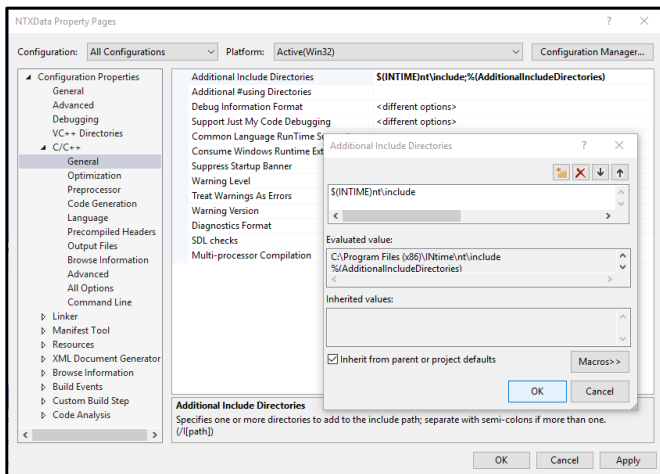
- 6) Click *Finish*. The Wizard generates template code.

Before proceeding with the MFC code, you must modify the project properties to include the NTX library and header files. This is required for any Windows program using the Windows Extension (NTX) system calls

Note: These instructions are specific to Visual Studio 2019, but should be similar for other versions.

- 1) From the Visual Studio menu select **Project Properties...** (or right click **NTXData** and select **Properties** in the Solution Explorer)
- 2) On the property pages dialog choose *All Configurations* in the *Configuration* pull down and either Win32 or x64 for Platform.
- 3) Navigate to **C/C++ | General | Additional Include Directories** and select <Edit...> in the field selector. Click the left icon for New Line and type `$(INTIME)nt\include` in the open field. Keep "Inherit from parent..." checked. Click OK.

Figure 28: Specifying Additional Include Directories



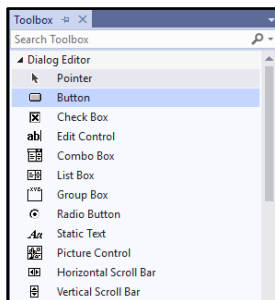
- 4) Similarly, navigate to **Linker | General | Additional Library Directories** and select <Edit...> in the field selector. Click the left icon for New Line and type `$(INTIME)nt\lib` in the open field.
- 5) And finally, navigate to **Linker | Input | Additional Dependencies** and select <Edit...> in the field selector. Type `ntx.lib` (`ntx64.lib` for x64 platform) in the open field.
- 6) Click **OK** to save changes and close the property pages dialog.

Creating a Graphical User Interface

The following steps create the GUI for the Windows process.

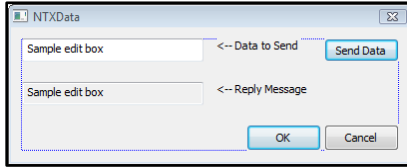
- 1) Under Resource Files in the Solution Explorer, double click `NTXData.rc`, then expand the **Dialog** category and double click `IDD_NTXDATA_DIALOG`.
- 2) Delete the `TODO: Place dialog controls here` test object.
- 3) Select **View|Toolbox** or type **Ctrl+Alt+X**. Expand the *Dialog Editor*.

Figure 29: Dialog Editor in the Toolbox



- 4) Add by dragging two *Edit Control* objects, two *Static Text* objects, and one *Button* object. The figure below shows a layout for the controls in the NTXData dialog box.

Figure 30: NTXData Dialog Box



- 5) Modify the properties of each control as follows. Right click each element to access *Properties*.

IDC_Button1	
ID	IDC_txDATA
Caption	Send Data
Default Button	True

IDC_Edit1	
ID	IDC_DATA

IDC_Static2	
ID	IDC_STDATA
Caption	<-- Data to Send

IDC_Edit2	
ID	IDC_rxDATA
Read Only	True

IDC_Static3	
ID	IDC_STRM
Caption	<-- Reply Message

Leave the *OK* and *Cancel* buttons as part of the dialog box. You can use them to close the NTXData application.

- 6) Save and build the solution, **Build|Build Solution** or type **Ctrl+Shift+B**, to make sure that it compiles without errors.

Edit the Code

These steps add code to start the *RTData.rta* process when *NTXData.exe* starts, using the INtime NTX API.

Changes to NTXDataDlg.h

- 1) Open the *NTXDataDlg.h* header file.
- 2) Add a `#include "ntx.h"` line at the top of the file.
- 3) Declarations for the real-time handles must be marked as protected. In the protected section of the class definition, add declarations for the handles needed to locate the *RTData* process and access the data mailboxes as well as the public declaration for the button action.

```

// NTXDataDlg.h : header file
//

#include "ntx.h"

#pragma once

...intervening lines removed for brevity...

// Implementation
protected:
    HICON m_hIcon;

    //The handles to the root processes, RTData, and mailboxes
    NTXHANDLE    m_RootProcess;
    NTXHANDLE    m_TestNTXProcess;
    NTXHANDLE    m_RtMailbox_1, m_RtMailbox_2;

    // Generated message map functions
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnBnClickedtxdata();
};

```

Changes to NTXDataDlg.cpp

4) Open *NTXDataDlg.cpp*.

Add this line after the #include statements:

```
NTXHANDLE    hNtx;
```

When *NTXData.exe* starts, it must load the *RTData.rta* application.

Add the following initialization code to load and start the *RTData.rta* application in `CNTXDataDlg::OnInitDialog`.

```

BEGIN_MESSAGE_MAP(CNTXDataDlg, CDialogEx)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_txDATA, OnBnClickedtxdata)
END_MESSAGE_MAP()

// CNTXDataDlg message handlers

BOOL CNTXDataDlg::OnInitDialog()
{
    NTXHANDLE    hRemoteApp;
    CString      tmp;

    ...intervening lines removed for brevity...

    // TODO: Add extra initialization here

    //Launch RTA Application
    //First set the location of the node
    //Typically you would want to use a browser
    //to select from the available nodes
    //For this example we will hard code the node to Local

```

```

        hNtx = ntxGetLocationByName("Local");
// Local node name: NodeB
// Local node name: ///NodeB
// Local node name: intime:///NodeB
// Remote DRTOS name: rtoshost.mydomain.com/NodeB
// Remote DRTOS name: intime://rtoshost.mydomain.com/NodeB
// Remote INtime name: host-NodeA.mydomain.com/NodeB
// Remote INtime name: intime://host-NodeA.mydomain.com/NodeB
//check to see that the node is there
If (ntxGetRtStatus(hNtx) != E_OK ) {
    MessageBoxEx(NULL, _T("RT Machine not present"),
        _T("NTXData"),
        MB_ICONERROR | MB_OKCANCEL, LANG_ENGLISH );
    exit(0);
}

//Now launch the RTData.rta application
hRemoteApp = ntxCreateRtProcess(hNtx,
// CHANGE THIS PATH for the location on the target system:
"C:\\Users\\[user]\\Doc...\\INtimeApps\\RTData\\Debug\\RTData.rta",
    NULL, NULL, NTX_PROC_SHOW_PROGRESS);
if (hRemoteApp == NTX_BAD_NTXHANDLE) {
    tmp = "Cannot load file";
    MessageBox(tmp);
    EndWaitCursor();
    exit(0);
}
return TRUE; // return TRUE unless you set the focus to
a control
}

```

Note: Make the appropriate changes to your code for the location of *RTData.rta* (loads from the windows host):
 "C:\Users\{User}\Documents\INtimeApps\RTData\Debug".

Note: *RTData.rta* can be executed on any INtime node. For Distributed RTOS, include the host/node instead of "Local".

The last change necessary to complete the application is in the code behind the *Send Data* button. This code consists of a sequence of NTX API calls to retrieve handles for the real-time root process, the *RTData* process, and the data mailboxes, *MY_MBOX_1* and *MY_MBOX_2*. The member variables defined in the *NTXDataDlg.h* header file are used here to store those handles. Once we have a handle to the data mailboxes, we can send the text typed into the *IDC_DATA Edit Control* to the *MY_MBOX_1 data mailbox* using *ntxSendRtData()*. The last part of the function waits for a return message from *RTData* from the *MY_MBOX_2 data mailbox* using *ntxReceiveRtData()*, and displays the message returned in the *IDC_rxDATA Edit Control*.

- 5) Insert the following code at the end of *NTXDataDlg.cpp*.
Build the solution after you finish editing the code.

Tip: Open the electronic (PDF) version of this guide and use the Adobe Acrobat "Text Tool" to copy and paste these code fragments directly from the documentation into your Visual Studio project.

```
void CNTXDataDlg::OnBnClickedtxdata()
{
    char          rt_my_mbx_1[] = "MY_MBOX_1";
    char          rt_my_mbx_2[] = "MY_MBOX_2";
    char          rt_TestNTXData_process[] = "RTData";
    char          send_buf[128];
    char          recv_buf[128];
    int           recv_buf_size;

#ifdef UNICODE
    TCHAR        wsend_buf[128];
    USES_CONVERSION;
#endif

    //Use the existing handle to the local INtime node
    //check to see that the INtime kernel is available
    If (ntxGetRtStatus(hNtx) != E_OK ) {
        MessageBoxEx(NULL,
            _T("RT Machine not present"),
            _T("NTXData"),
            MB_ICONERROR | MB_OKCANCEL, LANG_ENGLISH );
        exit(0);
    }

    //Get root process handle, needed to get RTData process handle
    if( (m_RootProcess = ntxGetRootRtProcess(hNtx))
        == NTX_BAD_NTXHANDLE) {
        MessageBoxEx( NULL,
            _T("Could not find INtime root process"),
            _T("NTXData" ),
            MB_ICONERROR | MB_OKCANCEL, LANG_ENGLISH);
        exit(0);
    }
    //Get RTData process handle
    if ((m_TestNTXProcess = ntxLookupNtxhandle(m_RootProcess,
        rt_TestNTXData_process, 0xffff)) == NTX_BAD_NTXHANDLE) {
        MessageBoxEx(NULL,
            _T("Could not find RTData process"),
            _T("NTXData"),
            MB_ICONERROR | MB_OKCANCEL, LANG_ENGLISH);
        exit(0);
    }
    //Now get a handle for each mailbox
    if (( (m_RtMailbox_1 = ntxLookupNtxhandle(m_TestNTXProcess,
        rt_my_mbx_1, 0xffff)) == NTX_BAD_NTXHANDLE)
        || ((m_RtMailbox_2 = ntxLookupNtxhandle(m_TestNTXProcess,
        rt_my_mbx_2, 0xffff)) == NTX_BAD_NTXHANDLE) ) {
        MessageBoxEx(NULL,
            _T("Could not find data mailboxes" ),
            _T("NTXData"),
            MB_ICONERROR | MB_OKCANCEL, LANG_ENGLISH);
        exit(0);
    }
}
```

```

        //Get the user information typed in IDC_DATA
        //and send it to mailbox MY_MBOX_1
#ifdef UNICODE
    GetDlgItemText(IDC_DATA, wsend_buf, 30);
    strcpy_s(send_buf, 30, W2A(wsend_buf));
#else
    GetDlgItemText(IDC_DATA, send_buf, 30);
#endif
    ntxSendRtData(m_RtMailbox_1, send_buf, 128);

    //Look for response back from RTData
    if((recv_buf_size
        = ntxReceiveRtData( m_RtMailbox_2, recv_buf, INFINITE))
        == NTX_ERROR ) {
        if(ntxGetLastRtError() != E_TIME) {
            MessageBoxEx( NULL,
                _T("Received data failed"),
                _T("NTXData"),
                MB_ICONERROR | MB_OKCANCEL, LANG_ENGLISH);
            exit(0);
        }
    }

    //Convert message from ASCII to Unicode
    LPTSTR lpsz = new TCHAR[recv_buf_size + 1];
#ifdef UNICODE
    _tcscopy_s(lpsz, (recv_buf_size + 1), A2W(recv_buf));
#else
    _tcscopy(lpsz, recv_buf);
#endif

    //Update Edit box with value
    SetDlgItemText(IDC_rxDATA, lpsz);
    UpdateData();
}

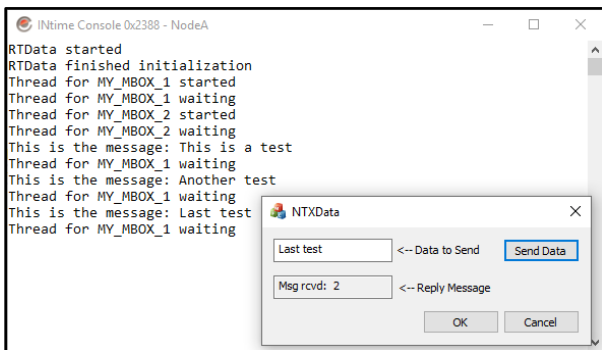
```

Running the Complete Solution

- 1) Start the INtime kernel.
- 2) Open the INtime Explorer, select the *Local* node, and press *OK*. INtex displays all processes running on the INtime kernel. It also shows any mailboxes associated with those processes.
- 3) Start *NTXData.exe* within Visual Studio by pressing the *F5* key. Recall that *NTXData* automatically loads and starts *RTData*.
- 4) After *NTXData* starts, locate *RTData* in the INtex process tree (remember to enable automatic refresh in the INtex options if you do not see *RTData* appear in the process list).
- 5) Expand the *RTData* process to see the data mailbox objects, *MY_MBOX_1* and *MY_MBOX_2*, and the INtime region object. (For Distributed RTOS, expand "Extended I/O System"|RINTM "INtime Run-Time Loader" first.)

- 6) Type something into the *IDC_DATA Edit Control* and click *Send Data*. Your message displays in the *RTData* console window, and the words *Msg rcvd: #* appears in the *IDC_rxDATA Edit Control*, where *#* corresponds to the message sequence number.

Figure 31: Running the Complete Solution



- 7) Close *NTXData* by pressing either *OK* or *Cancel*.

RTData continues to run, even though you closed *NTXData*, because we did not include any code to stop *RTData* when *NTXData* terminates.

Use *INtime Explorer* to shut down the *RTData* process by right-clicking the *RTData* process icon in the *INtime Explorer* window and selecting *Delete* from the context menu.

EXAMPLE #3 – Working with multiple INtime Nodes

Note:

To run this example in **INtime for Windows**, the host running the Windows and INtime SDK requires a four-core processor, or dual-core with Hyperthreading enabled.

To run this example in **INtime Distributed RTOS** configuration setup, the deployment host needs have a multi-core processor.

A key feature of INtime is the ability for processes to communicate with each other even when they run on different nodes. This communication uses the same methods – interaction with system objects – as between two processes running on the same node.

In this example we will use the same RTData.rta application built in the previous example, and create a new one, RTSend.rta, to replace the NTXData.exe application. We will use the same interface to the RTdata application, but from an INtime application using the console.

Tip: Example #3 uses “RT Data Project” shortcut for the “RTData” project (unchanged from Example #2) and the “RT Client Data Project” shortcut for the “RTSend” project in the sample projects directory.

The example goes through the following steps:

- a. Creating the RTSend application.
- b. Running the processes RTData and RTSend on the same node.
- c. Stopping the processes.
- d. Setting up a second INtime node.

- *INtime for Windows* configuration:

Note: This requires that the host has a four-core processor, or dual-core with Hyperthreading enabled.

- *INtime Distributed RTOS* configuration:

Note: This requires that the deployment host have a multi-core processor.

- e. Modifying the RTData application to enable the applications to run on separate nodes.
- f. Running the applications.

Creating the RTSend application

This real-time process looks for the RTData application and its mailboxes, prompts the user for the input string, and sends it. It then receives a reply.

- 1) Open Visual Studio, create a real-time project called *RTSend*, and place it in the *INtimeApps* directory you created in the *HelloWorld* example.
- 2) Choose *A full-featured application* from the INtime Application Wizard and click *OK* (leave *C++* unchecked for this example).
- 3) From the elements setup dialog, add a *Client Thread* (last element in the list).
- 4) Check the *Send to data mailbox* item (upper left), leave all other items unchecked, then click *OK* to return to the elements setup dialog.
- 5) Click *Finish* followed by *OK*. The wizard automatically generates real-time code templates.

The client thread in *Client1.c* sends messages to the RTData process via *MY_MBOX_1*, then receives the response from *MY_MBOX_2*.

Edit the code

Open *Client1.c*. Add the process and mailbox names, and modify *Client1*, adding the following code:

```
// Process and mailbox catalog names
char rt_RTData_process[] = "RTData";
char rt_my_mbx_1[] = "MY_MBOX_1";
char rt_my_mbx_2[] = "MY_MBOX_2";

#undef _MULTI_NODE_
// #define _MULTI_NODE_

void Client1(void *param)
{
    LOCATION    hLoc;
    RTHANDLE    hOtherRoot;
    RTHANDLE    hProcess = NULL_RTHANDLE;
    RTHANDLE    hDmbx;
    RTHANDLE    hRmbx;
    char        nodename[32];
    char        message[128];
    WORD        status;
    WORD        n_recvd;

#ifdef _MULTI_NODE_
    do {
        do {
            printf("Enter the name of the target node: ");
// Local node: NodeB
// Local node: ///NodeB
// Local node: intime:///NodeB
// Remote DRTOS name: rtoshost.mydomain.com/NodeB
// Remote DRTOS name: intime:///rtoshost.mydomain.com/NodeB
// Remote INtime name: host-NodeA.mydomain.com/NodeB
```

```

// Remote INtime name: intime://host-NodeA.mydomain.com/NodeB
    gets(nodename);
    hLoc = GetRtNodeLocationByName(nodename);
    if (BAD_LOCATION == hLoc)
        printf("Could not find location of node \"%s\"\n",
nodename);
    } while (BAD_LOCATION == hLoc);

    if ((status = GetRtNodeStatus(hLoc)) != E_OK) {
        printf("Node \"%s\" is not ready: %s\n", nodename,
GetRtErrorText(GetLastRtError()));
        continue;
    }
    hOtherRoot = GetRemoteRootRtProcess(hLoc);
    if (BAD_RTHANDLE == hOtherRoot) {
        printf("Could not get remote root process: %s\n",
GetRtErrorText(GetLastRtError()));
        continue;
    }
} while (BAD_RTHANDLE == hOtherRoot);
#else
    hOtherRoot = GetRtThreadHandles(ROOT_PROCESS);
#endif

    // TODO: adjust process and mailbox name
    // TODO: remove the next lines if the data mailbox was created
in this process
    hProcess = LookupRtHandle(hOtherRoot, rt_RTData_process, 5000);
    if (BAD_RTHANDLE == hProcess)
    {
        Fail("Cannot find data mailbox process");
        return;
    }

    // TODO: replace hProcess by NULL_RTHANDLE if the data mailbox
was created in this process
    // Look up MY_MBOX_1
    hDmbx = LookupRtHandle(hProcess, rt_my_mbx_1, 5000);
    if (BAD_RTHANDLE == hDmbx)
    {
        Fail("Cannot find data mailbox 1");
        return;
    }

    // Look up MY_MBOX_2
    hRmbx = LookupRtHandle(hProcess, rt_my_mbx_2, 5000);
    if (BAD_RTHANDLE == hRmbx)
    {
        Fail("Cannot find data mailbox 2");
        return;
    }

    // tell that this thread is alive
    gInit.htClient1 = GetRtThreadHandles(THIS_THREAD);

    // attempt to catalog the thread but ignore error
    Catalog(NULL_RTHANDLE, gInit.htClient1, "TClient1");

    while (!gInit.bShutdown)
    {

```

```

// TODO: put client code that must be repeated here
// prompt the user for a message
printf("\nType a message: ");
gets(message);

// send the message
if (!SendRtData(hDmbx, message, 128))
{
    Fail("Cannot send to data mailbox");
    break;
}
// receive a response message
n_recvd = ReceiveRtData(hRmbx, message, WAIT_FOREVER);
if (n_recvd == 0) {
    break;
}
printf("Received %u bytes: \"%s\"\n", n_recvd, message);
}

Fail("Failed to receive message from mailbox 2\n");

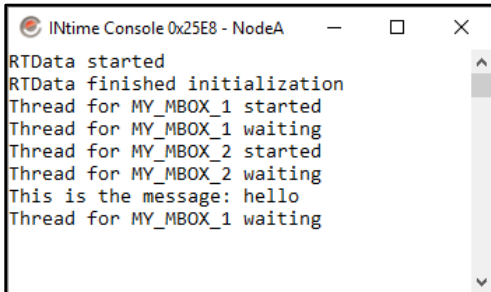
// tell that this thread is dead
gInit.htClient1 = NULL_RTHANDLE;
}

```

Running the solution

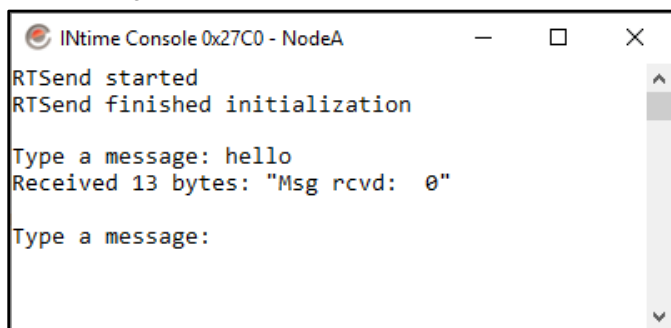
- 1) Start Node A:
 - With INtime for Windows host, start the local NodeA
 - With INtimeDistributed RTOS host, make sure the target node is booted. The default name is NodeA.
- 2) Start RTData.rta on the appropriate host, NodeA.
- 3) Start RTSend.rta on the same node on the same host.
- 4) At the prompt, type a message and observe the response when the message is returned.

Figure 32: RTData process console output.



Shown running in debug mode (with notifications) waiting for a message from the RTSend process. Then displaying the message that it received.

Figure 33: RTSend process console output.



Shown running in debug mode (with notifications), prompting (Type a message:) for message to be entered. Upon sending the message (hello) the application acknowledges receipt of the message and prompts for another message.

Adding a second node

So far, we have two cooperating processes running on the same node on the same host. Now we will create a second node on the same host and run the processes on different nodes on the same host.

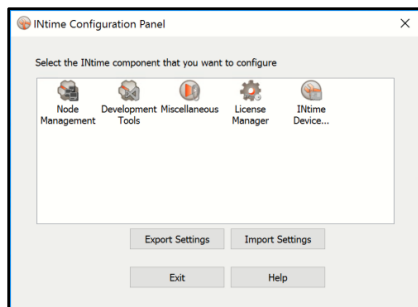
INtime for Windows: - Setting up a second Node

This section explains how to create a second node in INtime for Windows. (For INtime Distributed RTOS see the next section.)

This section requires that you have at least a four-core processor, or dual-core with Hyperthreading enabled. (1 Windows node and 2 INtime nodes minimum.)

Open the **INtime Configuration Panel|Node Management** applet.

Figure 34: INtime Configuration Panel



On the left side of the dialog, you see a map of all the known INtime nodes. Currently there is just one – NodeA – showing.

(1) NodeA has a dedicated processor core

(2) Click *New Node*, and

(3) Select Local and insert Node name “NodeB” or some other preferred name.

Click OK.

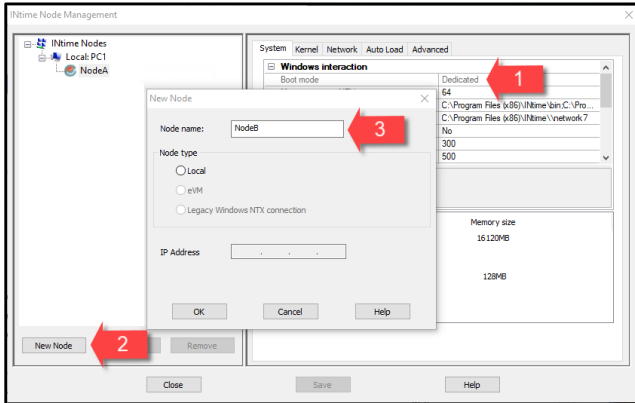
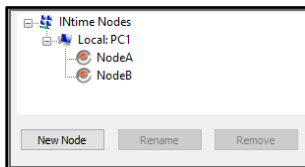


Figure 35: INtime Node Management applet

Reboot the host.

Check that you have two local Nodes (on the same host) running after rebooting by going to the **INtime Configuration Panel|Node Management** applet – left side window should show both local nodes.

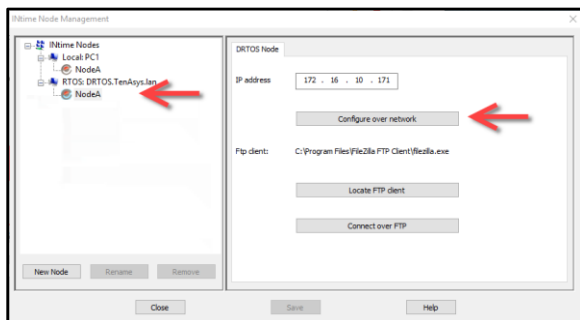
Figure 36: NodeA and NodeB are shown as local nodes.



INtime Distributed RTOS: - Setting up a second Node

Open the **INtime Configuration Panel|Node Management** applet, as in the previous section. This operation requires the INtime Distributed RTOS host to have a multi-core processor.

Figure 37: Configure Distributed RTOS

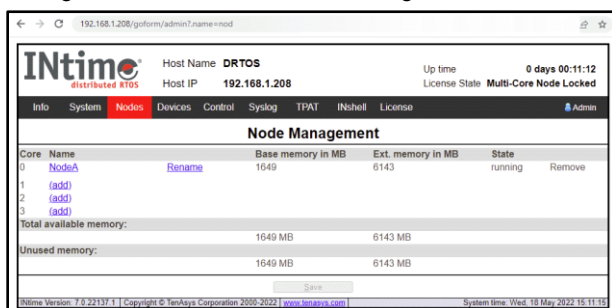


Select your node (“DRTOS.TenAsys.Ian/NodeA” in this case) and click “Configure over network”. A web browser appears.

Enter the password you assigned during installation.

Select the Nodes option from the bar.

Figure 38: Distributed RTOS configuration interface



Click the (add) link for one of the unused nodes, accept the defaults, and click OK.

Select the Control option from the bar and Reboot the Distributed RTOS host.

Modifying RTSend application for a second node

Edit the project once more and in Client1.c comment out this line:

```
///undef _MULTI_NODE_
```

And uncomment this line:

```
#define _MULTI_NODE_
```

Rebuild the RTSend application.

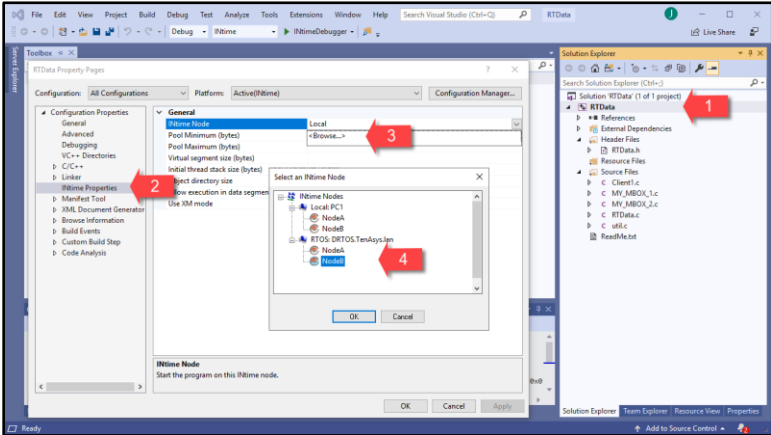
This adds a sequence of code which prompts for a node name, searches for it, and discovers its root process handle.

End of Adding a second node section

Running the complete solution with a second node

- 1) Start both INtime nodes on the same host.
- 2) In a second instance of Visual Studio open the RTData project. Set its target node to NodeA in the INtime Properties for the project: Right click **RTData** in the Solution Explorer, then Click **Properties**. Select **INtime Properties**, then **INtime Node** <browse>, and then select NodeA. Launch the application from within Visual Studio.

Figure 39: Selecting a Node within Visual Studio



- 3) In the first instance of Visual Studio, open the RTSend project. Set its target node to NodeB. Launch the application.
- 4) At the prompt for the target node name, enter "NodeA".
- 5) At the message prompt type a message and observe the response when the message returns.

The text output to the NodeA and NodeB console ports should be the same as the example running on the same Node, but with the addition of the target node query.

Tip: For INtime Distributed RTOS, use the Channel Select HotKey: ALT-TAB or ALT-SYSREQ to switch screen I/O between the nodes.

Tip: To use this example between nodes on two different hosts the INtime for Windows host needs the following:

- 1) a network bridge connecting the TenAsys Virtual Ethernet Adapter and the physical Ethernet Adapter
 - 2) gobs_net.rta running on the INtime for Windows node
- Start both from the Node Manager, Auto Load tab of NodeA.

(These are started automatically on a Distributed RTOS node.)

Example #4: The INScope Performance Analyzer

Determinism is a key attribute of real-time systems. Speed is always a useful attribute to have in any embedded system, but the ability to ensure the correct timing and sequence of events can be even more important. This is a key difference between a real-time system and a system that is simply fast.

The INScope performance analyzer is a software tool that provides you with precise information regarding the timing and sequence of real-time events in a multi-threaded application, so you can measure the determinism of your real-time process. INScope traces events while your application runs in real-time.

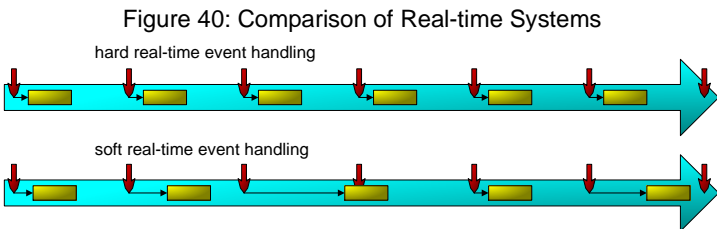
In this section we will use the INScope tool to monitor a multi-threaded real-time process.

Tip: Example #4 uses the “Multithread Sample” shortcut for the “MultiThread” project in the sample projects directory.

How Fast is Deterministic?

The deterministic nature of a real-time system forces a unique set of requirements upon software applications. A simple definition of a real-time system is one in which the time required to respond to an event is just as important as the logical correctness of that response. Hard real-time systems require the highest degree of determinism and performance. Typically, their worst-case event response requirements are measured in microseconds.

Bounded response to events is the key to defining a hard real-time system. Real-time systems require determinism to ensure predictable behavior of the system. Without determinism, a system cannot be called real-time, and, without bounded determinism, a system cannot be classified as *hard* real-time.



The specific degree of determinism required is a function of the frequency of the real-time events (size of the time interval between events) and the effect of delays on the dynamic characteristics of that system. That is, how often do events occur and how quick and repeatable must the system be in response to those events. Being able

to place a finite and acceptable bound on the value of these numbers is what distinguishes a *hard* real-time system from *soft* real-time systems.

Fast Does Not Equal Deterministic

Faster processors, memory, and peripherals improve the aggregate performance of a system, but they generally do not directly affect the bounded determinism of a system. The worst-case response time to an event may not be significantly changed by using a faster processor; increased speed can decrease the average jitter, the spread and intensity of the variations in response to an event, but it will not eliminate the worst-case jitter.

Improving the performance (or speed) of a real-time system is useful. More performance allows one to increase the complexity of the algorithms that can be implemented in a given period of time (i.e., within a sample interval or cycle). Therefore, the quality of the control and data acquisition system that one can implement in software is improved by using a faster system. However, bounded determinism is still needed to ensure that a stable and accurate system, regardless of the performance level, can be deployed.

A Multi-threaded Example

This example application contains three alarm threads, or fixed interval timing events. Two will be set for the same priority level, and the third will be set one priority level higher.

Tip: Complete the *HelloWorld* example before performing this example to familiarize yourself with the INtime development system.

- 1) Open Visual Studio.
- 2) Create an INtime project called *MultiThread* and place it in the *INtimeApps* directory you created for the *HelloWorld* example.
- 3) Select *A full-featured application* from the INtime application wizard dialog and click *OK*.
- 4) In the next dialog, add a *Thread that operates at a regular interval* element.
- 5) Change the *Method for waiting* parameter from *Sleep* to *Alarm*, change the *Number of microseconds* to wait from *1000* to *5000*, and change the *Thread Priority* from *170* to *160*. This creates *Poll1* as a thread that will start on a precise time interval of every five-thousand microseconds (every 5 milliseconds). Click *OK*.

- 6) Click on *Thread that operates at a regular interval* again. Change the *Method for waiting* parameter from *Sleep* to *Alarm*, leave the *Number of microseconds to wait* parameter at *1000*, and change the *Thread Priority* from *170* to *165*.

Figure 41: Modifying Thread Parameters

The screenshot shows a dialog box titled "Polling Thread 2" within the "INtime Application Wizard - MultiThread" window. The dialog is divided into two sections. The top section contains two parameters: "Method for waiting:" with a dropdown menu set to "Alarm", and "Number of microseconds to wait:" with a text input field containing "1000". The bottom section, titled "Thread Properties", contains two parameters: "Thread Priority: (Range: 0 - 254)" with a text input field containing "165", and "Stack Size in Kilobytes: (Min = 4, multiple of 4)" with a text input field containing "4". At the bottom right of the dialog are three buttons: "OK", "Cancel", and "Help".

This sets up *Poll2* as a thread that will be started by the INtime scheduler at a precise time interval of every one millisecond. Click *OK*.

- 7) Choose *Thread that operates at a regular interval* a third time. However, this time, specify the following parameters for the thread: *Method for waiting* is *Sleep*, *Number of milliseconds to wait* is *20* and *Thread Priority* is *170*.

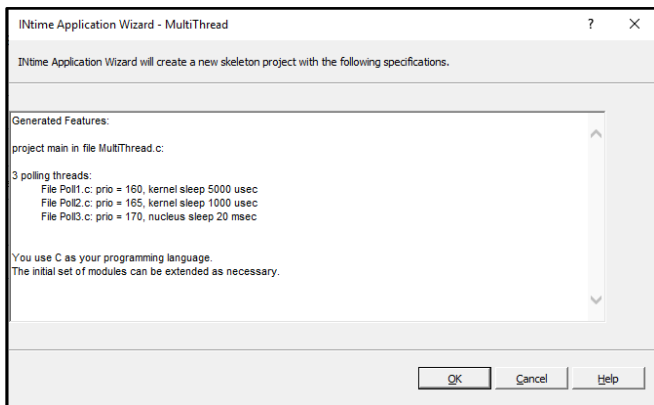
Figure 42: Modifying Thread Parameters

The screenshot shows a dialog box titled "Polling Thread 3" within the "INtime Application Wizard - MultiThread" window. The dialog is divided into two sections. The top section contains two parameters: "Method for waiting:" with a dropdown menu set to "Sleep", and "Number of milliseconds to wait: (Range: 0 - 655349)" with a text input field containing "20". The bottom section, titled "Thread Properties", contains two parameters: "Thread Priority: (Range: 0 - 254)" with a text input field containing "170", and "Stack Size in Kilobytes: (Min = 4, multiple of 4)" with a text input field containing "4". At the bottom right of the dialog are three buttons: "OK", "Cancel", and "Help".

This sets up *Poll3* as a simple delay thread, not a precise timer-based interval thread like the previous two threads. As a simple delay thread, *Poll3* will run approximately once every twenty milliseconds. The imprecision of *Poll3* is due to the variable amount of processing, especially by higher-priority threads that can occur between each sleep call.

- 8) Click *OK*. You now have three time-based threads.
- 9) Click *Finish* and double-check the summary screen to be sure it lists the following threads and parameters for those threads.

Figure 43: MultiThread Project Summary



- 10) If everything is fine, click *OK* at the summary screen; otherwise push the *Cancel* button, and create a *MultiThread* project that matches the parameters specified above.
- 11) After clicking *OK*, the wizard builds your project files. Three *Poll#.c* files are created. Each *Poll#.c* file corresponds to one of the three polling thread elements we created using the INtime application wizard.
- 12) Add the two global variables shown below to the beginning of *MultiThread.c* for communicating between our timing threads.

```
// global variables
DWORD      dwPoll11;
DWORD      dwPoll12;
RTHANDLE   hRootProcess;
DWORD      dwKtickInUsecs;
INIT_STRUCT gInit;
```

- 13) Remember to include external declarations in the header file *MultiThread.h* for the two global variables we added above.

```
// global variables
extern DWORD      dwPoll11;
extern DWORD      dwPoll12;
extern RTHANDLE   hRootProcess;    // RTHANDLE of root process
```

```
extern DWORD      dwKtickInUsecs; // length of one low level tick in
usecs
extern INIT_STRUCT gInit;        // structure describing all global
objects
```

- 14) *Poll1.c* and *Poll2.c* have nearly identical code. Make the following modifications to each of these files and be sure the variable specified after the `TODO` line matches the thread number.

```
while (!gInit.bShutdown)
{
    if (!WaitForRtAlarm(gInit.hAlarmPoll1, KN_WAIT_FOREVER)
    {
        Fail("Cannot wait for alarm Poll1");
        break;
    }
}

#ifdef _DEBUG
//      fprintf(stderr, "Poll1 waking up\n");
#endif

// TODO: do what has to be done every 5000 microseconds
++dwPoll1;
}
```

Note: The code immediately following the `while()` statement differs for each thread, as a function of the time interval and the sleep method specified when you used the wizard to generate the template code. Also, unlike the previous examples, in this example remove (or comment out) the `#ifdef _DEBUG` lines of code inside the `while()` statement; we do not want the `printf()` statements to interfere with the output and timing of these threads.

- 15) *Poll3.c* contains more code than the prior two. Make the following modifications to this file; again, make sure that the number specified in the `putchar('#')` line matches the thread number.

```
void Poll3(void* param)
{
    int i = 0;
    int x = 0;

#ifdef _DEBUG
    printf(stderr, "Poll3 started\n");
#endif

...intervening lines removed for brevity...

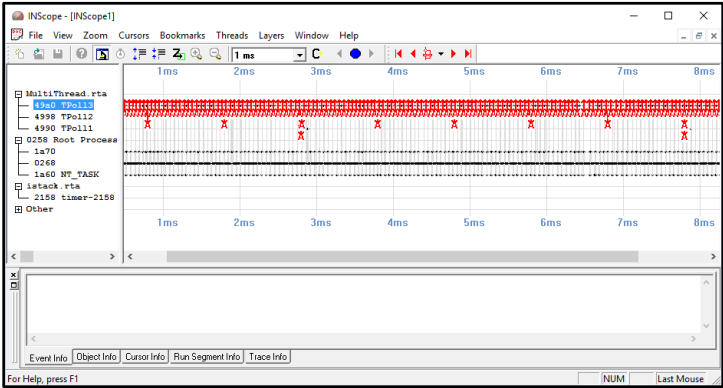
    while (!gInit.bShutdown)
    {
        RtSleep(20);

#ifdef _DEBUG
//      printf(stderr, "Poll3 waking up\n");
#endif

// TODO: do what has to be done every 20 milliseconds
for (i = 0; i < 10; i++) {
    putchar(0x0a);
}
```


- 6) In a few moments the *View Trace* button appears, indicating that the trace buffer is full. Click *View Trace*. The event trace for *MultiThread* appears in the INscope upper-right pane.

Figure 45: INscope Event Trace



The left pane lists the INtime processes that were running in the kernel when the trace started, and each of the threads running inside those processes. The name of the *MultiThread* executable file appears along with the three polling threads, also listed by name. The thread names appear courtesy of the `CatalogRtHandle()` calls. The exact length of time associated with the trace, and the order of the threads on the display, may differ from the figure above. The time it takes *Poll3* to run through the `while()` loop depends on the speed and configuration of your machine; remember that all INtime `printf()` statements (and other console I/O functions) go through Windows, which affects some of the timing in this example program.

Scrolling left to right you will see that *Poll1* and *Poll2* execute at precise five and ten millisecond intervals, but the timing of *Poll3* is variable.

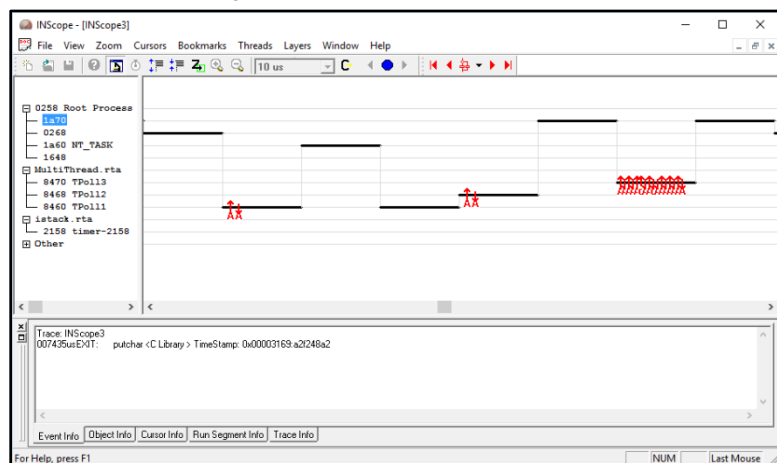
Note: INscope timestamps are derived from the processor's *Time Stamp Counter (TSC)*. If your host has a variable speed clock (such as a laptop with *SpeedStep*) the timing measurements within INscope may be inconsistent. For more information regarding this phenomenon, visit the TenAsys Knowledge Base at www.tenasys.com.

When all real-time threads are in an idle state the *NT_TASK* and *WIN_EXEC_TSK* threads run. These threads represent Windows, its drivers, applications, and the transfer of information between the INtime kernel and Windows; In shared mode CPU cycles are allocated to Windows only when all real-time processes are idle.

The exact set of Windows threads you observe with the INScope tool, and the rate at which those threads run, depends on the number of CPU cores in the host and how the host is configured. In a multi-core host INtime is configured to use one or more cores exclusively for real-time threads. All remaining CPU cores are allocated to Windows.

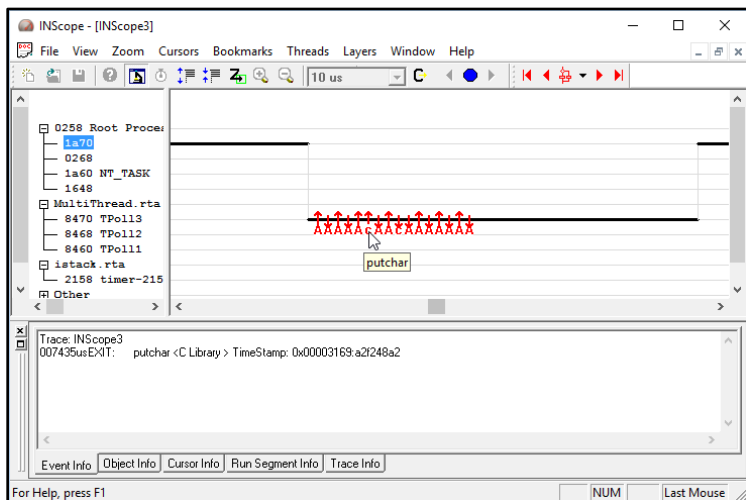
Use the zoom controls on the menu or toolbar to see the trace more clearly and inspect the task switches between threads. By depressing the 'Z' button on the toolbar and tracing a rectangular region with the mouse, you can zoom to a specific segment. In the screenshot below we can see all three threads running in a zoomed view.

Figure 46: Zoomed INScope Trace



Zoom in further on the Poll3 thread. To see close events, toggle to the view Event Based mode with the stopwatch icon.

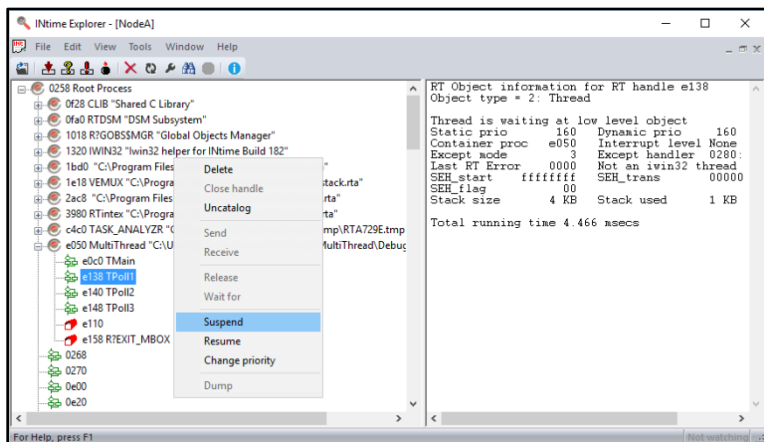
Figure 47: Event Based detail



Hover the mouse over one of the arrows on a 'C' event and you can see it is the `putchar()` function call made inside the `Poll3` `while()` loop. An up arrow is a return from a prior `putchar()` call, and the down arrow is a new call into `putchar()`. Since a `putchar()` call results in a transfer of data to the Windows side of the host, it forces `Poll3` into an idle state. Hovering over the 'A' events shows similar information for `Poll1` and `Poll2`. Right-click an event arrow and select the *Display Details* item that appears, and data regarding that event appears in the *Event Info* tab at the bottom of the screen.

An interesting and useful feature of INtime Explorer is the ability to suspend and resume threads on the fly.

Figure 48: INtEx View of the Multithread App



- 1) Right-click the *Poll2* thread icon in the INtEx Explorer process tree while *MultiThread* is running (expand the *MultiThread* process to see its individual threads).
- 2) Select *Suspend* from the context menu.
- 3) **Note** the change in the *MultiThread* console window.

The numbers at the end of each line of dots in the console window indicate how many times each of the two high-priority threads ran since the last time the low-priority thread ran. These high-priority threads can and will pre-empt the low-priority thread (as shown by the previous figure). If no number appears after the dots, it means zero precise timer events were detected. The numbers vary because the time to run *Poll3* varies in length.

- 4) Suspend *Poll1* and again watch the console window's output.
- 5) Suspend and resume any of the threads, including *Poll3*. Do the results match your expectation?

Next Steps

This guide introduces a variety of INtime development tools and features. The example applications were designed to help you become familiar with developing INtime real-time applications for Windows. The next step is to become familiar with the INtime architecture and API. See the online help and *User's Manual* for more detailed information about these subjects.

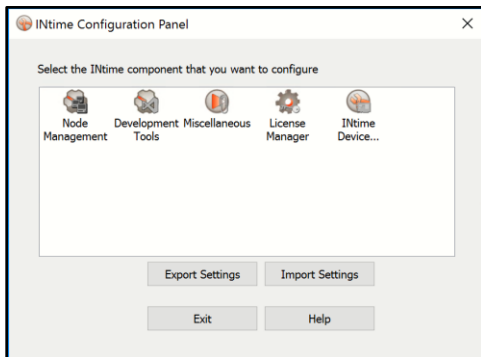
Once you are familiar with the INtime kernel architecture, you might want to review the sample real-time applications that were installed along with the INtime development package. Appendix B includes a list of the sample applications with their descriptions.

The final step is to review how to deploy INtime real-time applications. You have the option of creating real-time applications that share the hardware host with Windows, or stand-alone INtime nodes with INtime Distributed RTOS. For more information, see the documentation.

A. Configuring the *INtime for Windows Kernel* (local Node)

The INtime Configuration applet opened from the Windows Control Panel or the INtime Configuration Panel can be used to modify run-time parameters in the INtime kernel and the development environment. This appendix describes some of those parameters.

Figure 49: INtime Control Panel



Double-clicking an icon in the window starts the individual configuration application.

The *Export Settings* button can be used to save a configuration file from a reference machine that can then be applied to other machines (i.e., for use on a production line) using the *Import Settings* button. The *Export Settings* button will export a single INtime configuration file for all components that have been selected.

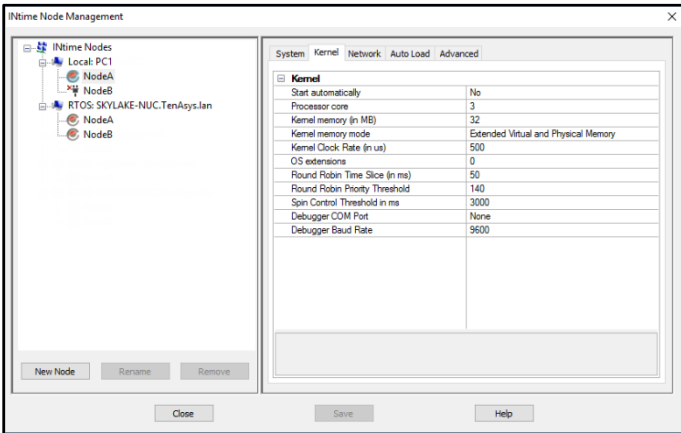
Tip: Hold the *Ctrl* key down while clicking the left mouse button to highlight multiple configuration components before exporting the INtime configuration file.

INtime for Windows Node Management

Use this configuration applet to select the best kernel timer rate for the INtime application. In the *MultiThread* example the fastest timer (or alarm) event we could specify was 500 microseconds, because that is the default rate at which the INtime kernel is configured. Changing the

Kernel Clock Rate to 100 microseconds would have allowed us to create threads that wake with 100 microsecond resolution.

Figure 50: Node Management Kernel Tab



Following are some useful details regarding this applet:

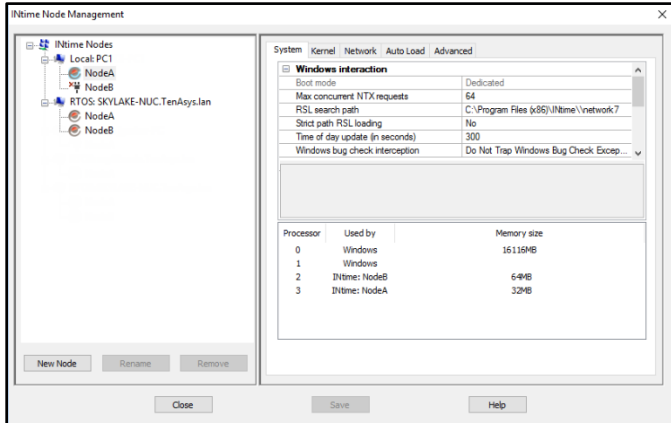
- *Kernel Clock Rate* specifies the number of microseconds that elapse between system clock interrupts. The default is 500 with a range of 100 to 10,000 microseconds.
- *Round Robin Priority Threshold* specifies the priority level at which threads will be scheduled to run using a round-robin schedule. The priority range is 128 to 254. Only threads with identical priorities that are *at or below* the *Round Robin Priority Threshold* are scheduled for round-robin operation.

Note: INtime priority levels are numbered from 0 to 254, where zero is the highest priority level in the system and 254 is the lowest. Thus, a priority level that is *at or below* the *Round Robin Priority Threshold* means a priority number equal to or higher than that specified as the *Round Robin Priority Threshold*.

- *Round Robin Time Slice* specifies the time allocated for a round-robin time slice. Values range from 20 to 100 milliseconds in multiples of 10 milliseconds.
- *Spin Control Threshold* and *AutoLoad Spin Control* (scroll the *Kernel Parameters* screen down to locate these items) specify the behavior of a special function of the INtime kernel that can be used to detect real-time threads that may be running without pause. In other words, it can be used to identify and stop misbehaving real-time threads that are “locking up” the system.

- **Kernel memory** specifies the total amount of physical memory allocated to the INtime real-time kernel. This is memory reserved exclusively for use by the INtime kernel and all real-time processes and threads. This memory is never paged and is, therefore, guaranteed to be deterministic.

Figure 51: Node Management System Tab



Settings which affect all nodes on this host are in the System Wide tab. Following are some useful details regarding this tab:

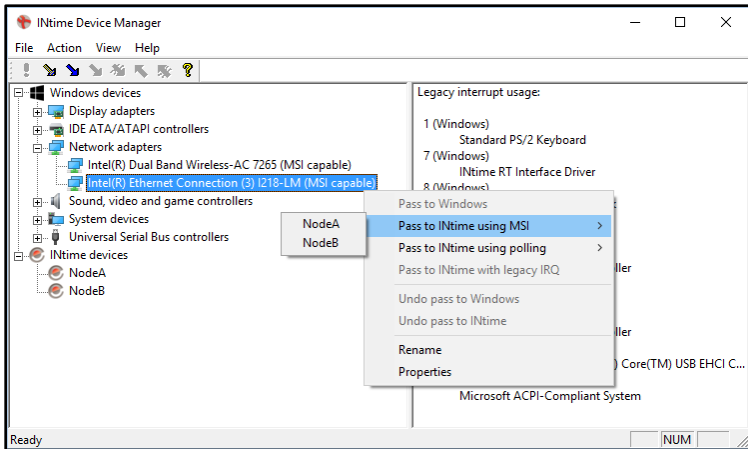
- **Boot Mode** specifies how INtime should allocate CPU resources between INtime and Windows: shared or dedicated.
 - Dedicated mode is required on Windows 8 forward. Dedicated means at least one core of a multi-core host is dedicated to an INtime kernel and all real-time applications. In this mode you may configure multiple INtime kernels on a multi-core host.

INtime for Windows Device Manager

Use this applet to allocate hardware device resources (especially interrupts) for use by an INtime kernel and the real-time applications. The applet presents a view of all hardware devices in the host, similar to that presented by the Windows Device Manager.

To remove this device from Windows and make it available to the real-time environment, right-click a device in the list of *Windows devices* and select *Pass to INtime* from the context menu.

Figure 52: Device Configuration applet



Passing a device to INtime results in Windows no longer recognizing and loading a device driver for that hardware. Your real-time applications now have exclusive access to the interrupt and hardware registers of that device. This process is needed to ensure that Windows drivers and applications do not interfere with your use of the device.

INtime includes support for MSI devices (Message Signaled Interrupts). If you have an MSI-capable device you can pass it to INtime, even if there is a potential legacy interrupt conflict with Windows, by right clicking the device, and selecting the *Pass to INtime using MSI*.

B. INtime for Windows Sample Applications

The following table describes the sample applications that are installed with the INtime host. These can be found in the *My Documents\INtime\Projects* folder of the user who installed INtime on the host.

Sample Application	Description
C and C++ Samples for Debugger	The C++ program demonstrates several components of the C++ language available to real-time applications, as well as basic classes, dynamic instantiation, operator overloading, and so on. It also shows the libraries and startup modules needed.
Distributed RTOS Configuration API Sample	Configure Distributed RTOS.
Global Objects Sample	Illustrates various aspects of global objects and node management.
Graphical Jitter Sample Project	Measures the minimum, maximum, and average times between low-level ticks using an alarm event handler (precise timer). This application is comprised of both real-time and Windows executables and illustrates use of the NTX API.
High Performance Ethernet Sample	Illustrates the use of the High Performance Ethernet drivers included with INtime.
HPE3 Extra Features Sample	Illustrates the additional features of the HPE3 interface with the Intel i210 Ethernet adapter.
Intel MKL Example	Shows how to use MKL in INtime.
INtime API Sample	Exercises most INtime software system calls.
INtime License Library Sample	For INtime for Windows and INtime Distributed RTOS.
INtimeDotNet Sample Application (VS2022)	Sample applications showing the use of the INtimeDotNet assembly for use in Windows applications that use the CLR to communicate to the RT side.
INtime Service Process Sample	A skeleton for creating a multi-process service.
Local Node Configuration Sample	Windows Console application showing read and write of INtime for Windows configuration.
Memory Heap Debug Sample	Demonstrates how to collect heap usage statistics as well as technique for trapping heap errors, invalid pointers, and memory leaks.
Message Queue Demo	Windows and INtime application using queues
Network Datagrams Sample	Examples of how to send unicast, multicast and broadcast datagrams.
Network Interface Information	Project showing how to find network interface information.

Sample Application	Description
NTX Sample Application (MsgBoxDemo)	This INtime application has both a Windows and a real-time portion. The Windows portion looks up a mailbox created by the real-time portion and waits at the mailbox. Whenever a real-time thread sends a message to the mailbox, the Windows portion displays the received data in a Windows message box. Also demonstrates use of semaphore and shared memory.
PCAP Sample Application	Illustrates the use of the PCAP library to filter specific Ethernet packets from the network stack.
RSL Examples	Demonstrates the creation and use of real-time <i>Shared Libraries</i> , the INtime analog of Windows DLLs.
Serial Communications Sample	This project demonstrates how to use the INtime Serial Communications library.
TCP Samples	Demonstrates TCP communications between a client and a server. Client and server code is provided for INtime and server code for Windows.
UDP Samples	UDP ping-pong sample application. Datagrams are exchanged between INtime and Windows.
USB Device List Utility	How to find all attached USB devices.
USB Keyboard Sample	Demonstrates how to use the INtime USB subsystem by monitoring a USB keyboard and printing a dump of each keystroke as it occurs.
Windows STOP Detection Sample	Shows how an INtime application can detect either a Windows crash (blue screen) or a Windows shutdown event and prevent Windows from completing its normal actions until the real-time application has had a chance to perform a “graceful” shutdown.
XCNT-HPE Sample (Ethernet connector)	Illustrates the use of the XCNT network driver to forward packets from an HPE application to and from the network stack.