tenAsys®

USER'S MANUAL

INtime® 4.0 Software

TenAsys Corporation
1400 NW Compton Drive, Suite 301
Beaverton, OR 97006 USA
+1 503 748-4720
FAX: +1 503 748-4730
info@tenasys.com
www.tenasys.com

31001-6
September 2009

September 2009

Before you begin

This guide describes INtime® software, both as an extension for Microsoft Windows or as a stand-alone RTOS (Real Time Operating System) running on an Intel architecture PC, that provides the tools you need to create and run real-time (RT) applications—robust, high-performance applications with predictable responses to external events.

This guide assumes that you know how to develop programs for Windows and understand RT system concepts.

Note

In this guide, the term “Windows” means any supported version of Windows. For a complete listing of supported Windows versions, see page 3.

About this guide

Guide contents

This guide introduces you to INtime software: how it makes RT applications possible and how to use the INtime development tools. Use this guide to get acquainted with INtime software, then refer to INtime Help for detailed information about INtime components. For more information about accessing help, see *Where to get more information* later in this chapter.

Note

For a quick start, read the following:

- *Chapter 1, Overview* to introduce you to all the basic INtime software concepts and to learn where to find detailed information about INtime software.
- *Chapter 10, INtime application development*, to learn about developing RT applications using INtime software.

Part I: Introducing INtime software

This part introduces INtime software and explains how INtime software and Windows work together to create RT applications.

Chapter		Description
1	Overview	Describes how INtime software works together with Windows to create and run RT applications, and lists INtime software's features. It also tells you where to find detailed information about INtime software topics.
2	Understanding INtime software architecture	Explains how INtime's RT kernel works with Windows to provide RT functionality. It also lists and describes INtime components.
3	About INtime software's RT kernel	Describes the RT kernel and its objects, the basic building blocks that application programs manipulate.
4	About RT programming	Describes processes unique to RT programming.
5	Designing RT applications	Provides general guidelines for RT system design.

Part II: Using INtime software

This part explains how to start INtime software and how to use the INtime software development tools.

Chapter		Description
6	Installation	Explains how to install and uninstall INtime software.
7	Configuration	Describes how to configure INtime software.
8	Connecting to an INtime host	Explains how to set up an NTX connection to an INtime host or a runtime system for remote debugging.
9	Operation	Describes how to start and run INtime software.

Part III: Appendices

The appendices provide additional information about INtime software.

Appendix		Description
A	INtime software system calls	Lists and describes system calls that threads in the RT portion of INtime applications use to communicate with each other and with Windows threads. You can find detailed information, including syntax and parameter values, in INtime Help.
B	The iwin32 subsystem	Describes the iwin32 subsystem, which provides a Win32 API for the INtime kernel. It is a parallel API to the INtime API that makes porting of existing Win32 applications easier.
C	INtime directory structure	Describes the INtime directory structure.
D	INtime software components	Lists and describes INtime software program files.

Appendix		Description
E	Visual Studio debugging for older INtime projects	Describes how to upgrade existing INtime projects to use the Visual Studio product and its debugger.
F	Adding INtime software to an XP Embedded configuration	Lists and describes how to add INtime components to a Windows XP Embedded development environment so you can produce XP Embedded images that include these INtime components.
G	Troubleshooting	Lists problems you may encounter while running INtime software, and explains how to avoid or resolve those problems.

Glossary

The glossary defines terms used to describe INtime software.

Notational conventions

This manual uses the following conventions:

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

Note	Indicates important information about the product.
Tip	Indicates alternate techniques or procedures that you can use to save time or better understand the product.
⚠ CAUTION	Indicates potentially hazardous situations which, if not avoided, may result in minor or moderate injury, or damage to data or hardware. It may also alert you about unsafe practices.

Where to get more information

About INtime software

You can find out more about INtime software from these sources:

- **World Wide Web**: TenAsys maintains an active site on the World Wide Web. The site contains current information about the company and locations of sales offices, new and existing products, contacts for sales, service, and technical support information. You can also send e-mail to TenAsys using the web site:

  `www.tenasys.com`

  You can contact TenAsys by email:

  `info@tenasys.com`

  You can contact TenAsys technical support by email:

support@tenasys.com

Note

When sending e-mail for technical support, please include information about both the hardware and software, including Windows and INtime versions, plus a detailed description of the problem, including how to reproduce it.

Requests for sales, service, and technical support information receive prompt response.

- **INtime Help**: Describes INtime software concepts and explains how to use INtime tools. INtime Help includes all system calls, including their syntax which you can cut and paste directly into your code. To access Intime Help, do one of these:
  - **Within Microsoft Visual Studio**: INtime content is integrated with the Visual Studio help collections. INtime content may be filtered with the keyword "INtime".
  - **Within source code**: Highlight a system call in your source code, then press F1. Help for that system call displays.
  - **Readme file**: Lists features and issues that arose too late to include in other documentation.
  - **Other**: If you purchased your TenAsys product from a third-party vendor, you can contact that vendor for service and support.

About Windows

For more information about Windows operation and program development, see these documents:

- Documentation that came with Windows.
- Documentation that came with Microsoft Visual Studio.

Contents

Part I **Introducing INtime software**

Chapter 1 **Overview**

How does INtime software work? ... 3
Running an INtime application in conjunction with Windows ... 4
Communication between Windows and RT threads ... 4
Considerations for INtime applications running on a single processor PC ... 5
Considerations for INtime applications running on a multiprocessor PC ... 6
Developing an INtime application ... 7
Design considerations ... 7
Code development ... 7
Features ... 8
Development environment ... 8
Wizards ... 8
Libraries ... 9
Debuggers ... 9
Sample applications ... 10
Runtime environment ... 12
RT enhancements to Windows ... 12
Memory protection ... 12
“Blue screen” protection ... 13

Chapter 2 **Understanding INtime software architecture**

Terminology ... 15
How INtime software and Windows work together to run RT applications ... 16
Transport mechanisms ... 17
About the OSEM ... 18
How the RT interface driver works ... 19
About the Windows HAL ... 20
About thread scheduling ... 21
Priority-based scheduling ... 21
Execution state ... 21
Round-robin scheduling ... 23
Handling interrupts ... 23
Interrupt handler alone ... 24
Interrupt handler/thread combination ... 24
Managing time ... 25

Chapter 3 **About INtime software’s RT kernel**

What does the RT kernel provide? ... 27
RT kernel objects ... 27
Threads ... 28

Processes 28
Virtual memory 29
Memory pools 30
Dynamic memory 30
Object directories 31
Exchange objects 32
Validation levels 32
Mailboxes 33
Semaphores 34
Regions 35
Priority inversions 35
Deadlocks 35
Ports 36
Services 36
Heaps 36
Global objects, references, and locations 36
Node architecture 37
New Objects 37
Location object 37
Reference object 38
Global objects 38

Chapter 4 About RT programming

Multi-threading 39
Preemptive, priority-based scheduling 41
Interrupt processing 42
Determinism 43
Multi-programming 44
Inter-thread coordination and communication 45
Messages 46
Synchronization 47
Mutual exclusion 47
Memory pools and memory sharing 48
Inter-node coordination and communication 49
System calls 50
Real time shared libraries 50
Exception handling 51
Fault Manager 52
Structured Exception Handling 53

Chapter 5 Designing RT applications

Define the application 55
Target environments 57
Methodology 57
A hypothetical system 58
Interrupt and event processing 59
Multi-tasking 59

Part II Using INtime software

Chapter 6 Installation

Install INtime software on a Windows system 63
Requirements 63
Before you begin 63
Running the Installation program 64
Installing hardware for use with the RT kernel 66

Chapter 7 Configuration

Configuring INtime software 67
Default configuration 67
Running the INtime Configuration Utility 68
Miscellaneous 69
RTIF.SYS driver 69
Interrupt resources 69
Configuring INtime applications 69
Configuring Windows for non-interactive logon 69
Configuring INtime Local Kernel service to execute automatically 70
Automatic loading of Realtime Applications 70
Configuring the INtime Network software 71
Before you begin 71
Hardware installation 71
Setting the TCP/IP configuration parameters 72
NIC driver configuration 72

Chapter 8 Connecting to an INtime host

Creating a connection to an INtime host 73
Fixed and Passive Connections 73

Chapter 9 Operation

Starting the RT kernel and related components 75
After you start the INtime kernel 76

Chapter 10 INtime application development

Create a project 80
Develop Windows source code 80
Adding the INtime RT Client Browser to your INtime application 80
Develop RT source code 82
Running the INtime Application wizard 82
Running the INtime process add-in wizard 84
Running the INtime Shared Library wizard 84
Running the INtime Static Library wizard 85
Compile 85
Visual Studio 2008 86
Visual Studio 2005 (aka Visual Studio 8) 88
Debug 89
Debugging tips 90
Performance monitor 90
Status messages 91

Prepare for release ... 92
Before you begin ... 92
Using launch-rt.exe ... 92
Sample INtime applications ... 93
EventMsg DLL Project ... 94
INtime API Sample ... 94
Serial Communications Sample ... 94
Graphical Jitter ... 94
Real-time Interrupt Sample ... 95
C and C++ Samples for Debugger ... 95
TCP Sample Applications ... 95
UDP Sample Applications ... 96
INtimeDotNet Sample Applications ... 96
Fault Handling (ntrobust) ... 96
Floating Point Exception Handling ... 96
RSL Examples ... 97
NTX Sample (MsgBoxDemo) ... 97
Windows STOP Detection sample (STOPmgr) ... 97
USB Client sample ... 97
Global Objects sample project ... 98
High-Performance Ethernet (HPE) sample project ... 98
PCAP Sample application ... 98

Part III Appendices

Appendix A INtime software system calls

System call types ... 102
NTX calls ... 103
Handle conversion ... 103
RT calls ... 104
High-level (validating) calls ... 104
Low-level (non-validating) calls ... 104
RT services ... 105
RT system calls ... 105
Distributed System Management (DSM) ... 105
NTX calls ... 105
High-level calls ... 105
Exception handling ... 106
High-level calls ... 106
Global objects ... 106
Interrupts ... 107
High-level calls ... 107
Mailboxes ... 108
NTX calls ... 108
High-level calls ... 108
Low-level calls ... 108
Memory management ... 109
Object directories ... 111

Ports 112
Service support 112
Port object management 113
Message transmission 113
Processes 114
Regions 114
Scheduler 115
Semaphores 115
Status 116
System data 117
Threads 117
Time management 118
Structures 119
Heaps and memory pools 120
High-performance gigabit Ethernet 123
INscope calls 124
Network stack 124
PCI library calls 124
Real-time shared library (RSL) calls 125
Registry calls 125
RT services and device drivers 126
RT service calls 126
RT service handlers 127
Serial Communications (COMM) 128
TCP/IP system calls 130
USB calls 130
INtimeDotNet calls 131
Input/Output Calls 133

Appendix B The iwin32 subsystem

Handles 135
Named objects 136
Processes 136
Threads 137
Mutexes 139
Critical section 139
Semaphores 140
Events 140
Shared memory 141
Timers 141
I/O handling 142
Interrupt handling 142
Registry handling 143
Miscellaneous 144

Appendix C INtime directory structure 147

Appendix D INtime software components

Blue.exe (Windows crash program) 152

Clk1Jitr.rta 152
EventMsg.dll 152
INconfCpl.cpl 152
INtime.chm 153
Main Help files 153
Utility Help files 153
C++ Help files 154
INscope.exe 154
INtex.exe 155
INtime local kernel (INtime.bin) 155
INtime remote kernel (Remote.bin) 155
INtime Visual Studio project type packages 155
INtime Performance Monitor (INtmPerf.* files) 155
INtime RT Client Browser 156
iWin32 header files 157
iWin32 interface library 157
iWin32x header files 157
iWin32x interface library 157
Jitter.exe 157
LdRta.exe (INtime RT Application Loader) 157
LoadRtk.exe (INtime Kernel Loader) 158
mDNSINtime.exe 159
MFC*.dll files 159
network7 utility files 159
NTX header files 159
NTX import libraries 159
NTX DLLs 160
NtxRemote2.exe (INtime Remote Connection Manager) 160
OvwGuide.pdf 160
Project files 160
Quick Start Guide 161
RT header files 162
RT interface libraries 162
RT Stack Services 162
RT USB Interface Drivers 163
RtClkSrv.exe (INtime Clock Synchronization Service) 163
RtDrvrW5.awx (RT Device Driver wizard) 163
RtELServ.exe (INtime Event Log Service) 164
RtIf.sys (RT Interface Driver) 164
RtIOCons.exe (INtime I/O console) 165
RtIOSrv.exe (INtime I/O Service) 165
RtNdSrv.exe (INtime Node Detection Service) 166
RtProcW5.awx (RT Process wizard) 166
RtProcAddinW5.awx (RT Process Add-in wizard) 166
RtRegSrv.exe (INtime Registry Service) 167
RtRslWiz.awx (RT Shared Library wizard) 167
Spider.exe (INtime standalone debugger) 167

Appendix E **Visual Studio debugging for older INtime projects**
Upgrading from Visual Studio 6.0 to newer Visual Studio 169
Converting to a .intp project 169
Setting project properties 170
Getting to work with the debugger 171
What if conversion did not work? 171
Upgrading from Visual Studio 2003 to newer Visual Studio 171
Appendix F **Adding INtime software to an XP Embedded configuration** 173
Appendix G **Troubleshooting**
Do a quick check 176
Look for symptoms 176
Other resources 180
Glossary 181
Index 187

Figures

Figure 1-1. Transport mechanism for NTX communication 4
Figure 1-2. Transferring control between Windows and INtime software's RT kernel 5
Figure 1-3. Control flows in a dedicated multiprocessor configuration 6
Figure 1-4. Creating INtime applications 8
Figure 2-1. How Windows threads and RT threads communicate with each other on an INtime node 16
Figure 2-2. How NTX communicates with other INtime hosts 17
Figure 2-3. Encapsulating Windows processes and threads into an RT thread 19
Figure 2-4. Execution state transitions for threads 22
Figure 2-5. Round-robin scheduling 23
Figure 2-6. Thread execution model 25
Figure 3-1. Processes in a process tree 29
Figure 3-2. Threads using their process's memory pool 30
Figure 3-3. Threads using the root process's object directory 31
Figure 3-4. Threads using an object mailbox 33
Figure 3-5. Threads using a semaphore for synchronization 34
Figure 3-6. Global object architecture 37
Figure 4-1. Thread switching in a multithreading environment 40
Figure 4-2. Multithreading and preemptive, priority-based scheduling 41
Figure 4-3. Interrupt handler interrupting a thread 42
Figure 4-4. Multiprogramming 44
Figure 4-5. Resources in a process 45
Figure 4-6. Object-based solution for message passing 46
Figure 4-7. Threads that use a semaphore for synchronization 47
Figure 4-8. Multithreading and mutual exclusion 48
Figure 4-9. Dynamic memory allocation between threads 49
Figure 4-10. Fault Manager Dialog 52
Figure 5-1. Typical development cycle for INtime applications 56
Figure 5-2. The hardware of the dialysis application system 58
Figure 6-1. Installing INtime software 65
Figure 7-1. INtime Configuration Panel 68
Figure 10-1. Developing an INtime application 79
Figure A-1. Converting NTXHANDLES to RTHANDLES 103
Figure G-1. Troubleshooting INtime software problems 175

Tables

Table 9-1. INtime software's Windows services 75
Table 9-2. INtime software tools 77
Table 10-1. INtime program directory 147
Table G-1. Symptom table 176
Table G-2. Solution table 177

I

Introducing INtime software

This part acquaints you with INtime software: its components, how they're put together, and how they work with Windows to run real-time applications.

This part contains:

Chapter 1: Overview

Describes how INtime software works together with Windows to create and run real-time applications, and lists INtime software's features. It also tells you where to find detailed information about INtime software topics.

Note
Read this chapter first. It introduces you to all the basic INtime software concepts and tells you where to find detailed information.

Chapter 2: Understanding INtime software architecture

Explains how INtime's real-time kernel works with Windows to provide real-time functionality. It also lists and describes INtime components.

Chapter 3: About INtime software's RT kernel

Describes the real-time kernel and its objects, the basic building blocks that application programs manipulate.

Chapter 4: About RT programming

Describes processes unique to real-time programming.

Chapter 5: Designing RT applications

Provides general guidelines for real-time system design.

1 Overview

INtime software extends Windows to provide the tools you need to create and run real-time (RT) applications. INtime software consists of:

- **Development environment**: tools you use to create RT applications that run in the INtime runtime environment in conjunction with Windows or on a PC running only the INtime RTOS.
- **Runtime environment**: additions to your Windows system that provide an RT platform for INtime applications.

This chapter describes how INtime software works together with Windows to create and run INtime applications, and lists INtime software features. It also tells you where to find detailed information about INtime software.

How does INtime software work?

You install INtime software on a system that already runs Windows. Once installed, Windows and INtime software work together to provide deterministic, RT support for INtime applications.

INtime software works with the following versions of Windows:

- Windows XP, Service Pack 2 or later
- Windows XP Embedded
- Windows Server 2003 and Windows Server 2003 Release 2
- Windows Vista
- Windows 7

For further details of supported Windows version, see the release notes.

Running an INtime application in conjunction with Windows

> **Note**
>
> For detailed information about how INtime software and Windows work together to run INtime applications, see *Chapter 2, Understanding INtime software architecture.*

An INtime application includes these components:

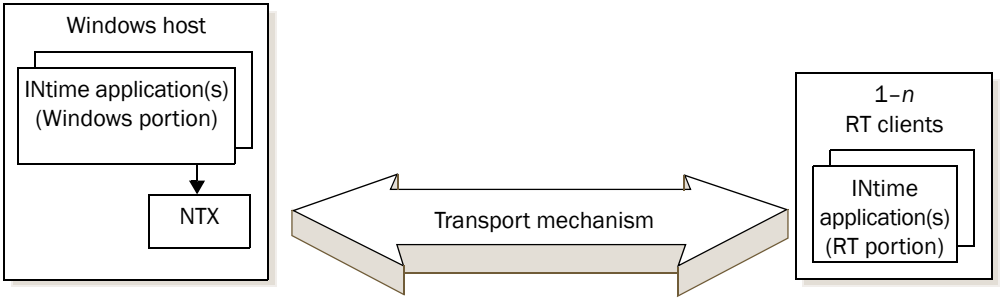
- **RT processes**: RT processes contain threads that typically handle time-critical I/O and control. Where Windows and RT processes share a CPU, RT threads preempt Windows threads.
- **Windows processes**: Windows processes contain threads that handle aspects other than time-critical I/O and control, including the user interface, network communication, data manipulation and computation, and data storage.

Communication between Windows and RT threads

When an INtime application runs, Windows threads communicate with RT threads via the Windows extension (NTX) API.

The RT threads in your INtime application(s) may reside on the same PC as the Windows threads or in a remote computer accessed via Ethernet cable. The NTX API automatically detects the connection type and determines the transport mechanism to use between Windows and RT threads: in-memory (local) or Ethernet:

Figure 1-1. Transport mechanism for NTX communication

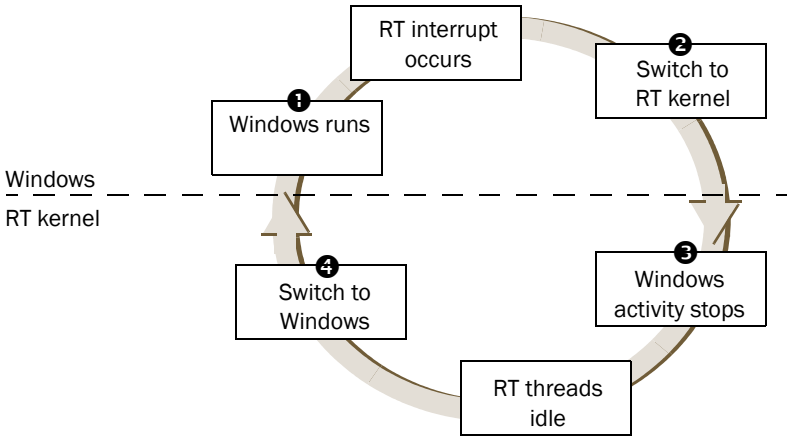


> **Note**
>
> For detailed information about INtime software's transport mechanisms, see *Transport mechanisms* in *Chapter 2, Understanding INtime software architecture*.

Considerations for INtime applications running on a single processor PC

When both the Windows and RT portions of an INtime application run on a single CPU hardware thread, INtime software transfers control between the Windows and RT environments as shown in this figure:

Figure 1-2. Transferring control between Windows and INtime software's RT kernel



❶) When a Windows thread runs, the full Windows environment exists, including its interrupts, interrupt masks, and handlers.

❷) When an RT interrupt occurs, control immediately switches to the RT kernel, where an RT interrupt handler deals with the event. This, in turn, may cause one or more RT threads to execute.

❸) Windows processes and interrupts stop until the RT threads complete.

❹) When all RT threads complete their work, leaving no RT threads ready to run, control switches back to the Windows environment, and standard Windows scheduling resumes.

When running on a single microprocessor, the INtime runtime environment encapsulates all Windows processes and threads into a single RT thread of lowest priority. As a result, RT threads always preempt running Windows threads, guaranteeing determinism for RT activities within the system.

The RT and Windows threads can share sections of memory allocated by INtime applications. A Windows thread can obtain a handle for this shared memory, then map the memory referenced by that handle into the thread's address space.

Note
For detailed information about memory usage, go to INtime Help and select About INtime software, RT kernel objects, then Memory Management.

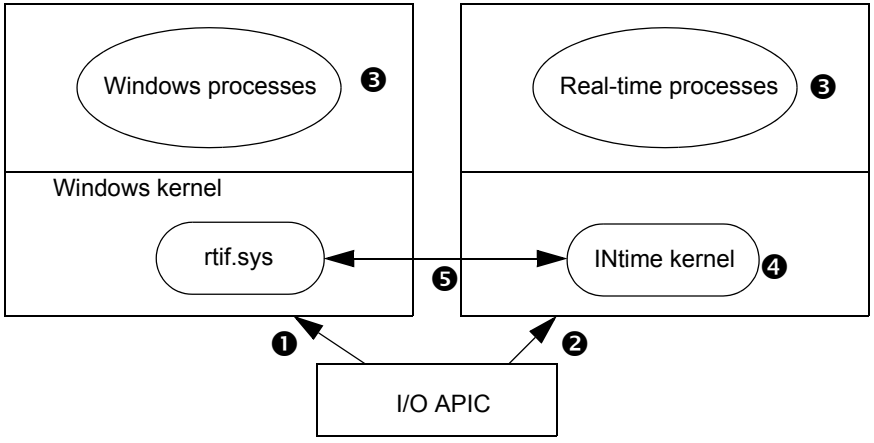
This INtime operating environment is referred to as Shared Mode.

Considerations for INtime applications running on a multiprocessor PC

When Windows and INtime run on a multiprocessor PC, by default the INtime kernel and Windows share one CPU and Windows uses the others. The shared hardware thread behaves in the same way as the previously mentioned single-CPU case.

INtime software may alternatively be configured such that an entire CPU hardware thread may be dedicated to the INtime kernel. In this case the architecture is rather different, as shown in this figure:

Figure 1-3. Control flows in a dedicated multiprocessor configuration



❶) When a Windows interrupt occurs, the I/O APIC delivers the interrupt to only the Windows CPU.

❷) When an RT interrupt occurs, the I/O APIC delivers the interrupt to only the INtime CPU.

❸) Windows processes are never preempted by real-time interrupts and processes and vice-versa, because each OS has a dedicated CPU.

❹) When all RT threads complete their work, leaving no RT threads ready to run, the INtime CPU executes an idle task until the next real-time interrupt occurs.

❺) The Windows and INtime kernels signal using IPIs (Inter Processor Interrupt) and shared memory.

On multiple processors configured so that INtime has a dedicated processor, the RT kernel does not need to encapsulate the Windows system in the same way because there are separate processors for each OS.

In both cases, RT and Windows processes can share sections of memory allocated by INtime applications. A Windows thread can obtain a handle for this shared memory, then map the memory referenced by that handle in Shared Mode into the thread's address space.

From version 4.0 it is possible to configure INtime software on a multicore system so that multiple cores each have an instance of the RT kernel. In this case each RT kernel instance has a dedicated hardware thread using a single chunk of memory, neither of which are shared with Windows or another instance of the RT kernel.

Developing an INtime application

Design considerations

Note

For detailed information about designing INtime applications, see *Chapter 5, Designing RT applications*.

When designing INtime applications, you must divide the labor appropriately between Windows processes and RT processes and, to a finer degree, between the threads in each process. For the best performance, limit RT processes to performing only time-critical functions, and determine which Windows threads require the greater relative priority.

Code development

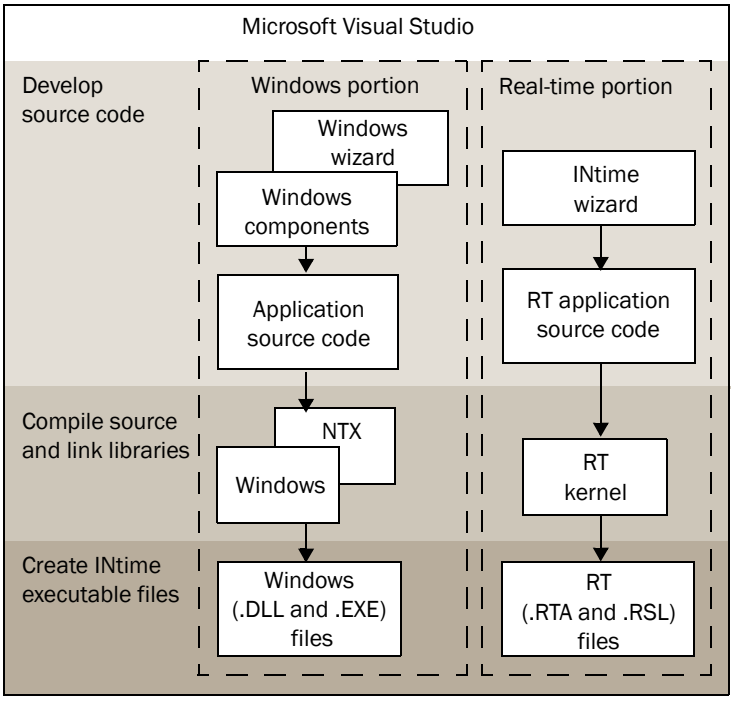
Note

For detailed information about developing INtime applications, see *Chapter 10, INtime application development*.

To develop an INtime application, you use Microsoft Visual Studio, including INtime wizards and Microsoft Visual Studio extensions for RT processes, a standard Windows debugger, and a Windows-based RT dynamic debugger that supports on-target debugging of RT threads. INtime includes a debugger which integrates with the Visual Studio debugger. This integration supports the Visual Studio versions from 2005 onward.

When developing INtime applications with Microsoft Visual Studio, you create executable files for both RT and Windows environments as shown in this figure:

Figure 1-4. Creating INtime applications



Features

Note

For detailed list of INtime software components, see *Appendix D, INtime software components*. For information about using these components, see *Chapter 10, INtime application development*.

Development environment

To develop INtime applications, you use standard Windows tools together with these INtime software tools:

Wizards

Accessed from within Microsoft Visual Studio, INtime wizards automatically prompt you for the information needed to create projects that contain source code for the RT portion of your INtime applications. Once you create the project, you manually edit the code.

INtime software provides these wizards:

- **Application wizard**: develops the RT portion of INtime applications.
- **Shared Library wizard**: develops RT shared library (RSL is the RT equivalent to Windows DLL).
- **Static Library wizard**: develops an RT static library which you can link to other RT applications.

Note

For information about using the INtime wizards, see *Chapter 10, INtime application development*.

Libraries

INtime software provides interface libraries that your threads use to obtain RT kernel services. INtime software libraries include:

- **Windows extension (NTX) library**: Contains system calls that the Windows portion of an INtime application uses to communicate with the RT portion of the system.
- **Real-time (RT) application library**: Contains system calls that the RT portion of an INtime application uses to access RT kernel services such as memory management and inter-thread communication.
- **Real-time (RT) DSM library**: Contains system calls that implement sponsorship and dependency registration of INtime RTAs (real-time applications) with their counter-part Windows applications.
- **Real-time C and C++ libraries**: Contains system calls that the RT portion of an INtime application uses to access standard ANSI C and C++ functions.
- **PCI library**: Contains system calls that provide access to the PCI bus configuration space.
- **iWin32 library**: Contains system calls which emulate a subset of the Win32 API.
- **Windows iWin32x library**: Contains system calls that the Windows portion of an INtime application uses to access real-time objects created using the iWIn32 library.

Note

For an overview of system calls included in the APIs, see *Appendix A, INtime software system calls*. For detailed information, including syntax and parameter values, see Help.

Debuggers

You debug the Windows portion of INtime applications using the debug tools provided in Microsoft Visual Studio. To debug the RT portion of INtime applications, you use the debug tools provided with INtime software:

- **Visual Studio debugger**: The INtime software debug engine is integrated with the Visual Studio 2005 or later IDE to provide debugging capabilities from within Microsoft Visual Studio. This is the recommended default debug environment.
- **Spider debugger (SPIDER.EXE)**: A Windows application that provides source level, multi-tasking debug capabilities. Spider can debug multiple RT threads simultaneously while other threads continue to run. Use the Spider debugger when you must be able to debug a thread in a process when other threads continue to run.
- **System debug monitor (SDM)**: A command-line interface for RT applications that provides low-level, static debugging capability.
- **System Debugger (SDB.RTA)**: An extension to SDM which provides information about RT kernel objects, threads, and processes.

You can simultaneously debug the Windows and RT portions of an INtime application.

Note

For detailed information about Spider, see Spider Help. For detailed infomration about SDM, see INtime Help. For detailed information about using SDM, access INtime Help, then select Debuggers>Low-level debugger>System Debug Monitor (SDM).

Sample applications

INtime software contains several sample applications that you can use as examples for your own program development.

- **EventMsg DLL Project**: This DLL allows you to customize event messages.
- **INtime API Sample**: This test application exercises most INtime software system calls.
- **Serial Communications Sample**: This project demonstrates how to use the INtime Serial Communications library. The library and accompanying drivers allows the user to access serial devices such as the COM PC ports, RocketPort multi-channel PCI devices, and Edgeport multi-channel USB devices.
- **Graphical Jitter**: This application measures the minimum, maximum, and average times between low-level ticks via an Alarm Event Handler. Because this application is made from both an RT and a Windows executable, it shows both INtime and INtimeDotNet API usage.
- **Real-time Interrupt Sample**: This application tests the INtime RT Interrupt system calls using the Transmitter Ready interrupt from COM1.
- **C and C++ Samples for Debugger**: These simple C and C++ programs are provided as a vehicle to demonstrate the Spider debugger's capabilities. The C++ program also demonstrates several components of the C++ language available to RT

applications, as well as basic classes, dynamic instantiation, operator overloading, and so on. It also shows the libraries and startup modules needed.

- **TCP Sample Applications**: Sample project that demonstrate TCP communications between a client and server. Client and server code is provided for INtime, and server code for Windows.
- **UDP Sample Applications**: Sample project that demonstrate a UDP ping-pong type application. Datagram packets are exchanged between INtime and Windows with an incrementing identifier in the payload.
- **INtimeDotNet Sample Applications**: Sample INtimeDotNet applications that demonstrate NTX communication via the INtime DotNet assembly.
- **Fault Handling (ntrobust)**: This INtime application has both a Windows and an RT portion. The Windows portion allows the user to set up timing parameters that control how often a thread in the RT portion causes a hardware fault. The application demonstrates how another RT thread can detect and log the failure, delete the offending thread, and recreate it, all without affecting Windows or other RT processes.
- **Floating Point Exception Handling**: This simple program demonstrates floating point exception handling.
- **RSL Examples**: These RT programs demonstrate the creation and use of RT Shared Libraries, the RT analog for Windows DLLs.
- **NTX Sample (MsgBoxDemo)**: This INtime application has both a Windows and a RT portion. The Windows portion looks up an RT mailbox created by the RT portion, and then waits at the mailbox. When an RT thread sends a message to the mailbox, the Windows portion displays the received data in a message box on the Windows side. RT semaphore and RT shared memory usage are also demonstrated.
- **Windows STOP Detection sample (STOPmgr)**: This sample application shows how an INtime application can detect either a Windows Crash (blue screen) or Windows Shutdown event and prevent Windows from completing its normal actions until the RT application has had a chance to do a “graceful” shutdown.
- **Global Objects sample project**: This project illustrates some aspects of the Global Objects feature of INtime. and how they are used.
- **High-Performance Ethernet (HPE) sample project**: This project illustrates the use of the HPE drivers included with INtime.
- **PCAP Sample application**: This project illustrates the use of the PCAP library to filter specific Ethernet packets from the network stack.

- **USB Client sample**: This sample application demonstrates how to use the INtime USB subsystem. It monitors a USB keyboard and prints a dump of each keystroke as it occurs.

> **Note**
> For detailed information about these sample applications, see *Chapter 10, INtime application development*.

Runtime environment

INtime's runtime environment includes RT enhancements to Windows, memory protection, and blue screen protection. Runtime features are described in detail in the following sections.

RT enhancements to Windows

These features enable Windows and one or more instances of the RT kernel to work together in the same system:

- **RT kernel**: provides deterministic scheduling and execution of RT threads.
- **OS encapsulation mechanism (OSEM)**: manages the simultaneous operation and integrity of the Windows kernel and the RT kernel where they occupy the same hardware thread.
- **RTIF**: Windows Driver that enables NTX interface between Windows threads and RT threads, and reserves system memory for exclusive use by each RT kernel instance. Also, in Shared Mode, intercepts certain HAL functions that ensures determinism of INtime applications running on a Windows host.

> **Note**
> For information about the RT kernel, see *Chapter 3, About INtime software's RT kernel*. For information about the OSEM and HAL, see *Chapter 2, Understanding INtime software architecture*.

Memory protection

During INtime node system initialization, memory is allocated by RTIF for use by the RT kernel and INtime applications. This memory is "locked down" so that it does not page to disk. This memory is either removed from the non-paged memory pool available for Windows applications, or allocated from memory that has been excluded from Windows use.

INtime's RT kernel provides several protection levels for RT memory:

- **32-bit segmentation**: INtime software keeps Windows and each RT process in separate address spaces. Keeping Windows from the RT kernel isolates and protects

addresses not only between complex RT processes but between RT processes and Windows processes.

- **Paging**: The RT kernel uses the processor's paging mode for virtual address translation, but does not implement demand paging. Each RT process loads into its own virtual address space, defined by a 32-bit virtual segment. Because code, data, and stack are automatically placed in non-contiguous areas of the application's virtual memory, memory overruns are trapped as page faults.
- **Virtual addressing**: Since each RT process resides in a separate memory space defined by a virtual segment created by the RT Application Loader, RT processes cannot address beyond the virtual segment. This effectively partitions every RT process into its own address space.

"Blue screen" protection

On an INtime node, the RT kernel enables successful execution of RT threads even in the event of a total Windows failure, also known as a "blue screen crash."

- **Failure diversion**: The RTIF driver captures Windows failures. Once captured, control transfers to the RT kernel, Windows operation suspends and RT threads continue to run, unaffected by the failure.
- **Application-specific recovery**: In the event of a Windows failure, your crash recovery RT threads run and you can execute an orderly shutdown of the hardware your INtime application controls.

  In the event of a Windows blue screen crash, INtime software keeps running until a graceful shutdown occurs. To start INtime software again, you must first restart Windows.

2 Understanding INtime software architecture

This chapter explains how the RT kernel works in conjunction with Windows to provide real-time functionality. It also lists and describes INtime components.

Terminology

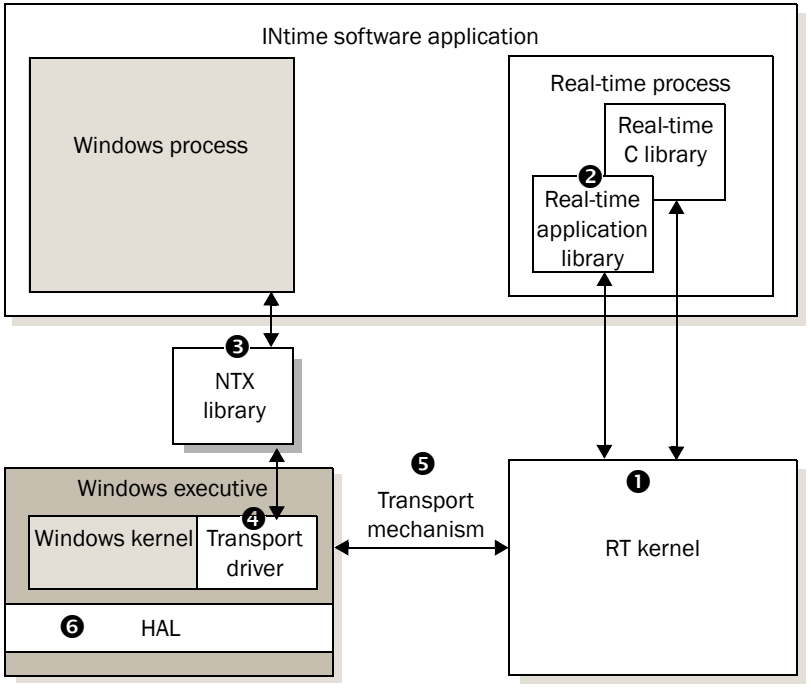
Some commonly-used terms are described here:

- **Host**: A computer consisting of one or more processing elements (cores or hardware threads).
- **Node**: An instance of an operating system. A node may be on its own on a host, or one of several on a multi-core host, or sharing a host with Windows.
- **Windows node**: An instance of the Windows operating system, whether running on a single or multiple hardware threads.
- **Node scope**: Accessible by all processes on a given node.
- **Remote node**: A node other than the node where the current process is running.
- **Host scope**: Accessible by all processes on all nodes of a given host.
- **Universal scope**: Accessible by all processes on all nodes.
- **Location**: A handle which uniquely identifies a node.

How INtime software and Windows work together to run RT applications

When an INtime application runs on an INtime node, Windows threads communicate with RT threads via the Windows extension (NTX) library as shown in this figure:

Figure 2-1. How Windows threads and RT threads communicate with each other on an INtime node



The INtime components include:

❶ **RT kernel**: Provides deterministic scheduling and execution of RT threads within RT processes. For detailed information about the kernel, see *Chapter 3, About INtime software's RT kernel*.

❷ **Real-time application, C, and C++ libraries**: Gives direct access to the RT kernel services for RT threads. For an overview of calls in the RT libraries, see *Appendix A, INtime software system calls*. For detailed information on all calls, including syntax and parameter values, see INtime Help.

❸ **NTX library**: Provides RT interface extensions for the Win32 API that allow Windows threads to communicate and exchange data with RT threads within the application. For an overview of calls in this library, see *Appendix A, INtime software system calls*. For detailed information, including syntax and parameter values, see INtime Help.

❹ **Transport driver**: A driver that converts information to the protocol needed by the specified transport mechanism. For details, see *Transport mechanisms* later in this chapter.

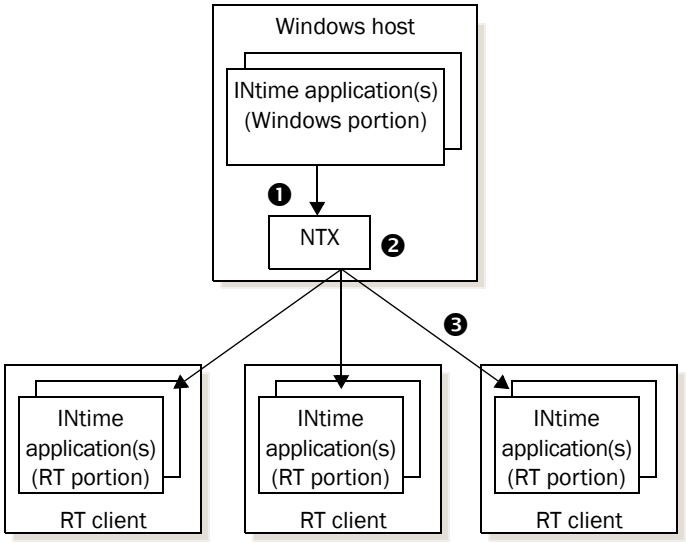
❺ **Transport mechanism**: The communication protocol or method used by NTX to communicate between Windows and RT threads. Whether the various portions of your INtime applications reside on a single PC or on multiple computers accessed via Ethernet cable, NTX provides this essential communication. For details, see *Transport mechanisms* later in this chapter.

❻ **Windows hardware abstraction layer (HAL)**: INtime software intercepts some HAL calls to ensure real-time performance. For details, see *About the Windows HAL* later in this chapter.

Transport mechanisms

With INtime software, NTX communicates between Windows and RT portions of INtime applications, whether they reside on a single PC (single or multi-core), or on separate computers accessed via Ethernet cable:

Figure 2-2. How NTX communicates with other INtime hosts



❶ The Windows portion of INtime applications, located on a Windows host, makes NTX calls that communicate to RT clients.

❷ NTX determines RT client locations, detects the connection method, and determines how to communicate between Windows and RT threads.

❸ NTX uses the appropriate transport method to communicate with the RT portion of the INtime applications, located on RT clients.

Transport methods available to NTX include:

Transport mechanism	Transport driver	Description
OSEM	RTIF.SYS	Used when the Windows host and RT client co-exist on a single PC (Shared Mode). For details, see *About the OSEM* and *How the RT interface driver works* later in this chapter.
Ethernet	UDP/IP	Used for Windows hosts and other INtime hosts connected via a local area network (LAN) cable.
Multicore IPC	RTIF.SYS	Used for Windows hosts and RT nodes to communicate on a multi-core system.

About the OSEM

The OSEM manages the simultaneous operation of Windows and the RT kernel on the same CPU. It encapsulates all of Windows as an RT thread, and then transparently switches execution to the appropriate kernel, based on interrupt activity and thread scheduling. Once encapsulated, Windows (with all its processes and threads) execute as a single, low priority, RT thread in the context of the RT root process.

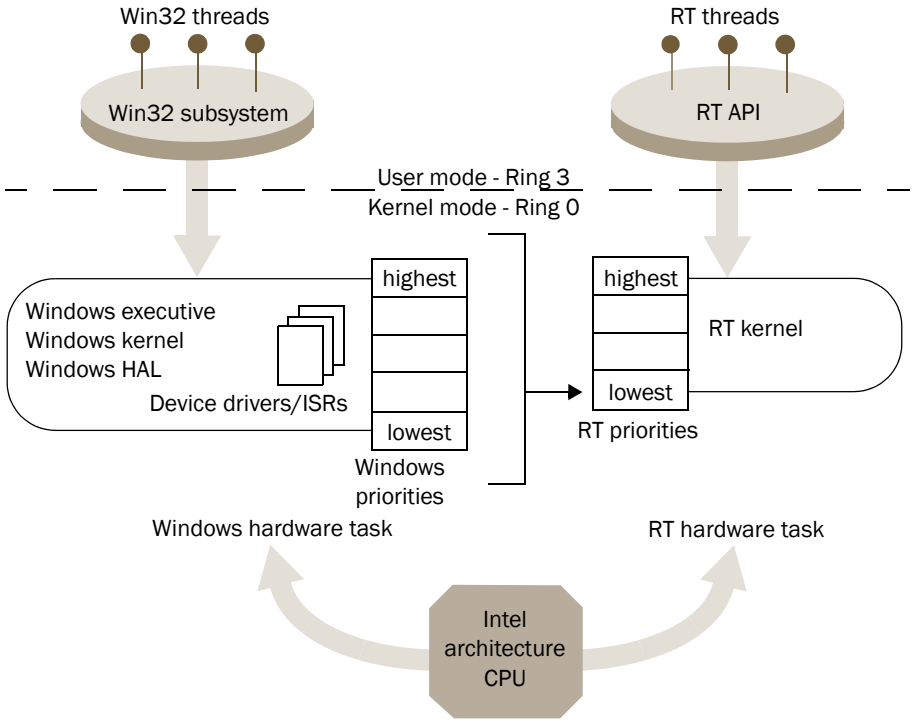
The OSEM provides:

- **Isolated processes**: Uses standard Intel architecture support for hardware multi-tasking to maintain proper address space isolation and protection between Windows processes and RT processes. This approach also ensures RT responsiveness, regardless of Windows activity.
- **Transparent thread creation and switching**: Transparently creates a hardware task for the RT kernel, and manages the switching and execution of both the standard Windows and INtime system hardware tasks.

  In a standard Windows configuration, the bulk of the OS runs in the confines of a single hardware task. Additional hardware tasks are defined only to handle catastrophic software-induced failures, such as stack faults and double faults, where a safe and known environment is required from which to handle the failure. INtime software's task switching approach guarantees the integrity of both Windows and the RT kernel, and enables the successful operation of RT threads even in the event of a total Windows failure (a blue screen crash).
- **Additional address isolation via 32-bit segmentation**: Provides additional address isolation and protection between RT processes, and between RT processes and Windows code. The RT kernel accomplishes this by using multiple sets of 32-bit segments, separate from those used by Windows.
- **Easy-to-use interface**: Provides a clean, well defined interface, which minimizes interaction with Windows to a few key areas. The result is improved product reliability and simplified compatibility between Windows releases.

With INtime applications running on a single PC, the INtime runtime environment encapsulates all Windows processes and threads into a single RT thread of lowest priority as shown in the next figure. As a result, RT threads always preempt running Windows threads, guaranteeing hard determinism for all RT activities within the system.

Figure 2-3. Encapsulating Windows processes and threads into an RT thread



When an interrupt occurs, the INtime runtime environment responds in one of these ways:

Interrupt type	Windows in control	RT in control	Shared control
Windows	Windows maintains control.	RT maintains control.	RT determines whether to maintain or relinquish control.
RT	RT takes control, pre-empting Windows activity.	RT maintains control.	RT maintains control.

How the RT interface driver works

RTIF.SYS is a Windows device driver that provides centralized support for the OS encapsulation mechanism (OSEM). The RT Interface Driver facilitates communications between RT kernel threads and Windows threads.

The RTIF driver begins execution as a Windows system service, early in the Windows boot process. During initialization, it allocates physically contiguous memory for the RT kernel's memory pool.

The RTIF driver cooperates with the RT Kernel Loader to load and start the RT kernel in its own environment. The driver queries the registry for various kernel parameters and passes them to the RT kernel at the kernel's initialization time. Parameters include:

- The number of Windows threads that can simultaneously make NTX library calls. The default is 64.
- The low-level tick duration used by the RT kernel.

If the RT kernel is running, the RTIF driver:

- Routes the clock interrupt (IRQ 0), based on who needs the next clock tick, to either the RT kernel or to the Windows clock interrupt entry point for processing.

  When neither environment needs the tick, the driver sends an EOI to the PIC for this level and returns control to the interrupted Windows thread.
- Immediately routes all other real-time interrupts to the RT kernel for processing.
- Relays NTX library requests to the RT kernel and blocks the calling Windows thread until the RT kernel responds to the request and/or until resources are available to complete the request.

  Otherwise, the RTIF driver terminates NTX library requests. When the RT kernel announces its termination, the RTIF driver terminates all pending requests.
- Manages the Windows portion of controlled shutdown during a Windows blue screen crash: the handler notifies the RT kernel to handle the RT portion of the controlled shutdown. If the kernel is not running, control is returned to Windows.

In summary, the RTIF.SYS device driver contains the Windows portion of the OSEM. It also acts as the NTX transport driver for a co-resident, or local, RT kernel. RTIF.SYS allocates physical memory for the RT kernel and locks that memory in place so it will not be used or paged to disk by the Windows kernel. A Windows service loads the RT kernel into the allocated memory and issues a "start kernel" command to RTIF.SYS. In response to the start command, the driver establishes a separate hardware task for the RT kernel and hands off control to the kernel's initialization code. After initializing its environment, the RT kernel creates a low-priority thread (priority level 254) which returns to Windows and becomes the Windows thread.

About the Windows HAL

INtime software uses the Windows HAL, but intercepts certain functions to perform the following actions (in Shared Mode):

- Traps attempts to modify the system clock rate so that the RT kernel can control the system time base.

- Traps attempts to assign interrupt handlers to interrupts reserved for RT kernel use.
- Ensures that interrupts reserved for RT kernel use are never masked by Windows software.

INtime software is compatible with all HAL files shipped with Windows XP and Windows 2003, and with the Multiprocessor HAL shipped with Windows Vista, Windows 7 and Server 2008.

About thread scheduling

The RT kernel switches between threads and makes sure the processor always executes the appropriate thread. The kernel's scheduling policy is that the highest priority thread that is ready to run is/becomes the running thread. The kernel maintains an execution state and a priority for each thread and enforces its scheduling policy on every interrupt or system call.

Priority-based scheduling

A priority is an integer value from 0 (the highest priority) through 255.

Range	Usage
0–127	Used by the OS for servicing external interrupts. Creating a thread that handles internal events here masks numerically higher interrupt levels.
128–130	Used for some system threads.
131–252	Used for application threads.

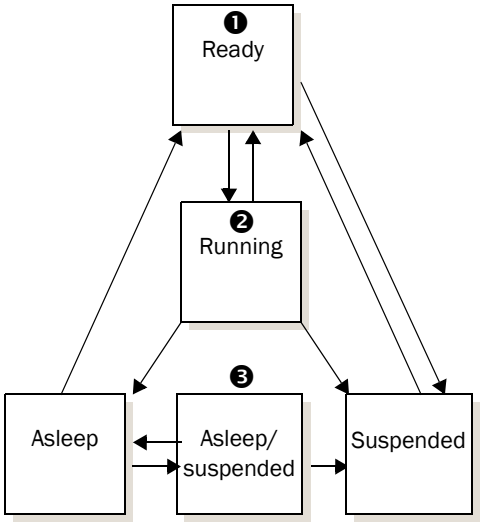
Interrupt threads mask lower-priority (numerically higher) interrupt levels. When you assign interrupt levels, give a higher-priority (numerically lower) level to interrupts that can't wait, such as serial input, and a lower priority (numerically higher) to interrupts that can wait, such as cached input.

Execution state

The execution state for each thread is, at any given time, either running, ready, asleep, suspended, or asleep-suspended. The RT kernel enforces the scheduling policy in which the highest priority ready thread is always the running thread.

Threads run when they have the highest (numerically lowest) priority of all ready threads in the system and are ready to run. Threads can change execution state, as shown in the next figure.

Figure 2-4. Execution state transitions for threads



❶ Threads are created in the ready state.

❷ The running thread, the ready thread with the highest priority, does one of these:

- Runs until it removes itself from the ready state by making a blocking system call.
- Runs until its time slice expires (when running with a priority lower—numerically higher—or equal to the configured round robin threshold priority with other threads at the same priority.
- Runs until preempted by a higher priority thread which has become ready due to the arrival of an interrupt, or through the receipt of a message/unit at an exchange at which the higher priority thread was blocked.

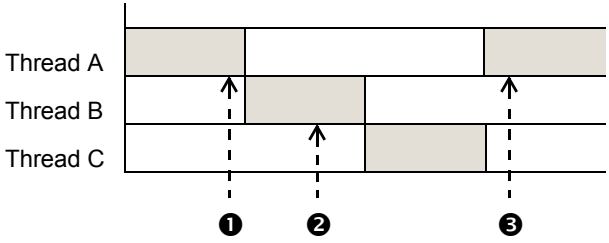
❸ A thread in any state except ready cannot run, even if it has the highest priority.

A thread can put itself to sleep or suspend itself by using system calls for that purpose. The RT kernel might indirectly put a thread to sleep if the thread makes a "blocking" call by, for example, waiting at a mailbox until a message arrives. The kernel puts the thread in the ready state when the message arrives.

Round-robin scheduling

INtime software also provides round-robin scheduling, where equal-priority threads take turns running. Each thread gets a time slice. If a thread is still running when its time slice expires, that thread moves to the end of a circular queue for that priority level where it waits until all threads ahead of it use up their time slices, as shown in the next figure. You can adjust the length of the time slice and set the priority level threshold where round-robin scheduling occurs.

Figure 2-5. Round-robin scheduling



Threads A, B, and C are of equal priority below the round-robin priority threshold.

❶ Thread A, the running thread, stops running when its time slice runs out. Thread A's state is saved and it moves to the end of the queue.

❷ Thread B, a ready thread, then becomes the running thread.

❸ Thread A runs again when all threads in the queue either finish running or are preempted when their time slice expires.

Higher-priority threads still preempt any running thread in the round-robin queue, regardless of the amount of time left in its time slice.

Note

Round-robin scheduling cannot guarantee a predictable worst-case response to events because the number of threads in the queue varies.

Handling interrupts

System hardware invokes an interrupt handler in response to an asynchronous interrupt from an external source, based on its entry number in the IDT (Interrupt Descriptor Table). The handler takes control immediately and saves the register contents of the running thread so it can be restarted later. There are two ways you can service an interrupt:

- Using a handler alone
- Using a handler/thread combination

Interrupt handler alone

An interrupt handler alone can process only interrupts that require very little processing and time. Handlers without threads can:

- Accumulate data from the device in a buffer. The data must have an associated thread to process the data.
- A handler begins running with all interrupts disabled. It must execute quickly and then exit to minimize its effect on system interrupt latency.
- Find the interrupt level currently serviced. This is useful if one handler services several interrupt levels.
- Send an EOI (End of Interrupt) signal to the hardware.

By itself, an interrupt handler can only do very simple processing, such as sending an output instruction to a hardware port to reset the interrupt source. Handlers can use only a few system calls. For a list and description of system calls, see *Appendix A, "INtime software system calls".*

During the time the interrupt handler executes, all interrupts are disabled. Since even very high level interrupts are disabled, it is essential that the handler execute quickly and exit.

When the handler finishes servicing the interrupt, it sends an EOI to the PIC (Programmable Interrupt Controller) via an INtime software system call, restores the register contents of the interrupted thread, and then returns to the interrupted thread.

Interrupt handler/thread combination

An interrupt handler/thread combination provides more flexibility. Although the handler may perform some processing, it typically signals the corresponding interrupt thread to do most or all interrupt processing. In general, use an interrupt handler/thread combination if the processing requires more than 50 microseconds or requires system calls that interrupt handlers cannot use.

When an associated interrupt thread exists, the handler can put accumulated information into a memory address, if the interrupt thread has set one up. The interrupt thread can access data in the memory address and perform the required processing.

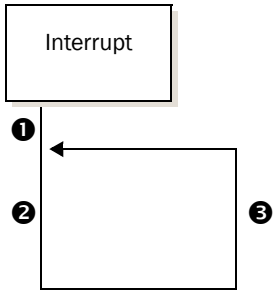
Interrupt threads have access to the same resources and use the same system calls as ordinary threads. The RT kernel assigns an interrupt thread's priority, which is based on the interrupt level associated with the handler. Ordinary threads have a priority assigned by the process.

In addition to the usual thread activities, an interrupt thread can also:

- Cancel the assignment of an interrupt handler to an interrupt level.
- Wait for an interrupt to occur.
- Enable and disable interrupts.

This shows how an interrupt thread enters an event loop where it waits to service an interrupt:

Figure 2-6. Thread execution model



❶ Upon creation, the interrupt thread uses an RT system call to set up an RT interrupt and associate itself with this interrupt. Normally, it then waits for a signal that indicates an interrupt occured.

❷ When signaled, the interrupt thread executes the required operations.

❸ The interrupt thread releases control by waiting for the next signal from the interrupt handler, which restarts the cycle shown in this figure.

Managing time

INtime software enables threads to:

- Create alarm objects that wake up the current thread at a regular interval.
- Start and stop scheduling by the RT kernel.

3 About INtime software's RT kernel

This chapter describes objects provided by the RT kernel.

What does the RT kernel provide?

The RT kernel provides:

Item	Description
Object management	Includes creating, deleting, and manipulating object types defined by the kernel. Memory for high-level kernel objects is automaticallly taken from your processor's memory pool. You must provide memory for low-level kernel objects and may allocate memory beyond the kernel's needs to store application specific state information associated with the low-level object.
Time management	Includes an RT clock, alarms that simulate timer interrupts, and the ability to put threads to sleep.
Thread management	Includes scheduling locks which protect the currently running thread from being preempted.
Memory management	Implements memory pools from which it allocates memory in response to application requests.

RT kernel objects

Objects, data structures that occupy memory, are building blocks that application programs manipulate. Each object type has a specific set of attributes or characteristics. Once you learn the attributes of, for example, a mailbox, you know how to use all mailboxes.

Object-based programming, which concentrates on objects and operations performed on them, is compatible with modular programming. Typically a single thread performs only a few related functions on a few objects.

The RT kernel provides basic objects and maintains the data structures that define these objects and their related system calls. When you create an object, the RT kernel returns a handle that identifies the object:

- **High-level objects** consume memory, but also a slot in the system GDT (Global Descriptor Table). Therefore, the maximum number of high-level objects allowed in the system at any one time is approximately 7600 (8192 slots in a GDT minus slots used by the operating system).

- **Low-level objects** consume only memory. Therefore, only the amount of system memory controls how many low-level objects can be present at a given time.

The RT kernel provides these objects. Each object is discussed on the indicated page:

Object	Description	Page
Threads	Do the work of the system and respond to interrupts and events.	28
Processes	Environments where threads do their work.	28
Exchange objects	Used by threads to pass information.	32
Mailboxes	Used by threads to pass objects and data. INtime software includes both object and data mailboxes.	33
Semaphores	Used by threads to synchronize.	34
Regions	Used by threads to provide mutual exclusion.	35
Ports	Used by threads to synchronize operations, pass messages, and access INtime services.	36
Dynamic memory	Addressable blocks of memory that threads can use for any purpose.	30

Note

For detailed information about RT kernel objects, how they operate, and system calls associated with each object, see INtime Help.

Threads

Threads, or threads of execution, are the active, code-executing objects in a system.

Threads typically respond to external interrupts or internal events. External interrupts include events such as a keystroke, a system clock tick, or any other hardware-based event. Internal events include events such as the arrival of a message at a mailbox. Threads have both a priority and an execution state, whether the thread is running or not.

There are system calls to create and delete threads, view and manipulate a thread's priority, control thread readiness, and obtain thread handles. For a list and description of these system calls, see *Appendix A*, "*INtime software system calls*".

Processes

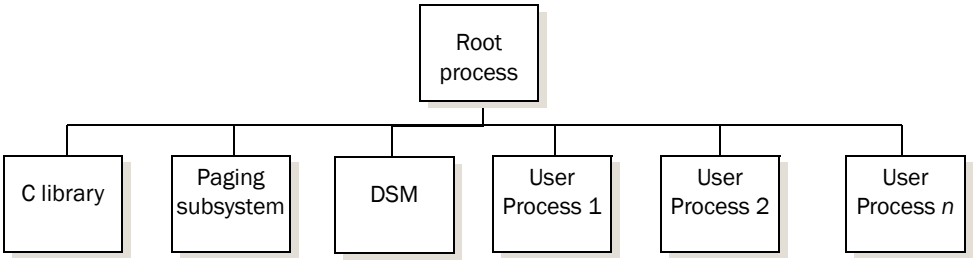
A process is an RT kernel object that contains threads and all their needed resources. Processes make up your INtime applications. The RT kernel processes have these characteristics:

- Cannot make system calls; they are passive.
- May include one or more threads.

- Isolate resources for their threads, particularly for dynamically allocated memory. Two threads of one process compete for the memory associated with their process. Threads in different processes typically do not.
- Provide error boundaries. Errors within one process do not corrupt other processes or the OS because they reside in separate virtual address spaces.
- When you delete processes, the objects associated with them also are deleted.

Each INtime application's executable loads as a separate, loadable process. The processes in a system form a process tree. Each application process obtains resources from the root:

Figure 3-1. Processes in a process tree



The RT Application Loader creates RT processes when an INtime application loads. There are system calls you can use to delete RT processes from within an application.

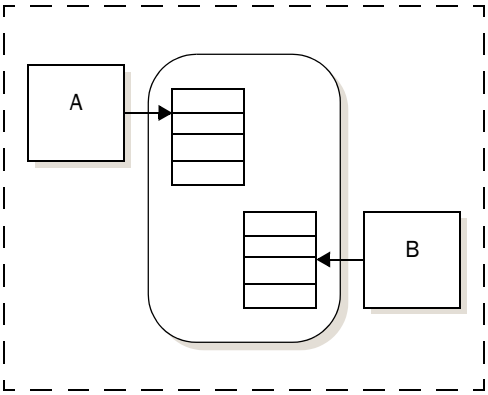
Virtual memory

Each process has an associated VSEG whose size is the amount of Virtual Memory available to the process. The VSEG size must be large enough to contain all the memory dynamically allocated by the threads within the process.

Memory pools

Each process has an associated memory pool, an amount of memory with a specified minimum and maximum, allocated to the process. Minimum memory is always contiguous. Usually, all memory needed for threads to create objects in the process comes from the process's memory pool, as shown in the next figure.

Figure 3-2. Threads using their process's memory pool



Threads A and B obtain memory from the process's memory pool.

If not enough contiguous memory exists (up to the maximum size of the process's memory pool), the RT kernel tries to borrow from the root process.

You can also statically allocate memory to processes, but you cannot free memory allocated in this manner for other processes. The system's total memory requirement is always the sum of the memory requirements of each process. Static memory allocation uses more memory than dynamic allocation, but may be safer.

Dynamic memory

Dynamic memory supports many uses, including communicating and storing data. The memory area is usually allocated from the memory pool of the thread's process, as shown in *Figure 3-2."Threads using their process's memory pool"*. If there is not enough memory available (up to the maximum size of the process's memory pool), the kernel tries to borrow from the root process.

INtime software includes system calls that allocate and free memory and create handles for allocated memory to share with other processes. For an overview of these calls, see *Appendix A, INtime software system calls*. For detailed information, including syntax and parameter values, see INtime Help.

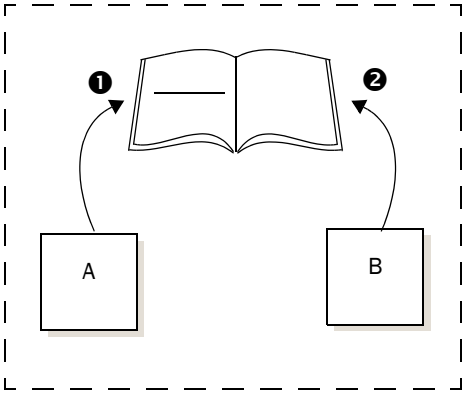
Object directories

Each process has an associated object directory. When a thread creates an object, the RT kernel creates a handle for it. A thread can catalog a high-level object, with its handle and a corresponding name, in the object directory of its own process or any other process it knows about. Typically, you catalog objects in the root directory so that threads in other processes can access them.

Threads that know the name can use the object directory to look up and access objects. Threads in the same process also can use global variables to identify and access objects within their process.

Note
You cannot catalog the handle for a low-level object in an object directory.

Figure 3-3. Threads using the root process's object directory



❶ Thread A catalogs an object such as a data mailbox in the root process's object directory.
❷ Thread B looks up the object in the object directory to use it.
Now thread A can send data to the mailbox and thread B can receive it.

Exchange objects

Validation levels

INtime software provides two levels of system calls for exchange objects:

Level	Description	Exchange objects
High (validating)	Provides higher protection and validation features. Memory is allocated automatically from the process's pool. High-level objects: • Validate parameters. • Are protected against unexpected deletion.	• Object mailboxes • Data mailboxes • Counting semaphores • Regions (for mutual exclusion with priority inversion protection)
Low (non-validating)	Provide higher performance and lower protection and validation features. Low-level objects provide functionality beyond that of high-level objects. You must allocate memory for low-level objects and may allocate memory beyond low-level object needs. You can use this additional memory to store application-specific state information associated with the object. Low-level objects: • Do not validate parameters. If you need parameter validation, use high-level system calls instead. • Are not protected against unexpected deletion. **Note**: System calls that manipulate low-level objects assume that all memory reference pointers received are valid.	• Data mailboxes • Single-unit semaphores • Region semaphores (with priority inversion protection) • Software alarm events (virtual timers) that invoke alarm event threads that you write.

Write, test, and debug your application using high-level calls with their protection and validation features. Use low-level objects when there is no other choice, such as with AlarmEvents and a mailbox or semaphore that must be used from an interrupt handler.

For more information about validation levels, see RT calls in *Appendix A, INtime software system calls*.

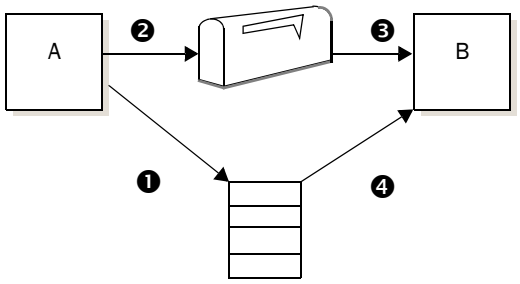
Mailboxes

Mailboxes provide communication between threads in the same process or in different processes. They can send information and, since a thread may have to wait for information before executing, they can synchronize thread execution. There are two mailbox types:

- **Object mailboxes**: Send and receive object handles. Available only as high level objects.
- **Data mailboxes**: Send and receive data. Available as both high- and low-level objects. High-level data mailboxes have a maximum message size of 128 bytes.

The next figure shows how threads use an object mailbox to send a handle for a memory address.

Figure 3-4. Threads using an object mailbox



❶ Thread A allocates a block of memory and creates a shared-memory handle for it. Data is placed in this shared memory object.
❷ Thread A sends the shared memory handle to a mailbox.
❸ Thread B waits to receive the shared memory handle at the mailbox. You can specify whether thread B should wait if the handle isn't in the mailbox.
❹ Thread B obtains the handle and accesses the data in the memory object by mapping the memory associated with the memory object into its memory address space.

Mailboxes have thread queues, where threads wait for messages, and message queues, where messages wait threads to receive them. The thread queue may be FIFO- or priority-based; the message queue is always FIFO-based.

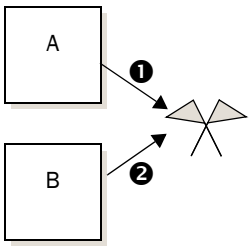
You use the same system calls to create and delete object and data mailboxes. However, you use different calls to send and receive messages or data.

Semaphores

A semaphore is a counter that takes positive integer values. Threads use semaphores for synchronization by sending units to and receiving units from the semaphores. When a thread sends *n* units to a semaphore, the value of the counter increases by *n*; when a thread receives *n* units from a semaphore, the value of the counter decreases by *n*.

The next figure shows a typical example of a binary (one-unit) semaphore used for synchronization.

Figure 3-5. Threads using a semaphore for synchronization



To ensure that thread A can do its work before thread B starts running, thread A creates a semaphore that contains one unit. To enable synchronization, threads A and B should request and obtain the unit before running.

❶ Thread A begins to run and obtains the semaphore unit, leaving the semaphore empty. While the semaphore has no units, thread B cannot run.
❷ When thread A completes, it returns the unit to the semaphore. Thread B can now obtain the unit and start running.

Semaphores:

- Enable synchronization; they don't enforce it. If threads do not request and obtain units from the semaphore before running, synchronization does not occur. Each thread must return the unit to the semaphore when it is no longer needed. Otherwise, threads can be permanently prevented from running.
- Provide mutual exclusion from data or a resource as follows:

  1. Thread A requests one unit from a binary semaphore, and uses the resource when it receives the unit.
  2. Thread B requests one unit from the semaphore before using the resource. Thread B must wait at the semaphore until thread A returns the unit.

- Enable mutual exclusion; they do not enforce it.
- Have a queue where threads wait for units. The queue may be FIFO- or priority-based. There are system calls to create and delete semaphores, and to send and receive units.

Regions

A region is a single-unit semaphore with special suspension, deletion, and priority-adjustment features. Regions provide mutual exclusion for resources or data; only one thread may control a region at a time; only the thread in control of the region can access the resource or data protected by a region. Once a thread gains control of a region, the thread cannot be suspended or deleted until it gives up control of the region. When the running thread no longer needs access, it exits the region, which enables a waiting thread to obtain control of the region and thus access the resource or data protected by that region.

Regions have a thread queue where threads wait for access to the region. The queue may be FIFO- or priority-based.

Priority inversions

Regions also have a priority-inversion avoidance mechanism when the region's thread queue is priority based.

Then, if a higher-priority thread tries to enter a busy region, the priority of the thread in the region is raised temporarily so that it equals the waiting thread's priority. This helps prevent priority-inversion, as shown in this example:

1. Thread A is the running thread. It is a low-priority thread with control of a region, accessing some data. The region has a priority queue. The only other thread that uses the data is thread C, a high-priority thread that is not ready to run.
2. Thread B, a medium-priority thread, becomes ready to run and preempts A.
3. Thread C becomes ready to run and preempts B. It runs until it tries to gain control of the region. Thread A's priority is raised to equal thread C's priority until thread A releases the region; then its priority returns to its initial level.
4. When thread A releases the region, thread C receives control of the region and uses the data. When thread C completes, thread B runs.

Without the priority inversion avoidance mechanism, thread B would have preempted A while A had control of the region; C would have preempted B, but would have been unable to use the data because A had control of the region.

Deadlocks

Regions require careful programming to avoid deadlock, where threads need simultaneous access to resources protected by nested regions, and one thread has control of one region while the other thread has control of another. To avoid deadlock, all threads must access nested regions in the same, arbitrary order, and release them in the same reverse order.

Ports

A port is the object which allows access to the features provided by an INtime service. A process that uses a port object can send messages through the port to the INtime service, or can receive messages through the port from the service. Other operations possible on ports include:

- Attach a heap object to the port for use by the service to store received messages.
- Link ports to a sink port, allowing a single thread to service multiple ports.

Services

An INtime service is an INtime real-time application (RTA) which provides access to one or more interfaces. Each interface is associated with a service descriptor. The interface generates events which are handled by the service. A process which uses a service creates a port for access to that service. A service may support more than one port and more than one user process may use a given port. A user process communicates with the service by sending and receiving messages via the port.

Heaps

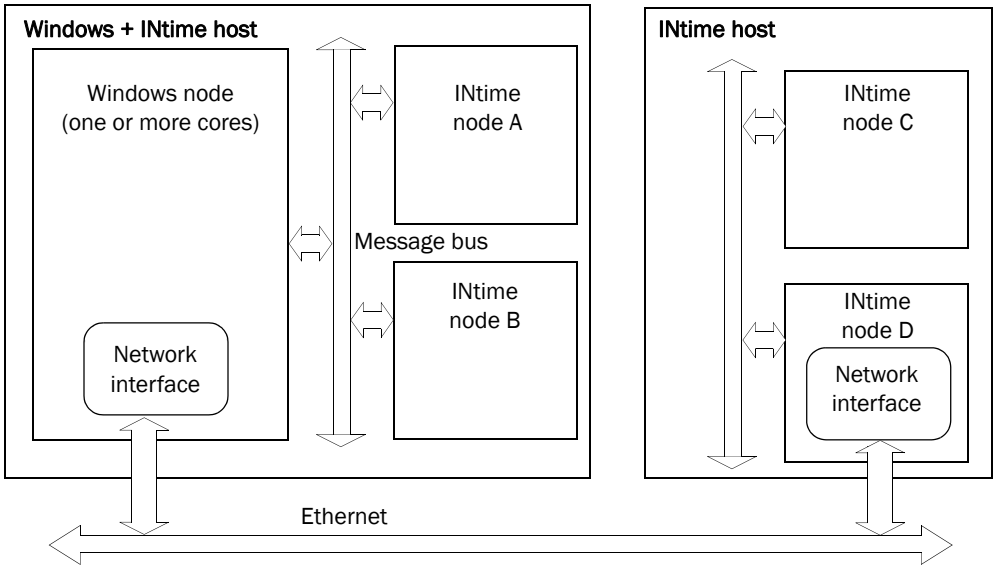
A heap is an INtime memory object that manages the chunk of dynamic memory allocated to it. A heap can be used by multiple processes that need to share large amounts of information. For instance, a heap can be associated with a port. Data placed in memory obtained from the heap by threads in one process (the thread using a port to communicate with an INtime service) can be manipulated by threads in another process (thread within the service accessing data passed through the port to the service).

Global objects, references, and locations

An INtime system is considered to be a set of one or more nodes. Multiple nodes are connected either through shared memory (multiple processor cores on the same host) or via Ethernet. The Address of a node is described in terms of its Local Logical APIC

ID and its network address. The Location of a node is a parameter used by processes on one node to identify another node.

Figure 3-6. Global object architecture



In the case where a multi-core host is connected to the INtime network from one node only, the requests will be forwarded to the target node by the connected node.

Node architecture

On each node a management process handles requests on behalf of remote nodes, and manages local connection objects. It is also responsible for managing objects created by other nodes, and the DSM relationships between nodes.

New Objects

Two new objects and a new object classification are defined under this specification. The Location object encapsulates information about the location of a given node and the spaces it occupies. The Reference object contains information allowing the user of its handle to use an object on a remote node using the standard APIs for an object of that type.

Location object

The Location object contains APIC ID and network address information for a given node. The handle for a location object (of type LOCATION) has Node scope, not universal scope.

Statically-configured locations are created at kernel boot time and are made visible to other nodes when they boot the first time. These would normally include all other nodes on the local host. Locations on other hosts may be created dynamically or on-demand.

Reference object

A reference object is created on a node when a process requests access to an object on another node. A reference object can be created explicitly, by a call to CreateRtReferenceObject. Sometimes a reference object is created implicitly, for example by CreateGlobalRtMailbox, or by LookupRtHandle. A reference object may also be deleted explicitly by DeleteRtReferenceObject or implicitly for example by DeleteRtMailbox.

A handle for a reference object has Node scope. The reference object thus represents the nexus between an RTHANDLE for a global object and the LOCATION of that global object. This is particularly important to remember when passing handles between nodes using the extended APIs.

Global objects

A special instance of a semaphore, mailbox or memory object can be created as a global object, which is a class of object which can be created either on the local node or a remote node and optionally it can be created so that it is not owned by the creating process. The APIs CreateGlobalRtSemaphore, CreateGlobalRtMailbox, CreateGlobalRtMemoryObject and CreateGlobalRtMemoryHandle may be used to create such objects. In the standard case the object created with these APIs is considered to be owned by the creating process, and thus is deleted implicitly when the process is deleted. However such objects may also be created using the REFCOUNT flag, which means that the object is not considered to be owned by the creating process. The object has an associated reference counter which is incremented each time a reference object is created for it and decremented each time one of its reference objects is deleted. The object itself is not deleted until its reference count decrements to zero.

Note that a reference object is always owned by the creating process and will be deleted when that process is deleted.

4 About RT programming

This chapter describes mechanisms appropriate to RT programming:

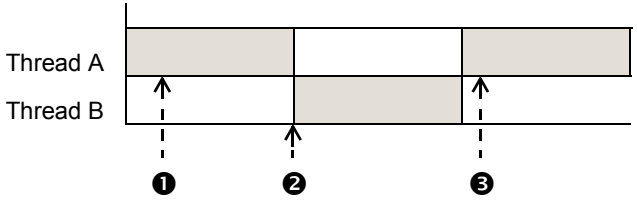
Note
For information about developing an RT application using INtime software, see *Chapter 10, INtime application development*.

Mechanism	Description	Page
Multi-threading	Switches between threads of execution.	39
Preemptive, priority-based scheduling	Determines which thread needs to run immediately and which can wait.	41
Interrupt processing	Responds to external interrupts that occur during system operation.	42
Determinism	Enables threads to execute in a predictable fashion, regardless of the arrival of both internal and external events.	43
Multi-programming	Allows more than one application to run at a time.	44
Inter-thread coordination and communication	Enables asynchronous threads, which run in a random order, to coordinate and communicate with one another.	45
Messages	Enables threads to exchange data, messages, or object handles.	46
Synchronization	Enables threads to signal the second thread when a task is completed.	47
Mutual exclusion	Prevents threads from accessing data currently in use until released.	47
Memory pools and memory sharing	Allocates memory to RT applications on request and manages multiple memory requests.	48
System calls	Programmatic interfaces you use to manipulate objects or control the computer's actions.	50
Real time shared libraries	Libraries you build that can be shared by one or more real-time applications.	50
Exception handling	Causes and proper handling of system exceptions.	51

Multi-threading

Multithreading means the computer stops running one thread and starts running another, as shown in the next figure. INtime software manages thread switching, saving the old thread's context on the old thread's stack and loads the new thread's context before starting execution. An INtime software thread is a thread of execution, similar to a Windows thread.

Figure 4-1. Thread switching in a multithreading environment



❶ The processor executes thread A.

❷ An event happens and a thread switch occurs. The processor then executes thread B.

❸ When thread B finishes, thread A becomes the running thread again.

Multithreading and modular programming go hand-in-hand. You start by breaking down a large, difficult application problem into successively smaller and simpler problems, grouping similar problems where you can. Finally, you solve the small problems in separate program modules. In the INtime software multithreading environment, each module is a thread.

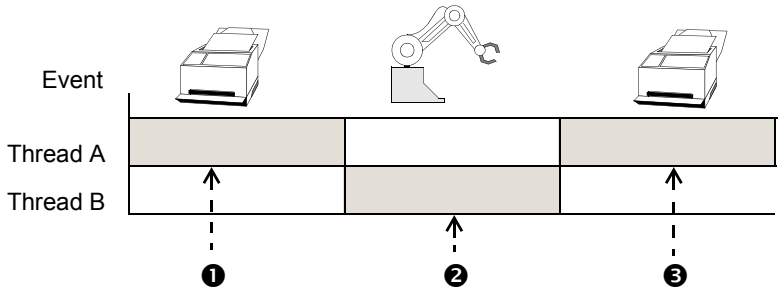
Multithreading simplifies building an application. When you need a new function, you just add a new thread.

When you combine multithreading with preemptive, priority-based scheduling, your application can switch as appropriate: from relatively unimportant threads, to important threads, to critical threads, and back again.

Preemptive, priority-based scheduling

In a preemptive, priority-based system, some threads are more critical than others. Critical threads run first and can preempt less critical threads, as shown in this figure:

Figure 4-2. Multithreading and preemptive, priority-based scheduling



❶ Thread A, a low-priority thread, prints data accumulated from the robotic arm in report form.

❷ Thread B, a high-priority thread, controls the robotic arm. If the arm needs to move while thread A runs, thread B preempts the print thread, then starts and moves the arm.

❸ After thread B repositions the arm, thread A finishes printing.

Multithreading allows an application to respond to internal events and external interrupts, such as clock ticks from the system clock or receiver ready from a serial device, based on how critical they are. You determine the priority of threads in your application; INtime software provides the thread scheduling algorithms.

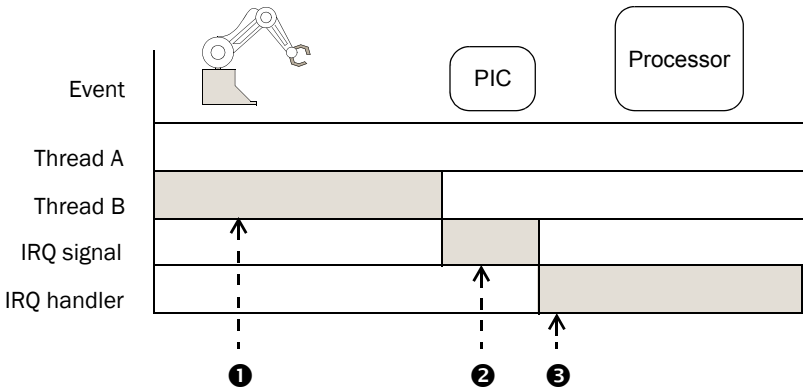
When you add interrupt processing to multithreading and preemptive, priority-based scheduling, your application can respond to interrupts as they occur. Your application becomes event-driven.

Interrupt processing

Interrupts are signals from devices such as a malfunctioning robot or interactive terminal. You connect interrupt sources to the processor through the PC's two PICs (Programmable Interrupt Controllers).

With interrupt processing, your application can handle interrupts occurring at random times (*asynchronously*) and can handle multiple interrupts without losing track of the running thread, or those threads waiting to run. Interrupts can occur while the processor is executing either an unrelated thread or a related thread, as shown in the next figure.

Figure 4-3. Interrupt handler interrupting a thread



❶ Thread B, the running thread, repositions the robotic arm.

❷ The robotic arm malfunctions and sends an interrupt signal through the PIC.

❸ As soon as it receives the signal, the microprocessor stops the running thread and starts an interrupt handler. The interrupt handler runs in the context of thread B. No new thread is loaded; thread B's state does not need to be saved. It remains loaded in RAM until the scheduler runs it again. Thread A, the print thread, is still waiting to run.

Typically, numerous sources of interrupts exist in an application. Some of them, like the malfunctioning robotic arm, are critical; some of them are not. You assign interrupt levels (which map directly to priorities) to the interrupt sources by the order in which you connect your external sources to the PIC. INtime software handles more critical interrupts first, and keeps track of which interrupts occurred, the order in which they occurred, and which ones have not been handled.

Interrupt handlers can perform very limited operations, so you typically write an interrupt handler to signal an interrupt thread. The interrupt thread's priority can be automatically assigned, based on the interrupt level of the external source.

Multithreading and interrupt processing simplify expanding an application. Because of the one-to-one relationship between interrupts and threads, you add a new thread when you need to respond to a new interrupt.

Determinism

INtime software provides deterministic response by establishing a predictable, worst-case response time to a high-priority interrupt. Deterministic response time includes these components:

- **Interrupt response time**: The time that elapses between a physical interrupt and the start of interrupt handler execution. A predictable worst-case response time to interrupt processing ensures that incoming data is handled before it becomes invalid.
- **Thread switch time**: The time that elapses between exiting one thread and starting another. To exit a thread, the RT kernel must save data registers, stack and execution pointers (the thread state) of one thread. Minimized thread switch time also provides a predictable response time to a high-priority thread.

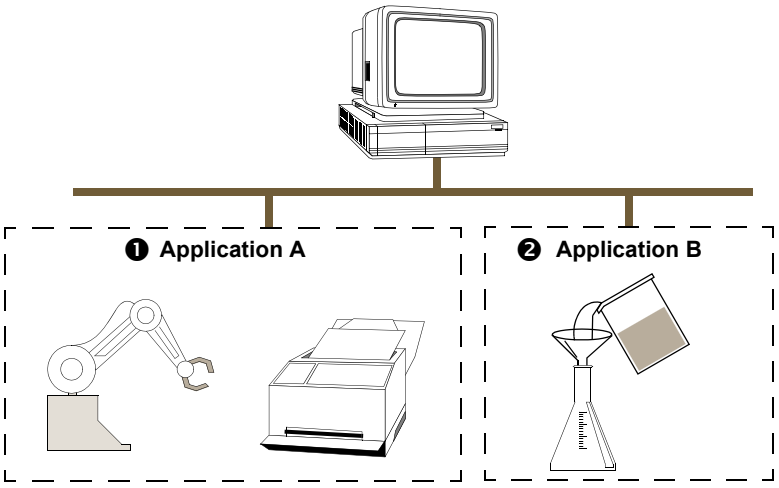
Since the typical response to an interrupt includes invoking a handler and then performing a thread switch to an interrupt thread, the deterministic response time includes both the interrupt response and thread switch times.

RT response does not mean instantaneous execution. A high-priority thread that is very long and performs many calculations uses as much processor time to execute on an RT system as on any other system. The length of time instructions take to execute is a function of processor speed.

Multi-programming

INtime software supports multiprogramming—running several unrelated applications on a single system at the same time.

Figure 4-4. Multiprogramming

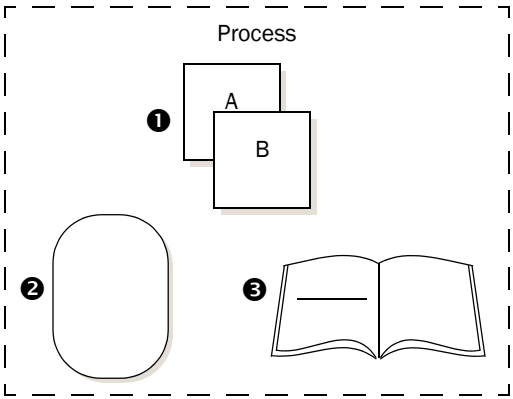


❶ Application A contains all the threads that relate to a robotic arm, including the print thread. It may also contain threads that control other devices on the factory floor.

❷ Application B contains all the threads that relate to an application that controls a chemical mixing system in one part of the factory.

To take full advantage of multiprogramming, you provide each application with a separate environment: separate memory and other resources. INtime software provides this kind of isolation in a process. Typically, a process includes a group of related threads and the resources they need, as shown in the next figure.

Figure 4-5. Resources in a process



❶ A group of related threads.

❷ The memory the threads need.

❸ An object directory where you can catalog thread resources.

You decide what processes to include in your system. INtime software coordinates the use of resources within and between processes so independently-developed applications do not cause problems for each other.

Multiprogramming simplifies adding new applications; you can modify your system by adding new processes (or removing old ones) without affecting other processes.

Inter-thread coordination and communication

INtime software exchange objects are mailboxes, semaphores, regions, and message ports. They enable asynchronous threads, which run in a random order, to coordinate and communicate with one another by:

- Passing messages.
- Synchronizing with each other.
- Mutually excluding each other from resources.

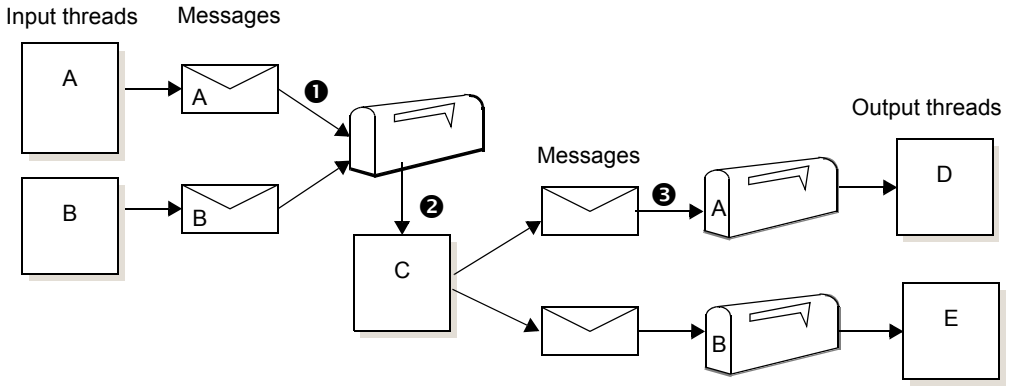
Messages

Threads may need to exchange data, messages, or object handles.

For example, a thread accumulates input from a terminal until it receives a carriage return. The thread then uses an exchange object to send the entire line of input as data to another thread that decodes the input.

This figure summarizes how you can solve a problem that requires routing several input types into several output types using a mailbox object. One mailbox and one manager thread can handle messages from multiple input and output threads.

Figure 4-6. Object-based solution for message passing



❶ System calls move data from input threads A and B to a waiting mailbox.

❷ Thread C, the manager thread, waits at the mailbox and determines which messages go to which output threads. If another message arrives during processing, the message waits in the mailbox queue until the manager thread can handle it.

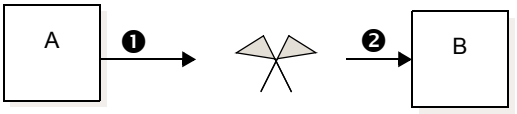
❸ The individual output threads receive data at their mailboxes and execute it.

Synchronization

When one thread needs to run before another thread, it can use an exchange object to signal the second thread when it has completed. For example, the thread that creates the transaction summary in an automated teller application shouldn't run until after the threads that handle withdrawals and deposits run. The transaction summary thread must synchronize with the other threads.

INtime software provides several objects for synchronization that accommodate a wide variety of situations. The next figure illustrates using a semaphore to send a signal to another thread.

Figure 4-7. Threads that use a semaphore for synchronization



Thread A, the running thread, preprocesses some data. Thread B needs to use the data after thread A finishes.

❶ When thread A finishes, it sends a signal (not data) to the semaphore.

❷ When thread B receives the signal, it processes the data.

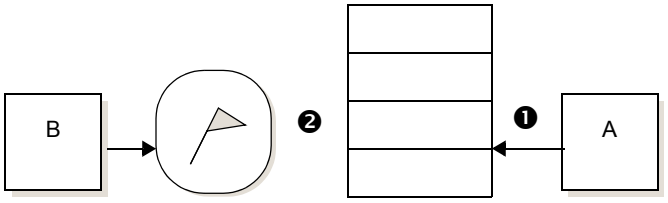
Mutual exclusion

INtime software includes regions that you can use to protect data from being accessed by multiple threads at the same time. This is called mutual exclusion.

When an INtime application runs, multiple threads can concurrently access the same data. This is useful in a multithreading system, such as a transaction processing system where a large number of operators concurrently manipulate a common database.

If a distinct thread drives each client, an efficient transaction system requires that threads share access to database data. When threads run concurrently, this situation occasionally arises:

Figure 4-8. Multithreading and mutual exclusion



❶ Thread A, the running thread, reads data from the database and performs computations based on the data.

❷ Thread B tries to preempt thread A and update the data while thread A works on it. Mutual exclusion, provided by a region, prevents two threads from accessing the same data concurrently.

Unless thread B is prevented from modifying the data until after thread A has finished, thread A may unknowingly use some old data and some new data, resulting in an invalid computation. It should, however, read and compute the new data after thread B updates it.

Memory pools and memory sharing

Memory pools are the basis of INtime software's memory management. Two types of memory pools exist:

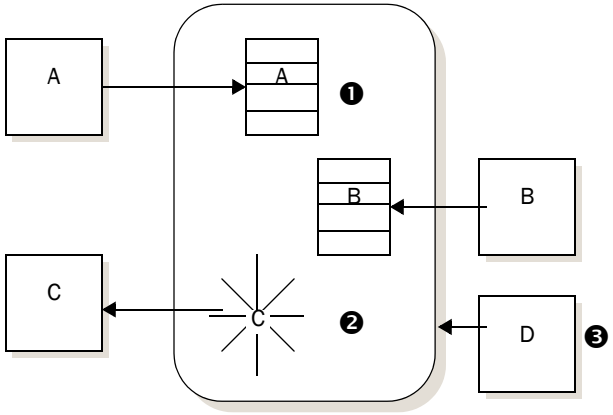
- **Initial memory pool**: All the memory available to the RT kernel (that is, free space memory). Managed by the RT kernel, the initial memory pool belongs to the root process and is allocated to INtime applications on request. Any amount of memory you allocate to an RT kernel is not available to Windows for that session, or any other RT node in the system. There is a practical maximum for the amount of memory allocated to INtime software. This value depends on the number and size of INtime applications that run on the RT kernel.
- **Process memory pools**: A portion of the initial memory pool assigned to an INtime application process when that process is created. Each process memory pool has a minimum and a maximum size. Once the minimum memory is allocated to a process, that memory is not available to other processes. When you delete a process, its memory returns to the initial memory pool.

  As threads in a process create and delete objects, the process memory pool's size increases and decreases as needed, provided minimum and maximum values are not yet encountered. This provides dynamic memory allocation of the memory pool.

The RT kernel uses dynamic memory allocation to free unused memory and assign freed memory to other processes. Threads within processes use dynamic memory allocation in the same manner.

For example, some threads periodically need additional memory to improve efficiency such as a thread that allocates large buffers to speed up input and output operations. Such threads can release memory for other threads when they complete, as shown in the next figure.

Figure 4-9. Dynamic memory allocation between threads



❶ Threads A and B use memory in the process's memory pool for objects they create.

❷ Thread C completes and then deletes its objects, and releases its memory to the process's memory pool.

❸ Thread D requests memory.

Inter-node coordination and communication

Threads within real-time processes can interact with each other even when resident on different instances of the real-time kernel. The Global Objects functionality is broadly similar to that provided to Windows applications by the NTX protocol, with enhancements to take into consideration issues of thread priority and multiple client access. A real-time thread can access objects which exist on another instance of the INtime kernel. Each of these “global” objects has a corresponding local object called a reference which points to it and may be used in standard APIs by the local thread. For an example a thread can obtain a reference to a mailbox on a different INtime node then access it using the standard SendRtData and ReceiveRtData calls.

There are new APIs for discovering other nodes, and for creating new global objects, and creating and deleting reference objects.

System calls

Each RT kernel object has an associated set of system calls: programmatic interfaces you use to manipulate objects or control the computer's actions. System calls for one object type cannot manipulate objects of another type. This protects objects from inappropriate actions or changes.

Most system calls have parameters, such as values and names, that you can set to tailor the call's performance. High-level system calls validate these parameters; a condition code returned by the call indicates whether or not the call completed successfully. Invalid parameter usage, such as trying to manipulate an object of the wrong type, or trying to read or write memory to which you have no access, results in the return of a 'call failed' condition code. You can use another system call (GetLastRtError) to determine the failure's exact cause.

Examples of tasks you can perform with system calls include:

Function	System call
Create a new mailbox	CreateRtMailbox
Set a thread's priority	SetRtThreadPriority
Send a handle to an object mailbox	SendRtHandle

Note

For an overview of INtime software's system calls, see *Appendix A, INtime software system calls*. For detailed information, including syntax and parameter values, see Help.

Real time shared libraries

You can build libraries that can be shared by one or more real-time applications. These Real time shared libraries (RSLs), function essentially like their Windows DLL counterparts.

An INtime Wizard is provided that integrates with Microsoft Visual Studio to allow you to easily create an RSL. It produces the framework for an RSL, and sets up the various Visual Studio settings to generate an RSL that the INtime loader can manage. For detailed information on creating an RSL, see INtime Help.

Exception handling

While running, an INtime application makes various system calls and interfaces with various hardware devices, the CPU's Numerics Processor, and both self- and system-defined memory structures. Incorrect usage of any of these resources will cause a system exception.

System exceptions include:

System exception	Cause
Programming error	Passing an invalid parameter to a system call.
Environmental error	Requesting resources outside the capabilities/limits of the operating system to grant. For example, a call to malloc can fail with an Environmental exception if the caller's process cannot provide the requested memory.
Numerics exception	Attempting an illegal floating point operation, such as a floating point divide-by-zero operation.
Hardware fault	Attempting to do an operation that violates the protection mechanisms built into the X86 architecture. For example, an integer divide-by-zero operation causes a Divide-by-Zero Fault. Likewise, trying to access a memory address in your VSEG that has not had physical memory assigned to it causes a Page Fault.

Programming and Environmental errors are signaled to threads making system calls as the return value for each call. Application code should check for successful completion of each system call and provide error handling subroutines to handle any possible Programming or Environmental errors.

Proper handling of Numerics exceptions requires that a user exception handler be set up for each thread that makes floating point calls. INtime software provides a Floating Point Exception handler sample project that you can use as a template for your own floating point exception handler.

When a thread causes a Hardware Fault, a system wide Hardware Fault Handler is called. This handler takes one of the following user specified actions:

- Suspends the offending thread (default).
- Deletes the offending thread.
- Deletes the offending thread and its process.

The handler also sends a notification message to the following cataloged data mailboxes:

- HW_FAULT_MBX cataloged in the object directory of the root process.
- HW_FAULT_MBX (if present) cataloged in the object directory of the process whose thread caused the Hardware Fault.

An INtime application can create and catalog a data mailbox in its object directory under the name HW_FAULT_MBX. It can then create a thread that waits at that mailbox for any hardware fault messages and take whatever actions are desired.

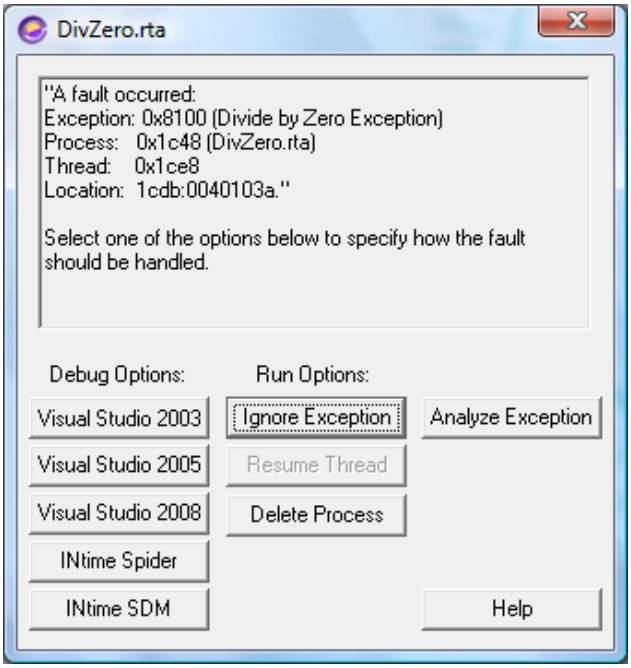
INtime software provides a Hardware Fault Exception handler sample project that shows the use of a local HW_FAULT_MBX mailbox and a thread that manages it.

For detailed information on the HW_FAULT_MBX and its notification messages, see INtime Help.

Fault Manager

INtime software includes an application which handles faults and gives the user an opportunity to debug the faulting condition. The Fault Manager is enabled by default in the Development Kit. If a fault occurs, the following dialog box displays:

Figure 4-10. Fault Manager Dialog



The action required to debug the fault may now be selected, or the process containing the faulting thread may be deleted. See the Fault Manager help information for how to configure a default action.

Structured Exception Handling

Any INtime application may use the Structured Exception Handling feature of the Microsoft compilers to handle faults in the application.

For more information about Structured Exception Handling, see Microsoft's compiler documentation.

5 Designing RT applications

This chapter provides general guidelines for RT system design using INtime software.

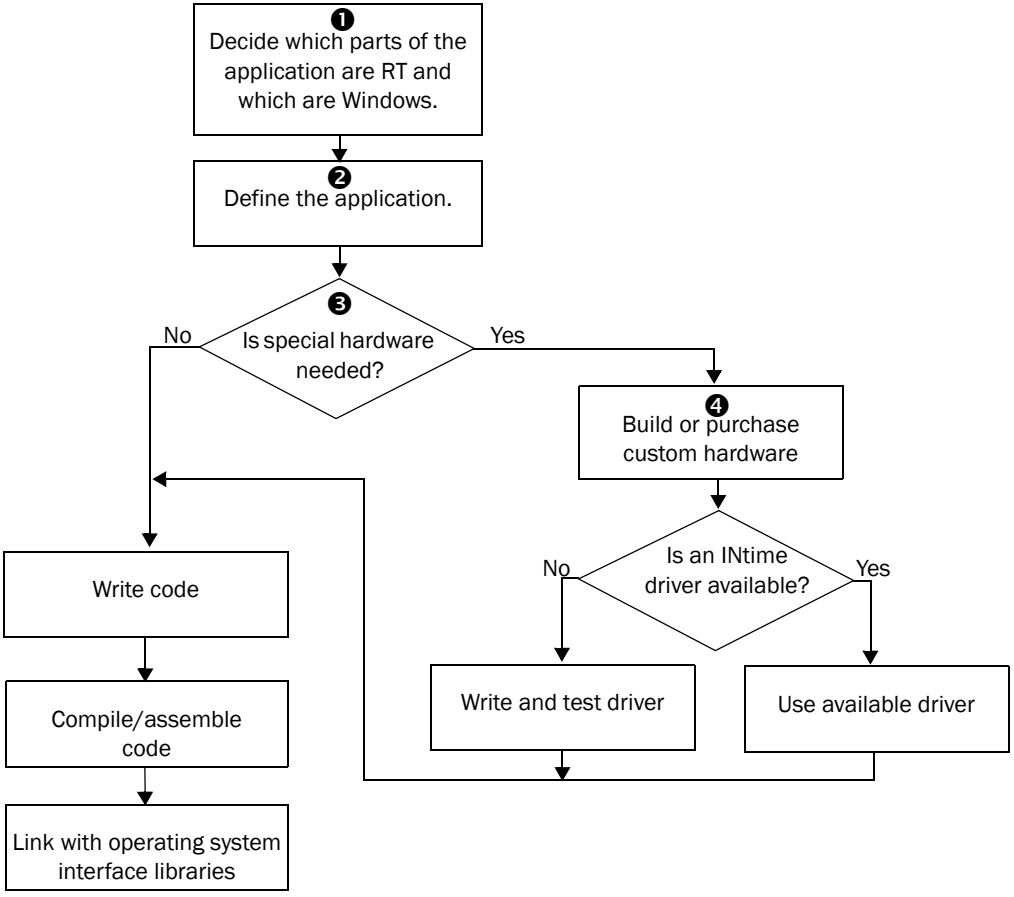
Define the application

When designing an RT application, you should include these steps:

- Partition the application into Windows and RT components.
- List the INtime application's inputs and outputs, and decide whether RT or Windows services each system component. Decide which objects to use for inter-thread synchronization and communication, both within the RT portion and between the RT and Windows portions.
- List all threads that require the input and output. Define interrupts and decide which ones require determinism. Assign ownership of the hardware which provide the interrupts, i.e., owned by Windows or INtime. Remember, an interrupt cannot be shared between Windows and INtime. Assign interrupt levels and priorities to take advantage of multitasking and preemptive, priority-based scheduling.
- Develop the detail for each thread in a block diagram.
- Decide if the application requires multiple processes, and if so, how they will use shared memory and dynamic memory allocation.
- Design your user interface.
- Determine if you require custom devices and drivers.
- Determine if your application will run on a single system or be distributed over multiple RT nodes.

This flowchart shows steps typically taken by RT software designers. When complete, the prototype system is ready to test, debug, and fine-tune.

Figure 5-1. Typical development cycle for INtime applications



❶ Partition the application into RT and Windows components.

❷ Define processes, then define threads, interrupt levels, and priorities. Decide which objects to use. Define interrupts, handlers, and levels.

❸ Decide on hardware and device drivers. Decide how to implement a multi-user environment and/or an operator interface.

❹ Decide if you need custom hardware that solves a unique problem or gathers data in a unique way.

Target environments

Developing INtime applications involves balancing between the target environments:

- The RT environment, where you create portions of the application that require RT robustness and determinism.
- The Windows environment, where you can add RT object access to a Win32 application.

Methodology

A critical aspect of any INtime application is the division of labor between Windows processes and RT processes and, to a finer degree, between the threads in each process.

Important guidelines for developing RT user applications include:

- **Limit RT processes to performing only time-critical functions**: For maximum performance in Shared Mode, applications must be divided appropriately between the RT and Windows portions. The INtime software scheduler gives precedence to RT processes and threads. This preferential scheduling guarantees the best possible RT response.

  For example, to optimize user interface (UI) performance, design INtime applications so that RT activity occurs in event-driven bursts. An INtime application that executes for too long on the RT kernel can consume too many CPU cycles and therefore degrade the system's Windows capabilities. This typically causes a sluggish GUI on the Windows side.

  Note

  Adverse effects in the Windows interface, disk and network operation become noticeable when RT CPU utilization on your system exceeds 70 percent. On a multi-core computer configured for dedicated mode, INtime software can consume close to 100 percent of its processor without affecting Windows.

- **Determine which Windows threads require the greater relative priority**: The relative priority given to each of the Windows threads of an INtime application can determine the perceived performance of the application in a heavily loaded system. The higher the relative priority given to a Windows thread, the more likely the thread will perform sufficiently in a heavily loaded system.

  Determining which threads require the greater relative priority depends on the nature of the application. For example, giving higher relative priority to data manipulation and data storage threads can sacrifice data display and user interface performance when the system is heavily loaded. If an application is data-intensive or the system has no keyboard or display, then sacrificing user interface performance may be desirable. Conversely, if a requirement of an application is a responsive user

interface, then data manipulation and data storage can be given a lower relative priority.

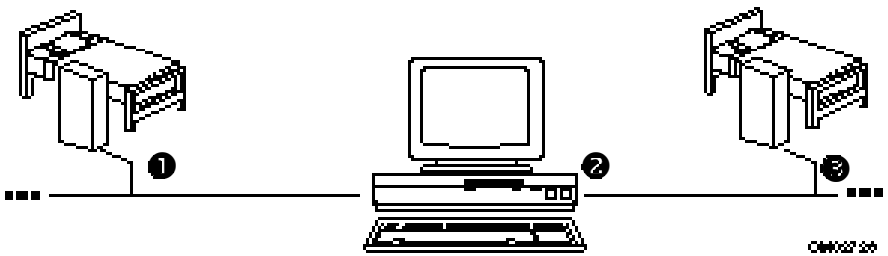
Note

- You can use the Windows Performance Monitor to observe CPU usage by Windows and INtime software's RT kernel. For more information, see *Performance monitor* in *Chapter 10, INtime application development*.
- You can dedicate 100% of one processor to INtime on a multi-core computer without negatively affecting Windows performance.

A hypothetical system

This hypothetical INtime application monitors and controls dialysis. The application consists of three main hardware components:

Figure 5-2. The hardware of the dialysis application system



❶ A bedside unit is located by each bed. Each unit runs INtime software, which performs these functions:
- Measures the toxins in the blood as it enters the unit
- Adjusts the rate of dialysis
- Removes toxins from the blood
- Generates the bedside display for bedside personnel
- Accepts commands from the bedside personnel
- Sends information to the MCU (Master Control Unit)

❷ The MCU, a PC with a screen and keyboard, runs INtime software. The MCU enables one person to monitor and control the entire system. It performs these functions:
- Accepts commands from the MCU keyboard
- Accepts messages from the bedside units (toxicity levels, bedside commands, emergency signals)
- Creates the display for the MCU screen

❸ A LAN connects the bedside units to the MCU.

The next sections describe how various INtime software features are used in the hypothetical system.

Interrupt and event processing

Interrupts and internal events occur at the bedside units: bedside personnel enter commands asynchronously and the system computes toxicity levels at regular intervals.

Toxicity levels, measured as the blood enters the bedside unit, are not subject to abrupt change. The machine slowly removes toxins while the patient's body, more slowly, puts toxins back in. The result is a steadily declining toxicity level. The bedside units must monitor toxicity levels regularly, but not too frequently. For instance, the bedside units could compute the toxicity levels once every 10 seconds, using a clock for timing. The measurement thread would measure and compute the toxicity, put the information in a mailbox for the MCU, and suspend itself for 10 seconds.

Command interrupts from the bedside unit occur when a medical operator types a command and presses Enter. Interrupts from command entries occur at random times. The interrupt handler signals the interrupt thread. The interrupt thread performs any required processing and waits for the next interrupt.

Processing commands from the bedside units: Each time a medical operator types a command and presses Enter, the bedside unit receives an interrupt signal from the terminal. The bedside unit stops executing the current instruction and begins to execute an interrupt handler.

1. The interrupt handler accumulates the characters in a buffer and puts them in memory. The interrupt handler signals the interrupt thread for bedside commands.
2. The interrupt thread gets the contents of the memory where the handler put the command. It parses the command and does the required processing.
3. The thread puts the command information, along with the number of the bedside unit, into a message.
4. The thread sends the message to the predetermined mailbox for the MCU.
5. The interrupt thread waits for the next interrupt. The system returns to its normal priority-based, preemptive scheduling.

Multi-tasking

Threads in the application run using preemptive, priority-based scheduling. This allows the more important threads, such as those that control the dialysis rate, to preempt lower-priority threads, such as those that update displays. New capabilities can be added to the system by simply adding new threads.

II Using INtime software

This part provides the information you need to install, run, and use INtime software to develop RT applications.

This part contains:

Chapter 6: Installation

Explains how to install and uninstall INtime software.

Chapter 7: Configuration

Describes how to configure INtime software.

Chapter 8: Connecting to an INtime host

Explains how to set up an NTX connection to another INtime host.

Chapter 9: Operation

Describes how to start and run INtime software.

Chapter 10: INtime application development

Explains how to use the INtime development environment to create INtime applications.

6 Installation

This chapter explains how to install your INtime software on a Windows system. Follow the directions in this chapter to install INtime software for a development system or a target system.

Note

For descriptions of INtime nodes, Windows hosts, and INtime hosts, see *Terminology* in *Chapter 2, Understanding INtime software architecture*.

Install INtime software on a Windows system

Requirements

Installation of INtime on a Windows system requires:

- A minimum of 4MB of RAM above that required to run Windows. The default allocation to INtime is 16MB.
- A minimum of 75MB of available disk space.
- A supported version of Microsoft Windows (for a listing, see page 3.)

The development environment requires the installation of one of these:

- Microsoft Visual Studio 2005, Standard Edition or better.
- Microsoft Visual Studio 2008.

Note

Support is provided for Visual C 6.0 and Visual Studio 2003 for support of legacy applications only. New development with these tools is not supported.

Before you begin

- Ensure that you are logged on with Administrator privileges.
- Exit all programs prior to installing INtime software.
- If your system has a previously installed version of INtime software, remove it:
  1. Ensure that no INtime services are running. If any are running, be sure they are set to Manual Start (using the Start>Control Panel> Administrative User/ Services applet), and then reboot the system.

2. Select the Add/Remove Programs Applet in the System Control Panel (Select Start>Control Panel>Add/Remove Programs).
3. Highlight INtime 3.1 (or earlier) program, and then click Remove.
4. Reboot the system.

Running the Installation program

To install the INtime software:

1. Insert the INtime software CD-ROM into the CD-ROM drive. A web browser automatically runs and displays instructions to launch the installer.

> **Note**
> If you've disabled the automatic start feature on your system, you must launch install.htm from an Explorer window.

   If the Installation program detects a previous version of INtime software, it prompts you to exit the installation and uninstall the previous version as described in *Before you begin* (above).

2. Review the INtime License and note its terms and conditions.

> **Note**
> If you continue and install the INtime software, your action shows that you agree to be bound by the terms and conditions outlined in the INtime License.

3. Review the Readme information for any late-breaking information not included in the product documentation.
4. Select a destination directory for the INtime software files, then click the Next button. The default is the default Program Files Path (%PROGRAMFILES%\INtime).

5. Select one of the installation options listed below, then follow the prompts in the installation.

Figure 6-1. Installing INtime software

Option	Description
Local Target Only	Installs the INtime runtime components on the local system. Choose this option to run INtime applications on this system.
Development and Local Target	Installs all product development and runtime components. Choose this option to develop and run INtime applications on this system.
Custom	Installs the product development and runtime feature groups you specify. Choose this option, for example, to install only the development components.

6. Refer to the license management document for explicit instructions for your product. Follow the instructions for installing the appropriate licenses.

   When the installation is complete, the Installation program may prompt you to reboot the system.

7. Click the Finish button to complete the installation process.
8. Restart your system using one of these methods:
   - Select YES at the last screen. The Installation program restarts your system.
   - Select NO to manually restart your system at a later time.

Note
You must restart your system before running INtime software.

Installing hardware for use with the RT kernel

Before loading an application to interface with a hardware device, you must make the hardware available to the RT kernel.

Windows Plug-and-Play software tries to automatically configure hardware it detects at boot time. To prevent this, INtime provides a Windows driver which causes Windows to believe it has claimed the hardware while isolating it from other Windows hardware so that an RT application can interact with it.

In an INtime system it is necessary to separate the IRQs of devices that that INtime will control from those controlled by Windows. If real-time and Windows devices shared the same interrupt line, deterministic handling of that interrupt is lost, since release of that signal would depend on the Windows kernel.

The INtime Device Configuration tool reserves hardware devices for INtime usage. To run the tool, go to Control Panel>INtime>Device Configuration. Use the tool to select which devices to reserve for INtime. The tool then analyzes the IRQ resources and ensures that the allocation is possible.

In cases where you cannot physically isolate the device, the hardware/motherboard combination in your system is not suitable for running the INtime device driver.

In the case where your device does not generate an interrupt or it supports the MSI method of interrupt delivery, you can pass the device to INtime by selecting the “No Interrupt or MSI” option. In this case, interrupt conflict is not an issue.

To install two real-time devices, configure both devices at the same time with the Device Configuration tool. It ensures that IRQ resources are not shared with Windows, but allows INtime devices to share the same IRQ when necessary.

7 Configuration

INtime software provides the flexibility to meet a variety of INtime application requirements that you set using configuration options available in the INtime Configuration utility. This chapter describes the following options:

Configuration option	Page
Configuring INtime software	67
Default configuration	67
Running the INtime Configuration Utility	68
Miscellaneous	69
Configuring INtime applications	69
Configuring Windows for non-interactive logon	69
Configuring the INtime Network software	71
Before you begin	71
Hardware installation	71
Setting the TCP/IP configuration parameters	72
NIC driver configuration	72

Configuring INtime software

Default configuration

By default, the Install program configures INtime software to:

- Require manual start up for the INtime Kernel service. The installation program configures all other INtime services to start automatically.
- Install INtime software files in the `%PROGRAMFILES%\INtime` directory.
- Access INtime application wizards, their components, and Help files from the directory appropriate for the version of Microsoft Visual Studio installed on your system.
- Install sample programs only for the user that installed INtime software.

The INtime Configuration utility's Help contains step-by-step instructions for many tasks. You can access this Help by running the utility, then pressing the Help button located at the bottom of the window.

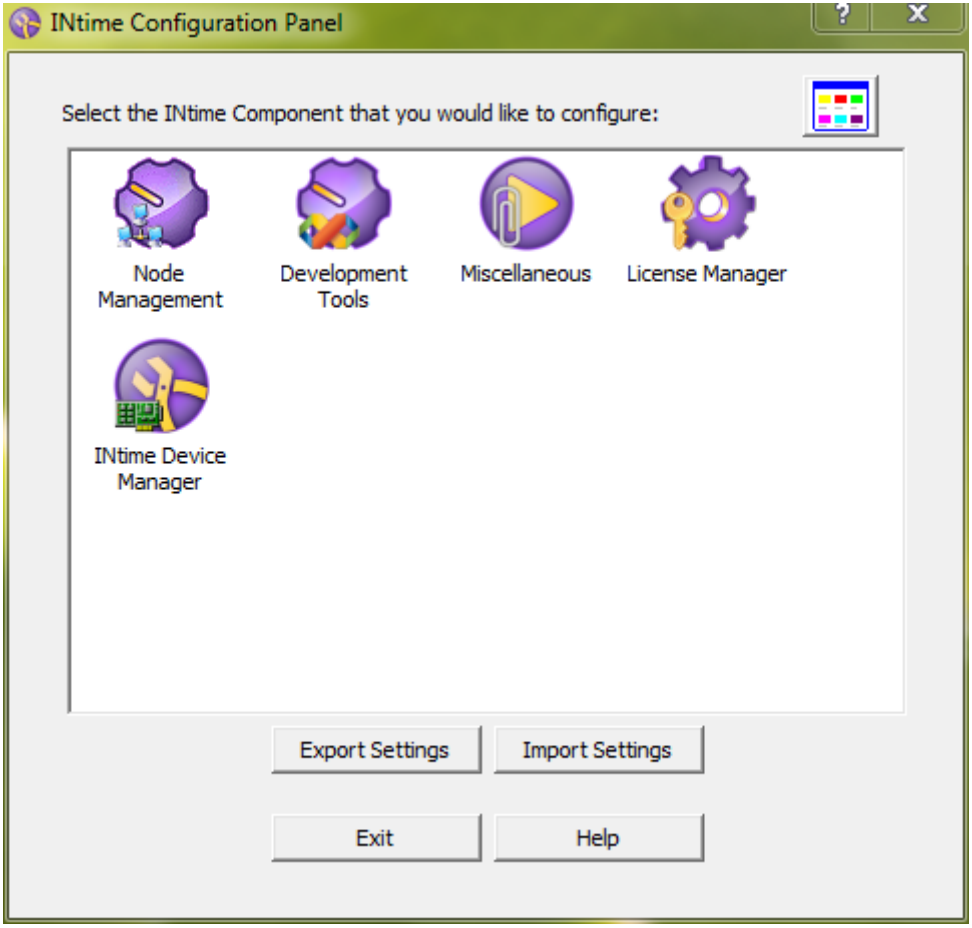
Running the INtime Configuration Utility

To access the Configuration utility, do one of these:

- Click (the INtime icon) in the Windows Notification Area (aka the System Tray), then select INtime Configuration.
- Select Start>Control Panel>INtime.

The INtime Configuration Panel displays:

Figure 7-1. INtime Configuration Panel



Note

Press F1 to view step-by-step instructions for many configuration tasks.

Tab	Description
Node Management	Configures settings that affect RT kernel operation for each configured RT kernel. These include the amount of memory allocated to the RT kernel, and the system clock period as well as other kernel configuration parameters.
License Manager	Collects information for node-locking the installation, and allows the installation of full license codes.
Development Tools	Installs, uninstalls, and verifies the status of INtime wizards for Microsoft Visual Studio.
Miscellaneous	Configures other settings such as automatic Windows logon, Fault Manager behavior and the size of the console windows where RT applications display text.
INtime Device Manager	Configures hardware devices for use with RT applications.

Miscellaneous

RTIF.SYS driver

When you install INtime software, the installation process sets up the Windows system to allow Rtif.sys, the RT kernel mode device driver, to load at boot time.

Interrupt resources

In Shared Mode, INtime software takes over the system clock (IRQ0) and multiplexes the clock interrupt between the RT and the Windows kernels. INtime applications can take over other interrupts, but must ensure that Windows drivers do not try to use the same interrupt signals (IRQs).

Use the INtime Device Manager applet to assign devices and interrupt resources to the INtime kernel.

Configuring INtime applications

This section details the steps required to configure INtime applications for specific conditions. You can also access this information in the INtime Configuration's Help file.

Configuring Windows for non-interactive logon

Executing Windows and INtime software without an attached keyboard and mouse requires that Windows be configured for non-interactive logon, also known as AutoAdmin Logon.

To configure Windows for non-interactive logon:

1. Log on to Windows with Administrator privileges.
2. Create/verify the presence of a Windows user with a password as to which you want to automatically log on.
3. Start the INtime Configuration application.
4. Select the Miscellaneous applet tab, then edit these fields:
   - **Automatic Logon Options**: Select Auto Logon Always.
   - **Auto Logon Parameters**: Enter the information in these fields:

     Domain Name
     User Name
     Password

     Note: Be sure to use the User name and Password established in step 2..
5. Exit the INtime Configuration application.
6. Restart Windows.

When Windows restarts, it automatically logs on as the specified user in the specified domain.

⚠ CAUTION
Deploying a Windows system with the non-interactive logon feature enabled represents a potential security breach: the Windows registry stores the specified user, domain, and password in clear text—they are not encrypted.

Configuring INtime Local Kernel service to execute automatically

To configure the INtime Local Kernel service to start automatically during Windows restart:

1. Go to the Node Manager applet and select the kernel you want to configure.
2. Select the Kernel tab, and then check the Start INtime Kernel Automatically checkbox.
3. Click the Save button
4. Restart your system.

Automatic loading of Realtime Applications

INtime software can load your realtime applications automatically when the Local Kernel service starts:

1. Open the Node Manager applet in the INtime Control Panel utlity and select the node you want to configure.
2. Select the Autoload tab, and then click the Add button.

3. Enter the Application Title, and the full path of the application (.RTA file) to load.
4. Add any command-line parameters you want to specify.
5. For other parameters, click the Advanced button.
6. If your application is dependent on another application being loaded first, enter the name of the dependent application in the Dependencies list by clicking the Add button.
7. Click the Save button. When the INtime Kernel restarts, your application automatically starts.

Configuring the INtime Network software

To configure the INtime kernel TCP/IP software, click the Network tab on the INtime Node Manager Panel for your selected node.

Before you begin

You will have to assign an IP address and host name to the network interface controlled by the INtime TCP/IP software. This should not be the same address used by your Windows networking software. Alternatively, you can specify that a DHCP server automatically provide the address.

Hardware installation

Before configuring INtime TCP/IP software, install and configure the network interface adapter you plan to use. Instructions for installing hardware for RT kernel use may be found in *Installing hardware for use with the RT kernel* in *Chapter 6, Installation*.

Setting the TCP/IP configuration parameters

1. Start the Node Manager applet.
2. Select your node, then click the Network tab.
3. Set the following:

To...	Select this checkbox...
Automatically start real-time networking services	Start the network automatically

4. Modify the host name as desired.
5. Click the NIC Configuration button. The NIC Management Dialog displays. Edit or Add the interfaces you require, as needed. Each configured interface needs one of these:
   - Check the Enable DHCP box.
   - Enter the following:

     IP address
     IP interface address mask.
6. Exit the NIC Configuration Dialog.
7. If you want to use a DNS server for name resolution, select the DNS checkbox, then enter the domain name and up to three IP addresses of DNS servers.
8. Click the Save button.

If the INtime kernel is already loaded, you must restart the kernel for changes to take effect.

NIC driver configuration

When adding or editing a NIC interface, you must select the NIC driver appropriate to your hardware. Devices are referred to by their INtime instance regardless of the actual number of devices in a system. Consider a quad-port Intel 1 Gbit/s card of which two devices, `ie1g0` and `ie1g1`, are passed to INtime. The instance number implies no ordering or grouping but simply indicates the first and second devices of that type passed to INtime.

8 Connecting to an INtime host

This release of INtime 4.0 does not include the ability to create a stand-alone INtime host, but it is still possible to connect to existing hosts. This feature will be re-added in a later product update.

Creating a connection to an INtime host

You can connect your INtime installation to these host types:

- **An existing INtime host (a stand-alone INtime installation, without Windows).** This type of host must have been previously created with an earlier version of INtime software.
- **An INtime runtime installation that runs WinNTXproxy.exe to debug your application**. In this case the WinNTXproxy application services the NTX connection in order to allow you to make NTX calls on the remote host. This can assists with debugging a runtime system.

Fixed and Passive Connections

NTX connection types include:

- **Fixed connections** use fixed IP addresses to identify each end of the connection.
- **Passive connections** use the mDNS protocol to advertise the connection automatically in order that you do not have to create a fixed configuration.

To Create a Fixed Connection

Before you begin, ensure that you know the IP address of each end of the connection and configure each end to address the connection's other end.

To create the connection:

1. Start the INtime Configuration Panel and open the Node Management applet.
2. Click New Node, and fill in the name of the node, select "Remote or Windows NTX connection" and fill in the IP address of the INtime host. If this is a WinNtxProxy connection to another Windows host, check the "WinNtxProxy Node" box.
3. On your INtime node, edit the rtload.sys file to change the IP address of the ntxproxy Windows host, if necessary.

To Create a Dynamic Connection

In this case you do not need to know the IP address of the connection endpoints and you do not need to create a connection using the INtime Control Panel.

1. Download the Apple Bonjour for Windows service from the Apple website. This is an implementation for Windows of the mDNS "zero-configuration" protocol.
2. Install the service.
3. Ensure that the INtime mDNS service is enabled.
   A. Go to the Windows Service Manager and locate the INtime mDNS Service.
   B. Open Properties and set Startup Type to "Automatic".

      After rebooting the system is now ready to accept mDNS connections.

To connect to an INtime host, make sure that the host is configured as a dynamic node by loading the mdnsintm.rta process.

To create a dynamic connection to a WinNtxProxy host:

1. Ensure that the target system is configured for mDNS operation, as above.
2. Start the WinNtxProxy process by entering this command line option:

   ```
   -passive
   ```

You can now connect to the INtime host from your development host.

9 Operation

This chapter describes how to start and operate INtime software.

You can treat the RT kernel and associated INtime applications as Windows services and start them automatically when Windows initializes.

In the event of a Windows blue screen crash, INtime software keeps running until a graceful shutdown occurs. To start INtime software again, you must first restart Windows.

Starting the RT kernel and related components

1. Open the Windows Service Manager (Start>Control Panel>Administrative Tools>Services).

   The Windows Services table lists these INtime services:

Table 9-1. INtime software's Windows services

Service	Default startup setting	Description
INtime Kernel Manager	Automatic	Loads the RT kernel binary image into memory and starts it. Uses loadrtk.exe.
INtime Clock Synchronization service	Automatic	A Windows program that synchronizes RT client time-of-day clock with the Windows host's time-of-day clock.
INtime Event Log service	Automatic	A Windows program that acts as a gateway to allow RT applications to log events in the system event log on the Windows host.
INtime I/O service	Automatic	A Windows program that acts as a server to RT "C" library to obtain a console window for display (via printf) of application data and to receive (via scanf) user input from the system keyboard. This service also provides Windows file system support to the RT "C" library.
INtime Node Detection service	Automatic	A Windows program that detects RT clients, both local and remote. This program checks for and registers RT clients that exist in both of these locations: • RT clients configured in the INtime Configuration program (INconfCpl.cpl). • RT clients available to the system.

Table 9-1. INtime software's Windows services

Service	Default startup setting	Description
Intime Remote Connection manager	Automatic	A Windows program that detects and manages connections between the Windows host other INtime hosts.. The manager includes NtxRemote2.exe, which is required for a Windows host to communicate with other INtime hosts.
INtime Registry service	Automatic	A Windows program that provides Windows registry access to RT processes via the RT application library.
INtime mDNS service	Manual	A Windows program that uses the mDNS protocol to automatically configure NTX connections.
INtime Debug service	Manual	Manages the console window on behalf of the INtime system Debug Monitor.

2. Start each INtime software service as desired. By default, all services except the INtime mDNS service and INtime Debug service start automatically.

   You can also start an INtime Kernel using these methods:

   - Select a kernel in the Node Management Control Panel applet and click the Start Local button.
   - Click the INtime icon in the system tray and selecting a kernel name from the menu that displays.
   - Open a command window and enter this command:

     ```
     nodemgr "start NodeA"
     ```

   - Call ntxStartLocalRtNode from a Windows application.

After you start the INtime kernel

- Use Microsoft Visual Studio to develop your INtime application.
- Use a debugger to load and debug your RT applications.
- Use the INtime Loader to load and run your RT applications. You can start this program by selecting Start>All Programs>INtime>RT Application Loader. You can also load your application by double-clicking the .rta file in an Explorer window, or by right-clicking the .rta file to open the RT Application Loader.

- INtime tools available from the Start>All Programs>INtime software menu include:

Note
One or more of these tools are installed, depending on which INtime Environment option you selected when you installed INtime software.

Table 9-2. INtime software tools

Tool	Description
INtime Configuration	A Windows program that configures INtime software.
RT Application Loader	A Windows program that loads/starts the RT portion of INtime applications.
Spider Debugger	A Windows program that communicates using NTX calls, with the INtime debug server to provide dynamic source level, multi-tasking debug capabilities for RT applications. **Note:** For detailed information about using Spider, see Spider Help.
INtime Explorer	A Windows program that uses NTX calls to communicate with its self-loaded RT counterpart to display objects inside an INtime node.
INscope Real-time Performance Analyzer	A Windows program that uses NTX calls to communicate with its self-loaded RT counterpart to trace execution of INtime applications.
INtime Status Monitor	A program that you use to start and stop the kernel, as well as open the INtime configuration panel.
NTX Connect-Disconnect utility	A tool that connects dynamic remote nodes.
Remote Maintenance utility	A tool that manages files on a remote node. Can also restart the remote node.

Also available from the Start>All Programs>INtime software menu are these resources:

- **INtime Help**: Describes INtime software concepts and explains how to use INtime tools. INtime Help includes all system calls, including their syntax which you can cut and paste directly into your code.

To access INtime Help, do one of these:

- **Within Microsoft Visual Studio**: INtime software content is integrated with the Visual Studio help collections. You can filter Visual Studio's help to display only INtime software content by entering the keyword "INtime software".

  Note

  You cannot select INtime Help from within Visual Studio 6. You must manually access the INtime Help file by selecting Start>Programs>INtime>Documentation>INtime Help.

- **From the Start menu**: Select Programs>INtime>Documentation>INtime Help.

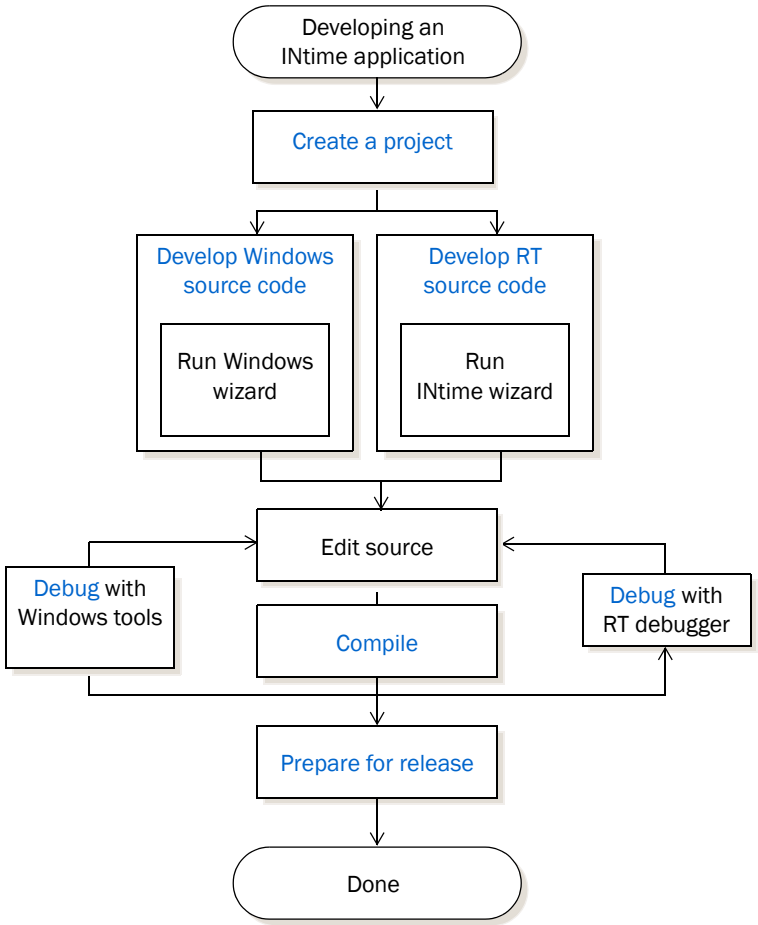
- **INtime Release Notes**: Lists features and issues that arose too late to include in other documentation.
- **Sample Code and Projects**: Menu of Visual Studio projects that demonstrate various aspects of INtime application programming.
- **Quick Start Projects:** Menu of projects described in the Quick Start Guide.

10 INtime application development

This chapter describes how to create INtime applications.

Typically, you develop both Windows and RT source code as shown in this illustration. The remainder of this chapter details each of these steps.

Figure 10-1. Developing an INtime application



Create a project

To develop RT applications using INtime software, you must have INtime installed on your system, and a supported version of Microsoft Visual Studio running. For a list of supported versions, see *Requirements* on page 63.

Before creating the project, decide how to set it up. Typical structures for INtime applications include:

- Set up the project as a single solution with the Windows portion as a project and the RT portion as a project. Use this approach when you want to start your INtime application in Windows and invoke the RT functions.
- Set up each portion as a solution.

An INtime application typically results in one Windows executable (a .DLL or .EXE file) and one RT executable (an .RTA file).

Develop Windows source code

To develop the Windows portion of an INtime application, use Microsoft Visual Studio as you normally would:

1. With Microsoft Visual Studio running, select a standard application wizard. Use the wizard to build the Windows portion of your INtime application.
2. Use the Microsoft Visual Studio's editor to edit the generated code and link the application.
3. When editing your Windows source code, use the NTX calls provided with INtime software to access RT functionality.
4. Debug with the debugger included as part of Microsoft Visual Studio.

Adding the INtime RT Client Browser to your INtime application

The INtime RT Client Browser (INBROW.OCX) is an ActiveX control that you can add to your INtime applications. Add the browser using the Microsoft Visual Studio menu. The Node Browser then displays as an available control on the Controls Toolbar. Add the control to the dialog you desire. For information about including ActiveX controls in projects, see the Microsoft Visual Studio documentation.

Once in your project, the control offers these functions:

Function	Description
GetCurrentName	Obtains the name of the item highlighted in the INtime RT Client Browser.
GetCurrentState	Obtains the state of the item highlighted in the INtime RT Client Browser. Valid states include: 0 Not an INtime node 1 ACTIVE 2 OFFLINE 3 CONFIG 4 DISCONNECTED Where: ACTIVE RT nodes the browser currently can communicate with. OFFLINE RT nodes the browser has but cannot currently communicate with. CONFIG RT nodes the browser has not communicated with. These nodes may not exist. DISCONNECTED RT nodes that are disconnected.
GetMask	Indicates which RT nodes display in the INtime RT Client Browser.
SetCurrentName	Not supported.
SetCurrentState	Not supported.
SetMask	For SetMask, the mask is any combination of these flags: 1 Show ACTIVE nodes 2 Show OFFLINE nodes 4 Show CONFIG nodes 8 Do not show LOCAL nodes 16 Do not show Remote nodes 32 Show DISCONNECTED nodes If not explicitly set, the mask is 0x27, which means ACTIVE, OFFLINE, CONFIG, DISCONNECTED. Where: ACTIVE RT nodes the browser currently can communicate with. OFFLINE RT nodes the browser has but cannot currently communicate with. CONFIG RT nodes the browser has not communicated with. These nodes may not exist.
SetSelected	Identifies which INtime RT node to select in the browser window when you specify a pointer to a string that contains a node name. This function returns TRUE if the name is valid and selected; otherwise it returns FALSE.

Develop RT source code

To develop the RT portion of an INtime application, use the INtime wizards available in Microsoft Visual Studio. The INtime wizards guide you through the decisions required to develop the RT portion of an INtime application and generate the corresponding code framework which you fine-tune in the Microsoft Visual Studio's editor.

You can select one of these INtime wizards:

- **Application wizard**: develops the RT portion of RT applications.
- **Shared Library wizard**: generates code for a Realtime Shared Library.
- **RT Static Library wizard**: Generates code for a Realtime Static Library.

Note

The RT portion of INtime applications support only source code written in C or C++. They do not support applications written using MFC and/or ATL.

Running the INtime Application wizard

To create the RT portion of an INtime application:

1. Select what you want to generate:
   - An empty project (the wizard simply sets up the correct compiler and linker settings for you). If you select this option, go to step 4.
   - A simple "Hello World" application (the wizard creates a simple source file and adds it to the project). If you select this option, go to step 4.
   - A minimal iwin32 application. The wizard creates a project with one source file, set up to use iwin32 API calls. If you select this option, go to step 4.
   - A full-featured INtime application. If you select this option, continue to the next step.
2. Add elements to your process:
   A. Select one or more elements from the main screen to add to your RT application:
      - Mailbox or semaphore server thread
      - Thread that operates at a regular interval
      - Interrupt handling
      - Shared memory allocation

- Client thread

> **Note**
> For detailed information about the fields on these screens, see Help. For information about accessing help, see *Where to get more information* on page v.

B. Click the Add element button. That element's detail screen displays.

C. Specify element parameter values.

D. When satisfied with an element's settings, click the OK button. The wizard's main screen displays again.

3. (Optional) Change the global process settings:

   A. Select the `-global-` option from the main screen.

   Note: You must click in the Name column to access this detail screen.

   B. Specify global process values.

   C. When satisfied with the global settings, click the OK button. The wizard's main screen displays again.

   Note: If you don't access this screen now, the wizard prompts you to verify global settings before generating the process.

4. Generate the process. The wizard creates a project from the settings you specified.

   A. Click the Generate Process button. If you did not access the Global Settings screen, the wizard prompts you to edit or accept the global process settings. The wizard then displays the New Project Information screen and prompts you to verify the process information.

   B. Click the OK button. The wizard creates a process from the settings you specified. A process may include these files:

   - Main file (which has the project name) with a .C or .CPP extension; for example: TEST.C.
   - INtime project file TEST.intp, c project file TEST.vcproj, and solution file TEST.sln.
   - Project header file with a .H extension.
   - Utility function file, UTIL.C.
   - A C source file with a .C extension for each thread.
   - A text file called README.TXT that describes each generated file.

Running the INtime process add-in wizard

The INtime Process Add-in wizard can be used only for a project generated by the RT Process wizard with the C++ and Full-Featured options selected. To start it, select Tools>TenAsys INtime C++ wizard.

To modify an existing RT process:

1. Add elements to your process:
   A. Select one or more elements from the main screen to add to your RT application:
      - Mailbox or semaphore server thread
      - Thread that operates at a regular interval
      - Interrupt handling
      - Shared memory allocation
      - Client thread

      Note

      For detailed information about the fields on these screens, see Help. For information about accessing help, see *Where to get more information* on page v.

   B. Click the Add element button. That element's detail screen displays
   C. Specify element parameter values.
   D. When satisfied with an element's settings, click the OK button. The wizard's main screen displays again and a message indicates what the wizard added to that process.
2. When you are satisfied with your additions, click the Close button to exit the wizard.

Running the INtime Shared Library wizard

The INtime Shared Library wizard generates the framework for a Real-time Shared Library. The code generated illustrates how to export both function and variable names, and also initializes the project settings to correctly generate an RSL.

The generated files include:

- Main file (which has the project name) with either a .C or a .CPP extension, depending on which option was chosen, for example TEST.C.
- Project files, for example TEST.intp and TEST.vcproj.
- Project header file, for example TEST.h.
- Utility function file, UTIL.C.

The main file contains a function RslMain which you can modify, if required. A default RslMain function is linked if this function is deleted from the source file. For further details about RSLs, see Help.

Running the INtime Static Library wizard

The INtime Static Library wizard generates a project that builds a static library fit for linking into INtime projects. After running the wizard, the library is empty. Add files as necessary. Successfully building a static library project results in a LIB file that can be linked into other projects.

The difference between a static and a shared library is that code and data from a static library is added to an executable file at link time, whereas code and data from a shared library is added only when the executable file loads. With a shared library, the shared code can be updated without rebuilding the executable file, but use of a shared library involves some minor overhead.

Compile

Use Microsoft Visual Studio to compile and link the application. The RT portion of INtime applications requires certain project settings. If you use the INtime wizards to develop the RT portion of your application, the wizards will set up your project properly. Settings vary, depending on whether you configured Microsoft Visual Studio to build a debug or a release version.

To view and verify settings required by INtime software, select Microsoft Visual Studio's Build>Settings menu option.

Note
Only required RT portion settings are listed. You can enter the values you want in fields for which no setting is listed.

Visual Studio settings vary, depending on the version you use. Go to the appropriate section to see the settings for your version:

- **Visual Studio 2008**: continue reading in the next section.
- **Visual Studio 2005 (aka Visual Studio 8)**: go to page 88.

Visual Studio 2008

General

Dbg	Rel	Field	Value
X	X	Use of MFC	Use Standard Windows Libraries
X	X	Use of ATL	Not Using ATL
X	X	Character Set	Not Set (Unicode not supported)
X	X	Common Language Runtime support	No Common Language Runtime support

Debugging

Dbg	Rel	Field	Value
X	X	Debugger Type	Auto
X		SQL Debugging	No

C/C++

Dbg	Rel	Category	Field	Value
X	X	General	Additional Include Directories	Must include %INtime%rt\include
X			Debug Information Format	Program Database (recommended for Debug configuration only)
X	X	Preprocessor	Preprocessor definitions	Add VS7_CPP if using C++
X	X		Ignore Standard Include Files	Yes
X	X	Code Generation	Enable C++ Exceptions	If Yes, select the Exception handling C++ libraries
X	X		Basic Runtime Checks	Default
X	X		Buffer Security Check	No
X	X	Language	Enable Run-Time Type Info	No

Linker

Dbg	Rel	Category	Field	Value
X	X	General	Version	21076.20052
X	X		Enable Incremental Linking	No
X	X		Additional Library Directories	Must include %INtime%rt\lib
X	X	Input	Additional Dependencies	rt.lib and a selection of iwin32.lib, pcibus.lib, netiff3m.lib, ciff3m.lib, rmxiff3m.lib, rtpp*.lib ecpp7*.lib
X	X		Ignore All Default Libraries	Yes
X	X	System	Subsystem:	Console
X	X		Heap Reserve Size	0 (zero) which sets this value to the maximum pool size (in bytes)
X	X		Heap Commit Size	0 (zero) which sets this value to the minimum pool size (in bytes)
X	X		Stack Reserve Size	Virtual segment (size in bytes)
X	X		Stack Commit Size	Stack size for the main thread (in bytes)

In the list of object/library modules, the following choices depend on other settings:

1. For C++ projects that use the INtime RT classes, add rtppd.lib for the debug version and rtpp.lib for the release version.
2. For C++ projects, add ecpp7*X*.lib, where *X* is one of these:

null	No exceptions, no namespaces.
`E`	Using exceptions, no namespaces.
`N`	No exceptions, using no namespaces.
`EN`	Using exceptions and namespaces.

3. The C runtime library is named ciff3m8.lib.

Visual Studio 2005 (aka Visual Studio 8)

General

Dbg	Rel	Field	Value
X	X	Use of MFC	Use Standard Windows Libraries
X	X	Use of ATL	Not Using ATL
X	X	Minimize CRT Use in ATL	No
X	X	Character Set	Not Set (Unicode not supported)
X	X	Common Language Runtime support	No Common Language Runtime support

Debugging

Dbg	Rel	Field	Value
X	X	Debugger Type	Auto
X		SQL Debugging	No

C/C++

Dbg	Rel	Category	Field	Value
X	X	General	Additional Include Directories	Must include %INtime%rt\include
X			Debug Information Format	Program Database (recommended for Debug configuration only)
X	X	Preprocessor	Preprocessor definitions	Add VS7_CPP if using C++
X	X		Ignore Standard Include Paths	Yes
X	X	Code Generation	Enable C++ Exceptions	If Yes, select the Exception handling C++ libraries
X	X		Buffer Security Check	No
X	X		Basic Runtime Checks	Default
X	X	Language	Enable Run-Time Type Info	No

1. The C runtime library is named ciff3m.lib.

Dbg	Rel	Category	Field	Value
X	X	General	Output filename	Should have an .RTA or .RSL extension
X	X	Output or General	**Important**: You must add value: /heap:0x100000,0x2000 to set the memory pool for the process	
X	X	General or Input	Ignore all default libraries	Enabled (checked)
X	X	Input or General	Additional library path	$(INtime)rt\lib
X	X	Input	Object/library modules or Additional Dependencies	rtpp[d].lib ecppXY.lib ciff3m[8].lib rt.lib rtserv.lib pcibus.lib netiff3m.lib rmxiff3m.lib
		Output		
X	X	Stack allocations	Reserve Commit	0x100000 0x2000
X	X	Version information	Major Minor	21076 20052

Debug

You must debug both portions of your INtime application using the appropriate tools:

- **Windows portion**: Use standard Windows development tools, including the Microsoft Visual Studio debugger.
- **RT portion**: Use Visual Studio debugger or Spider plus the other debug tools provided with INtime software.

Using the two debuggers, you can simultaneously view and debug on-target application code in both the Windows and the RT environments.

- **Spider (SPIDER.EXE)**: A Windows-based RT debugger that supports on-target debugging of RT threads. The Spider debugger fully comprehends the RT constructs supported by INtime and supports dynamic debugging. Spider can debug multiple RT threads simultaneously while other threads continue to run.
- **System debug monitor (SDM)**: A command-line interface that provides low-level, static debugging capabilities.

Note

For detailed information about using Spider, see Spider's Help. For detailed information about using SDM, select Debuggers>Low-level debugger>System Debug Monitor (SDM) in INtime Help.

- **Fault Manager:** This application pops up a dialog when a hardware fault is detected on the INtime kernel. The user may then choose one from a list of actions to handle the fault condition.
- **INtime Explorer (INtex)**: A Windows-based RT object browser. The INtex program, through its self-loaded RT counterpart, can show the state of all RT processes, threads, and objects. It can also create a crash report when a thread in a process has had a hardware exception (crash analysis).
- **INscope**: The INScope Real-time Performance Analyzer is a Windows application that allows you to trace execution of INtime applications. Trace information for thread switches, system calls, and interrupt handling is displayed in a graphical user interface with various tools available to allow system analysis.

Debugging tips

Performance monitor

To view CPU usage in both the Windows and RT kernels, you can run the Windows Performance monitor. Viewing CPU activity provides the feedback you need to determine if you divided labor appropriately between Windows processes and RT processes. For more information about designing your INtime applications, see *Chapter Chapter 5, "Designing RT applications"*.

To view Windows and RT kernel activity in the Windows Performance Monitor:

1. Open the Performance Monitor by selecting this option from the Start menu:

   Control Panel>Administrative Tools>Performance>System Monitor

2. Open a chart by selecting the File menu's New Chart option.
3. Identify the performance metrics you want to view by choosing the Edit menu's Add to Chart option, then select these options:

Object	Counter	Purpose
Processor	% Processor time	Displays the percent of time devoted to Windows work.
INtime RT kernel	RT Kernel CPU usage (%)	Displays the percent of time devoted to RT work.
	NTX Outstanding Requests	Displays the number of NTX requests awaiting completion
	NTX Requests per second	Displays the number of NTX requests started each second

You can now run your INtime application and observe the CPU usage of both the Windows and RT kernels. When viewing the CPU usage, keep in mind that the Performance Monitor displays total CPU usage which includes more than the activity

generated by an INtime application. Each counter has a separate instance for each local INtime node, or you can select the total for all local nodes.

The NTX counters indicate how busy the Windows to INtime interface is. The outstanding requests counter shows how full the NTX queue is. The size of the queue may be adjusted in the System Wide configuration in the Node Manager, by changing the value of Max Concurrent NTX Requests.

Status messages

When the RT kernel is running, a protection fault which occurs in the RT portion of an INtime application is handled by the default system hardware exception handler which suspends the faulting thread. Such threads display in the INtime Explorer with the state '`suspended because of an exception`'. For threads in this state, the CPU frame displays the thread's CPU context when the hardware fault occurred, including the CS:EIP address of the faulting instruction.

The error code indicates the fault encountered as follows:

Fault	Code	Description
EH_ZERO_DIVIDE	`0x8100`	Divide by Zero error
EH_SINGLE_STEP	`0x8101`	Single Step
EH_NMI	`0x8102`	NMI
EH_DEBUG_TRAP	`0x8103`	Debug Interrupt (Ignored by handler)
EH_OVERFLOW	`0x8104`	Overflow error
EH_ARRAY_BOUNDS	`0x8105`	Array Bounds error
EH_INVALID_OPCODE	`0x8106`	Invalid Opcode error
EH_DEVICE_NOT_PRESENT	`0x8107`	NPX device not present
EH_DOUBLE_FAULT	`0x8108`	Double Fault error
EH_DEVICE_ERROR	`0x8109`	NPX device error
EH_INVALID_TSS	`0x810A`	Invalid TSS error
EH_SEGMENT_NOT_PRESENT	`0x810B`	Segment Not Present error
EH_STACK_FAULT	`0x810C`	Stack Fault
EH_GENERAL_PROTECTION	`0x810D`	General Protection Fault
EH_PAGE_FAULT	`0x810E`	Page Fault

If a debugger is running when the fault occurs, the debugger traps the fault and allows you to debug or delete the offending program.

Prepare for release

Before you begin

- On the target system, ensure that you are logged on with Administrator privileges.
- Exit all programs prior to installing INtime software.
- If your system has a previously installed version of INtime software, remove it using the Add/Remove Programs Applet in the System Control Panel (select: Start>Control Panel>Add/Remove Programs. Highlight INtime 3.1 (or earlier) program, and then click Remove). Make sure none of the INtime services are running. If they are running, ensure they are set to Manual Start (using the Start>Control Panel>Administrative User/Services applet) and reboot the system before you can successfully complete the uninstall operation.

Using launch-rt.exe

To install INtime runtime software:

1. Start the launch-rt.exe program by double-clicking it. You can find launch-rt.exe in the redistribution folder of the CD-ROM, or in the most recent launch-rt.exe installer obtained form TenAsys.

   If the Installation program detects a previous version of INtime software, it prompts you to exit the installation and uninstall the previous version.

   After the prerequisites have been satisfied the launch program will launch runtime400.exe and exit.
2. Review the INtime License and note its terms and conditions.
3. Select a destination directory for the INtime software files, then click the Next button. The default is %PROGRAMFILES%\INtime.
4. Click the Next button to install the software.

   The runtime400.exe program creates the directory you specified, then installs the INtime software files.
5. Click the Finish button to complete the installation process.
6. Restart your system using one of these methods:
   - Select OK at the last screen. The runtime400.exe program restarts your system.

- Restart your system at a later time. You must restart your system before running INtime software.

Note

The runtime400.exe executable installs those portions of INtime software that you can include in your derivative works. Use of runtime400.exe is governed by the INtime Software Redistribution License Agreement you must enter into in order to include INtime Software in your product.
For information about obtaining a Software Redistribution License and on the associated per unit royalty due to TenAsys Corporation for use of the INtime software in your product, contact sales@tenasys.com.

Note

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

Sample INtime applications

INtime software includes a number of sample applications that you can use as samples for your own INtime applications. The source code for these applications are provided in the following Visual Studio project formats:

- Visual Studio 2005
- Visual Studio 2008

The files reside in separate directories per demo. For instance, the INtime API test program source files reside in the following directory:

```
My Documents\INtime\Projects\rttest
```

You can open the projects for these applications from the Start>All Programs>INtime>Sample Code and Projects menu.

Note

- Sample applications are installed only in the My Documents folder of the user that installed INtime.
- Sample program cannot run until you compile them.

EventMsg DLL Project

This DLL allows you to customize event messages.

Item	Source
Pathname	My Documents\INtime\Projects\eventmsg
Runtime requirements	The EventMsg Project builds the EventMsg.DLL file properly only if you first build the project under release mode from within the Microsoft Visual Studio. After building the project under release mode, you may then build the project under debug mode.

INtime API Sample

This test application exercises most INtime software system calls.

Item	Source
Pathname	My Documents\INtime\Projects\rttest

Serial Communications Sample

This project demonstrates how to use the INtime Serial Communications library. The library and accompanying drivers allows the user to access serial devices such as the COM PC ports, RocketPort multi-channel PCI devices, and Edgeport multi-channel USB devices.

The driver (C:\Program Files\INtime\Projects\serialio\debug\serdrvr.rta) operates as an INtime service that manages a serial device attached to COM1 or COM2 on an INtime system. The demo (C:\Program Files\INtime\Projects\serialio\debug\serialio.rta) uses an INtime Message Port to obtain read/write access to the COMn channel managed by the Serdrvr.rta service. Both components are provided in source and binary form. For more information, see the README.TXT file in the source code directory for this sample application.

Item	Source
Pathname	My Documents\INtime\Projects\commsamp

Graphical Jitter

This application measures the minimum, maximum, and average times between low-level ticks via an Alarm Event Handler. Because this application is made from both an RT and a Windows executable, it shows both INtime and INtimeDotNet API usage.

INtime Graphical Jitter includes these executables:

- **Jitter.exe**: The Windows executable. Automatically starts Clk1Jitr.rta, then processes output and displays a histogram.

- **Clk1Jitr.rta**: The RT executable. Started by Jitter.exe.

Item	Source
Pathname	My Documents\INtime\Projects\jittercs
Invocation	Invoke the Windows jitter.exe program.

Real-time Interrupt Sample

This application tests the INtime RT Interrupt system calls using the Transmitter Ready interrupt from COM1.

The INtime RT Interrupt API Test takes over COM1 and toggles its Transmitter Ready Interrupt. When the test ends, COM1 is disabled. Make sure COM1 is available on your system before running this test application. When you run the test, continuous activity occurs on the RT side, preempting Windows activity for eight 10-second time periods.

Item	Source
Pathname	My Documents\INtime\Projects\inttest

C and C++ Samples for Debugger

These simple C and C++ programs are provided as a vehicle to demonstrate the Spider debugger's capabilities. The C++ program also demonstrates several components of the C++ language available to RT applications, as well as basic classes, dynamic instantiation, operator overloading, and so on. It also shows the libraries and startup modules needed.

Item	Source
Pathname	My Documents\INtime\Projects\csamp (C program) My Documents\INtime\Projects\cppsamp (C++ program)

TCP Sample Applications

Sample project that demonstrate TCP communications between a client and server. Client and server code is provided for INtime, and server code for Windows.

Item	Source
Pathname	My Documents\INtime\Projects\tcpsample
Invocation	See the ReadMe.txt file in the source code directory.

UDP Sample Applications

Sample project that demonstrate a UDP ping-pong type application. Datagram packets are exchanged between INtime and Windows with an incrementing identifier in the payload.

Item	Source
Pathname	My Documents\INtime\Projects\Projects\udpsample
Invocation	See the ReadMe.txt file in the source code directory.

INtimeDotNet Sample Applications

Sample INtimeDotNet applications that demonstrate NTX communication via the INtime DotNet assembly.

Item	Source
Pathname	My Documents\INtime\Projects\INtimeDotNetSample
Invocation	See the ReadMe.txt file in the source code directory.

Fault Handling (ntrobust)

This INtime application has both a Windows and an RT portion. The Windows portion allows the user to set up timing parameters that control how often a thread in the RT portion causes a hardware fault. The application demonstrates how another RT thread can detect and log the failure, delete the offending thread, and recreate it, all without affecting Windows or other RT processes.

Item	Source
Pathname	My Documents\INtime\Projects\NtRobust My Documents\INtime\Projects\NtRobust\RtRobust\
Invocation	See the ReadMe.txt file in the source code directory.

Floating Point Exception Handling

This simple program demonstrates floating point exception handling.

Item	Source
Pathname	My Documents\INtime\Projects\FpExcep
Invocation	See the ReadMe.txt file in the source code directory.

RSL Examples

These RT programs demonstrate the creation and use of RT Shared Libraries, the RT analog for Windows DLLs.

Item	Source
Pathname	My Documents\INtime\Projects\RslTest
Invocation	See the ReadMe.txt file in the source code directory.

NTX Sample (MsgBoxDemo)

This INtime application has both a Windows and a RT portion. The Windows portion looks up an RT mailbox created by the RT portion, and then waits at the mailbox. When an RT thread sends a message to the mailbox, the Windows portion displays the received data in a message box on the Windows side. RT semaphore and RT shared memory usage are also demonstrated.

Item	Source
Pathname	My Documents\INtime\Projects\MsgBoxDemo My Documents\INtime\Projects\MsgBoxDemo\RtMsgBox
Invocation	See the ReadMe.txt file in the source code directory.

Windows STOP Detection sample (STOPmgr)

This sample application shows how an INtime application can detect either a Windows Crash (blue screen) or Windows Shutdown event and prevent Windows from completing its normal actions until the RT application has had a chance to do a "graceful" shutdown.

Item	Source
Pathname	My Documents\INtime\Projects\stopmgr

USB Client sample

This sample application demonstrates how to use the INtime USB subsystem. It monitors a USB keyboard and prints a dump of each keystroke as it occurs.

Item	Source
Pathname	My Documents\INtime\Projects\usbsample

Global Objects sample project

This project illustrates some aspects of the Global Objects feature of INtime.

Item	Source
Pathname	My Documents\INtime\Projects\globsample

High-Performance Ethernet (HPE) sample project

This project illustrates the use of the HPE drivers included with INtime.

Item	Source
Pathname	My Documents\INtime\Projects\hpeif2

PCAP Sample application

This project illustrates the use of the PCAP library to filter specific Ethernet packets from the network stack.

Item	Source
Pathname	My Documents\INtime\Projects\nicio

III Appendices

The appendices include:

Appendix A: INtime software system calls

Lists and describes system calls that threads in the RT portion of INtime applications use to communicate with each other and with Windows threads. You can find detailed information, including syntax and parameter values, in Help.

Appendix B: The iwin32 subsystem

Describes the iwin32 subsystem, which provides a Win32 API for the INtime kernel. It is a parallel API to the INtime API that makes porting of existing Win32 applications easier.

Appendix C: INtime directory structure

Describes the INtime directory structure.

Appendix D: INtime software components

Lists and describes INtime software program files.

Appendix E: Visual Studio debugging for older INtime projects

Describes how existing INtime projects may be upgraded to use the Visual Studio .NET product and its debugger.

Appendix F: Adding INtime software to an XP Embedded configuration

Lists and describes how to add INtime components to a Windows XP embedded development environment so that XP Embedded images can be produced that include these INtime components.

Appendix G: Troubleshooting

Lists problems you may encounter while running INtime software, and explains how to avoid or resolve those problems.

A INtime software system calls

This appendix lists and describes INtime software system calls. Use this appendix to identify the system calls you want to use for each RT kernel exchange object. The calls are arranged first by system call types, and then by objects available to that object type: NTX, high-, or low-level. Other calls are listed at the end of the appendix.

Note

For detailed information about system calls, including syntax and parameter values, refer to Help. For more information about accessing Help, see *Where to get more information* on page page v

Object	Page
System call types	102
RT system calls	105
Distributed System Management (DSM)	105
Exception handling	106
Global objects	106
Interrupts	107
Mailboxes	108
Memory management	109
Object directories	111
Ports	112
Processes	114
Regions	114
Scheduler	115
Semaphores	115
Status	116
System data	117
Threads	117
Time management	118
Heaps and memory pools	120
High-performance gigabit Ethernet	123
INscope calls	124
Network stack	124
PCI library calls	124
Real-time shared library (RSL) calls	125
Registry calls	125
RT services and device drivers	126
Serial Communications (COMM)	128
TCP/IP system calls	130
USB calls	130
INtimeDotNet calls	131
Input/Output Calls	133

System call types

Several types of calls exist, and each kernel exchange object has calls of one or more type associated with it:

- **Windows eXtension (NTX) calls**: Windows uses these calls to communicate with the INtime kernel. NTX calls allow Windows applications to operate on objects created by, stored in, and controlled by the RT kernel.
- **Real-time (RT) calls**: The RT kernel uses these calls to run the RT portion of INtime applications and communicate with Windows. INtime software provides two levels of system calls for RT kernel exchange objects:
  - **High-level (validating) calls**: Write, test, and debug your application using high-level calls with their protection and validation features.
  - **Low-level (non-validating) calls**: Use low-level calls where there is no other choice, such as with AlarmEvents and where a semaphore or a mailbox must be used with an interrupt handler.
- **RT services**: INtime real-time applications (RTAs) which process messages from a user and events from an interface, and the service handlers required to perform service-dependent functions of the system calls.

RT kernel objects provide these RT call levels:

Object	NTX	High-level	Low-level
Distributed System Management (DSM)	X	X	
Exception handling		X	
Global objects		X	
Interrupts		X	
Mailboxes	X	X	X
Memory management	X	X	
Object directories	X	X	
Ports	X	X	
Processes	X	X	
Regions		X	
Scheduler			X
Semaphores	X	X	X
Status	X	X	
System data	X		
Threads		X	
Time management			X
RT service handlers	X		

Regarding INtime software system calls:

- System call names correspond to related Windows functions, where appropriate. Where it is confusing to associate the function of an INtime software object with a Windows object, the names reflect the difference.

  For example, no Windows object corresponds closely to an INtime software region, so these are left as regions. On the other hand, INtime software alarms are encapsulated as an object which somewhat correspond to a specialized Windows event object, so this is called AlarmEvent. It does not, however, perform all the same functions as a Windows event.

- To avoid confusion with similarly-named Windows calls, INtime software system call names usually include the letters 'Rt'.

- RT kernel objects and the kernel interface which differ substantially from corresponding items in the other INtime software layers, include a 'kn' prefix to indicate calls that map to low-level RT kernel functions and deal with low-level kernel objects.

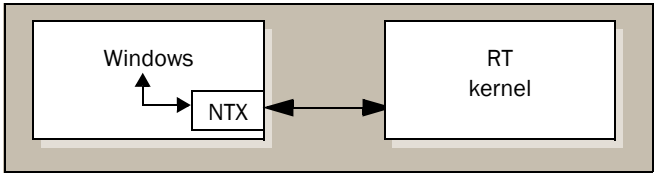
NTX calls

NTX calls allow Windows applications to operate on objects created by, stored in, and controlled by the RT kernel. This allows Windows and INtime applications to communicate and coordinate their activities.

Handle conversion

The NTX DLL converts in transit NTXHANDLES sent to the RT kernel to RTHANDLES, and vice-versa. This is necessary as the RT kernel cannot operate on NTXHANDLES and NTX can't operate on RTHANDLES.

Figure A-1. Converting NTXHANDLES to RTHANDLES



If a handle passes from RT to Windows in data, ntxImportRtHandle must convert it. Object directories and object mailboxes are converted automatically.

RT calls

You use these calls when developing the real-time portion of an INtime application.

High-level (validating) calls

High-level calls provide higher protection and validation features. Memory is allocated automatically from the process's pool. Each high level object consumes a slot from the system GDT (8000 objects maximum)

High-level system calls validate a call's parameters; a condition code returned by the call indicates whether you used invalid parameters. Condition codes for trying to read or write memory to which you have no access also exist.

High-level exchange objects validate parameters and are protected against unexpected deletion. High-level calls exist for these exchange objects:

- Object and data mailboxes
- Counting semaphores
- Regions (for mutual exclusion with priority inversion protection)

Low-level (non-validating) calls

Low-level calls provide higher performance, but lower protection and validation features. Low-level objects provide functionality beyond that of high-level objects.

You must allocate memory for low-level objects and may allocate memory beyond low-level object needs. You can use this additional memory to store application-specific state information associated with the object.

Low-level objects are not protected against unexpected deletion and do not validate parameters (if you need parameter validation, use high-level system calls). Low-level calls use the flat, 4 Gbyte addressing capabilities of the microprocessor. They do not use segmentation. Therefore, they do not consume a slot from the system GDT.

Use low-level objects in these situations:

- For isolated parts of the application, such as signaling ordinary threads from an interrupt handler.
- When performance is critical, such as high-performance, unvalidated sending and receiving of data mailbox messages and semaphore units.

Note

System calls that manipulate low-level objects assume that all memory reference pointers received are valid.

Low-level calls exist for these exchange objects:

- Data mailboxes
- Single-unit semaphores
- Region semaphores (with priority inversion protection)
- Software alarm events (virtual timers) that invoke alarm event threads that you write.

RT services

Real-time services include:

- **RT service calls**: An INtime real-time application (RTA) which processes messages from a user and events from an interface. A service is defined by the set of messages and the actions provoked by those messages. Each interface is associated with a service descriptor.
- **RT service handlers**: Subroutines invoked by the kernel to perform service-dependent functions of the system calls.

RT system calls

Distributed System Management (DSM)

NTX calls

System call	Description
ntxRegisterDependency	Registers a dependency
ntxUnregisterDependency	Unregisters a dependency
ntxRegisterSponsor	Registers a sponsor
ntxUnregisterSponsor	Unregisters a sponsor
ntxNotifyEvent	Notifies of an event

High-level calls

System call	Description
RegisterRtDependency	Registers a dependency
UnregisterRtDependency	Unregisters a dependency
RegisterRtSponsor	Registers a sponsor
UnregisterRtSponsor	Unregisters a sponsor
RtNotifyEvent	Notifies of an event

Exception handling

High-level calls

System call	Description
SetRtExceptionHandler	Assigns an exception handler and exception mode or changes the current mode for any of the following: • Current thread exception handler • Current process exception handler • System-wide exception handler
GetRtExceptionHandlerInfo	The GetRtExceptionHandlerInfo function returns the address and exception-handling mode for any of the following: • Current thread's exception handler • Current process' exception handler • System-wide exception handler • System-wide hardware exception handler (trap handler)

Global objects

Global object calls allow real-time processes to interact even when resident on different instances of the real-time kernel. The functionality is broadly similar to that provided to Windows applications by the NTX protocol, with enhancements to take into consideration issues of thread priority and multiple client access.

System call	Description
GetFirstRtLocation GetNextRtLocation	Returns the first or next LOCATION handle from the local location table.
GetRtNodeLocationByName	Returns a LOCATION for the given name.
GetRtNodeStatus	Returns a status value indicating the operational state of the target node.
GetRtNodeInfo	Returns information about a node given its location.
GetGlobalRootRtProcess	Returns a handle for a reference to the target node's root process.
CreateRtReferenceObject	Creates a reference object from a LOCATION and an RTHANDLE for a valid object on the node indicated by the LOCATION value and returns a RTHANDLE for the reference object.
DeleteRtReferenceObject	Deletes a reference object.
GetRtObjectInfo	Returns information about a reference object and its referenced object.
CreateGlobalRtSemaphore	Creates a semaphore on a remote node indicated by the LOCATION parameter.

System call	Description
CreateGlobalRtMailbox	Creates a mailbox on a remote node indicated by the LOCATION parameter.
CreateGlobalRtMemoryObject	Creates a memory area on a remote node indicated by the LOCATION parameter and returns a handle to a reference object for that area.
CreateGlobalRtMemoryHandle	Creates a handle for a physical memory area on a remote node indicated by the LOCATION parameter, and on the same host as the caller, and returns a handle to a reference object for that handle.

Interrupts

This group provides the system calls required to manage interrupts. Interrupt requests are encoded to indicate their source; the resulting coding is called the interrupt level. This encoded interrupt level is required in a number of the system calls in this group. Macros are provided in the header files for the 15 standard PC interrupt levels, IRQ0_LEVEL to IRQ15_LEVEL.

High-level calls

System call	Description
SetRtInterruptHandler SetRtInterruptHandlerEx	Assigns an interrupt handler to the specified interrupt level, and optionally makes the calling thread the interrupt thread for that level.
ResetRtInterruptHandler	Cancels the assignment of the current interrupt handler to the specified level and disables the level.
EnterRtInterrupt	Allows the interrupt handler to have access to static data belonging to the RT process which owns the handler. This function is called from an interrupt handler.
GetRtInterruptLevel	Returns to the calling thread the highest (numerically lowest) level that an interrupt handler has started servicing but has not yet finished.
SignalRtInterruptThread	Sends an EOI signal to the interrupt hardware then schedules the interrupt thread associated with the specified level. This function is called from an interrupt handler.
SignalEndOfRtInterrupt	Sends an EOI signal to the interrupt hardware. This function is called from an interrupt handler.
WaitForRtInterrupt	Used by an interrupt thread to signal its readiness to service an interrupt. It blocks for the given number of milliseconds.
DisableRtInterrupt	Disables the specified interrupt level. It has no effect on other levels.
EnableRtInterrupt	Enables a specific interrupt level which must have an interrupt handler assigned to it.

Mailboxes

These calls manage various RT kernel mailbox object types.

INtime software includes two kinds of RT mailboxes:

- Object mailboxes manage RTHANDLES.
- Data mailboxes manage short sections of arbitrary data.

NTX calls

System call	Description
ntxCreateRtMailbox	Creates an RT mailbox.
ntxDeleteRtMailbox	Deletes an RT mailbox.
ntxReceiveRtHandle	Receives RTHANDLES from an object mailbox.
ntxSendRtHandle	Sends RTHANDLES to an object mailbox.
ntxSendRtData	Copies arbitrary data to a data mailbox (up to 128 bytes).
ntxReceiveRtData	Copies arbitrary data out of a data mailbox (up to 128 bytes).

High-level calls

System call	Description
CreateRtMailbox	Creates a new mailbox and returns an RTHANDLE for the object. The mailbox type is determined by the flags parameter.
DeleteRtMailbox	Deletes the mailbox specified by the RTHANDLE given.
SendRtHandle	Sends an RT object RTHANDLE to a mailbox which has been created to pass RT objects.
ReceiveRtHandle	Receives an RTHANDLE from an object mailbox.
SendRtData	Copies arbitrary data to a data mailbox (up to 128 bytes).
ReceiveRtData	Copies arbitrary data out of a data mailbox (up to 128 bytes).

Low-level calls

System call	Description
knCreateRtMailbox	Creates a kernel mailbox with the given specifications.
knDeleteRtMailbox	Deletes the kernel mailbox associated with the given kernel handle.
knSendRtData	Sends a data message to the given kernel mailbox.
knSendRtPriorityData	Sends high-priority data to a kernel mailbox, bypassing the queue.
knWaitForRtData	Requests a message from a specific kernel mailbox.

Memory management

The Windows and RT portions of INtime applications can share regions of memory created by the RT portion of INtime applications.

These system calls implement the flat-model memory management interface for RT applications. There are additional calls for the management of page-aligned segments for sharing between applications (both RT and Windows applications). Also included are the calls that an application requires to allocate memory from its own virtual segment.

Physical memory may be allocated and mapped into an application's virtual segment and then an RTHANDLE created for this allocated memory so that it may be shared between applications (both RT and Windows applications)

NTX calls

System call	Description
ntxMapRtSharedMemory ntxMapRtSharedMemoryEx	Obtains a Windows pointer to the section of RT memory defined by the call's SharedRTMemoryHandle parameter.
ntxGetRtSize	Determines the size of the call's RT object parameter
ntxUnmapRtSharedMemory	Removes the mapping of a RT memory section and returns Windows system resources mapped to that memory.
ntxCopyRtData	Copies data directly to/from Windows application space from/to an RT memory object.

High-level calls

System call	Description
AllocateRtMemory	Allocates memory from the current process's memory pool to the current thread's virtual segment.
FreeRtMemory	Frees physical memory associated with the calling thread's virtual segment.
CreateRtMemoryHandle	Creates a handle for an area of memory in the process's virtual segment.
DeleteRtMemoryHandle	Deletes a memory handle created with CreateRtMemoryHandle.
MapRtSharedMemory	Maps a memory area, previously created by another process using CreateRtMemoryHandle, into the current thread's memory space.
MapRtPhysicalMemory	Maps a physical memory area, defined by its absolute address and contiguous length before any translation imposed by the paging system, into the current process' virtual segment.
GetRtPhysicalAddress	Returns the physical address for a valid buffer described by the call parameters.

System call	Description
GetRtSize	Returns the number of bytes in a previously allocated segment.
CreateRtHeap	Creates a heap object by allocating a segment of *cbHeapSize* bytes (plus overhead) and creating the heap structure in this segment.
DeleteRtHeap	Deletes a heap object, given its handle. Any flat-model mappings are deleted.
RequestRtBuffer	Allocates a memory buffer from a heap object and returns a pointer to the caller.
ReleaseRtBuffer	Returns a previously-allocated buffer to its heap.
GetRtHeapInfo	Returns a structure containing information about a heap object.
GetRtBufferSize	Returns the allocated size of a buffer previously allocated from a heap.
CopyRtData	Copies data directly between RT memory objects.

Object directories

The object directory provides a rendezvous mechanism between RT threads or between an RT thread and a Windows thread in an INtime application. An RT or Windows process can catalog objects it wants to share. Other processes can wait for an object to be cataloged.

These system calls manage RT object directories. Objects may be cataloged, looked up, and uncataloged. The default directory is the one for the current process. Handles for the other processes may be obtained using the GetRtThreadHandles call. Catalog names may be up to 12 characters long and are case-sensitive.

NTX calls

System call	Description
ntxLookupNtxHandle	Looks up a name-to-handle association. Given a name, an RT or Windows application can look up the handle associated with it.
ntxImportRtHandle	Obtains an NTXHANDLE corresponding to an RTHANDLE.
ntxCatalogRtHandle	Creates a name-to-handle association in the object directory of the RT process specified in the call.
ntxUncatalogRtHandle	Removes the name-to-handle association in the object directory of the RT process specified in the call.
ntxGetRootRtProcess	Obtains the root RT process handle for the NTX location specified in the call.
ntxGetType	Checks the RT object's type.

High-level calls

System call	Description
CatalogRtHandle	Creates a name-to-handle association in the object directory of the RT process specified in the call.
LookupRtHandle	Searches the given process' object directory for a given name and returns the object handle, if found.
UncatalogRtHandle	Removes an entry from a process' object directory.
InspectRtProcessDirectory	Returns the contents of a process' object directory.
GetRtHandleType GetRtHandleTypeEx	Returns a value indicating the type of an RT object. The handle must be for a valid RT object.

Ports

NTX calls

System call	Description
ntxBindRtPort	Binds an address to a port.
ntxCreateRtPort	Creates a port for access to an RT service module.
ntxDeleteRtPort	Destroys an RT port created by a Windows process.
ntxConnectRtPort	Creates a connection between one local port identified by a handle, and another port identified by an address.
ntxAttachRtPort	Forwards messages from one port to another on the same service.
ntxDetachRtPort	Stops forwarding messages from the specified port.
ntxGetRtPortAttributes	Obtains information about the specified port.
ntxSendRtMessage	Sends a message from a port to a service.
ntxSendRtMessageRSVP	Sends the request part of an RSVP transaction.
ntxCancelRtTransaction	Cancels an RSVP message transmission in progress, or the status reporting phase of an asynchronous send operation.
ntxReceiveRtMessage	Receives a message at a port.
ntxReceiveRtReply	Receives the reply phase of an RSVP transaction.
ntxGetRtServiceAttributes	Allows the caller to receive parameters from the service.
ntxSetRtServiceAttributes	Allows the caller to specify run-time parameters for the service.
ntxRequestRtBuffer	Allocates a memory buffer from the heap object associated with a port connected to a service, and returns a pointer to the Windows mapping of the buffer.
ntxReleaseRtBuffer	Returns memory to the heap object from which it was taken.

Service support

System call	Description
InstallRtServiceDescriptor	Adds a service to the operating system by linking the service descriptor to the service descriptor list.
UninstallRtServiceDescriptor	Removes a service from the operating system by unlinking the service descriptor from the service descriptor list.
GetRtServiceAttributes	Interrogates certain attributes of the interface controlled by the service.
SetRtServiceAttributes	Sets or changes some attributes of the interface controlled by the service.

Port object management

System call	Description
AttachRtHeap	Makes a heap's memory resources available to one or more message port objects.
AttachRtPort	Enables an application to monitor several ports simultaneously.
BindRtPort	Binds an address to a port.
ConnectRtPort	Creates a connection between a local port and a remote port.
CreateRtPort CreateRtPortEx	Creates a message port for access to a given service.
DeleteRtPort	Deletes a port object. Any messages queued at the port are discarded and, if the port is forwarded, forwarding is severed.
DetachRtHeap	Ends the association between a heap object and a message port.
DetachRtPort	Ends message forwarding from the specified message port.
GetRtPortAttributes	Returns a structure giving information about the port object indicated by the supplied handle.

Message transmission

System call	Description
SendRtMessage	Sends a data message from a port to a service.
SendRtMessageRSVP	Sends the request phase of a transaction and allocates a storage for the response part of the transaction.
SendRtReply	Sends a response message to an earlier receive SendRtMessageRSVP message.
CancelRtTransaction	Performs synchronous cancellation of RSVP message transmission.
ReceiveRtMessage	Receives a message at a port.
ReceiveRtReply	Receives a reply message to an earlier RSVP transmission. The port cannot be a sink port.
ReceiveRtFragment	Receives a fragment of an RSVP data message request.

Processes

NTX calls

System call	Description
ntxCreateRtProcess	Creates a process.
ntxRegisterDependency	Creates a dependency relationship between the calling process and the specified sponsor.
ntxUnregisterDependency	Removes the dependency relationship between the calling process and the specified sponsor.
ntxRegisterSponsor	Registers the calling process as a Sponsor with the given name.
ntxUnregisterSponsor	Removes the current sponsor name from the active sponsor state.
ntxNotifyEvent	Blocks until one of the desired notifications has been received.

High-level calls

System call	Description
DeleteRtProcess	Deletes the current process.
ExitRtProcess	Deletes the current process, all of the process' threads, and all objects created by the threads.
RegisterRtDependency	Looks up the name in the sponsor list and creates a dependency relationship to that sponsor process.
UnregisterRtDependency	Removes the dependency relationship from the database between the RT process and the Windows sponsor registered with the given name.
RegisterRtSponsor	Allows the RT process to register as a sponsor under the given name.
UnregisterRtSponsor	Removes the RT process registered as a sponsor from the database.

Regions

High-level calls

System call	Description
CreateRtRegion	Creates a region object.
DeleteRtRegion	Deletes a region object.

System call	Description
AcceptRtControl	Receives ownership of a region object only if it is immediately available.
WaitForRtControl	Gains ownership of a region. This function blocks until the current owner gives up the region.
ReleaseRtControl	Releases this thread's most recently obtained region object.

Scheduler

Low-level calls

System call	Description
knRtSleep	Puts the calling thread to sleep for the specified number of kernel ticks.
knStartRtScheduler	Cancels one scheduling lock imposed by the knStopRtScheduler function.
knStopRtScheduler	Temporarily locks the scheduling mechanism or places an additional lock on the mechanism for the running thread.

Semaphores

Semaphores contain units. These system calls deal with semaphore objects and the units associated with them.

RT semaphores differ from Windows semaphore objects in these respects:

- Choice of FIFO or priority queuing.
- A thread may wait for multiple units from a semaphore.
- Multiple-object waiting is not supported.
- High-level semaphores differ from the low-level semaphores in the following respects:
    - High-level semaphores have parameter validation.
    - Multiple units may be sent to and received from high-level semaphores.
    - High-level semaphore may be created with any initial count other than one or zero (low-level semaphores may be created only with zero or one unit).
    - Low-level handles may not be cataloged.

NTX calls

System call	Description
ntxCreateRtSemaphore	Creates an RT semaphore.
ntxDeleteRtSemaphore	Deletes an RT semaphore.
ntxWaitForRtSemaphore	Waits for and removes units from an RT semaphore.
ntxReleaseRtSemaphore	Adds units to an RT semaphore.

High-level calls

System call	Description
CreateRtSemaphore	Creates a semaphore with the given initial and maximum number of units.
DeleteRtSemaphore	Deletes a semaphore.
WaitForRtSemaphore	Waits for and removes a specified number of units from a semaphore.
ReleaseRtSemaphore	Sends a given number of units to a semaphore.

Low-level calls

System call	Description
knCreateRtSemaphore	Creates 1 of 3 kinds of kernel semaphores with 0 or 1 initial units.
knDeleteRtSemaphore	Deletes a kernel semaphore.
knWaitForRtSemaphore	Waits for and removes a unit from the specified kernel semaphore.
knReleaseRtSemaphore	Sends a single unit to a specified kernel semaphore.

Status

NTX calls

System call	Description
ntxGetLastRtError	Returns the status code of the last failing NTX call made in this Windows thread.
ntxGetRtErrorName	Returns a pointer to a constant string.
ntxLoadRtErrorString	Copies a short sentence (no punctuation) that describes "Status" into the buffer at *`lpBuffer`*.
ntxGetRtStatus	Verifies whether the RT kernel is currently running.

High-level calls

System call	Description
GetLastRtError	Returns the status code of the last failing RT system call made by this RT thread.
SetLastRtError	Sets the calling thread's last error field.
CopyRtSystemInfo	Copies the contents of the RQSYSINFO segment, cataloged in the root process, into the SYSINFO structure.
ReportRtEvent	Collects log data in the same format as the Windows ReportEvent function, and passes it to the INtime Event Log Service for logging in the Windows Event Log.

System data

NTX calls

System call	Description
ntxGetLocationByName	Gets a handle to a specified location.
ntxGetFirstLocation	Returns a handle to the first known location.
ntxGetNextLocation	Gets the handle that follows the one return by the last location call.
ntxGetNameOfLocation	Gets the name NTX uses for a handle.
ntxFindINtimeNode	Invokes the INtime RT Client Browser to allow target INtime node selection.

Threads

High-level calls

System call	Description
CreateRtThread	Creates a thread to execute within the context of the calling process.
DeleteRtThread	Deletes a thread referenced by the given handle.
GetRtThreadAccounting	Returns information about when a thread was created and the amount of time the thread has run.
GetRtThreadHandles	Returns a handle for either the calling thread, the calling thread's process, the parameter object of the calling thread's process, the root process, or the parent process of the calling thread's process, depending on the encoded request.
GetRtThreadInfo	Returns information about a thread, including such items as priority, exception handler, containing process, and execution state.

System call	Description
GetRtThreadPriority	Returns the specified thread's current priority.
GetRtThreadState	Returns information about the state of any thread in the system, including such items as the execution state and the CPU registers for that thread's execution context (if the thread has been suspended due to its causing a hardware fault).
ResumeRtThread	Decreases by one the suspension depth of the specified non-interrupt thread.
RtSleep	Places the current thread in the sleep state until the required number of system ticks have occurred. System ticks are always 10ms apart.
SetRtThreadPriority	Dynamically changes the priority of a non-interrupt thread. The new value must not exceed the containing process' maximum priority.
SetRtProcessMaxPriority	Dynamically change the maximum priority of threads in a process.
SetRtSystemAccountingMode	Toggles accounting tracking. You can return accounting information using GetRtThreadAccounting.
SuspendRtThread	Increases by one the suspension depth of a specified thread.

Time management

INtime software provides low-level time management calls that allow threads to create alarm events and to sleep for a specified amount of time. The kernel also provides an RT clock.

An alarm event is an object which is signaled when a pre-determined time interval has expired. The alarm event mode may be single-shot or repeatable.

The kernel's RT clock is a counter that the kernel uses to keep track of the number of kernel clock ticks that have occurred. When the kernel is initialized, the count is set to 0 (zero). The period of a kernel clock tick is configurable via the INtime Configuration utility and you can read the current configuration using CopyRtSystemInfo.

Low-level calls

System call	Description
knCreateRtAlarmEvent	Creates an alarm event object which is triggered by an alarm.
knWaitForRtAlarmEvent	Waits at an alarm object for the given time interval, or until the alarm triggers.
knResetRtAlarmEvent	Resets a one-shot alarm after it has triggered.

System call	Description
knDeleteRtAlarmEvent	Deletes an alarm event object and releases the memory used to store its state for reuse.
knGetKernelTime	Returns the value of the counter the kernel uses to tally the number of low-level ticks that have occurred.
knSetKernelTime	Sets the value of the counter that the kernel uses to tally the number of low-level ticks that have occurred.

Structures

Structure	Description
CONTROLBUFFER	Tracks the progress of each message during its life in the service.
CPUFRAME	N/A
EVENTINFO	Returns notifications about system state, sponsor processes, dependent processes, and shutdown notifications.
EXCEPTION	N/A
FILETIME	Provides the file system time stamp.
GENADDR	Provides port addresses which fully differentiate ports within a particular service.
HEAPINFO	N/A
HWEXCEPTIONMSG	Specifies the format for data sent to the HW_FAULT_MBX data mailbox.
INTERRUPTINFO	N/A
KNTIME	The kernel stores and reads the time value.
NTXEVENTINFO	Returns notifications about system state, sponsor processes, and dependent processes.
NTXPROCATTRIBS	Specifies memory usage fields the RT Application Loader.
OBJECTDIR	N/A
PCIDEV	Passes parameters to library calls and to return values from the PCI configuration space.
POOLINFO	N/A
PORTINFO	N/A
RECEIVEINFO	Provides information about received messages to the thread or returns information about the operation just completed.
SECURITY_ATTRIBUTES	Contains the security descriptor for an object and specifies whether the handle retrieved by specifying this structure is inheritable.
SERVICEATTRIBUTES	The header for every structure passed to GetRtServiceAttributes and SetRtServiceAttributes.

Structure	Description
SERVICEDESC	Contains all configuration information needed to run the service.
SYSINFO	Contains information about the current RT machine configuration.
THREADACCOUNTING	N/A
THREADINFO_SNAPSHOT	N/A
THREADSTATE_SNAPSHOT	N/A
TRANSACTION	Tracks many operations through the service. For example: a simple send, a send-RSVP or certain receive operations.
urb	The USB Request Block structure, which identifies USB transfer requests.
usbClient	Identifies the USB client to the USB subsystem.
usbConfigDescriptor	The USB-defined configuration descriptor, plus some implementation-specific fields that link to the configuration interfaces.
usbCtrlRequest	Defines a device control request.
usbDeviceDescriptor	The USB-defined device descriptor; there is one descriptor per device.
usbDeviceId	Identifies USB devices and interfaces for hotplugging and enumeration purposes.
usbEndPointDescriptor	The USB-defined endpoint descriptor. The USB subsystem copies all endpoint descriptors from the device.
usbInterface	References the alternate settings for a given interface. This structure is referenced from the usbConfigDescriptor structure.
usbInterfaceDescriptor	The USB-defined interface descriptor. The USB subsystem copies all interface descriptors from the device.

Heaps and memory pools

System call	Description
_CrtCheckMemory	Confirms the integrity of the memory blocks allocated in the debug heap (debug version only).
_CrtDbgReport	Generates a report with a debug message and sends the report to three possible destinations (debug version only).
_CrtDoForAllClientObjects	Calls an application-supplied function for all _CLIENT_BLOCK types in the heap (debug version only).
_CrtDumpMemoryLeaks	Dumps all the memory blocks in the debug heap when a memory leak occurs (debug version only).
_CrtGetAllocHook	Retrieves the current client-defined allocation function for hooking into the C runtime debug memory allocation process (debug version only).

System call	Description
_CrtGetDumpClient	Retrieves the current application-defined function for dumping the _CLIENT_BLOCK type memory blocks (debug version only).
_CrtGetReportHook	Retrieves the client-defined debug reporting function.
_CrtIsMemoryBlock	Verifies that a specified memory block is in the local heap and that it has a valid debug heap block type identifier (debug version only).
_CrtIsValidHeapPointer	Verifies that the heap contains the specified pointer (debug version only).
_CrtMemCheckpoint	Obtains the debug heap's current state and stores it in an application-supplied _CrtMemState structure (debug version only).
_CrtMemDifference	Compares two memory states and returns their differences (debug version only).
_CrtMemDumpAllObjectsSince	Dumps information about objects in the heap from the start of program execution or from a specified heap state (debug version only).
_CrtMemDumpStatistics	Dumps the debug header information for a specified heap state in a user-readable form (debug version only).
_CrtReportBlockType	Returns the block type/subtype associated with a given debug heap block pointer.
_CrtSetAllocHook	Installs a client-defined allocation function by hooking it into the C runtime debug memory allocation process (debug version only).
_CrtSetBreakAlloc	Sets a breakpoint on a specified object allocation order number (debug version only).
_CrtSetDbgFlag	Retrieves or modifies the _crtDbgFlag flag state to control the debug heap manager's allocation behavior (debug version only).
_CrtSetDumpClient	Installs an application-defined function to dump _CLIENT_BLOCK type memory blocks (debug version only).
_CrtSetReportFile	After specifying _CRTDBG_MODE_FILE with _CrtSetReportMode, you can specify the file handle to receive the message text. _CrtDbgReport also use _CrtSetReportFile to specify the text destination (debug version only).
_CrtSetReportHook	Installs a client-defined reporting function by hooking it into the C runtime debug reporting process (debug version only).
_CrtSetReportMode	Specifies the destination(s) for a specific report type generated by _CrtDbgReport and any macros that call _CrtDbgReport or _CrtDbgReportW, such as these macros (debug version only): _ASSERT, _ASSERTE, _RPT, and _RPTF.
_filelength64	Gets the specified file's length.

System call	Description
_findfirst75, _findnext64, _findclose64	Find functions include: • **_findfirst64**: Finds the first file which matches the specified pattern. • **_findnext64**: Finds the next file after an initial call to findfirst. • **_findclose64**: Closes the find handle and releases internal resources used to maintain context.
_stat64	Gets the specified file's status information.
Memory heaps	
heap_buffer_cnt	Returns the current number of buffers creaed by alloc/realloc/calloc.
heap_buffer_size	Returns the size of a buffer returned by alloc/realloc/calloc.
heap_check	Checks the heap for internal consistency.
heap_compact	Returns to the operating system pages of memory freed via free and no longer in use.
heap_dump	Dumps heap statistics and lists to the console or to a file.
heap_get_clk_size	Gets the size of the basic heap allocation block.
heap_get_config	Gets heap configuration.
heap_get_page_size	Gets the size of the system page size value in bytes.
heap_set_blk_size	Sets the size of the basic heap allocation block.
heap_set_config	Sets heap configuration.
heap_stats	Gets heap statistics.
heap_stats_reset	Resets heap statistics.
heap_validate_buffer	Determines whether the buffer address is valid and exists within the heap.
heap_zap	Instructs the heap manager to clear all buffers when freed.
Memory pools	
mpool_alloc	Allocates a new buffer from a memory pool.
mpool_buffer_cnt	Returns the current number of buffers created by mpool_alloc/mpool_realloc.
mpool_buffer_size	Returns the size of a buffer returned by mpool_alloc or mpool_realloc.
mpool_check	Checks the memory pool for internal consistency.
mpool_compact	Returns to the operating system pages of memory freed via mpool_free and no longer in use.
mpool_create	Creates a memory pool and return a handle to it.
mpool_delete	Deletes a memory pool and frees all its resources.
mpool_dump	Dumps memory pool statistics and lists to the console or to a file.
mpool_free	Frees a buffer allocated with mpool_alloc or mpool_realloc.

System call	Description
mpool_get_config	Gets a pool's configuration.
mpool_pool2name	Retrieves the name associated with a pool.
mpool_ptr2pool	Retrieves the pool associated with a buffer.
mpool_realloc	Changes the size of a buffer allocated with mpool_alloc or mpool_realloc.
mpool_reset	Restores a pool to its creation state.
mpool_set_config	Sets pool configuration.
mpool_stats	Gets the memory pool's statistics.
mpool_stats_reset	Resets pool statistics.
mpool_validate_buffer	Determines whether the buffer address is valid and exists within a pool buffer.

High-performance gigabit Ethernet

System call	Description
hpeAllocateReceiveBufferSet	Allocates a receive buffer set which is compatible with the network device.
hpeAttachReceiveBufferSet	Attaches a set of receive buffers to the driver for use by the DMA engine to receive Ethernet frames.
hpeAttachTransmitBufferSet	Attaches a set of transmit buffers to the driver.
hpeClose	Closes an Ethernet interface indicated by the handle parameter.
hpeConfigOptions	Configures options for the Ethernet controller, such as multicast packet reception.
hpeFreeReceiveBufferSet	Frees memory allocated by hpeAllocateReceiveBufferSet.
hpeGetMacAddress	Reports the interface's 6-byte MAC address.
hpeGetMediaStatus	Reports the current status of the media interface of the Ethernet controller.
hpeGetReceiveBuffer	Returns a pointer to the next receive buffer which has a fully-received frame in it.
hpeGetTransmitterState	Returns a value indicating the current state of the transmitter.
hpeOpen	Initializes an Ethernet interface.
hpeOpenWithOptions	Similar to hpeOpen; specifies additional options.
hpeStartTransmitter	Causes the transmitter to transmit any and all frames which are ready to be sent.
hpeWaitForReceiveComplete	Instructs the caller to sleep until the next receive interrupt occurs.
Structures	
hpeWaitForTransmitComplete	Instructs the caller to sleep until the next transmit interrupt occurs.
HPE_CONFIG_OPTIONS	Specifies options; used by hpeConfigOptions.

System call	Description
HPE_OPEN_OPTIONS	Specifies options; used by hpeOpenWithOptions.
HPEBUFFER	Describes an individual buffer to hold frame data.
HPEMEDIASTATUS	Obtains information about the currently-connected media.
HPERXBUFFERSET	Describes a set of receive frame buffers.
HPETXBUFFER	Describes a single transmit buffer.
HPETXBUFFERSET	Describes a transmit buffer set, consisting of multiple transmit buffers.

INscope calls

System call	Description
get_RT_trace_state	Retrieves the current state of the INscope RT Server Component and any current traces.
log_RT_event	Logs a user-defined event in the trace buffer which can be used for reference when viewing the trace.
pause_RT_trace	Pauses a trace, writing no new information to the trace buffer until the trace resumes.
RT_I_am_alive	Resets the watchdog timer.
start_RT_trace	Starts a trace using the current configuration settings.
stop_RT_trace	Stops the trace, locks the buffer, and sends a notification to INscope that a trace complete.

Network stack

The networking stack supports a rich API. For detailed information, select INtime APIs>Other system calls>TCP/IP in the INtime help file.

PCI library calls

System call	Description
PciInitialize	Initializes the PCI library by determining the PCI configuration access method used by the local chipset.
PciReadHeader	Reads the PCI configuration header fields to the supplied PCI_DEV structure.
PciFindDevice	Locates a PCI device given the vendor and device IDs, and an instance number.
PciSetConfigRegister	Writes a value to a given PCI configuration register.
PciGetConfigRegister	Reads a value from a given PCI configuration register.
PciVendorName	Returns a text string corresponding to the vendor ID supplied as a parameter.

System call	Description
PciDeviceName	Returns a text string corresponding to the vendor and device IDs supplied as parameters.
PciClassName	Returns a text string corresponding to the class ID supplied as parameters.
PciEnableDevice	Brings a PCI device out of a ACPI power down state to fully operational condition.

Real-time shared library (RSL) calls

System call	Description
LoadRtLibrary	Dynamically loads a Real-time Shared Library module.
GetRtProcAddress	Searches for a given symbol in a loaded module.
GetRtModuleHandle	Returns the handle of a loaded module, given its name.
FreeRtLibrary	Unloads a Real-time Shared Library module given its handle.
RtTlsAlloc	Allocates a thread-local storage index.
RtTlsSetValue	Stores a value in the calling thread's local storage slot for a specified index.
RtTlsGetValue	Retrieves a value from the calling thread's specified local storage slot.
RtTlsFree	Marks a thread local storage slot free for reuse.

Registry calls

System call	Description
RtRegCloseKey	Releases a handle to a key.
RtRegConnectRegistry	Establishes a connection to a registry handle on another computer.
RtRegCreateKeyEx	Creates a key.
RtRegDeleteKey	Deletes a subkey from the registry.
RtRegDeleteValue	Deletes a value from a registry key.
RtRegEnumKeyEx	Enumerates subkeys of an open registry key.
RtRegEnumValue	Enumerates a value for an open registry key.
RtRegFlushKey	Writes attributes of an open key into the registry.
RtRegLoadKey	Creates a subkey and stores registration information into that subkey.
RtRegOpenKeyEx	Opens a key.
RtRegQueryInfoKey	Retrieves information about a registry key.
RtRegQueryValueEx	Retrieves type and data for a value name associated with an open registry key.

System call	Description
RtRegReplaceKey	Replaces the file backing a key and all its subkeys with another file.
RtRegRestoreKey	Reads registry information in a file and copy it over a key.
RtRegSaveKey	Saves a key and all its subkeys and values to a new file.
RtRegSetValueEx	Sets the data and type of a value under a registry key.
RtRegUnLoadKey	Unloads a key and subkeys from the registry.

RT services and device drivers

RT service calls

System call	Description
DeliverMessage	Delivers a complete transactionless message to a port. This call is typically made from the service thread.
DeliverStatus	Terminates a transmit operation, usually from the service thread.
DeliverTransaction	Delivers a transaction. This call is used after a service thread receives a response message, and the transaction is complete.
DequeueInputTransaction	Dequeues the transaction at the head of the service input queue and returns a pointer to it.
DequeueOutputTransaction	Dequeues the transaction at the head of the service output queue and returns a pointer to it.
EnqueueInputTransaction	Enqueues a transaction on the service input queue. This allows the service to maintain an ordered list of requests for later completion by the service thread.
EnqueueOutPutTransaction	Enqueues a transaction on the service output queue when (for example) the transmitter hardware is busy.
EnterServiceRegion	Enters the region associated with the service. Currently the SendMessage handler is called while in this region. If mutual exclusion is desired between the service thread and the SendMessage handler, the service thread can make this call.
ExitServiceRegion	Exits the service region previously entered with EnterServiceRegion.
GetPortId	Returns the port ID for a given port handle.
GetPortParameter	Retrieves the parameter previously associated with a port by a call to SetPortParameter.
GetTransaction	Upon receiving a response message from the interface, instructs the service thread to tie the message with its transaction structure.
LookupPortHandle	Looks up a port handle given a port ID.

System call	Description
QueryInputTransactionQueue	Returns the transaction at the head of the service input queue.
QueryOutputTransactionQueue	Returns the transaction at the head of the service output queue.
ReleaseControlBuffer	Returns a control buffer to the service pool.
ReleaseTransaction	Returns a transaction structure to the pool.
RequestControlBuffer	Requests a control buffer from the service pool.
RequestTransaction	Requests a TRANSACTION buffer from the service transaction pool.
SetPortParameter	Sets the port parameter for the given port to a value given by the caller.

RT service handlers

Service handlers are subroutines invoked by the kernel to perform service-dependent functions of the system calls.

For example, calling SendRtMessage causes the kernel to invoke the handler supplied by the service to handle the transmission of the message to the interface. Some of these handlers must be supplied by the service while others are optional.

The control and transaction buffer pools are the static resources used for allocating internal data structures. The size of these pools is determined from parameters in the service descriptor at installation time.

System call	Description
CancelTransaction	implement special actions as a result of calling CancelRtTransaction or DeleteRtPort.
CreatePort	Invoked when CreateRtPort is called. A status of E_OK must be returned for the port to be created, else the port object is deleted and the status code is returned to the caller.
DeletePort	Invoked when DeleteRtPort is called. It must return a status of E_OK for the port to be deleted, else the port object is deleted and the status code is returned to the caller.
Finish	Ensures that any service-dependent resources are cleaned up before the service process is killed.
GetAttributes	Gets attributes from the service when an application calls GetRtServiceAttributes.
GetFragment	Invoked when an application calls ReceiveRtFragment.
Initialize	Performs any service-specific initialization functions and returns a suitable status code to the caller.
SendMessage	Implemented by all services which require a transmission function.
Service	Invoked by the service thread when an event occurs.

System call	Description
SetAttributes	Passes parameters from SetRtServiceAttributes.
UpdateReceiveInfo	Invoked just before returning results to either ReceiveRtMessage or ReceiveRtReply, caused by a service handler calling DeliverStatus. The kernel handles the receipt of a message at a port, and then may call this routine in order that the RECEIVEINFO structure may be filled out.
VerifyAddress	Validates the address parameter passed to the call.

Serial Communications (COMM)

System call	Description
ClearCommBreak	Restores character transmission for a specified communications device and places the transmission line in a nonbreak state.
ClearCommError	Retrieves information about a communications error and reports the current status of a communications device.
CloseComm	Closes a communications device handle.
EscapeCommFunction	Directs a specified communications device to perform an extended function.
FlushCommBuffers	Causes all buffered data to be written to a communications device.
GetCommConfig	Retrieves the current configuration of a communications device.
GetCommMask	Retrieves the value of the event mask for a specified communications device.
GetCommModemStatus	Retrieves modem control-register values.
GetCommProperties	Retrieves information about the communications properties for a specified communications device.
GetCommState	Retrieves the current control settings for a specified communications device.
GetCommTimeouts	Retrieves the time-out parameters for all read and write operations on a specified communications device.
OpenComm	Opens a handle to a communications device.
PurgeComm	Discards all characters from the output or input buffer of a specified communications resource.
ReadComm	Reads data from a communications device.
ResetCommEvent	Resets a particular communications device event.
SetCommBreak	Suspends character transmission for a specified communications device and places the transmission line in a break state.
SetCommConfig	Sets the current configuration of a communications device.

System call	Description
SetCommMask	Specifies a set of events to monitor for a communications device.
SetCommState	Configures a communications device according to the specifications in a device-control block.
SetCommTimeouts	Sets the time-out parameters for all read and write operations on a specified communications device.
TransmitCommChar	Transmits a specified character ahead of any pending data in the output buffer of the specified communications device.
WaitCommEvent	Waits for an event to occur for a specified communications device.
WriteComm	Writes data to a communications device.

COMM Drivers

System call	Description
comedgeport.rta	Driver for the Digi International Edgeport line of USB multi-channel serial device.
compc.rta	Driver for the onboard PC COM ports.
comrocket.rta	Driver for the Comtrol family of RocketPort serial multi-channel PCI cards.

COMM Utlities

System call	Description
comlist.rta	Displays the name and status of all COM ports.

COMM Structures

System call	Description
COMMCONFIG	Contains information about the configuration state of a communications device.
COMMPROP	Used by GetCommProperties to return information about a given communications driver.
COMMTIMEOUTS	Used in SetCommTimeouts and GetCommTimeouts to set and query the time-out parameters for a communications device. The parameters determine the behavior of ReadComm and WriteComm operations on the device.
COMSTAT	Contains information about a communications device. This structure is filled by ClearCommError function.
DCB	Defines the control setting for a serial communications device.

TCP/IP system calls

System call	Description
accept	Accepts a connection on a socket.
bind	Assigns a name to an unnamed socket.
bstring	Executes binary string operations.
byteorder	Converts short and long quantities between network byte order and host byte order.
connect	Initiates a connection on a socket.
gethostname - sethostname	Gets and sets the local host name.
getpeername	Returns the socket name of the connected remote socket.
getsockname	Returns the current name for the specified socket.
getsockopt - setsockopt	Returns or sets options associated with a socket.
inet	Manipulates Internet addresses.
listen	Listens for connection requests on a socket.
recv - recvfrom	Receives a message from a socket.
send - sendto	Sends a message from one socket to another.
shutdown	Shuts down all or part of a full-duplex connection.
socket	Creates an endpoint for communication.
select	Checks whether sockets are ready to receive or send, or have out-of-band data pending.
socktout	Sets a timeout for completion of calls on a socket.
gethostent	Sets and returns entries that identify the network host.
getnetent	Returns information about a network entry from the :config:networks database.
getprotoent	Returns an entry from the :config:protocols database file.
getservent	Sets or Returns an entry from the :config:services database file.

USB calls

System call	Description
UsbAllocUrb	Allocates a URB for use by the client, initializes some internal fields and marks it as in-use.
UsbBulkMsg	Creates a bulk transfer and submits it synchronously, returning from the call upon completion or time out.
UsbClearHalt	Clears a halt condition on an endpoint.
UsbConnect	Connects a client to the USB subsystem.
UsbControlMsg	Creates a control transfer and synchronously submits it, returning upon completion or time out.
UsbDisconnect	Disconnects a client from the USB subsystem.

System call	Description
UsbFillBulkUrb	Fills an URB for a bulk transfer.
UsbFillControlUrb	Fills an URB for a control transfer.
UsbFillintUrb	Fills an URB for an interrupt transfer.
UsbFillIsoUrb	Fills an URB for an isochronous transfer.
UsbFreeUrb	Frees the memory of a URB when all users of it are finished.
UsbGetAsciiString	Gets a string descriptor in ASCII format and US/English language ID.
UsbGetConfigDescriptor	Returns a configuration descriptor given a device handle.
UsbGetConfiguration	Gets the current configuration number of a given device.
UsbGetDescriptor	Returns the descriptor of given type and index.
UsbGetDeviceDescriptor	Returns the device descriptor for a given handle.
UsbGetEndpointCount	Returns the number of endpoints for a given interface.
UsbGetEndpointDescriptor	Returns the endpoint descriptor for the given interface handle and endpoint index.
UsbGetInterfaceDescriptor	Returns the interface descriptor for a given handle.
UsbGetLanguageString	Gets a string descriptor in UTF-16LE format.
UsbGetStatus	Gets the device, interface, or endpoint status.
UsbInterruptClose	Closes an open interrupt pipe handle.
UsbInterruptOpen	Opens a handle on an interrupt pipe for synchronous I/O.
UsbInterruptRead	Reads from an interrupt endpoint.
UsbInterruptWrite	Writes to an interrupt endpoint.
UsbKillUrb	Cancels a transfer request for an endpoint.
UsbMatchId	Finds a matching entry in a table of device descriptions.
UsbSetConfiguration	Creates a control transfer and synchronously submits it, returning from the call when it completes or times out.
UsbSetInterface	Sets an alternative setting number of a given interface.
UsbSubmitUrb	Submits a transfer request to the USB subsystem.
UsbUnlinkUrb	Cancels a transfer request for an endpoint.

INtimeDotNet calls

System call	Description
ntxCatalogNtxHandle	Names an object in a process directory.
ntxCreateRtMailbox	Creates an RT mailbox.
ntxCreateRtProcess	Loads an RT executable and runs it in a new process.
ntxCreateRtSemaphore	Creates an RT semaphore.
ntxDeleteRtMailbox	Deletes an RT mailbox.
ntxDeleteRtSemaphore	Deletes an RT semaphore.
ntxGetFirstLocation	Returns a handle to the first known location.

System call	Description
ntxGetLocationByName	Returns a handle to the specified location.
ntxGetNameOfLocation	Returns the name by which the specified location handle is known to NTX.
ntxGetNextLocation	Returns the handle to the location following the one returned by the last call to ntxGetFirstLocation or ntxGetNextLocation in the current thread.
ntxGetRootRtProcess	Obtains the root RT process handle.
ntxGetRtErrorName	Returns a string that contains the name of the status code passed.
ntxGetRtSize	Returns a memory region's size.
ntxGetRtStatus	Verifies whether the RT kernel is successfully initialized.
ntxGetType	Returns the type of an NTX handle.
ntxImportRtHandle	Obtains an NTXHANDLE that corresponds to an RTHANDLE.
ntxLoadRtErrorString	Returns a short sentence (no punctuation) that describes `Status`.
ntxLookupNtxHandle	Searches the given process's object directory for the given name and return the object handle, if found.
ntxNotifyEvent	Blocks until one of the desired notifications is received.
ntxReadRtXxx	Reads from a Byte array or an INtime shared memory object.
ntxReceiveRtDataXxx	Waits for and then copies data out of an RT data mailbox.
ntxReceiveRtHandle	Receives handles from an object mailbox.
ntxRegisterDependency	Creates a dependency relationship between the calling process and the specified sponsor.
ntxRegisterSponsor	Registers the calling process as a Sponsor with the given name.
ntxReleaseRtSemaphore	Releases units to an RT semaphore.
ntxSendRtDataXxx	Copies data to an RT data mailbox.
ntxSendRtHandle	Sends an object handle to an object mailbox.
ntxUncatalogNtxHandle	Removes an entry from a process' object directory.
ntxUnregisterDependency	Removes the dependency relationship between the calling process and the specified sponsor.
ntxUnregisterSponsor	Removes the current sponsor name from the active sponsor state. No notifications are made to dependents and the name remains in use until the sponsor is removed from all relationships.
ntxWaitForRtSemaphore	Requests a specified number of units to be received from the RTsemaphore.
ntxWriteRtXxx	Writes to a Byte array or an INtime shared memory object.

INtimeDotNet structures

System call	Description
NTXEVENTINFO	Passes into ntxCreateRtProcess to overrule process creation defaults.
NTXPROCATTRIBS	Returns notifications about system state, sponsor processes, and dependent processes.

Input/Output Calls

System call	Description
inbyte, inhword, inword	Inputs data from an I/O port.
outbyte, outhword, outword	Outputs data to an I/O port.

B The iwin32 subsystem

This appendix describes the iwin32 subsystem, which provides a Win32 API for the INtime kernel. It is provided as a parallel API to the INtime API, and is intended to make porting of existing Win32 applications easier. A subset of the Win32 functions is implemented, and some extensions are defined to handle INtime features such as interrupt handling and shared memory. The functionality of the subset is broadly similar to the Windows CE version of the Win32 API, since Windows CE and INtime have similar goals. Some groups of functions have been omitted where INtime does not require the functionality, such as with the GUI functions.

The elements covered by the iwin32 API include the following:

- Processes and threads
- Mutexes, critical sections, semaphores and events
- I/O handling
- Registry handling
- Miscellaneous

In addition a number of real-time extension (RTX) functions are provided where more real-time functionality is required; this includes functions for:

- Interrupt handling
- Shared memory
- Timers

This appendix also describes the iwin32x API, which gives access to real-time iwin32 objects from a Windows application, much in the same way that the NTX API gives access to INtime objects from a Windows application.

Handles

Each object is identified by a handle. In INtime an object is uniquely identified by a single handle value (16 bits for INtime, 32 bits for NTX). This handle can be used in any INtime process and in Windows processes (using NTX). When the object is deleted with a type-specific deletion function such as DeleteRtSemaphore, the handle becomes invalid.

Iwin32 has a different handle system: the Create and Open functions return a handle and different callers may receive different handles for the same object. A handle is stored in 32 bits; an iwin32 handle can be distinguished from an INtime handle

because its value is 0x10000 or greater. Every iwin32 object includes a handle count. When the last handle for an object is closed, the object is implicitly deleted.

In Windows, a handle is normally specific to a process and the same handle in different Windows processes may refer to different objects. Iwin32 implements a slightly different method, where all handle values are unique. This allows a handle to be shared between processes, which would be against the Win32 rules.

There is a limit on the number of objects that can exist at any time in the system, because INtime uses a table to define each object; the size of this table (GDT) is configurable up to a maximum of about 8000 entries. For information about GDT configuration, see *Running the INtime Configuration Utility* on page 68.

Each iwin32 object requires one, two (thread, timer, interrupt), or three (process) INtime objects. Additional handles for a given iwin32 object do not require additional INtime objects. Iwin32 uses a fixed size table for all handles, the size of which is configurable.

iwin32 calls	iwin32x calls
CloseHandle or RtCloseHandle	RtCloseHandle
	RtImportHandle
	RtSetNode

Named objects

Event, mutex, semaphore and shared memory objects have a Create and an Open function. CreateXxx checks if the named object of that type already exists and if so, returns an error and the handle of the found object. If the name exists but belongs to another object type, the function fails. If the name does not occur yet, the object is created and the name remembered. If no name is supplied, the name check does not take place. OpenXxx only does the name check and if that fails, the whole operation fails.

All object types share one name space, which is not process specific but has system scope. Iwin32 allows names up to 128 characters.

There are no specific functions for named objects. For details on named objects, see the functions listed in *Events* (page 140), *Mutexes* (page 139), *Semaphores* (page 140), and *Shared memory* (page 141).

Processes

A process is a container for objects and resources; it includes a virtual address space that is only accessible to the threads in the process. When a process is created, a primary thread is always created inside it (this is the function named main).

A process can refer to itself by a so-called pseudo handle, which is not a fixed value, but must be obtained by the GetCurrentProcess function. A pseudo handle can only be used by the process itself, and it cannot be closed (it is implicitly closed when the process terminates).

A process can be explicitly terminated with the TerminateProcess or ExitProcess functions; implicit termination obeys these rules:

- When the primary thread returns, the process is terminated.
- When the primary thread calls ExitThread explicitly, the process is not terminated.
- When any thread terminates and it was the last thread in the process, the process is terminated.

Terminating a process does not necessarily delete the process! It only closes the pseudo handle for the process and only if that is the last handle, the process is deleted. When a process is terminated, all handles created by its threads are closed; again, this need not imply that all objects are deleted.

Waiting for a process to be signaled means waiting until the process has terminated.

Process functions include:

iwin32 calls	iwin32x calls
ExitProcess or RtExitProcess	–
GetCurrentProcess	–
GetCurrentProcessId	–
GetExitCodeProcess or RtGetExitCodeProcess	RtGetExitCodeProcess
OpenProcess or RtOpenProcess	RtOpenProcess
TerminateProcess or RtTerminateProcess	RtTerminateProcess
CreateProcess or CreateRtProcess	RtCreateProcess
WaitForMultipleObjects or RtWaitForMultipleObjects	RtWaitForMultipleObjects
WaitForSingleObject or RtWaitForSingleObject	RtWaitForSingleObjects

Threads

A thread is the active element type in the system. Each thread has a priority, a state and a stack. The priority indicates the importance of the thread when it is in the ready state.

A thread is in one of these states:

- **Ready**: The thread wants to execute; out of the set of ready threads (called the ready list) the thread with the best priority becomes the running thread.

- **Asleep**: The thread waits for an object or one of a set of objects to be signaled, or for a specific timeout, or both. While in this state, the thread will never be running.
- **Suspended**: The thread is waiting for a resume operation. More than one suspend can be done, and each such suspend must be undone by a resume. While in this state, the thread will never be running.
- **Asleep suspended**: While in the asleep state, the thread was suspended. Both the suspend state and the asleep state must be undone before the thread becomes ready again.

A thread has a stack for calling functions and storing local variables and parameters. The stack must be big enough to contain all necessary data; when it overflows, it is not extended but the hardware exception EH_STACK_FAULT occurs.

A thread can refer to itself by a so-called pseudo handle, which is not a fixed value, but must be obtained by the GetCurrentThread function. A pseudo handle can only be used within the owning process, and it cannot be closed (it is implicitly closed when the thread terminates).

Waiting for a thread to be signaled means waiting until the thread has terminated.

Thread handling functions include:

iwin32 calls	iwin32x calls
CreateThread or RtCreate Thread	–
ExitThread or RtExitThread	–
GetCurrentThread	–
GetCurrentThreadId	–
GetExitCodeThread	–
GetLastError or RtGetLastError	–
GetThreadPriority or RtGetThreadPriority	–
RtGetThreadTimeQuantum	–
OpenThread	–
ResumeThread or RtResumeThread	–
SetLastError or RtSetLastError	–
SetThreadPriority or RtSetThreadPriority	–
RtSetThreadTimeQuantum	–
Sleep or RtSleep	–
RtSleepFt or RtSleepFt	RtSleepFt
SuspendThread or RtSuspendThread	–
TerminateThread or RtTerminateThread	–
WaitForMultipleObjects or RtWaitForMultipleObjects	RtWaitForMultipleObjects
WaitForSingleObject or RtWaitForSingleObject	RtWaitForSingleObjects

Mutexes

A mutex is an object for getting exclusive access to a resource used by more than one thread, possibly in different processes.

When a thread attempts to get ownership of a mutex and that mutex is free, ownership is given to that thread; until the thread releases ownership, no other thread can own the same mutex. A thread can own the same mutex more than once, in which case it must release the mutex the same number of times. When a thread with INtime priority `Pw` wishes to own a mutex and that mutex is already owned by another thread with priority `Po`, then if `Pw` is better than `Po`, the priority of the owning thread is changed to `Pw` until it releases all mutexes it owns. This avoids the infamous priority inversion, as described in *Priority inversions* on page 35.

Termination of a thread that owns one or more mutexes causes all threads waiting for such mutexes to be woken up with a WAIT_ABANDONED exception. Deleting a mutex causes all threads waiting for that mutex to be woken up with an ERROR_INVALID_HANDLE error code.

Mutex manipulation functions are:

iwin32 calls	iwin32x calls
CreateMutex or RtCreateMutex	RtCreateMutex
OpenMutex or RtOpenMutex	RtOpenMutex
ReleaseMutex or RtReleaseMutex	RtReleaseMutex
WaitForMultipleObjects or RtWaitForMultipleObjects	RtWaitForMultipleObjects
WaitForSingleObject or RtWaitForSingleObject	RtWaitForSingleObjects

Critical section

A critical section is a mutex that has no name; it can therefore only be used in the process that creates it. A critical section is identified by a CRITICAL_SECTION structure, which in turn contains the handle of the mutex. For more details see mutexes.

Functions for critical sections include:

iwin32 calls	iwin32x calls
DeleteCriticalSection	Part of Win32
EnterCriticalSection	Part of Win32
InitializeCriticalSection	Part of Win32
LeaveCriticalSection	Part of Win32
TryEnterCriticalSection	Part of Win32

Semaphores

A semaphore is a counter that takes positive integer values called units. Threads release units to and wait for units from the semaphore. A semaphore can synchronize a thread's actions with other threads and can also be used to provide mutual exclusion for data or a resource (although a mutex may be better in that case).

A thread can release one or more units to a semaphore. Waiting can be done for a single unit only. A semaphore does not protect against priority inversion (described in *Priority inversions* on page 35). Deleting a semaphore causes all threads waiting for that semaphore to be woken up with an ERROR_INVALID_HANDLE error code.

Semaphore functions include:

iwin32 calls	iwin32x calls
CreateSemaphore or RtCreateSemaphore	RtCreateSemaphore
OpenSemaphore or RtOpenSemaphore	RtOpenSemaphore
ReleaseSemaphore or RtReleaseSemaphore	RtReleaseSemaphore
WaitForMultipleObjects or RtWaitForMultipleObjects	RtWaitForMultipleObjects
WaitForSingleObject or RtWaitForSingleObject	RtWaitForSingleObjects

Events

An event is a flag that can be set (signaled) or reset; it can be reset manually (once set, it remains set until explicitly reset by a ResetEvent call, independent of how many threads are woken up) or automatically (after waking up one thread, the event is reset).

Deleting an event causes all threads waiting for that event to be woken up with an ERROR_INVALID_HANDLE error code.

Event functions include:

iwin32 calls	iwin32x calls
CreateEvent or RtCreateEvent	RtCreateEvent
OpenEvent or RtOpenEvent	RtOpenEvent
PulseEvent or RtPulseEvent	RtPulseEvent
ResetEvent or RtResetEvent	RtResetEvent
SetEvent or RtSetEvent	RtSetEvent
WaitForMultipleObjects or RtWaitForMultipleObjects	RtWaitForMultipleObjects
WaitForSingleObject or RtWaitForSingleObject	RtWaitForSingleObjects

Shared memory

Mutexes and semaphores allow threads to synchronize, but what if you want to exchange data? You can use INtime objects such as mailboxes, but in iwin32 you also find shared memory. Shared memory is memory that has been allocated by one process and that can be accessed by other processes as well. To access shared memory created by another process you need to know its name.

Since every process has its own virtual address space, a shared memory object must be mapped into a process' address space. Different processes may use different local addresses to access the same shared memory! The shared memory is only deleted when all its handles are closed.

Space for shared memory objects comes from an iwin32 virtual memory pool; the maximum size of that pool is configurable. For information about memory configuration, see *Running the INtime Configuration Utility* on page 68.

It is up to the communicating threads to agree on a method of queuing data in the shared memory as necessary.

Shared memory functions include:

iwin32 calls	iwin32x calls
RtCreateSharedMemory	RtCreateSharedMemory
RtOpenSharedMemory	RtOpenSharedMemory
RtGetPhysicalAddress	RtGetPhysicalAddress
RtMapMemory	RtMapMemory
RtUnmapMemory	–
RtUnmapSharedMemory	RtUnmapSharedMemory

Timers

Any thread can be made to wait for a given time by using Sleep or RtSleepFt. A timer is simply a thread that gets woken up when its time passes. Creating a timer means that a thread is created that calls a user-provided function after a given time. This thread is a special one: it can not be suspended or resumed and its priority can not be changed. It should not call ExitThread and can not be terminated by TerminateThread.

Timer functions include:

iwin32 calls	iwin32x calls
RtCancelTimer	–
RtCreateTimer	–
RtDeleteTimer	–
RtGetClockResolution	RtGetClockResolution
RtGetClockTime	RtGetClockTime

iwin32 calls	iwin32x calls
RtGetClockTimerPeriod	RtGetClockTimerPeriod
RtGetTimer	–
QueryPerformanceCounter	–
QueryPerformanceFrequency	–
RtSetClockTime	–
RtSetTimer	–
RtSetTimerRelative	–

I/O handling

In iwin32 a few general file handling functions are present. For many functions, the C-library offers alternatives. Device dependent functions (as provided by DeviceIoControl in Win32) can either be programmed using port I/O, or can be delegated to INtime device drivers.

In contrast to Win32, port I/O (accessing hardware ports directly) is allowed in all INtime threads.

I/O functions in iwin32 include::

iwin32 calls	iwin32x calls
CreateFile	Part of Win32
DeleteFile	Part of Win32
RtDisablePortIo	Part of Win32
RtEnablePortIo	–
RtGetBusDataByOffset	RtGetBusDataByOffset
ReadFile	Part of Win32
RtReadPort...	–
RemoveDirectory	Part of Win32
RtSetBusDataByOffset	RtSetBusDataByOffset
RtTranslateBussAddress	RtTranslateBussAddress
SetFilePointer	Part of Win32
WriteFile	Part of Win32
RtWritePort...	–

Interrupt handling

Win32 does not provide interrupt handling functions, as this always takes place in the Windows kernel environment. Since interrupts are critical in INtime, we have extended iwin32 with interrupt handling. There are two choices for handling an interrupt:

Using RtAttachInterruptVector: A thread is created that is woken up when an interrupt occurs. The thread may use all INtime functions, which makes this a simple- to-understand approach. There is a penalty in processing time, as each interrupt requires two thread switches for switching to and from the interrupt thread.

Using RtAttachInterruptVectorEx: As with the previous function, a thread is created. But in addition a function may be specified that gets called from the hardware interrupt handler, which then determines the need to wake up the thread. In this way many thread switches can be avoided, such as in the case of a terminal: the interrupt function can cause thread wake up for a carriage return character and do internal buffering (and maybe editing) for all other characters. Such an interrupt function can only use the I/O functions RtReadPortXxx and RtWritePortXxx.

When an interrupt comes from a PCI source, the actual interrupt line can be determined using RtGetBusDataByOffset. Access to I/O ports on the device for determining interrupt details is provided by the RtReadPortXxx and RtWritePortXxx functions.

The thread created for interrupt handling is a special one: it can not be suspended or resumed and its priority can not be changed. It should not call ExitThread and cannot be terminated by TerminateThread.

Interrupt handling functions include:

iwin32 calls	iwin32x calls
RtAttachInterruptVector	–
RtAttachInterruptVectorEx	–
RtDisableInterrupts	–
RtEnableInterrupts	–
RtReleaseInterruptVector	–

Registry handling

This lists common operations on registry keys and the registry system calls that do the operations.

To...	iwin32 call	iwin32x call
Create a key	RegCreateKeyEx	Part of Win32
Create a subkey and store registration information into that subkey	RegLoadKey	Part of Win32
Delete a subkey from the registry	RegDeleteKey	Part of Win32
Delete a value from a registry key	RegDeleteValue	Part of Win32
Enumerate subkeys of an open registry key	RegEnumKeyEx	Part of Win32
Enumerate a value for an open registry key	RegEnumValue	Part of Win32

To...	iwin32 call	iwin32x call
Establish a connection to a registry handle on another computer	RegConnectRegistry	Part of Win32
Open a key	RegOpenKeyEx	Part of Win32
Read registry information in a file and copy it over a key	RegRestoreKey	Part of Win32
Release a handle to a key	RegCloseKey	Part of Win32
Replace the file backing a key and all its subkeys with another file	RegReplaceKey	Part of Win32
Retrieve information about a registry key	RegQueryInfoKey	Part of Win32
Retrieve type and data for a value name associated with an open registry key	RegQueryValueEx	Part of Win32
Save a key and all its subkeys and values to a new file	RegSaveKey	Part of Win32
Set the data and type of a value under a registry key	RegSetValueEx	Part of Win32
Unload a key and its subkeys from the registry	RegUnLoadKey	Part of Win32
Write attributes of an open key into the registry	RegFlushKey	Part of Win32

Miscellaneous

In iwin32 (these all have a counterpart in Win32):

FreeLibrary
GetModuleHandle
GetProcAddress
LoadLibrary

Miscellaneous functions include:

iwin32 calls	iwin32x calls
FreeLibrary	Part of Win32
GetModuleHandle	Part of Win32
GetProcAddress	Part of Win32
LoadLibrary	Part of Win32

Provided for easy porting of existing code:

To...	iwin32 calls
Allocate a memory block of the specified size. (Provides the same features as malloc; provided only for compatibility.)	HeapAlloc
Deallocate a memory block. (Provides the same features as free; provided only for compatibility.)	HeapFree
Change the size of a previously allocated memory block or allocate a new one. (Provides the same features as realloc; provided only for compatibility.)	HeapRealloc
Obtain the size, in bytes, of the given memory unit.	HeapSize
Collects log data in the same format as the Windows ReportEvent function, and passes it to the Windows machine for logging.	ReportEvent
Allocates contiguous memory from the current process's memory pool. Provided only for compatibility.	RtAllocateContiguousMemory
Allocates locked memory from the current process' memory pool.	RtAllocateLockedMemory
Convert to an integer value. (Provides the same features as atoi; provided only for compatibility.)	RtAtoi
Register a shutdown notification handler.	RtAttachShutdownHandler
Provided only for compatibility.	RtCommitLockHeap
Provided only for compatibility.	RtCommitLockProcessHeap
Provided only for compatibility.	RtCommitLockStach
Free memory allocated with RtAllocateContiguousMemory or RtAllocateLockedMemory.	RtFreeContiguousMemory
Free memory allocated with RtAllocateContiguousMemory or RtAllocateLockedMemory.	RtFreeLockedMemory
Indicates which environment the process runs in—the INtime real-time environment or a simulated one.	RtIsInRtss
Provided only for compatibility.	RtLockKernel
Provided only for compatibility.	RtLockProcess
Print formatted data to stdout. (Provides the same features as printf; provided only for compatibility.)	RtPrintf
Destroy the shutdown handler object created by RtAttachShutdownHandler.	RtReleaseShutdownHandler

C INtime directory structure

This appendix describes the INtime directory structure.

Note
These directory paths assume INtime is installed in default locations.

Table 10-1. INtime program directory

Directory	Description	File Types
%PROGRAMFILES%\INtime\bin	Contains executables and libraries that support INtime software development an its tools.	Dynamic link libraries (dll), executables (exe), help files for INtime tools (chm), real-time applications for INtime tools and services (rta), real-time shared libraries for INtime tools and services (rsl).
%PROGRAMFILES%\INtime\help	Contains the INtime overview guide, quick-start guide, main INtime help file, and other INtime documentation	Pdf documents, help files (chm), Wordpad documents (rtf).
%PROGRAMFILES%\INtime\help\ecpp	Contains the documentation for the C++ available for use in INtime real-time code	Web pages (htm).
%PROGRAMFILES%\INtime\msdev	Contains the files needed for INtime to work with Visual Studio 6	Custom application wizard (awx), help files (chm).
%PROGRAMFILES%\INtime\Network7	Contains files for INtime networking.	Real-time applications (rta).
%PROGRAMFILES%\INtime\Network	Contains files for INtime legacy networking.	Real-time applications (rta).
%PROGRAMFILES%\INtime\nt\include	Contains include files for windows processes that communicate via NTX to INtime	Include files (h).
%PROGRAMFILES%\INtime\nt\lib	Contains library files fro windows processes that communicate via NTX to INtime	Library files (lib).
%PROGRAMFILES%\INtime\ remote\common	Contains files needed to create remote nodes (NOTE: these files are only available if you have purchased a development kit with remote node support).	Binary images (bin), real-time applications (rta), DOS executable (exe)

Table 10-1. INtime program directory

Directory	Description	File Types
%PROGRAMFILES%\INtime\rt\include %PROGRAMFILES%\INtime\rt\include*	Include files for real-time applications.	Include files (h).
%PROGRAMFILES%\INtime\rt\lib	Library files for real-time applications.	Library files (lib).
%PROGRAMFILES%\INtime\system32\	File to allow events to be logged.	Dynamic link library (dll).
%PROGRAMFILES%\INtime\ system32\drivers	Driver files for the virtual Ethernet device	Security catalog file (cat), driver (sys), driver information file (inf).
%PROGRAMFILES%\INtime\Tools	Real time application tools useful for system evaluation and debugging.	Real-time applications (rta).
%PROGRAMFILES%\INtime\vstudio		
%PROGRAMFILES%\INtime\vstudio*	Files used by Visual Studio 2003 for creating and debugging real-time applications.	Dynamic link libraries (dl), Windows executables (exe), Web pages (htm), Java script files (js), etc.
%PROGRAMFILES%\INtime\vstudio80		
%PROGRAMFILES%\INtime\vstudio80*	Files used by Visual Studio 2005 for creating and debugging real-time applications.	Dynamic link libraries (dl), Windows executables (exe), Web pages (htm), Java script files (js), etc.
%USERPROFILE%\My Documents\ INtime\Projects %USERPROFILE%\Documents\ INtime\Projects (Vista)	Sample applications (see sample applications chapter)	Visual Studio Solutions (sln), Developer studio files (dsw), Project files (vcproj, csproj, jsproj, dsp), source files (h, c, cpp, js, cs).
%USERPROFILE%\My Documents\ INtime\remote %USERPROFILE%\Documents\ INtime\remote (Vista)	Files for remote nodes you have created. (NOTE: these files are only available if you have purchased a development kit with remote node support).	Binary images (bin), real-time applications (rta), DOS executable (exe), DOS batch file (bat), configuration files (ini)
%ALLUSERSPROFILE%\ Application Data\TenAsys\INtime %ALLUSERSPROFILE%\TenAsys\ INtime (Vista)	Global files for INtime configuration	License file (lservrc), data file (dat)
%ALLUSERSPROFILE%\TenAsys\Node Name\etc INtime (Vista)	Global files for specific network configuration files	Network startup, configuration, and data files
%ALLUSERSPROFILE%\Application Data\ TenAsys\INtime\Drivers %ALLUSERSPROFILE%\ TenAsys\INtime\Drivers (Vista)	Driver files for passing devices to INtime.	Setup information files (inf), Drivers (sys)

D INtime software components

This appendix describes product components. Descriptions are based on the default installation path:

`C:\Program Files\INtime\...`

Configuration option	Page
Blue.exe (Windows crash program)	152
Clk1Jitr.rta	152
EventMsg.dll	152
INconfCpl.cpl	152
INtime.chm	153
INscope.exe	154
INtex.exe	155
INtime local kernel (INtime.bin)	155
INtime Performance Monitor (INtmPerf.* files)	155
INtime RT Client Browser	156
Jitter.exe	157
LdRta.exe (INtime RT Application Loader)	157
LoadRtk.exe (INtime Kernel Loader)	158
MFC*.dll files	159
network7 utility files	159
NTX header files	159
NTX import libraries	159
NTX DLLs	160
NtxRemote2.exe (INtime Remote Connection Manager)	160
OvwGuide.pdf	160
Project files	160
RT header files	162
RT interface libraries	162
RT Stack Services	162
RtClkSrv.exe (INtime Clock Synchronization Service)	163
RtDrvrW5.awx (RT Device Driver wizard)	163
RtELServ.exe (INtime Event Log Service)	164
RtIf.sys (RT Interface Driver)	164
RtIOCons.exe (INtime I/O console)	165
RtIOSrv.exe (INtime I/O Service)	165
RtNdSrv.exe (INtime Node Detection Service)	166
RtProcW5.awx (RT Process wizard)	166
RtProcAddinW5.awx (RT Process Add-in wizard)	166
RtRegSrv.exe (INtime Registry Service)	167
RtRslWiz.awx (RT Shared Library wizard)	167

Blue.exe (Windows crash program)

A Windows program that causes the Windows system to have a 'blue screen crash'. Use this program to validate the operation of the INtime software after Windows experiences a "blue screen crash".

Item	Description
Pathname	C:\Program Files\INtime\bin\blue.exe
Invocation	Invoke from the command prompt as follows: `blue -really`

Clk1Jitr.rta

An RT application started by Jitter.exe. This application measures the minimum, maximum, and average times between low-level ticks via an Alarm Event Handler. For more information, see Graphical Jitter.

Item	Description
Pathname	C:\Program Files\INtime\projects\jittercs\clk1jitr.rta
Invocation	Jitter.exe loads this RT application as it starts.

EventMsg.dll

A resource DLL that associates an event ID with a message. You can add your own messages and event IDs to this DLL by using Microsoft Visual Studio on the project C:\Program Files\INtime\projects\eventmsg.

Item	Description
Pathname	C:\My Documents\INtime\Projects\eventmsg
Invocation	The INtime Event Log service loads this DLL at runtime.

INconfCpl.cpl

A Windows program that configures INtime software.

Item	Description
Pathname	C:\Program Files\INtime\bin\INconfCpl.cpl
Invocation	Double-click the icon associated with the file in the INtime program folder (Start>Programs>INtime>INtime Configuration).

INtime.chm

INtime software contains the following Help files:

- Main Help files
- Utility Help files
- C++ Help files

Main Help files

A Windows Help file that describes INtime software.

Item	Description
Pathname	C:\Programs\INtime\help\INtime.chm
Invocation	Double-click the icon associated with the help file in the INtime program folder (Start>Programs>INtime>Documentation>INtime Help).

Utility Help files

Help files that describe INtime software's utilities. Utilities include:

Utility	Help files
INtime Configuration	C:\Program Files\INtime\bin\INConfig.chm
RT Process wizard	C:\Program Files\INtime\bin\RtProcW5.chm
RT Process Add-in wizard	C:\Program Files\INtime\bin\RtProcAddin5.chm
RT Device Driver wizard	C:\Program Files\INtime\bin\RtDrvrW5.chm
RT Application Loader	C:\Program Files\INtime\bin\LdRta.chm

C++ Help files

HTML and GIF files that describe C++ calls and syntax. Files include:

Pathname	Files			
C:\Program Files\ INtime\help\ecpp\	_index.html	cstdlib.html	iosfwd.html	setjmp.html
	assert.html	cstring.html	iostrea2.html	signal.html
	cassert.html	ctime.html	iostream.html	sstream.html
	cctype.html	ctype.gif	istream.html	stdarg.html
	cerrno.html	ctype.html	lib_cpp.html	stddef.html
	cfloat.html	errno.html	lib_file.html	stdexcep.html
	charset.html	escape.gif	lib_over.html	stdio.html
	climits.html	exceptio.html	lib_prin.html	stdlib.html
	clocale.html	express.html	lib_scan.html	stream.gif
	cmath.html	float.html	limits.html	streambu.html
	complex.html	format.gif	locale.html	string.html
	crit_pb.html	fstream.html	math.html	string2.html
	crit_pjp.html	fstream2.html	new.html	strstrea.html
	csetjmp.html	function.html	new2.html	strtod.gif
	csignal.html	index.html	ostream.html	strtol.gif
	cstdarg.html	iomanip.html	preproc.html	time.gif
	cstddef.html	iomanip2.html	print.gif	time html
	cstdio.html	ios.html	scan.gif	

Item	Description
Pathname	C:\Program Files\INtime\help\ecpp_index.html
Invocation	Do one of these: • Access INtime software Help, then select Using INtime software>Other system calls>C++ calls. • Double-click the _index.html file in the C++ help folder.

INscope.exe

A Windows program that uses NTX calls to communicate with its self-loaded RT counterpart to trace execution of INtime applications.

Item	Description
Pathname	C:\Program Files\INtime\bin\inscope.exe
Invocation	Double-click the icon associated with the file in the INtime Program folder (Start>Programs>INtime>INtime Real-time Performance Analyzer).

INtex.exe

A Windows application which allows you to browse the objects in an INtime kernel.

Item	Description
Pathname	C:\Program Files\INtime\bin\intex.exe
Invocation	Double-click the icon associated with the file in the INtime Program folder (Start>Programs>INtime>INtime Explorer).

INtime local kernel (INtime.bin)

The RT kernel binary image , loaded by LoadRtk.exe (INtime Kernel Loader).

Item	Description
Pathname	C:\Program Files\INtime\intime.bin
Invocation	Launched by LoadRtk.exe (INtime Kernel Loader).

INtime remote kernel (Remote.bin)

The RT kernel binary image for use with RT nodes.

Item	Description
Pathname	C:\Program Files\INtime\Remote\common\remote.bin
Invocation	Booted on remote node.

INtime Visual Studio project type packages

A collection of DLLs and Wizards which implements the INtime project type in Visual Studio.

Item	Description
Pathname	C:\Program Files\INtime\vstudio*\...

INtime Performance Monitor (INtmPerf.* files)

The INtime Performance Monitor reports INtime Kernel CPU usage to the Windows Performance Monitor.

INtmPerf.ini is a setup file required as a part of the INtime installation process. It is used by the Windows LOADCTR utility to add the proper registry keys and settings.

Item	Description
Pathname	C:\Program Files\INtime\system\intimperf.dll C:\Program Files\INtime\system\intimperf.ini

INtime RT Client Browser

An ActiveX control that you can add to your INtime applications. For information about adding this browser to INtime applications, see *Adding the INtime RT Client Browser to your INtime application* on page 80.

Item	Description
Pathname	C:\Program Files\INtime\system32\inbrow.ocx

iWin32 header files

Pathname	Files
C:\Program Files\INtime\rt\include\...	iWin32 header files.

iWin32 interface library

Pathname	Files
C:\Program Files\INtime\rt\lib\iwin32.lib	iWin32 interface library

iWin32x header files

Pathname	Files
C:\Program Files\INtime\nt\lib\iwin32x.h	iWin32x header file

iWin32x interface library

Pathname	Files
C:\Program Files\INtime\rt\lib\iwin32x.lib	iWin32x import library

Jitter.exe

A Windows program that automatically starts Clk1Jitr.rta, then processes output and displays a histogram. This application measures the minimum, maximum, and average times between low-level ticks via an Alarm Event Handler. For more information, see Graphical Jitter.

Item	Description
Pathname	C:\Program Files\INtime\projects\jitternt\jitter.exe
Invocation	Select Start>Programs>INtime>INtime Graphical Jitter.

LdRta.exe (INtime RT Application Loader)

A Windows program that loads and starts the RT portion of INtime applications. The loader has two parts: the program that executes on Windows, and an RT “helper” process that performs the RT portion of a load operation. The “helper” process is part of the RT kernel.

The Windows-resident portion of the RT Application Loader is a 32-bit Windows program that uses NTX library calls to load and start INtime applications under the RT kernel.

The RT Application Loader:

- Supports loading of both 32-bit Microsoft PE code (the output of Visual Studio) and 32-bit OMF386 code.
- Supports both command line and dialog-based operation. Supports specification of the file to load, optional debug arguments, and optional program arguments.
- Recognizes the file extension ".RTA" (for RT application).

The INtime Installation processes set up a file association so that an INtime application loads automatically when a user double-clicks the file name in a supporting Windows application (such as Windows Explorer). If a default node has not been established, such an invocation (double-click of the filename) displays ldrta.exe's user interface so you can establish a default node.

Item	Description
Pathname	C:\Program Files\INtime\bin\ldrta.exe
Invocation	Do one of these: • Click the "INtime RT Application Loader" shortcut locatedin the INtime start menu folder (Start\Programs\INtime). Click the Browse button and locate the rta you want to load. • Double-click an INtime application executable which has a .RTA extension. This launches the application on the default node with no command line options. • Right-click an rta file and click the Open button. You can then select the node and set command line options.

LoadRtk.exe (INtime Kernel Loader)

A 32-bit Windows program that loads the RT kernel after Windows starts. When set to automatically start, the INtime Kernel Loader launches the RT kernel at system startup. In this case, the loader loads the RT kernel after the Windows kernel and after other Windows services, but before users log on. Otherwise, the RT kernel is started manually using the INtime Status applet.

The INtime Kernel Loader cooperates with the RT Interface Driver (Rtif.sys) to load the RT kernel image set up by the INtime Configuration Utility.

First it loads the specified image into the memory allocated by the RT Interface Driver, then it makes a request to the RT Interface Driver to start the RT kernel.

Note

You can configure the RT kernel to start automatically at boot time by using the INtime Configuration utility.

Item	Description
Pathname	C:\Program Files\INtime\bin\loadrtk.exe
Invocation	Start INtime software's RT kernel either in Manual or Automatic mode, using the INtime Configuration utility.

mDNSINtime.exe

A Windows application which configures remote NTX connections automatically.

MFC*.dll files

Microsoft DLLs required by MFC programs; included in the event they were not installed with Windows.

Item	Description
Pathname	C:\Program Files\INtime\system32\mfc71.dll C:\Program Files\INtime\system32\msvcp71.dll C:\Program Files\INtime\system32\msvcr71.dll

network7 utility files

Item	Description
Pathname	%intime%network7 %intime%rt\include\network7 (header files)

NTX header files

Pathname	Files
C:\Program Files\ INtime\nt\include\...	ntx.h.....C NTX header file

NTX import libraries

Pathname	Files
C:\Program Files \INtime\nt\lib\...	ntx.lib..... NTX import library ntxext.lib.....Extended NTX import library.

NTX DLLs

DLLs provided with INtime software used by Windows applications to communicate with INtime applications using NTX system calls.

Item	Description
Pathname	C:\windows\system32\ntx.dll C:\windows\system32\ntxext.dll
Invocation	Windows loads these DLLs when a Windows application attempts the first NTX call.

NtxRemote2.exe (INtime Remote Connection Manager)

Manages connections with remote INtime nodes. Runs as the INtime Remote Connection Manager.

Item	Description
Pathname	C:\Program Files\INtime\bin\ntxremote2.exe
Invocation	Start INtime software's ntxremote2.exe either in Manual or Automatic mode using the Windows Services Manager (Start>Control Panel>Administrative Tools>Services>INtime Remote Connection Manager).

OvwGuide.pdf

The INtime Software User's Guide in PDF format. This file requires Acrobat32.exe.

Item	Description
Pathname	C:\Program Files\INtime\help\OvwGuide.pdf
Invocation	Double-click the file in the Windows Explorer. This launches Acrobat32.exe which, in turn, opens the OvwGuide.pdf file for viewing.

Project files

INtime software has a number of sample applications that you can use as samples for your own INtime applications.

The source code for these applications is provided in both Microsoft Visual Studio 6.0 and .Net 2003 generation format, and reside in separate directories. For example, the INtime API test program source files reside in the My Documents\INtime\Projects\RTTest directory.

Click on the desired project's name (Start>Programs/INtime/Sample Code and Projects) to open the Microsoft Visual Studio and find out more about each of these projects:

- INtime API Sample
- Serial Communications Sample
- Graphical Jitter
- Real-time Interrupt Sample
- C and C++ Samples for Debugger
- TCP Sample Applications
- UDP Sample Applications
- INtimeDotNet Sample Applications
- Fault Handling (ntrobust)
- Floating Point Exception Handling
- RSL Examples
- NTX Sample (MsgBoxDemo)
- Windows STOP Detection sample (STOPmgr)
- USB Client sample

Item	Description
Pathname	My Documents\INtime\Projects*SampleDirectory*\for source code My Documents\INtime\Projects*SampleDirectory**CompilerVersion*\ debug for executables
Invocation	Use Microsoft Visual Studio to edit/compile these sample applications. Use the INtime RT Application Loader to load the resulting sample application executables.

Quick Start Guide

The INtime Quick Start Guide in PDF format. This file requires the Adobe Acrobat reader.

Item	Description
Pathname	C:\Program Files\INtime\help\QuickStartGuide.pdf
Invocation	Double-click the file in Windows Explorer. This launches the Acrobat Reader which in turn opens the QuickStartGuide.pdf file for viewing.

RT header files

Pathname	Files
C:\Program Files\INtime\rt\include\...	C RT and C library header files.
C:\Program Files\INtime\rt\include\sys\...	Additional C RT and C library header files.
C:\Program Files\INtime\rt\include\\network7\...	Network C header files.
C:\Program Files\INtime\rt\include\cpp\...	C++ header files.
C:\Program Files\INtime\rt\include\services\...	C RT Services header files.

RT interface libraries

Pathname	Files
C:\Program Files\ INtime\rt\lib\...	clib.lib ..... CLIB function interface library cpplib.lib ..... C++ library net3m.lib ..... Sockets utility flat library pcibus.lib ..... PCI library rmxiff3m.lib ..... Flat RMX library rt.lib ..... INtime API interface library rtpp400.lib ..... INtime C++ library rtpp400d.lib ..... INtime C++ debug library rtserv.lib ..... INtime port interface library usbss.lib ..... INtime USB subsystem interface library

RT Stack Services

INtime RT components that make up the RT TCP/IP Stack. These components include NIC drivers and TCP/IP Stack Layers.

Pathname	Files
C:\Program Files\ INtime\network	3c59x.rta ..... INtime 3COM driver bcomg.rta ..... INtime Broadcom Gigabit driver dhcpcInt.rta ..... Intime DHCP Client application e1000.rta ..... Intie Intel Gigabit drivr eepro100.rta ..... INtime Pro 100 driver ip.rta ..... INtime IP loopback.rta ..... INtime Loopback driver ne.rta ..... INtime NE2000 driver rip.rta ..... INtime Raw IP rtl8139.rta ..... Realtek driver tcp.rta ..... INtime TCP udp.rta ..... INtime UDP

RT USB Interface Drivers

INtime RT components that make up the RT USB Subsystem. These components include Host Controller drivers for UHCI, OHCI, and EHCI (USB 2.0) interfaces.

Pathname	Files
C:\Program Files\INtime\bin	usbss.rsl ..... INtime USB Subsystem Shared Library uhci.rta ..... INtime USB Universal Host Controller Interface Driver ohci.rta ..... INtime USB Open Host Controller Interface Driver ehci.rta ..... INtime USB Enhanced Host Controller Interface (USB 2.0) Driver

RtClkSrv.exe (INtime Clock Synchronization Service)

A Windows program that provides time-of-day and time interval services used to synchronize the RT time-of-day clock to the Windows time-of-day clock.

Note

You can have the INtime Clock Synchronization Service start automatically at boot time by using the Windows Services Manager to set up the INtime Clock Synchronization Service for Automatic Startup. If the INtime Node Detection Service has not yet started, it is automatically started by the INtime Clock Synchronization Service.

Item	Description
Pathname	C:\Program Files\INtime\bin\rtclksrv.exe
Invocation	This program is run as a Windows service (Start>Control Panel>Administrative Tools>Services> INtime Clock Synchronization Service).

RtDrvrW5.awx (RT Device Driver wizard)

An MSVC 6.0 Application Wizard that you use to develop device drivers for INtime applications.

Item	Description
Pathname	C:\Program Files\INtime\msdev\template\rtdrvrw5.awx
Invocation	Invoked by the MSVC 6.0 IDE.

RtELServ.exe (INtime Event Log Service)

A 32-bit Windows program that supports Windows event log manipulation by RT threads. The RT Event Log Service receives, processes, and responds to requests from the RT application library's event log entry points. When the RT Event Log Service receives a request, it blocks the calling thread, executes the appropriate Win32 event log function, and replies to the original request (unblocking the calling thread). If the Windows host and/or the RT Event Log Service terminates execution, it terminates all pending requests with an E_EXIST error.

The RT Event Log Service supports a single request: to write an entry from the RT client event source at the end of the local PC's Application event log.

Neither the RT application library nor the RT Event Log Service support event logging to the system or security event logs, event logging by a source other than RT client, event logging to a remote PC, or backing up an event log file.

Note

You can have the INtime Event Log Service start automatically at boot time by using the Windows Services Manager to set up the INtime Event Log Service for Automatic Startup. If the INtime Node Detection Service has not yet started, the INtime Event Log Service automatically starts it.
The factory default is for this service to automatically start.

Item	Description
Pathname	C:\Program Files\INtime\bin\rtelserv.exe
Invocation	Start the INtime Event Log Service using the Windows Services Manager (Start>Control Panel>Administrative Tools>Services).

RtIf.sys (RT Interface Driver)

A Windows kernel mode device driver that co-manages the OSEM used by the INtime product to add RT capabilities to Windows. This driver:

- Establishes the initial RT kernel environment.
- Co-manages switching between the Windows and INtime runtime environments.
- Supports communication and synchronization mechanisms between Windows threads and RT threads by relaying NTX library requests.

Note

For detailed information, see INtime Help. For information about accessing help, see *Where to get more information* on page v.

Item	Description
Pathname	C:\windows\system32\drivers\rtif.sys
Invocation	Windows loads this driver at system initialization time.

RtIOCons.exe (INtime I/O console)

The RT I/O Console is a 32-bit Windows program that provides support for Windows console I/O for RT threads. It creates and manages a single console window and executes keyboard data (input) requests and display (output) requests from the RT thread.

Item	Description
Pathname	C:\Program Files\INtime\bin\rtiocons.exe
Invocation	Invoked by RtIOSrv.exe (INtime I/O Service)

RtIOSrv.exe (INtime I/O Service)

A 32-bit Windows program that provides Windows file system and console I/O support for RT threads. This program acts as a server to RT "C" library to obtain a console window for display (via printf) of application data and to receive (via scanf) user input from the system keyboard.

When the RT "C" library receives an stdio request from a real-time thread, it checks to see if a console exists. If a console exists, it relays the request to the appropriate RT I/O console If no console exists, it blocks the thread making the request, creates an RT I/O console window for the thread via the INtime I/O Service, and relays the request to the RT I/O console. When a request completes, the INtime I/O Service unblocks the corresponding thread and relays the reply. If Windows terminates execution, the RT I/O Console terminates all pending requests with an E_EXIST error.

When the RT "C" library receives a file I/O request from an RT thread, it blocks the thread making the request and forwards the request to the INtime I/O Service for completion. When a request completes, the RT "C" library unblocks the thread making the request and relays the reply to the thread.

Use of the INtime I/O Service is restricted to the RT "C" library.

Note

You can have the INtime I/O Service start automatically at boot time by using the Windows Services Manager to set up the INtime I/O Service for Automatic Startup. If the INtime Node Detection Service has not yet started, the INtimeIO Server automatically starts it.

The factory default is for this service to automatically start.

Item	Description
Pathname	C:\Program Files\INtime\bin\rtioserv.exe
Invocation	Start the INtimeIO Server using the Windows Services Manager (Start>Control Panel>Administrative Tools>Services).

RtNdSrv.exe (INtime Node Detection Service)

A 32-bit Windows program that detects RT clients, both local and remote. This program checks for and registers RT clients that exist in both of these locations:

- RT clients configured in INtime Configuration utility.
- RT clients available to the system.

Item	Description
Pathname	C:\Program Files\INtime\bin\rtndsrv.exe
Invocation	Start the INtime Node Detection Service using the Windows Services Manager (Start>Control Panel>>Administrative Tools>Services).

RtProcW5.awx (RT Process wizard)

An MSVC 6.0 Application Wizard that you use to develop the RT portion of INtime applications.

Item	Description
Pathname	C:\Program Files\INtime\msdev\template\rtprocw5.awx
Invocation	Invoked by the MSVC 6.0 IDE.

RtProcAddinW5.awx (RT Process Add-in wizard)

An Application wizard for Microsoft Visual Studio, version 6.0, that you use to add supplemental files to the already-generated RT portion of INtime applications.

Item	Description
Pathname	C:\Program Files\INtime\msdev\addins\rtprocaddinw5.awx
Invocation	Invoked by the MSVC 6.0 IDE.

RtRegSrv.exe (INtime Registry Service)

A Windows program that provides RT Registry Clients access to the Windows registry.

Item	Description
Pathname	C:\Program Files\INtime\bin\rtregsrv.exe
Invocation	Start the INtime Registry Service using the Windows Services Manager (Start>Control Panel>Administrative Tools>Services).

RtRslWiz.awx (RT Shared Library wizard)

An MSVC 6.0 Application Wizard that you use to develop a Realtime Shared Library (RSL) for an INtime application.

Item	Description
Pathname	C:\Program Files\INtime\msdev\template\rtrslwiz.awx
Invocation	Invoked by the MSVC 6.0 IDE.

Spider.exe (INtime standalone debugger)

A Windows program that provides standalone debug capabilities for any INtime application.

Item	Description
Pathname	C:\Program Files\INtime\bin\spider.exe
Invocation	Double-click the icon associated with the file in the INtime Program folder (Start>Programs>INtime>Spider debugger).

E Visual Studio debugging for older INtime projects

This appendix describes how existing INtime projects may be upgraded to use the newer Visual Studio product and its debugger. INtime software continues to provide support the Microsoft Visual C 6.0 and Visual Studio 2003 products for legacy applications, but the use of newer Visual Studio products allows you to take advantage of newer debugging features of the product.

To avoid confusion, we will refer to the newer Visual Studio products as "Visual Studio"; for earlier versions we will include the version number.

Upgrading from Visual Studio 6.0 to newer Visual Studio

When you open an existing project (.dsp or .dsw file) in Visual Studio, you are asked whether you want to upgrade; if you refuse, the workspace is not opened. If you agree, Visual Studio modifies your file set and you now have these files:

Filename	Contents
proj.c	Main source files
proj.sln	Solution properties
proj.suo	User options file
proj.vcproj	Project properties

The solution file, proj.sln, is the modern version of the workspace: it contains solution properties (a solution is the new word, but it means the same: a solution or workspace is a container for one or more projects). There can be different types of projects, but for INtime we are interested only in the C/C++ project type, which is referred to as a VC project and identified by the file extension .vcproj; a VC project file contains compiler and linker settings.

After upgrading, you now have access to all Visual Studio features, but you cannot use the integrated INtime debugger yet.

Converting to a .intp project

To get access to the integrated INtime debugger, we need to tell Visual Studio that we have an INtime project. This is accomplished by converting the VC project into an INtime project (represented by a file with an .intp extension) which encapsulates the VC project. When Visual Studio reads an INtime project, it provides access to features specific to INtime projects, which includes the integrated INtime debugger. Before leading you there, let us see how you convert a project.

You have already upgraded your VS6 workspace to a Visual Studio solution. Start by opening that solution in Visual Studio. Load the macros provided by INtime 4.0 into Visual Studio as follows (loading the macro project needs to be done only once):

1. Select Tools>Macros>Load macro project.
2. Navigate to the vstudio80 directory in INtime's installation directory.
3. Open the tointp.vsmacros file.
4. Locate ToIntp in the Macro Explorer window:
   A. Click the plus icon to expand ToIntp.
   B. Click the plus icon to expan Module1.
5. Double-click ToIntp, or click it once and press Enter.

   Progress messages display in the Output window, located at the bottom of the Visual Studio workspace. You are then prompted to save files.
6. Click Yes to save files.

You now have a solution with one or more INtime projects identified by an INtime icon; each INtime project contains a C/C++ project. Your set of files now looks like this:

Filename	Contents
proj.c	Main source files
proj.sln	Solution properties
proj.suo	Same, but in binary
proj.intp	INtime project properties
proj.vcproj	VC project properties

Setting project properties

An INtime project has an additional set of options: when you right-click the VC project and select properties, you see the compiler and linker settings, just like before. Right clicking the INtime project and selecting properties shows two pages (there are also menu items to achieve the same results):

- Launch settings, that determine on which INtime noe the application will run and what command line arguments are passed to it, and
- INtime settings, which control resources in the INtime system.

The pool maximum parameter on the INtime settings page is the same as the Heap reserve size on the Linker/System page of the C/C++ project properties.

Getting to work with the debugger

All of these steps had to be done only once. From now on you can use the integrated INtime debugger, just like you are used to doing so with Windows applications:

- Press F9 to set a breakpoint on a source line.
- Press F5 (run), Control-F5 (run without debugging), or F10 (single step) to start debugging.
- When at a breakpoint, view threads, variables, registers, call stack, modules, etc.

Note that the integrated debugger will refuse to debug an INtime RSL—you must select a project that represents a .RTA file.

What if conversion did not work?

The ToIntp macro makes some assumptions about projects that can be converted to INtime projects:

- Such a project is identified by the Version=21076.20052 option in the Linker/General properties page. Projects that do not have this option are skipped by the convertion macro.
- There are two configurations, named Debug and Release.
- compiler and linker options are set to default values applicable for INtime.

There is a second macro ToIntpSelected that only converts the selected project.

If for any reason you choose not to use the conversion macro but still want to use the integrated debugger, you can proceed as follows in Visual Studio:

1. Create a new solution if you wish.
2. In the solution, create a new project; use the INtime projects>Application wizard for this, and select an empty project.
3. When the wizard is finished, modify the VC project to suit your needs (add configurations, change compiler and linker settings). Note that you will not be able to modify the platform in the configuration manager; it will always ge “INtime”.
4. Add project items such as .c and .cpp source files, .h and .hpp include files, and so on.

Upgrading from Visual Studio 2003 to newer Visual Studio

All that is required is to open the existing solution (proc.sln) with the newer Visual Studio. You will be asked to upgrade and then the solution will be converted to the newer format. If you do not have a .intp project, proceed as described above under “Converting to a .intp product”.

F Adding INtime software to an XP Embedded configuration

You can add INtime components to your XP Embedded configuration using the standard mechanism provided in the XP Embedded Developer's Kit.

The INtime.sld file (C:\Program Files\INtime\Xpembedded\intime.sld) defines the INTime components and should be imported into the component database using the Component Database Manager. After creating new target XPE images, you can view and add INtime components from the Database.

INtime components are found in the component hierarchy under this key:

```
Software\System\OEM System Extensions\Infrastructure\
```

Note

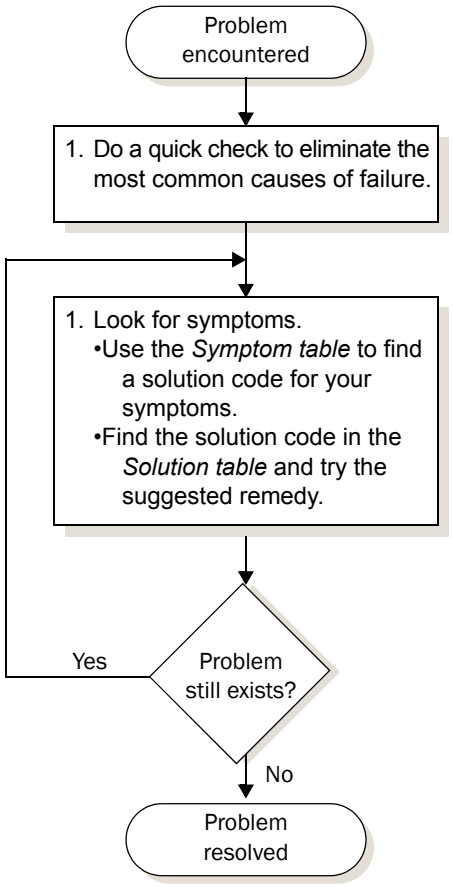
Before starting work with INtime on XP Embedded, obtain the latest sld file from TenAsys at support@tenasys.com.

G Troubleshooting

This appendix lists problems you may encounter while running INtime software, and explains how to avoid or resolve those problems.

Complete these steps to resolve INtime software problems:

Figure G-1. Troubleshooting INtime software problems



Do a quick check

Many problems are solved by correcting these:

- **Is the RT kernel running?** You must start the RT kernel prior to starting most INtime development tools and all INtime applications.
- **Did you use the INtime RT Application wizard to set up the project?** Using this wizard automatically starts an INtime project and ensures that the project includes proper settings.

Look for symptoms

Scan the *Symptom table* until you find a symptom that your system exhibits. The Symptom table lists solution codes in the order most likely to occur.

Locate the corresponding solution codes in the *Solution table* and take the recommended action. You may need to try several solutions before you successfully resolve the problem.

Table G-1. Symptom table

Symptom	Solution codes
INtime software does not install.	1, 2, 3, 4, 5, 6
Tools in the INtime development environment do not run.	11
The INtime application reports build errors.	9, 10
A serial mouse does not respond after you start the RT kernel.	12
The INtime application does not load.	11, 13
The file is not recognized as a valid INtime application.	14
The INtime application terminates unexpectedly.	15
The INtime application displays this message: `Running out of Virtual Address Space (error code 0xf0)`	15
The INtime application reports an unsupported C library function.	10
The INtime application performs terminal I/O, and no terminal window appears.	17
Only a blue screen displays.	20
Windows window events and applications respond slowly.	19, 16
The system doesn't respond to attempts to move the cursor or make entries from the keyboard.	18, 20, 19, 16, 8
You're running SDM and neither the system console nor the SDM terminal respond to attempts to move the cursor or make entries from the keyboard.	7

Table G-2. Solution table

Category	Solution	
Installing INtime software	1	Adjust your system configuration. Your system has an incompatible system configuration.
	2	Check the System Event Log for error messages from source RTIF. Also check the Application Event Log for error messages from LOADRTK.
	3	Ensure that your PC is set up as required for INtime software. For details, see *Requirements* on page 63.
	4	Ensure that you exit all programs before you install INtime software.
	5	Ensure that your system does not contain a previous version of INtime. If a previous version exists, uninstall it by selecting Start>Control Panel>Add/Remove programs.
	6	Ensure that you are logged on with Administrator privileges.
Developing INtime applications	7	Use the SDM PDP <logical address> command to get the physical address of memory in question, followed by S c80:physical address (returned by PDP). Do not use the SDM D command to display Ring 3 code/data in a Virtual Segment at an address not populated with memory. Do not use the SDM S command to modify memory using a Ring 3 code segment (read-only).
	8	If you're running SDM, no action is required. When you run SDM, you access the INtime software via the SDM terminal and the Windows system console no longer responds.
	9	Verify project build settings and ensure that you: • Select Not Using MFC to disable use of MFC classes. • Include the <INtime install path>\rt\include directory as a preprocessor directive. • Ignore all default libraries as a link option.
	10	Remove C library calls not supported by INtime. For a list of C library calls that INtime supports, run INtime Help and select Programmer's reference>C library reference.

Table G-2. Solution table

Category	Solution	
Loading INtime applications	11	Start the RT kernel before starting an INtime application. For information about starting the RT kernel, see *Chapter 9, Operation*.
	12	Do one of these: • If your mouse is on COM2, switch it to COM1. • Switch the RT kernel debug port to COM1 or disable RT kernel debug.
	13	Run the RT Application Loader and select the Advanced option to increase the amount of virtual address space in 4 MByte increments. Not enough memory exists to load the application. The program size and its heap and stack requirements specified when it was built may exceed the amount of memory available for INtime applications. For more information about changing memory options for loading INtime applications, see Help in the RT Application Loader.
	14	Rebuild the project. You may have tried to run a corrupted executable. This can occur, as an example, when a project is open and Windows halts.

Table G-2. Solution table

Category	Solution	
Running INtime applications	15	Run the INtime Explorer to obtain debug information. For information about running the INtime Explorer, see *After you start the INtime kernel* in *Chapter 9, Operation*.
	16	Set the RT kernel tick interval above 200usec. When the kernel tick drops below this rate, Windows slows down because the CPU devotes too much time to switching between the Windows and INtime kernels.
	17	Ensure that the INtime I/O Service is running. INtime applications that perform terminal I/O require that the INtime I/O Service runs. For information about starting this service, see *Starting the RT kernel and related components* in *Chapter 9, Operation*.
	18	End the INtime application: 1. Invoke the INtime Explorer (Start>All Programs>INtime>INtime Explorer). 2. Highlight the INtime application you want to terminate. 3. Right click the mouse. A pulldown menu displays. 4. Select Delete. INtex prompts you to confim the deletion process. 5. Select Yes. If you cannot display the INtime Explorer (i.e., the Windows screen seems frozen and the mouse does not respond), the INtime application may have halted or may be monopolizing system resources. If you suspect the latter, see solution 19. **Note**: • To verify whether Windows has halted, try to access the file system from another system. If you can, Windows is still running but cannot respond. • Ensure that you wait for at least ten seconds, in case the Spin Doctor detects a spinning thread.
	19	Adjust your INtime application so that the RT portion of your INtime applications do not dominate CPU time. Your INtime application may be designed to monopolize too many system resources. For more information about designing applications to balance RT and Windows activity, see *Methodology* in *Chapter 5, Designing RT applications*. **Note**: To aid in diagnosing system resource misuse, try to access the file system from another system. If you can, Windows is still running but cannot respond.
Running INtime applications (continued)	20	Exit INtime applications, if possible, then reboot your system: One of these situations may have occured: • Windows halted. • A Windows application halted. • An INtime application monopolized system resources. **Note**: Try other solutions before trying this one.

Other resources

If the information in this chapter doesn't solve the problem, you may want to contact TenAsys as described in *Where to get more information* on page v.

Glossary

Location. a handle which uniquely identifies a node.

application loader The layer of an INtime application that loads programs into memory for execution, such as when a user enters a command at the console.

asynchronous Events that occur at random times.

BIOS (Basic I/O System) On a PC system, the code that resides in ROM to supply OS-independent access to the computer's I/O system.

BSOD (Blue Screen of Death) An acronym used to describe total Windows failure.

client On a network, a client is a computer which makes requests of a remote system that acts as a server. For example, a client could request a remote server to supply it with data from one of the server's disk files.

descriptor An 8-byte data structure taken from a descriptor table in memory. Descriptors provide the CPU with the data it needs to map a logical address into a linear address. The fields of a descriptor include information concerning segment size, base address, access rights, and segment type (such as read/write segment, executable segment, call gate, task gate, trap gate, etc).

determinism Predictable response time. Enables threads to execute before their deadlines expire.

device controller The hardware interface between the CPU or system bus and a device unit.

device driver The software interface between the I/O system and a device controller.

Distributed INtime A configuration of INtime where an INtime kernel (RT client) runs on a CPU that does not run Windows.

encapsulation A characteristic of object-based systems. The representation of an object is hidden from the user of that object. Only the object's type manager can manipulate an object. Users of an object can manipulate the object only by invoking type manager functions for the object.

EOI (End of Interrupt) A command sent to a PIC to indicate that an interrupt handler completed processing an interrupt.

event-driven Applications can respond to interrupts as they occur; they do not waste time polling for interrupts.

exception handler A program that receives control when either the operating system or hardware detects an error.

exchange object	Generic name for object types managed by INtime software that allow threads to synchronize and communicate with each other. Exchange objects include: semaphores, mailboxes, and regions.
execution state	Thread state. Thread execution states include: running, ready, asleep, suspended, or asleep-suspended.
FIFO	First in, first out.
GDT	(Global Descriptor Table) A memory segment that contains descriptors for code, data, and descriptor table segments.
HAL	Hardware Abstraction Layer.
handle	An object identifier.
host	A computer consisting of one or more processing elements (cores or hardware threads).
host scope	Accessible by all processes on all nodes of a given host. See *node scope* on page 183 and *universal scope* on page 185.
IDT	Interrupt descriptor table.
interrupt	A signal from a device such as a NIC (Network Interface Card) or IDE hard disk controller. You connect interrupt sources to the processor through a PIC (Programmable Interrupt Controller).
interrupt handler	Code executed first in response to a hardware interrupt. This code runs in the context of the thread that was running when the interrupt occurred. Interrupt handlers must save the current context on the stack before using any CPU registers.
interrupt levels	PICs manage interrupts by presenting them to the system processor as discreet levels in priority order. INtime software handles more critical interrupts first, and keeps track of which interrupts occurred, the order in which they occurred, and which ones have not been handled.
interrupt response time	The time between a physical interrupt happening and the system beginning to execute the interrupt handler. By being able to calculate a predictable worst-case response time to interrupt processing, a real-time system can be designed to ensure that incoming data is handled before it becomes invalid.
INtime host	A computer which is running one or more instances of the INtime RT kernel, but not Windows. See *Windows host* on page 185.
IPI	(Inter Processor Interrupt) A way to communicate between multiple processors in a system.
MAC	(Media Access Control) The 6-byte Ethernet address for a computer node on a network.
mailbox	An RT kernel object type managed by the RT kernel. Mailboxes are used for interthread synchronization and communication. Two types of mailboxes exist: • **Data mailbox**: sends and receives data. Available in both high and low level.

- **Object mailbox**: sends and receives object handles. Available only in high level.

memory address The architectural mechanism used by x86 processors to access an individual byte of system physical memory. In the 32 bit protected mode environment of Windows and INtime, memory addresses are 32 bit linear addresses relative to a 16 bit descriptor that is loaded into a CPU segment register. The INtime kernel manages the 16 bit descriptors (virtual segments - VSEGs) for INtime applications that only use 32 bit linear addresses to access code and data (flat model applications generated with MSVC tools).

memory area Provides memory for threads to use for many purposes, including communicating and storing data.

memory pool An amount of memory, with a specified minimum and maximum, allocated to a process. The basis of INtime software's memory management. The initial memory pool is all the memory available to the application (that is, free space memory). It is managed by the OS and allocated to the application on request.

message port An RT kernel object type managed by the kernel. Used to provide an access point for an INtime application thread to communicate with an INtime service.

multi-programming Ability of an operating system to simultaneously run several unrelated applications on a single system. Allows more than one application to run at a time.

multithreading Ability of an operating system to run multiple threads at virtually the same time. When the operating system stops executing one thread, it resumes/starts executing another thread. This transition from one thread to another is called a thread switch.

node An instance of an operating system. A node may be on its own on a host, or one of several on a multi-core host, or sharing a host with Windows. See *remote node* on page 184 and *Windows node* on page 185.

node scope Accessible by all processes on a given node. See *host scope* on page 182 and *universal scope* on page 185.

object An instance of a data structure that can be accessed only through a set of functions provided by a type manager.

object directory A storage area within an INtime process where objects can be cataloged, i.e. have a name associated with the objects so that other theads may refer to/access the cataloged object by name.

OSEM (OS Encapsulaton Mechanism) Manages the simultaneous operation and integrity of the Windows kernel and the RT kernel. Used only with INtime nodes.

PIC (Programmable Interrupt Controller) An integrated circuit that can resolve simultaneous interrupt requests and negotiate a sequence of CPU interrupt requests based on the priorities of the requesting device controllers.

priority-based scheduling Abiltiy of an operating system to assign execution order importance values (priority) to each thread in the system. In an INtime system, the scheduling policy enforced by the INtime kernel is that the highest priority ready thread is/will immediately become the running thread. Thus, when thread A is running, if thread B becomes ready

	through some system event, and thread B is higher priority than thread A, then the INtime kernel will immediately preempt thread A, and make thread B the running thread.
priority inversion	A situation in which a high-priority thread is effectively prevented from running by a lower-priority thread. Proper use of the RT kernel's region objects eliminates this problem.
process	An RT kernel object type. Processes have memory pools and contain/own execution threads and other objects.
region	An RT kernel object type managed by the kernel. Regions are binary semaphores with special suspension, deletion, and priority-adjustment features. You can use regions to provide mutual exclusion for resources or data.
remote node	A host other than the host where the current process is running. See *Windows node* on page 185.
round-robin scheduling	A scheduling method where equal priority threads take turns running. Each thread gets a time slice, an equal portion of the processor's time.
RT	Real-time.
RT client	An RT subsystem designated to consume INtime services (booting, configuration, file system proxy, etc.) provided by a Windows host.
RT kernel	(Real-time kernel) An operating system that provides full RT functionality.
RT subsystem	A self-contained collection of software containing an RT kernel, a number of RT support processes, and the RT portion of zero or more INtime applications.
semaphore	An RT kernel object type managed by the kernel. Semaphores are used for interthread synchronization. A counter that takes positive integer values. Threads use semaphores for synchronization by sending units to and receiving units from the semaphores.
server	On a network, a server is any computer which responds to requests from remote systems. For example, file or print servers allow remote systems to access local disks or printers.
system call	A subroutine supplied by INtime software to provide a service, such as I/O processing or memory allocation. A programmatic interface you use to manipulate objects or control the computer's actions.
thread	An RT kernel object type managed by the RT kernel. Threads, or threads of execution, are the active, code-executing objects in a system.
thread switch time	The time required to save the context (data registers, stack and execution pointers) of one thread, and to start another thread by moving its context into the processor registers.
time slice	An equal portion of the processor's time.

universal scope Accessible by all processes on all nodes. See *host scope* on page 182 and *node scope* on page 183.

Windows host A computer that simultaneously executes both a Windows host and one or more RT nodes that are connected via the OSEM. See *INtime host* on page 182.

Windows node An instance of Windows whether running on a single or multiple hardware threads. See *node* on page 183 and *remote node* on page 184.

Windows subsystem A self-contained collection of software containing Windows, a number of Windows support processes, and the Windows portion of zero or more INtime applications.

Index

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

accept *130*
AcceptRtControl *115*
alarms *32*
AllocateRtMemory *109*
allocating
    memory *48*
application development
    design *55*
    examples *58*
applications
    INscope *90*
    INtex *90*
    RT Event Log Server *164*
    RT kernel loader *158*
    RT loader *157*
    RT Node Detection Server *166*
    sample, defined *10*
    SDM *89*
    Spider *89*
    Windows NT Crash program *152*
AttachRtHeap *113*
AttachRtPort *113*

B

bind *130*
BindRtPort *113*
blue screen protection *13*
Blue.exe *152*
browser, INtime RT Client *156*
BSOD. *See blue screen protection.*
bstring *130*
byteorder *130*

C

C and C++ sample for debugger *161*
C++ Help files *154*
calls
    NTX *103*
    real-time *104*
        low-level *104*
calls. *See system calls.*
CancelRtTransaction *113*
CancelTransaction *127*
CatalogRtHandle *111*
ClearCommBreak *128*
ClearCommError *128*
CloseComm *128*
comedgeport.rta *129*
comlist.rta *129*
COMM calls *128*
    ClearCommBreak *128*
    ClearCommError *128*
    CloseComm *128*
    Drivers
        comedgeport.rta *129*
        compc.rta *129*
        comrocket.rta *129*
    drivers *129*
    EscapeCommFunction *128*
    FlushCommBuffers *128*
    GetCommConfig *128*
    GetCommMask *128*
    GetCommModemStatus *128*
    GetCommProperties *128*
    GetCommState *128*
    GetCommTimeouts *128*
    OpenComm *128*
    PurgeComm *128*
    ReadComm *128*
    ResetCommEvent *128*
    SetCommBreak *128*
    SetCommConfig *128*
    SetCommMask *129*
    SetCommState *129*
    SetCommTimeouts *129*
    structures *129*
    TransmitCommChar *129*
    Utilities
        comlist.rta *129*
    utilities *129*
    WaitCommEvent *129*
    WriteComm *129*

COMMCONFIG structure *129*
COMMPROP structure *129*
COMMTIMEOUTS structure *129*
communicating
between threads *46*
between Windows NT and RT threads *4*
compc.rta *129*
comrocket.rta *129*
COMSTAT structure *129*
configuring INtime software
default settings *67*
interrupt resources *69*
options *67*
connect *130*
connecting to INtime host *73*
ConnectRtPort *113*
CONTROLBUFFER structure *119*
conventions, notational *v*
CopyRtData *110*
CopyRtSystemInfo *117*
CPUFRAME structure *119*
crash program, Windows NT *152*
CreateGlobalRtMailbox *107*
CreateGlobalRtMemoryHandle *107*
CreateGlobalRtMemoryObject *107*
CreateGlobalRtSemaphore *106*
CreatePort *127*
CreateRtHeap *110*
CreateRtMailbox *108*
CreateRtMemoryHandle *109*
CreateRtPort *113*
CreateRtPortEx *113*
CreateRtReferenceObject *106*
CreateRtRegion *114*
CreateRtSemaphore *116*
CreateRtThread *117*

D

data, validity of *48*
DCB structure *129*
deadlock *35*
debug tools
INscope *90*
INtex *90*
SDM *89*
Spider *89*
debuggers *9*
C and C++ sample *161*
default configuration settings *67*
DeletePort *127*
DeleteRtHeap *110*
DeleteRtMailbox *108*
DeleteRtMemoryHandle *109*
DeleteRtPort *113*
DeleteRtProcess *114*
DeleteRtReferenceObject *106*
DeleteRtRegion *114*
DeleteRtSemaphore *116*
DeleteRtThread *117*
DeliverMessage *126*
DeliverStatus *126*
DeliverTransaction *126*
DequeueInputTransaction *126*
DequeueOutputTransaction *126*
desigining RT applications
RT processes, appropriate tasks *57*
sample system *58*
Windows NT processes, assigning priority *57*
designing RT applications
guidelines *55*
DetachRtHeap *113*
DetachRtPort *113*
determinism *43*
interrupt response time *43*
thread switch time *43*
developing INtime applications *7*
development environment. *See INtime development environment.*
directories, object *31*
DisableRtInterrupt *107*
drivers
RT interface *19* , *164*
RT USB interface *163*
DSM calls *105*
dynamic memory *30*

E

e-mail address, TenAsys *vi*
email, technical support *vi* , *93* , *173*
email, TenAsys *v*
EnableRtInterrupt *107*
encapsulating Windows NT
See OSEM 18

encapsulating Windows NT as an INtime software thread *19*
encapsulation mechanism. *See OSEM.*
EnqueueInputTransactionTransaction *126*
EnqueueOutPutTransaction *126*
EnterRtInterrupt *107*
EnterServiceRegion *126*
EscapeCommFunction *128*
Ethernet *18*
Ethernet, high performance calls *123*
event log service *164*
event-driven applications *41*
EVENTINFO structure *119*
EventMsg.dll *152*
examples
application systems *58*
interrupt handlers *24* , *59*
multitasking *59*
mutual exclusion *34* , *35*
regions *35*
sample system *58*
semaphores *34*
synchronizing threads *47*
exception handling calls *106*
exception handling, floating point *161*
EXCEPTION structure *119*
exchange objects *32*
mailboxes *33*
ports *36*
regions *35*
semaphores *34*
validation levels *32*
exclusion, mutual *48*
execution state *21*
ExitRtProcess *114*
ExitServiceRegion *126*

F

Fault Handling (ntrobust) *161*
Fault Manager *52*
file types
.DLL *8*
.EXE *8*
.RTA *8*
FILETIME structure *119*
Finish *127*
Floating Point Exception Handling *161*
FlushCommBuffers *128*
FreeRtLibrary *125*
FreeRtMemory *109*
functions. *See system calls.*

G

GENADDR structure *119*
get_RT_trace_state *124*
GetAttributes *127*
GetCommConfig *128*
GetCommMask *128*
GetCommModemStatus *128*
GetCommProperties *128*
GetCommState *128*
GetCommTimeouts *128*
GetFirstRtLocation *106*
GetFragment *127*
GetGlobalRootRtProcess *106*
gethostent *130*
gethostname *130*
GetLastRtError *117*
getnetent *130*
GetNextRtLocation *106*
getpeername *130*
GetPortId *126*
GetPortParameter *126*
getprotoent *130*
GetRServiceAttributes *112*
GetRtBufferSize *110*
GetRtExceptionHandlerInfo *106*
GetRtHandleType *111*
GetRtHandleTypeEx *111*
GetRtHeapInfo *110*
GetRtInterruptLevel *107*
GetRtModuleHandle *125*
GetRtNodeInfo *106*
GetRtNodeLocationByName *106*
GetRtNodeStatus *106*
GetRtObjectInfo *106*
GetRtPhysicalAddress *109*
GetRtPortAttributes *113*
GetRtProcAddress *125*
GetRtSize *110*
GetRtThreadAccounting *117*
GetRtThreadHandles *117*
GetRtThreadInfo *117*
GetRtThreadPriority *118*

GetRtThreadState *118*
getservent *130*
getsockname *130*
getsockopt *130*
GetTransaction *126*
global object calls *106*
    CreateGlobalRtMailbox *107*
    CreateGlobalRtMemoryHandle *107*
    CreateGlobalRtMemoryObject *107*
    CreateGlobalRtSemaphore *106*
    CreateRtReferenceObject *106*
    DeleteRtReferenceObject *106*
    GetFirstRtLocation *106*
    GetGlobalRootRtProcess *106*
    GetNextRtLocation *106*
    GetRtNodeInfo *106*
    GetRtNodeLocationByName *106*
    GetRtNodeStatus *106*
    GetRtObjectInfo *106*
guidelines for designing applications *55*
guidelines for designing INtime applications *55*

H

HAL, modified for INtime software *12* , *20*
handlers
    interrupt *42*
    service *127*
handlers, interrupt
    alone *24*
    handler/thread combination *24*
header files
    NTX *159*
    RT *162*
HEAPINFO structure *119*
help *iii* , *v* , *vi* , *3* , *93* , *153* , *173*
high-performance gb Ethernet calls *123*
host, connecting to *73*
HWEXCEPTIONMSG structure *119*

I

INBrow.ocx *156*
inbyte *133*
INCnfgcpl.cpl *152*
INConfig.hlp *153*
inet *130*
inhword *133*
Initialize *127*
Input/Output calls *133*
    inbyte *133*
    inhword *133*
    inword *133*
    outbyte *133*
    outhword *133*
    outword *133*
INscope application *90*
INscope calls *124*
    get_RT_trace_state *124*
    log_RT_event *124*
    pause_RT_trace *124*
    RT_I_am_alive *124*
    start_RT_trace *124*
    stop_RT_trace *124*
Installation program, running *64*
installing INtime software
    before you begin *63*
    running the Installation program *64*
InstallRtServiceDescriptor *112*
interface driver, RT *19*
interface, RT driver *164*
internal loop. *See loop.*
interrupt calls *107*
interrupt handlers *24* , *42*
    alone *24*
    examples *24* , *59*
    handler/thread combination *24*
INTERRUPTINFO structure *119*
interrupts
    during RT operation *19*
    levels *42*
    processing *42*
    resources, configuring *69*
    response time *43*
    threads *24* , *42*
INtex application *90*
INtime API Test *161*
INtime Application wizard *82*
INtime applications
    described *4*
    developing *7* , *55*
    files *8*
    how they run *16*
    RT processes, appropriate tasks *57*
    runtime behavior *19*

sample system *58*
Windows NT processes, assigning priority *57*
INtime development environment
debuggers *9*
libraries *9*
wizards *8*
INtime Driver Model Serial Driver Test Demo *161*
INtime Graphical Jitter *161*
INtime Performance Monitor.INtmPerf.*
files.Performance monitor *155*
INtime RT Client Browser *156*
INtime RT process add-in wizard *84*
INtime runtime environment
blue screen protection *13*
encapsulating Windows NT as a thread *19*
HAL, modified for Windows NT *12*
HAL, modified for windows NT *20*
how INtime applications run *16*
kernel *12*
memory protection *12*
OSEM *12* , *18*
RT interface driver *19*
sample applications *10*
INtime Serial Driver Sample *161*
INtime services
INtime Registry service *167*
INtime software
configuring *67*
default settings *67*
interrupt resources *69*
debuggers *9*
defined *3*
libraries *9*
requirements *63*
wizards *8*
INtime Software Overview Guide, PDF format *160*
INtime USB Client sample *161*
INtime Windows STOP Detection sample (STOPmgr) *161*
INtime.hlp *153*
INtimeDotNet calls *131*
ntxCatalogNtxHandle *131*
ntxCreateRtMailbox *131*
ntxCreateRtProcess *131*
ntxCreateRtSemaphore *131*
ntxDeleteRtMailbox *131*
ntxDeleteRtSemaphore *131*
ntxGetFirstLocation *131*
ntxGetLocationByName *132*
ntxGetNameOfLocation *132*
ntxGetNextLocation *132*
ntxGetRootRtProcess *132*
ntxGetRtErrorName *132*
ntxGetRtSize *132*
ntxGetRtStatus *132*
ntxGetType *132*
ntxImportRtHandle *132*
ntxLoadRtErrorString *132*
ntxLookupNtxHandle *132*
ntxNotifyEvent *132*
ntxReadRtXxx *132*
ntxReceiveRtDataXxx *132*
ntxReceiveRtHandle *132*
ntxRegisterDependency *132*
ntxRegisterSponsor *132*
ntxReleaseRtSemaphore *132*
ntxSendRtDataXxx *132*
ntxUncatalogNtxHandle *132*
ntxUnregisterDependency *132*
ntxUnregisterSponsor *132*
ntxWaitForRtSemaphore *132*
ntxWriteRtXxx *132*
structures *133*
NTXEVENTINFO *133*
NTXPROCATTRIBS *133*
INtimeDotNet callsntxSendRtHandle *132*
INtimeDotNet sample applications *161*
inword *133*
ITWrpSrv.hlp *153*
iWIn32 header files *157*
iWin32 library files *157*
iWin32x header files *157*
iWin32x import library *157*

J

Jitter *161*

K

kernel
blue screen protection *13*
loader *158*
memory protection *12*
RT *155*
time management *25*

knCreateRtAlarmEvent *118*
knCreateRtMailbox *108*
knCreateRtSemaphore *116*
knDeleteRtAlarmEvent *119*
knDeleteRtMailbox *108*
knDeleteRtSemaphore *116*
knGetKernelTime *119*
knReleaseRtSemaphore *116*
knResetRtAlarmEvent *118*
knRtSleep *115*
knSendRtData *108*
knSendRtPriorityData *108*
knSetKernelTime *119*
knStartRtScheduler *115*
knStopRtScheduler *115*
KNTIME structure *119*
knWaitForRtAlarmEvent *118*
knWaitForRtData *108*
knWaitForRtSemaphore *116*

L

LdRta.exe *157*
LdRta.hlp *153*
levels, interrupt *42*
libraries *9*
    NTX import *159*
    RT interface *162*
limitations
    maximum objects in system *28*
listen *130*
loader
    RT application *157*
    RT kernel *158*
LoadRtk.exe *158*
LoadRtLibrary *125*
log_RT_event *124*
LookupPortHandle *126*
LookupRtHandle *111*
loop. *See internal loop.*
low-level calls *104*
    when to use *104*

M

mailbox calls *108*
mailboxes, kernel *32* , *33* , *104*
managing time *25*
MapRtPhysicalMemory *109*
MapRtSharedMemory *109*
mDNSINtime.exe *159*
memory
    allocating and sharing *48*
    dynamic *30*
    pools *30*
    segments *30*
memory calls *109*
memory heap calls *120*
memory pool calls *120*
memory pools
    defined *30*
memory protection *12*
memory/buffer management *109*
message port calls *112*
message transmission calls *113*
MFC*.dll files *159*
modified Windows NT Hardware Abstraction Layer.
        See HAL.
modular programming *40*
MsgBoxDemo (NTX Sample) *161*
multiprogramming *44*
multitasking, examples *59*
multi-threading *39*
mutual exclusion
    defined *48*
    examples *34* , *35*

N

network stack calls *124*
network7 utility files *159*
Node Detection Server *166*
non-validating calls *104*
notational conventions *v*
NT Crash program *152*
ntrobust (Fault Handling) *161*
NTX calls *103*
NTX header files *159*
NTX import libraries *159*
NTX Sample (MsgBoxDemo) *161*
NTX.dll *160*
ntxAttachRtPort *112*
ntxBindRtPort *112*
ntxCancelRtTransaction *112*
ntxCatalogNtxHandle *131*
ntxCatalogRtHandle *111*
ntxConnectRtPort *112*

ntxCopyRtData *109*
ntxCreateRtMailbox *108* , *131*
ntxCreateRtPort *112*
ntxCreateRtProcess *114* , *131*
ntxCreateRtSemaphore *116* , *131*
ntxDeleteRtMailbox *108* , *131*
ntxDeleteRtPort *112*
ntxDeleteRtSemaphore *116* , *131*
ntxDetachRtPort *112*
NTXEVENTINFO structure *119* , *133*
ntxFindINtimeNode *117*
ntxGetFirstLocation *117* , *131*
ntxGetLastRtError *116*
ntxGetLocationByName *117* , *132*
ntxGetNameOfLocation *117* , *132*
ntxGetNextLocation *117* , *132*
ntxGetRootRtProcess *111* , *132*
ntxGetRtErrorName *116* , *132*
ntxGetRtPortAttributes *112*
ntxGetRtServiceAttributes *112*
ntxGetRtSize *109* , *132*
ntxGetRtStatus *116* , *132*
ntxGetType *111* , *132*
ntxImportRtHandle *132*
ntxImportRthandle *111*
ntxLoadRtErrorString *116* , *132*
ntxLookupNtxHandle *132*
ntxLookupNtxhandle *111*
ntxMapRtSharedMemory *109*
ntxMapRtSharedMemoryEx *109*
ntxNotifyEvent *105* , *114* , *132*
NTXPROCATTRIBS structure *119* , *133*
ntxReadRtXxx *132*
ntxReceiveData *108*
ntxReceiveHandle *108*
ntxReceiveRtDataXxx *132*
ntxReceiveRtHandle *132*
ntxReceiveRtMessage *112*
ntxReceiveRtReply *112*
ntxRegisterDependency *105* , *114* , *132*
ntxRegisterSponsor *105* , *114* , *132*
ntxReleaseRtBuffer *112*
ntxReleaseRtSemaphore *116* , *132*
NtxRemote2.exe *160*
ntxRequestRtBuffer *112*
ntxSendData *108*
ntxSendHandle *108*
ntxSendRtDataXxx *132*
ntxSendRtHandle *132*
ntxSendRtMessage *112*
ntxSendRtMessageRSVP *112*
ntxSetRtServiceAttributes *112*
ntxUncatalogNtxHandle *132*
ntxUncatalogRtHandle *111*
ntxUnmapRtSharedMemory *109*
ntxUnregisterDependency *105* , *114* , *132*
ntxUnregisterSponsor *105* , *114* , *132*
ntxWaitForRtSemaphore *116* , *132*
ntxWriteRtXxx *132*

O

OBJECTDIR structure *119*
objects
    defined *27*
    directories *31* , *111*
    exchange *32*
        mailboxes *33*
        ports *36*
        regions *35*
        semaphores *34*
    maximum in system *28*
    memory pools *30*
    processes *28*
    threads *28*
OpenComm *128*
OS encapsulation mechanism. *See OSEM.*
OSEM *12* , *18*
outbyte *133*
outhword *133*
outword *133*
OvwGuide.pdf *160*

P

pause_RT_trace *124*
PCI library calls *124*
    PciClassName *125*
    PciDeviceName *125*
    PciEnableDevice *125*
    PciFindDevice *124*
    PciGetConfigRegister *124*
    PciInitialize *124*
    PciReadHeader *124*
    PciSetConfigRegister *124*
    PciVendorName *124*
PciClassName *125*

PCIDEV structure *119*
PciDeviceName *125*
PciEnableDevice *125*
PciFindDevice *124*
PciGetConfigRegister *124*
PciInitialize *124*
PciReadHeader *124*
PciSetConfigRegister *124*
PciVendorName *124*
PDF file, *INtime Software Overview Guide* *160*
POOLINFO structure *119*
pools, memory *30*
port calls
    message transmission *113*
    port object management *113*
    service support *112*
port callst *112*
port object management calls *113*
PORTINFO structure *119*
ports *36*
pre-emptive scheduling *41*
priority-based scheduling *21* , *41*
process management calls *114*
processes *28*
    about *28*
    components *44*
    tree *29*
processing, interrupts *42*
Project files *160*
protection
    blue screen *13*
    memory *12*
PurgeComm *128*

Q

QueryInputTransactionQueue *127*
QueryOutputTransactionQueue *127*
Quick Start Guide *161*

R

ReadComm *128*
real-time
    debuggers *9*
    interface driver *19*
    INtime applications, designing *55*
    response *39*
    RT kernel *12*
real-time calls *104*
Real-time Interrupt Sample *161*
Real-time Shared Library (RSL) calls *125*
RECEIVEINFO structure *119*
ReceiveRtData *108*
ReceiveRtFragment *113*
ReceiveRtHandle *108*
ReceiveRtMessage *113*
ReceiveRtReply *113*
recursive. *See recursive.*
recv *130*
recvfrom *130*
region calls *114*
regions *35*
RegisterRtDependency *105* , *114*
RegisterRtSponsor *105* , *114*
Registry calls *125*
registry calls *125*
ReleaseControlBuffer *127*
ReleaseRtBuffer *110*
ReleaseRtControl *115*
ReleaseRtSemaphore *116*
ReleaseTransaction *127*
ReportRtEvent *117*
RequestControlBuffer *127*
RequestRtBuffer *110*
RequestTransaction *127*
requirements for INtime software *63*
ResetCommEvent *128*
ResetRtInterruptHandler *107*
response time, predictability *43*
ResumeRtThread *118*
round-robin scheduling *23*
RSL calls
    FreeRtLibrary *125*
    GetRtModuleHandle *125*
    GetRtProcAddress *125*
    LoadRtLibrary *125*
    RtTlsFree *125*
    RtTlsGetValue *125*
    RtTlsSetValue *125*
RSL Example *161*
RT application loader *157*
RT Clock Synchronization Server *163*
RT Device Driver wizard *163*
RT Event Log Server *164*
RT header files *162*

RT I/O console *165*
RT I/O Server *165*
RT interface driver *164*
RT interface libraries *162*
RT kernel *12* , *155*
  blue screen protection *13*
  communicating with Windows NT *4*
  loader *158*
RT Node Detection Server *166*
RT Process Add-in wizard *166*
RT process wizard *166*
RT service handlers *127*
  CancelTransaction *127*
  CreatePort *127*
  DeletePort *127*
  Finish *127*
  GetAttributes *127*
  GetFragment *127*
  Initialize *127*
  SendMessage *127*
  Service *127*
  SetAttributes *128*
  UpdateReceiveInfo *128*
  VerifyAddress *128*
RT services *105*
RT Shared Library wizard *167*
RT USB Interface Drivers *163*
RT_I_am_alive *124*
rta files
  comedgeport.rta *129*
  comlist.rta *129*
  compc.rta *129*
  comrocket.rta *129*
RtAddinW.hlp *153*
RtClkSrv *163*
RtDrvrW.hlp *153*
RtDrvrW5.awx *163*
RtELServ.exe *164*
RTIF.SYS *18* , *19*
RtIf.sys *164*
RtIOCons.exe *165*
RtIOSrv.exe *165*
RtkImage.dbg *155*
RtNdSrv.exe *166*
RtNotifyEvent *105*
RtProc5.awx *166*
RtProcAddinW5.awx *166*
RtProcW.hlp *153*
RtRegCloseKey *125*
RtRegConnectRegistry *125*
RtRegCreateKeyEx *125*
RtRegDeleteKey *125*
RtRegDeleteValue *125*
RtRegEnumKeyEx *125*
RtRegEnumValue *125*
RtRegFlushKey *125*
RtRegLoadKey *125*
RtRegOpenKeyEx *125*
RtRegQueryInfoKey *125*
RtRegQueryValueEx *125*
RtRegReplaceKey *126*
RtRegRestoreKey *126*
RtRegSaveKey *126*
RtRegSetValueEx *126*
RtRegSrv.exe *167*
RtRegUnLoadKey *126*
RtRslWiz.awx *167*
RtSleep *118*
RtTlsAllocRSL calls
  RtTlsAlloc *125*
RtTlsFree *125*
RtTlsGetValue *125*
RtTlsSetValue *125*
runtime environment. *See INtime runtime environment.*

S

sample applications *10* , *160*
scheduler calls *115*
scheduling
  priority-based *21* , *41*
  round-robin *23*
SDM application *89*
SECURITY_ATTRIBUTES structure *119*
segments *30*
select *130*
semaphores *34* , *115*
send *130*
SendMessage *127*
SendRtData *108*
SendRtHandle *108*
SendRtMessage *113*
SendRtMessageRSVP *113*
SendRtReply *113*

sendto *130*
Serial Communications calls *128*
    ClearCommBreak *128*
    ClearCommError *128*
    CloseComm *128*
    Drivers
        comedgeport.rta *129*
        compc.rta *129*
        comrocket.rta *129*
    EscapeCommFunction *128*
    FlushCommBuffers *128*
    GetCommConfig *128*
    GetCommMask *128*
    GetCommModemStatus *128*
    GetCommProperties *128*
    GetCommState *128*
    GetCommTimeouts *128*
    OpenComm *128*
    PurgeComm *128*
    ReadComm *128*
    ResetCommEvent *128*
    SetCommBreak *128*
    SetCommConfig *128*
    SetCommMask *129*
    SetCommState *129*
    SetCommTimeouts *129*
    TransmitCommChar *129*
    Utilities
        comlist.rta *129*
    WaitCommEvent *129*
    WriteComm *129*
Service *127*
service handlers *127*
service support calls *112*
SERVICEATTRIBUTES structure *119*
SERVICEDESC structure *120*
Services
    RT Event Log Server *164*
    RT I/O Server *165*
    RT Node Detection Server *166*
services, RT *105*
ServicesLRT Clock Synchronization Server *163*
SetAttributes *128*
SetCommBreak *128*
SetCommConfig *128*
SetCommMask *129*
SetCommState *129*
SetCommTimeouts *129*
sethostname *130*
SetLastRtError *117*
SetPortParameter *127*
SetRtExceptionHandler *106*
SetRtInterruptHandler *107*
SetRtInterruptHandlerEx *107*
SetRtProcessMaxPriority *118*
SetRTServiceAttributes *112*
SetRtSystemAccountingMode *118*
SetRtThreadPriority *118*
setsockopt *130*
sharing memory *48*
shutdown *130*
SignalEndOfRtInterrupt *107*
SignalRtInterruptThread *107*
socket *130*
socktout *130*
Spider
    application *89*
    debugger *9*
Spider debugger *167*
start_RT_trace *124*
states
    execution *21*
    thread *43*
stop_RT_trace *124*
STOPmgr (INtime Windows STOP Detection sample) *161*
Structured Exception Handling *53*
structures
    COMM *129*
        COMMCONFIG *129*
        COMMPROP *129*
        COMMTIMEOUTS *129*
        COMSTAT *129*
        DCB *129*
    CONTROLBUFFER *119*
    CPUFRAME *119*
    EVENTINFO *119*
    EXCEPTION *119*
    FILETIME *119*
    GENADDR *119*
    HEAPINFO *119*
    HWEXCEPTIONMSG *119*
    INTERRUPTINFO *119*
    INtime *119*

INtimeDotNet *133*
KNTIME *119*
NTXEVENTINFO *119*
NTXPROCATTRIBS *119*
OBJECTDIR *119*
PCIDEV *119*
POOLINFO *119*
PORTINFO *119*
RECEIVEINFO *119*
SECURITY_ATTRIBUTES *119*
SERVICEATTRIBUTES *119*
SERVICEDESC *120*
SYSINFO *120*
THREADACCOUNTING *120*
THREADINFO_SNAPSHOT *120*
THREADSTATE_SNAPSHOT *120*
TRANSACTION *120*
urb *120*
usbClient *120*
usbConfigDescriptor *120*
usbCtrlRequest *120*
usbDeviceDescriptor *120*
usbDeviceId *120*
usbEndPointDescriptor *120*
usbInterface *120*
usbInterfaceDescriptor *120*
StrvAlrm.hlp *153*
support *iii* , *v* , *3* , *153*
support, technical *vi* , *93* , *173*
SuspendRtThread *118*
synchronizing threads *47*
SYSINFO structure *120*
system calls *9* , *50*
global objects *106*
high-level calls
DSM *105*
exception handling *106*
interrupts *107*
mailboxes *108*
memory *109*
message ports *112*
object directories *111*
process management *114*
regions *114*
semaphores *115*
thread management *117*
high-performance gb Ethernet *123*
INscope *124*
INtimeDotNet *131*
low-level calls *115*
mailboxes *108*
scheduler *115*
memory heaps *120*
memory pools *120*
network stack *124*
NTX calls *103*
mailboxes *108*
memory *109*
object directories *111*
semaphores *115*
ntx calls
system data *117*
PCI library *124*
real-time *104*
low-level *104*
Real-time Shared Library (RSL) *125*
registry *125*
Serial Communications (COMM) *128*
structures
COMM *129*
INtime *119*
INtimeDotNet *133*
TCP/IP *130*
types *102*
USB *130*
system data calls *117*
systemcalls
input/output *133*

T

TCP Server *161*
TCP/IP calls *130*
accept *130*
bind *130*
bstring *130*
byteorder *130*
connect *130*
gethostent *130*
gethostname *130*
getnetent *130*
getpeername *130*
getprotoent *130*
getservent *130*
getsockname *130*

getsockopt 130
inet 130
listen 130
recv 130
recvfrom 130
select 130
send 130
sendto 130
sethostname 130
setsockops 130
shutdown 130
socket 130
socktout 130
TCPServ 161
technical support vi , 93 , 153 , 173
TenAsys, contacting vi
thread management calls 117
THREADACCOUNTING structure 120
THREADINFO_SNAPSHOT structure 120
threads 28
about 28
communicating between Windows NT and the RT kernel 4
communication 45
coordination 45
encapsulating Windows NT 19
execution state 21
interrupt 42
message passing 46
multi-threading 39
priority of 21
scheduling
defined 21
priority-based 21
round-robin 23
states 43
switch time 43
synchronizing 47
THREADSTATE_SNAPSHOT structure 120
time, managing 25
top-down programming 40
TRANSACTION structure 120
TransmitCommChar 129
troubleshooting iii , v , 3 , 153

U

UDP sample applications 161
UDP/IP 18
UncatalogRtHandle 111
UninstallRtServiceDescriptor 112
UnregisterRtDependency 105 , 114
UnregisterRtSponsor 105 , 114
UpdateReceiveInfo 128
urb structure 120
USB calls 130
UsbAllocUrb 130
UsbBulkMsg 130
UsbClearHalt 130
UsbConnect 130
UsbControlMsg 130
UsbDisconnect 130
UsbFillBulkUrb 131
UsbFillControlUrb 131
UsbFillintUrb 131
UsbFillIsoUrb 131
UsbFreeUrb 131
UsbGetAsciiString 131
UsbGetConfigDescriptor 131
UsbGetConfiguration 131
UsbGetDescriptor 131
UsbGetDeviceDescriptor 131
UsbGetEndpointCount 131
UsbGetEndpointDescriptor 131
UsbGetInterfaceDescriptor 131
UsbGetLanguageString 131
UsbGetStatus 131
UsbInterruptClose 131
UsbInterruptOpen 131
UsbInterruptRead 131
UsbInterruptWrite 131
UsbKillUrb 131
UsbMatchId 131
UsbSetConfiguration 131
UsbSetInterface 131
UsbSubmitUrb 131
UsbUnlinkUrb 131
USB Client sample 161
UsbAllocUrb 130
UsbBulkMsg 130
UsbClearHalt 130

usbClient structure *120*
usbConfigDescriptor structure *120*
UsbConnect *130*
UsbControlMsg *130*
usbCtrlRequest structure *120*
usbDeviceDescriptor structure *120*
usbDeviceId structure *120*
UsbDisconnect *130*
usbEndPointDescriptor structure *120*
UsbFillBulkUrb *131*
UsbFillControlUrb *131*
UsbFillintUrb *131*
UsbFillIsoUrb *131*
UsbFreeUrb *131*
UsbGetAsciiString *131*
UsbGetConfigDescriptor *131*
UsbGetConfiguration *131*
UsbGetDescriptor *131*
UsbGetDeviceDescriptor *131*
UsbGetEndpointCount *131*
UsbGetEndpointDescriptor *131*
UsbGetInterfaceDescriptor *131*
UsbGetLanguageString *131*
UsbGetStatus *131*
usbInterface structure *120*
usbInterfaceDescriptor structure *120*
UsbInterruptClose *131*
UsbInterruptOpen *131*
UsbInterruptRead *131*
UsbInterruptWrite *131*
UsbKillUrb *131*
UsbMatchId *131*
UsbSetConfiguration *131*
UsbSetInterface *131*
UsbSubmitUrb *131*
UsbUnlinkUrb *131*
Utilities
    INtime Configuration utility *152*

V

validation levels *32*
VerifyAddress *128*

W

WaitCommEvent *129*
WaitForRtControl *115*
WaitForRtInterrupt *107*
WaitForRtSemaphore *116*
Windows NT
    communicating with the RT kernel *4*
    encapsulated as an INtime software thread *19*
    modified HAL *20*
    running INtime applications *4* , *16*
    where to find information *vi*
wizards *8* , *82* , *84*
    RT Device Driver *163*
    RT Process Add-in, MSVC 5.0 *166*
    RT process, MSVC 5.0 *166*
    RT Shared Library *167*
WriteComm *129*