

SCRAMNet GT Network

Software and API Guide

Document No. G-T-ML-G1AP1###-A-0-A5


FOREWORD

The information in this document has been carefully checked and is believed to be accurate; however, no responsibility is assumed for inaccuracies. Curtiss-Wright Controls, Inc., reserves the right to make changes without notice.

Curtiss-Wright Controls, Inc., makes no warranty of any kind with regard to this printed material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

© Copyright 2007 Curtiss-Wright Controls, Inc. All rights reserved.

SCRAMNet[®] is a registered trademark of Curtiss-Wright Controls, Inc., US Patent #4928289

 **LinkXchange**[®] is a registered trademark of Curtiss-Wright Controls, Inc.

Visual C++ is a trademark and a product of Microsoft Corporation.

Windows[®] is a trademark and a product of Microsoft Corporation.

Solaris[®] is a registered trademark of Sun Microsystems, Inc.

Linux[®] is a registered trademark of Linus Torvalds.

All Curtiss-Wright Controls LinkXchange products referred to in this document are protected by one or both of the following U.S. patents 6,751,699 and 5,982,634.

Any reference made within this document to equipment from other vendors does not constitute an endorsement of their product(s).

Revised: August 29, 2007

Curtiss-Wright Controls Embedded Computing

Data Communications
4126 Linden Avenue
Dayton, OH 45432-3068 USA
Tel: (800) 252-5601(U.S. only)
Tel: (937) 252-5601

TABLE OF CONTENTS

1. INTRODUCTION.....	1-1
1.1 How to Use This Manual.....	1-1
1.1.1 Purpose	1-1
1.1.2 Scope	1-1
1.1.3 Style Conventions.....	1-1
1.2 Related Information.....	1-1
1.3 Quality Assurance	1-2
1.4 Technical Support.....	1-3
1.5 Ordering Process	1-3
2. SOFTWARE OVERVIEW	2-1
2.1 Overview	2-1
2.2 Device Driver	2-1
2.3 API Library	2-1
2.4 Utility Applications	2-1
2.5 Software Directory Structure.....	2-1
3. NETWORK DESIGN AND OPERATION	3-1
3.1 Overview	3-1
3.2 Network Topology	3-1
3.3 Node ID.....	3-1
3.4 Accessing GT Memory.....	3-1
3.4.1 Access Methods.....	3-1
3.4.2 Preparing for GT Memory Access	3-2
3.4.3 Notes on Memory Access	3-2
3.4.4 Choosing an Access Method.....	3-2
3.5 Network Interrupts	3-3
4. API DESCRIPTION	4-1
4.1 Overview	4-1
4.2 Structures.....	4-2
4.2.1 scgtDeviceInfo.....	4-2
4.2.2 scgtInterrupt.....	4-3
4.3 API Functions.....	4-5
4.3.1 scgtOpen.....	4-6
4.3.2 scgtClose.....	4-7
4.3.3 scgtMapMem, scgtUnmapMem.....	4-8
4.3.4 scgtRead, scgtWrite	4-9
4.3.5 scgtGetInterrupt	4-11
4.3.6 scgtGetDeviceInfo	4-12
4.3.7 scgtGetState, scgtSetState.....	4-13
4.3.8 scgtGetErrStr	4-15
4.4 Return Codes	4-16
5. UTILITY APPLICATIONS.....	5-1
5.1 Application Overview	5-1
5.1.1 Running SCRAMNet GT Applications Under VxWorks	5-1
5.2 Application Descriptions	5-2
5.2.1 gtmon – Device Monitor and Configuration Tool	5-2
5.2.2 gttp – Network Throughput Graphing Tool	5-2
5.2.3 gtnex – Network Exerciser	5-2
5.2.4 gtmem – Memory/Register Access Utility.....	5-3
5.2.5 gtint – Network Interrupt Utility	5-3
5.2.6 gtprog – Firmware Programmer.....	5-3

APPENDICES

APPENDIX A PRIMITIVES A-1

APPENDIX B INSTALLATION..... B-1

APPENDIX C UTILITY HELP C-1

1. INTRODUCTION

1.1 How to Use This Manual

1.1.1 Purpose

This manual describes how to use the SCRAMNet GT software. The SCRAMNet GT Application Program Interface (API), utility applications, and software installation procedures are discussed.

1.1.2 Scope

This manual is intended for system and software developers who want to call the API Library routines in applications programming.

Operating-system-specific installation information is included in appendix B.

1.1.3 Style Conventions

- Called functions are italicized. For example, *OpenConnect()*.
- Data types are italicized. For example, *int*.
- Function parameters in textual references are bolded. For example, **Action**.
- Path names are italicized. For example, *utility/sw/cfg*.
- File names are bolded. For example, **config.c**.
- Path file names are italicized and bolded. For example, ***utility/sw/cfg/config.c***.
- Hexadecimal values are written with a “0x” prefix. For example, 0x7e.
- Code and monitor screen displays of input and output are boxed and indented on a separate line. Text that represents user input is bolded. Text that the computer displays on the screen is not bolded. For example:

```
ls
file1          file2          file3
```

- Large samples of code are Courier font, at least one size less than context, and are usually on a separate page or in an appendix.

1.2 Related Information

- *SCRAMNet GT Hardware Reference Manual for PCI and PMC cards*
- *LinkXchange GLX4000 Physical Layer Switch User Reference Manual (Doc. No F-T-MR-L5XL144) Curtiss-Wright Controls, Inc.*
- Curtiss-Wright Controls, Inc. www.cwcembedded.com

1.3 Quality Assurance

Curtiss-Wright Controls, policy is to provide our customers with the highest quality products and services. In addition to the physical product, the company provides documentation, sales and marketing support, hardware and software technical support, and timely product delivery. Our quality commitment begins with product concept, and continues after receipt of the purchased product.

Curtiss-Wright Controls' Quality System conforms to the ISO 9001 international standard for quality systems. ISO 9001 is the model for quality assurance in design, development, production, installation, and servicing. The ISO 9001 standard addresses all 20 clauses of the ISO quality system, and is the most comprehensive of the conformance standards.

Our Quality System addresses the following basic objectives:

- Achieve, maintain, and continually improve the quality of our products through established design, test, and production procedures.
- Improve the quality of our operations to meet the needs of our customers, suppliers, and other stakeholders.
- Provide our employees with the tools and overall work environment to fulfill, maintain, and improve product and service quality.
- Ensure our customer and other stakeholders that only the highest quality product or service will be delivered.

The British Standards Institution (BSI), the world's largest and most respected standardization authority, assessed Curtiss-Wright Controls' Quality System. BSI's Quality Assurance division certified we meet or exceed all applicable international standards, and issued Certificate of Registration, number FM 31468, on May 16, 1995. The scope of Curtiss-Wright Controls' registration is: "Design, manufacture and service of high technology hardware and software computer communications products." The registration is maintained under BSI QA's bi-annual quality audit program.

Customer feedback is integral to our quality and reliability program. We encourage customers to contact us with questions, suggestions, or comments regarding any of our products or services. We guarantee professional and quick responses to your questions, comments, or problems.

1.4 Technical Support

Technical documentation is provided with all of our products. This documentation describes the technology, its performance characteristics, and includes some typical applications. It also includes comprehensive support information, designed to answer any technical questions that might arise concerning the use of this product. We also publish and distribute technical briefs and application notes that cover a wide assortment of topics. Although we try to tailor the applications to real scenarios, not all possible circumstances are covered.

Although we have attempted to make this document comprehensive, you may have specific problems or issues this document does not satisfactorily cover. Our goal is to offer a combination of products and services that provide complete, easy-to-use solutions for your application.

If you have any technical or non-technical questions or comments, contact us. Hours of operation are from 8:00 a.m. to 5:00 p.m. Eastern Standard/Daylight Time.

- Phone: (937) 252-5601 or (800) 252-5601
- E-mail: DTN_support@curtisswright.com
- Fax: (937) 252-1465
- World Wide Web address: www.cwcembedded.com

1.5 Ordering Process

To learn more about Curtiss-Wright Controls, Inc., products or to place an order, please use the following contact information. Hours of operation are from 8:00 a.m. to 5:00 p.m. Eastern Standard/Daylight Time.

- Phone: (937) 252-5601 or (800) 252-5601
- E-mail: DTN_info@curtisswright.com
- World Wide Web address: www.cwcembedded.com

This page intentionally left blank

2. SOFTWARE OVERVIEW

2.1 Overview

The SCRAMNet GT software is designed to facilitate application software development. It is an easy-to-use interface that abstracts the complex details of device drivers, DMA transactions etc., and allows you to write effective software with minimal complication. The software consists of a device driver, an API library, and a set of utility applications. See Appendix B for the software [installation procedure](#) for your OS.

2.2 Device Driver

The GT device driver performs operations on the GT hardware in response to requests from the API library and the hardware itself. There is a separate device driver for each platform supported by the GT software.

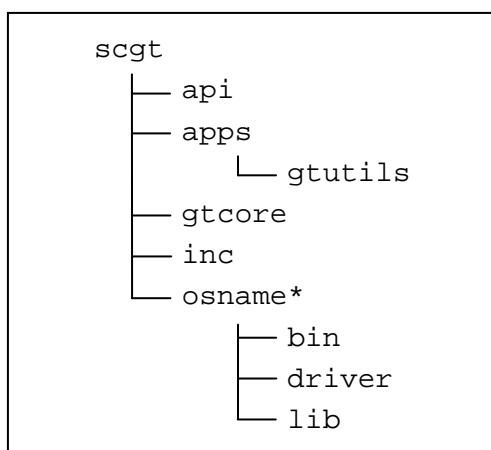
2.3 API Library

The API library is the interface applications use to communicate with the device driver. It is a simple platform independent interface that covers all the features supported by the hardware. The full set of [API functions](#) is described in Chapter 4.

2.4 Utility Applications

A set of utility applications is distributed with the GT software. These applications serve both as useful utilities and as examples of how to write software using SCRAMNet GT.

2.5 Software Directory Structure



- *api* directory contains API source code
- *gtutils* directory contains source code for SCRAMNet GT utilities.
- *gtcore* directory contains core driver files.
- *inc* directory contains API header files.
- *osname** directory is the operating-system-specific directory.
- *bin* directory contains binaries for the utilities.
- *driver* directory contains the driver(s), and where applicable, the *gtload* and *gtunload* scripts.
- *lib* directory contains the built API library.

* Replace the name “osname” with your specific operating system.

This Page Intentional Left Blank

3. NETWORK DESIGN AND OPERATION

3.1 Overview

This chapter provides an overview of basic SCRAMNet GT concepts and system design.

3.2 Network Topology

The standard GT network topology is a ring topology where nodes are connected forming a continuous loop. This can be done directly or through a switch solution like the LinkXchange™ GLX4000. Each node in the ring must have a unique node ID. Other topologies are possible by simply connecting the fiber in a different way. The **gtmon** application may be used to view the connection status between nodes with the '-N' option.

3.3 Node ID

Each node on the GT network has an associated node ID. When a node transmits a native data packet, it sends its node ID as part of the packet header. When a node receives a data packet, it compares its own node ID with the node ID in the received packet header. If the node IDs match, the packet is identified as a native packet and is not retransmitted. If they do not match, the packet is identified as a foreign packet and is retransmitted without modification. This packet removal mechanism prevents traffic from circulating around the ring more than once.

It follows from above that if two nodes on a ring share the same node ID, they will remove non-native packets from the ring.

Valid node IDs range from 0 to 255. However, node ID 0 is used by default when boards are first powered-on and its use is not recommended.

A device's node ID may be assigned using the **gtmon** application or by calling the *scgtSetState()* function. Execute the **gtmon** application from a system-startup script or batch file after the driver is loaded to set the node ID.

3.4 Accessing GT Memory

3.4.1 Access Methods

Management of data in GT memory can be accomplished using either of two transfer methods. These methods are Programmed Input Output (PIO) and Direct Memory Access (DMA).

The PIO transfer method allows GT memory to be written and read as though it is system memory. During PIO transfers, the CPU (or other PCI master) initiates the movement of data over the PCI bus, between host memory (or other PCI-visible memory) and GT memory. This means that a program can read or write GT memory simply by executing a CPU instruction that dereferences an address within the GT memory address space.

The DMA transfer method allows GT memory to be written and read in contiguous blocks, and produces results equivalent to a memory copy. During DMA transfers, the GT hardware initiates the movement of data over the PCI bus between host memory (or other PCI-visible memory) and GT memory. The CPU is available to perform other system tasks.

3.4.2 Preparing for GT Memory Access

Accessing GT memory requires several steps. The program must call the function *scgtOpen()*, which verifies the existence of the GT device and initializes a handle to the device. If a PIO operation is desired, call *scgtMapMem()* to map GT memory into the program's address space. Next, call the function *scgtGetDeviceInfo()*, which returns static information about the GT device in an *scgtDeviceInfo* data structure.

The relevant information contained in the **scgtDeviceInfo** structure is the amount of mapped memory (**mappedMemSize**) for PIO. Check this for validity. If the base PIO address and the mapped memory size are non-zero, then the valid address range for PIO access to GT memory is [**memPtr**, **memPtr** + **mappedMemSize** - 1]. Operations such as `((int*)memPtr)[41] = ((int*)memPtr)[40] + 5;` can now be performed. PIO transfer mode is also available using the API functions *scgtWrite()* and *scgtRead()*.

For DMA, the relevant information contained in the **scgtDeviceInfo** structure is the amount of available/populated GT memory (**popMemSize**). The valid offset range for use in DMA transfers is [0, **popMemSize** - 4]. DMA transfers are performed in 32 or 64-bit quantities, so GT memory offsets and user buffer addresses must be at least 32-bit (4-byte) aligned. Some host systems may require stricter alignment for user buffers, for example, page alignment. DMA transfers can be performed using the API functions *scgtWrite()* and *scgtRead()*.

3.4.3 Notes on Memory Access

The base PIO address as seen by a program, though non-zero, is in fact a pointer to offset zero of GT memory. PIO access to GT memory at address “base address + 40” is equivalent to DMA access at offset 40.

Dependent upon available host system resources, it may be impossible to map all of the GT device's populated memory for PIO access. For this reason, it is important to use addresses within the valid range provided by the API. Access to the remainder of the populated memory is achievable through DMA access with the on-board DMA controller.

3.4.4 Choosing an Access Method

Identical operations on GT memory can be achieved with PIO or DMA. Although it is still important to select the method that meets application performance demands. Each transfer method has its own strengths. However, it is not necessary for a program or system to use only one transfer method, as the GT hardware supports simultaneous PIO and DMA data transfers.

Characteristics of PIO access:

- CPU required to perform data transfer
- No API and driver overhead
- One byte transfer granularity
- Minus-one-copy data transfer (*)

Characteristics of DMA access:

- GT hardware performs data transfer
- Frees CPU for other system tasks
- API and driver overhead
- Four byte transfer granularity
- Zero-copy data transfer (**)

In general, system throughput will be maximized when data is transferred in large blocks using DMA. As transfer size decreases, driver overhead in setting up DMA has an increasing impact on throughput. Therefore, applications that need to transfer small blocks of data may experience higher throughput using PIO, thereby eliminating the DMA setup time. In any case, a design will benefit from some analysis.

The **gttp** application is provided to assist in such analysis. The **gttp** throughput graphing option(s) -GA, can be used to collect performance numbers for varying transfer sizes. These numbers provide a good starting point for evaluating system performance. Factors including system load, PCI bus performance, memory speed, and GT network load can have considerable impact on overall performance. Analysis of performance under expected operating conditions is recommended.

- * The term minus-one-copy refers to the capability of transferring data to/from the device without also having the data in system memory. This is achieved with PIO by using device memory locations as the operands in computations.
- ** The term zero-copy refers to the capability of performing DMA transfers to the device without first copying the user's buffer into driver/kernel memory buffers. Technically speaking, one copy still occurs between the user's buffer and device memory.

3.5 Network Interrupts

Network interrupts are messages that are passed across the network that can be used for event notification and process synchronization. For example, let A and B be two nodes that are both working on the same region of GT memory. Node A updates the region after some processing and needs to tell Node B it has been updated. Node A can send Node B a network interrupt to indicate the buffer has been updated.

There are two types of network interrupts, broadcast and unicast. Every node on the network can receive Broadcast interrupts. Unicast interrupts are sent to (and received by) only one node. Both types of interrupts include a user-defined 32-bit value specified by the sender.

There are 32 broadcast interrupts designated with an ID in the range [0, 31]. Each node can choose which of the 32 broadcast interrupts it would like to receive by setting the broadcast interrupt mask using the [scgtSetState\(\)](#) function or using **gtmon**.

Unlike broadcast interrupts, unicast interrupts are targeted at a specific node based on the node ID. The reception of unicast interrupts can be enabled or disabled by setting the unicast interrupt mask using the [scgtSetState\(\)](#) function or using **gtmon**. The sender of the interrupt specifies the node to which the interrupt will be sent.

The system designer defines the meaning of the network interrupts and their associated 32-bit value.

The [scgtWrite\(\)](#) function is used to send network interrupts and [scgtGetInterrupt\(\)](#) is used to receive them.

This page intentionally left blank

4. API DESCRIPTION

4.1 Overview

This chapter describes the SCRAMNet GT API in detail. First, the structures defined by the API are described in section 4.2. Then the API functions and their arguments are described in detail in section 4.3. Lastly, possible return codes of the API functions are described in section 4.4. Note that base data types (for example, int32) are used throughout the API. This ensures compatibility across platforms and compilers. See Appendix A for a detailed description of these base types.

4.2 Structures

Structures are used to retrieve the status of the driver and to send/receive network interrupts. These structures are described here.

4.2.1 *scgtDeviceInfo*

The type *scgtDeviceInfo* returns information about the device. The definition of the device info structure is located in *gtcore/gtucore.h*.

```
typedef struct _scgtDeviceInfo
{
    char driverRevisionStr[128];
    char boardLocationStr[128];
    uint32 unitNum;
    uint32 popMemSize;
    uint32 mappedMemSize;
    uint32 numLinks;
    uint32 revisionID;
} scgtDeviceInfo;
```

MEMBERS:

driverRevisionStr:.....	NULL-terminated string giving revision information of the GT driver currently running.
boardLocationStr:	NULL-terminated string describing the physical location of the device.
unitNum:	Unit number of the device.
popMemSize:	Memory size in bytes that is populated on the device.
mappedMemSize:	Memory size in bytes that is available for PIO access. Under rare circumstances, where the platform cannot map the amount of memory that is populated on the device, this can be less than the total populated memory size.
numLinks:	The number of links supported by the device.
revisionID:	The firmware revision ID.

4.2.2 scgtInterrupt

The definition of the interrupt structure is located in *gtcore/gtucore.h*.

```
typedef struct _scgtInterrupt
{
    uint32 type;
    uint32 sourceNodeID;
    uint32 id;
    uint32 val;
} scgtInterrupt;
```

MEMBERS:

type: SCGT_UNICAST_INTR for unicast interrupt, SCGT_BROADCAST_INTR for broadcast interrupt, or SCGT_ERROR_INTR for error interrupt.

sourceNodeID: When receiving interrupts, this member is set by the driver to the node ID that sent the interrupt. This member is not used when sending an interrupt, and is never valid for error interrupts.

id: When sending or receiving broadcast interrupts, this is the interrupt ID (in the range [0,31]) of the interrupt. When sending a unicast interrupt, set this member to the destination node ID. This member is not valid when receiving unicast or error interrupts.

val: The value of the user-defined data sent or received with a broadcast or unicast interrupt. If an error interrupt is received, this member holds the error code associated with the error. See the table of possible error-interrupt values below.

Sending Interrupts

Unicast:

- Set **type** to SCGT_UNICAST_INTR
- Set **id** to the desired destination node ID
- Set **val** to the desired 32-bit value

Broadcast:

- Set **type** to SCGT_BROADCAST_INTR
- Set **id** to the desired interrupt ID (in the range [0,31])
- Set **val** to the desired 32-bit value

Receiving Interrupts

Unicast:

- **type** is set to SCGT_UNICAST_INTR
- **sourceNodeID** is set to the node ID that sent the interrupt
- **val** contains the 32-bit user defined value
- **id** does not contain valid data

Broadcast:

- **type** is set to SCGT_BROADCAST_INTR
- **sourceNodeID** is set to the node ID that sent the interrupt
- **id** contains the interrupt ID (in the range [0,31])
- **val** contains the 32-bit user defined value

Error:

- **type** is set to SCGT_ERROR_INTR
- **val** contains the error value. See the table below for possible values
- **sourceNodeID** and **id** do not contain valid data

Possible error-interrupt values:

SCGT_LINK_ERROR	Link error occurred.
SCGT_DRIVER_MISSED_INTERRUPTS	Driver interrupt queue is overrun because the card is receiving interrupts faster than can be processed.

These values are defined in *gtcore/gtucore.h*.

4.3 API Functions

The SCRAMNet GT API contains a variety of functions. They allow you to write and read data from a buffer into SCRAMNet GT memory, configure the driver, and get driver information in a variety of ways. The function prototypes are found in the file **scgtapi.h** (located in the *inc* directory) and detailed below. All functions return SCGT_SUCCESS upon successful completion unless otherwise specified. For a complete description of possible return codes see Section 4.4.

Table 4-1 API Functions

Function	Description	Page
<i>scgtOpen</i>	Returns a handle (through the pHandle variable) to a specific adapter that is then passed to all other API calls that will operate on that device.	4-6
<i>scgtClose</i>	Closes a handle to a device.	4-7
<i>scgtMapMem</i>	Map GT memory for PIO	4-8
<i>scgtUnmapMem</i>	Unmap GT memory	4-8
<i>scgtRead</i>	Read data from GT memory.	4-9
<i>scgtWrite</i>	Write data to GT memory.	4-9
<i>scgtGetInterrupt</i>	Retrieve network and error interrupts from the driver.	4-11
<i>scgtGetDeviceInfo</i>	Retrieves information about the device.	4-12
<i>scgtGetState</i>	Get the value of a specified configuration parameter.	4-13
<i>scgtSetState</i>	Set the value of a specified configuration parameter.	4-13
<i>scgtGetErrStr</i>	Returns a string describing a specific error code.	4-15

In addition to these functions, the API provides a global variable **scgtapiRevisionStr** that points to a NULL-terminated string containing API revision information.

4.3.1 scgtOpen

FUNCTION PROTOTYPE:

*uint32 scgtOpen (uint32 unitNum, scgtHandle *pHandle);*

DESCRIPTION:

Initializes a handle to a specific GT device. Pass this handle to all other API calls to operate on the device.

INPUT:

unitNum: Unit number of the device to open. Valid unit numbers range from 0 to 15. Use **gtmon** with the **'-a'** option to see the available unit numbers.

OUTPUT:

pHandle: Pointer to a **scgtHandle** variable. After successful completion ***pHandle** will contain a handle to a GT device.

4.3.2 scgtClose

FUNCTION PROTOTYPE:

*uint32 scgtClose (scgtHandle *pHandle);*

DESCRIPTION:

Closes a handle to a device.

INPUT:

pHandle: Pointer to a device handle.



WARNING: The handle becomes invalid once closed. If the handle is used after it has been closed, the results are undefined.

4.3.3 *scgtMapMem*, *scgtUnmapMem*

FUNCTION PROTOTYPE:

```
void *scgtMapMem          void scgtUnmapMem
      (scgtHandle *pHandle);      (scgtHandle *pHandle);
```

DESCRIPTION:

The *scgtMapMem()* function maps GT memory into the program's address space. Upon successful completion, *scgtMapMem()* returns a pointer to GT memory. A NULL pointer is returned otherwise.

The *scgtUnmapMem()* function frees the resources used by the *scgtMapMem()* function.

INPUT:

pHandle:Pointer to a device handle.

4.3.4 scgtRead, scgtWrite

FUNCTION PROTOTYPE:

```
uint32 scgtRead(scgtHandle *pHandle,      uint32 scgtWrite(scgtHandle *pHandle
                uint32 gtMemoryOffset,    uint32 gtMemoryOffset,
                void *pDataBuffer,        void *pDataBuffer,
                uint32 bytesToTransfer,    uint32 bytesToTransfer,
                uint32 flags,              uint32 flags,
                uint32 *pBytesTransferred, uint32 *pBytesTransferred,
                                           scgtInterrupt *pInterrupt);
```

DESCRIPTION:

The *scgtRead()* and *scgtWrite()* functions performs read and write DMA or PIO transfers to and from GT memory. The *scgtRead()* function reads data from GT memory into the data buffer. The *scgtWrite()* function writes data from the data buffer to GT memory and then sends a network interrupt if one is specified. The *scgtRead()* and *scgtWrite()* functions perform 32-bit sized transfers at 32-bit aligned GT memory offsets. For memory transfers smaller than 32 bits, use a pointer to GT memory provided by *scgtMapMem()*.



NOTE: The current GT firmware (version 2.25) does not support sending an interrupt via DMA. As a result, the *pInterrupt* part of a single *scgtWrite* is disregarded when used to send a network interrupt. To send a network interrupt, enter the *scgtWrite* with any byte transfer size and with or without a *pInterrupt* value (it will be ignored), followed by a second *scgtWrite* with a zero byte transfer size and the desired *pInterrupt* value.

INPUT:

pHandle: Pointer to a device handle.

gtMemoryOffset: Byte offset of GT memory to start reading from or writing to. This value is rounded down to the nearest multiple of 4.

pDataBuffer: Pointer to a user buffer. Set to NULL when using *scgtWrite()* to send a network interrupt without writing data.

bytesToTransfer: Number of bytes to transfer. This value is rounded down to the nearest multiple of 4.

flags: Bit vector with the following bits defined:

 SCGT_RW_PIO: PIO operation. All other flags are invalid when SCGT_RW_PIO is set. If SCGT_RW_PIO is not set, this call performs a DMA transfer.



NOTE: Starting offsets for DMA reads and writes must be on 64-bit boundaries.

SCGT_RW_DMA_BYTE_SWAP: ...Invert the SCGT_BYTE_SWAP_ENABLE state for this DMA transfer. Only affects this transfer.

SCGT_RW_DMA_WORD_SWAP: ...Invert the SCGT_WORD_SWAP_ENABLE state for this DMA transfer. Only affects this transfer.

SCGT_RW_DMA_PHYS_ADDR: ...Indicates that the address given by **pDataBuffer** is a PCI physical address rather than a virtual address.

pInterrupt: Pointer to an interrupt structure that defines a network interrupt to send immediately after the write completes. No interrupt is sent if pInterrupt is NULL.



WARNING: Use caution when using PCI physical addresses with these calls. Using SCGT_RW_PHYS_ADDR with an incorrect PCI physical address will result in corrupted memory and may crash the system.

OUTPUT:

*pBytesTransferred: Contains the number of bytes transferred. This value is always a multiple of 4.

4.3.5 scgtGetInterrupt

FUNCTION PROTOTYPE:

```
uint32 scgtGetInterrupt(scgtHandle *pHandle,
                       scgtIntrHandle *intrHandle,
                       scgtInterrupt *interruptBuffer,
                       uint32 numInterrupts,
                       uint32 timeout,
                       uint32 *numInterruptsRet);
```

DESCRIPTION:

Retrieves one or more unicasts, broadcasts, or error interrupts and places them in the buffer pointed to by **interruptBuffer**. If no interrupts are available and none have occurred within the specified timeout time, *scgtGetInterrupt()* returns with SCGT_TIMEOUT.

Also see SCGT_UNICAST_INT_MASK, SCGT_BROADCAST_INT_MASK, and SCGT_INT_SELF_ENABLE in section 4.3.7.

INPUT:

pHandle: Pointer to a device handle.

intrHandle: Handle used by the driver to mark which interrupts the application has already received. You must initialize the value pointed to by intrHandle to -1 before calling *scgtGetInterrupt()* for the first time. By setting the intrHandle to -1, you will receive only new interrupts that take place starting at the time of the call to *scgtGetInterrupt()*.

numInterrupts: The maximum number of interrupts to get.

timeout: Maximum time to wait for an interrupt specified in milliseconds. Specify a timeout value of 0 to return immediately.

OUTPUT:

interruptBuffer: Pointer to an array of scgtInterrupt structures.

numInterruptsRet: Contains the number of interrupts written to interruptBuffer.

4.3.6 scgtGetDeviceInfo

FUNCTION PROTOTYPE:

```
uint32 scgtGetDeviceInfo (scgtHandle *pHandle,  
                          scgtDeviceInfo *pDeviceInfo);
```

DESCRIPTION:

Retrieves information about a SCRAMNet GT device.

INPUT:

pHandle: Pointer to a device handle.

OUTPUT:

pDeviceInfo: Pointer to a SCRAMNet GT device info structure. The structure is updated with the status of the device.

4.3.7 scgtGetState, scgtSetState

FUNCTION PROTOTYPE:

```
uint32 scgtGetState(scgtHandle,
                   *pHandle, uint32 stateID);
```

```
uint32 scgtSetState(scgtHandle
                   *pHandle, uint32 stateID,
                   uint32 val);
```

DESCRIPTION:

The *scgtGetState()* function returns the current state of a driver resource identified by the value of **stateID**. The *scgtSetState()* function sets the current state of a driver resource identified by the value of **stateID**. See Table 4-2 State ID to determine which driver resources are settable.

INPUT:

pHandle: Pointer to a device handle.

stateID: State identifier. Can be one of the following values.

Table 4-2 State ID

stateID	Settable	Description
SCGT_NODE_ID	Yes	Node ID of unit.
SCGT_ACTIVE_LINK	Yes	Unit's active link. For boards with multiple links (link 0 and 1), you may get/set the active link.
SCGT_NUM_LINK_ERRS		Number of link errors.
SCGT_EWRAP	Yes	Set to '1' to enable electronic wrap and bypass the link. Set to '0' to disable.
SCGT_WRITE_ME_LAST	Yes	Set to '1' to specify that writes to shared memory will occur only after corresponding native packets traverse the ring. Set to '0' to specify normal operation, where writes to shared memory will not rely on the reception of native packets.
SCGT_UNICAST_INT_MASK	Yes	Set to '1' to enable receives of Unicast Interrupts. Set to '0' to disable.
SCGT_BROADCAST_INT_MASK	Yes	Bit vector describing which of the 32 Broadcast Interrupts to receive. Bit 0 corresponds to broadcast interrupt 0, and bit 31 corresponds to broadcast interrupt 31. Set the corresponding bit to '1' to receive the broadcast interrupt, set to '0' otherwise. For example, to receive interrupt 16 set the mask as follows: mask = mask (1 << 16);
SCGT_INT_SELF_ENABLE	Yes	Set to '1' to enable reception of network interrupts sent by this node. Set to '0' to disable.
SCGT_RING_SIZE		The number of nodes on the ring.
SCGT_UPSTREAM_NODE_ID		Node ID of the node upstream to this node.
SCGT_NET_TIMER_VAL		Timer that is incremented on every network transmit clock period. For 2.5 Gbps configurations, the network transmit clock period is 8 ns. See the hardware manual for additional information.
SCGT_LATENCY_TIMER_VAL		Timer that measures transit latency of the network.

stateID	Settable	Description
SCGT_SM_TRAFFIC_CNT		Number of 32-bit shared memory data phases.
SCGT_SPY_SM_TRAFFIC_CNT	Yes	Number of 32-bit data phases transmitted by node SCGT_SPY_NODE_ID.
SCGT_SPY_NODE_ID	Yes	The node ID for which 32-bit data phases will be counted by SCGT_SPY_SM_TRAFFIC_CNT.
SCGT_TRANSMIT_ENABLE	Yes	Set to '1' to enable native packet transmissions. Set to '0' to disable. Does not affect retransmission of the network packets.
SCGT_RECEIVE_ENABLE	Yes	Set to '1' to enable network packet reception. Set to '0' to disable. This includes data packets and network interrupt packets. Does not affect retransmission of the network packets.
SCGT_RETRANSMIT_ENABLE	Yes	Set to '1' to enable retransmission of foreign network traffic. Set to '0' to disable.
SCGT_LASER_0_ENABLE	Yes	Set to '1' to enable laser. Set to '0' to disable.
SCGT_LASER_1_ENABLE	Yes	Set to '1' to enable laser. Set to '0' to disable.
SCGT_LINK_UP		Link up status for active link. '1' if link up, '0' if down.
SCGT_LASER_0_SIGNAL_DET		Signal detect state for laser 0: '1' if signal detected, '0' if no signal detected.
SCGT_LASER_1_SIGNAL_DET		Signal detect state for laser 1: '1' if signal detected, '0' if no signal detected.
SCGT_D64_ENABLE	Yes	Set to '1' to enable 64-bit PCI data transfers by the GT DMA engine. Set to '0' to allow only 32-bit PCI data transfers by the GT DMA engine.
SCGT_BYTE_SWAP_ENABLE	Yes	Set to '1' to enable byte swapping on accesses to GT memory. Set to '0' to disable.
SCGT_WORD_SWAP_ENABLE	Yes	Set to '1' to enable swapping of 32-bit words on 64-bit accesses to GT memory. Set to '0' to disable.



NOTE: Traffic on the network is transmitted in 32-bit data phases. When performing byte-sized PIO, single byte transfers will be sent using 32-bit data phases. As a result, the traffic counters can only be interpreted as a word count for 32-bit or 64-bit transactions.

val: The *scgtSetState()* function sets the state of the driver resource to this value.

4.3.8 scgtGetErrStr

FUNCTION PROTOTYPE

*char * scgtGetErrStr(uint32 retcode);*

DESCRIPTION

Returns a NULL-terminated string describing the return code of a SCRAMNet GT API call. Returns the string “UNKNOWN ERROR” if the return code is not valid.

INPUT:

retcode:.....Return code of the error to describe.

4.4 Return Codes

Each function in the SCRAMNet GT API returns SCGT_SUCCESS or an error code unless otherwise specified. The possible return codes are listed in Table 4-3. They are defined in *gtcore/gtucore.h*.

Table 4-3 Return Codes

Name	Description
SCGT_SUCCESS	Successful completion.
SCGT_SYSTEM_ERROR	System error.
SCGT_BAD_PARAMETER	An invalid parameter was received by an API call. An invalid parameter can be a value outside the specified range or a NULL pointer.
SCGT_DRIVER_ERROR	The driver encountered an internal error.
SCGT_TIMEOUT	The time-out expired before the call completed.
SCGT_CALL_UNSUPPORTED	The requested call was not supported.
SCGT_INSUFFICIENT_RESOURCES	Resources not available to complete a call.
SCGT_MISSED_INTERRUPTS	Application missed network interrupts. This can happen when the driver is receiving interrupts faster than the application can process them.
SCGT_DMA_UNSUPPORTED	DMA transfers are unsupported in some preliminary versions of the SCRAMNet GT product line. Contact Curtiss-Wright Controls, Inc. technical support for more information.

5. UTILITY APPLICATIONS

5.1 Application Overview

The SCRAMNet GT software comes with a variety of SCRAMNet GT utility applications. These samples are provided to assist in verifying the card's functionality, to assist in application development, and to provide examples using the SCRAMNet GT API.



CAUTION: Some of these applications will modify memory and should be used with caution on production systems.

5.1.1 Running SCRAMNet GT Applications Under VxWorks

The utility applications and API library binaries are compiled on a per-CPU basis, rather than a per-computer basis. Computers with PowerPC 604 compatible processors, for example, can run the binaries in directory *vxworks/bin_ppc604*.

Prior to executing SCRAMNet GT applications under VxWorks, the driver must be loaded (see Appendix B, Section B.5.3) followed by the loading of the SCRAMNet GT API binary:

```
ld < scgtapi.o
```

Next load the application binary. For example:

```
ld < gtmon.o
```



NOTE: Utility application binaries are also provided in an all-in-one binary named **gtutils.o**, intended for easy compilation into the VxWorks kernel. The API library is not included in the **gtutils.o** binary, and must be included separately.

Execute the application enclosing any parameters inside quotes. For example:

```
gtmon "-u 0"
```



NOTE: Quotes are required around GT application parameters on VxWorks only. Further examples and descriptions will not use quotes.

The application and API binaries can be unloaded using the **unld** command. This will fully remove the binaries from system memory. For example,

```
unld "gtmon.o"  
unld "scgtapi.o"
```

5.2 Application Descriptions

Application options are documented within each application's help menus. Running an application without options will display a condensed help menu. Use option '-h' for descriptive help with options. Use option '-h 1' for extensive application usage and options information. For example:

```
gtmon -h
```

```
gttp -h 1
```

See Appendix C for a listing of the output.

5.2.1 gtmon – Device Monitor and Configuration Tool

Use the **gtmon** tool to monitor device status or modify driver and device configuration settings. For example:

- Change the node identification number
- Determine the device status and configuration
- Determine the number of nodes on the loop
- Display performance statistics
- Display network topology

While **gtmon** is a valuable stand-alone tool, it is also a powerful tool for use in shell scripting.

5.2.2 gttp – Network Throughput Graphing Tool

The **gttp** tool is used to test device performance in a computer system. Though GT devices are capable of transmitting data at their documented maximum data rate, many factors can reduce the total system performance. Some of these factors are bus speed, CPU frequency, and memory speed. For example:

- Graph network throughput versus data packet size
- Determine optimal packet sizes for use in new applications
- Generate network loads during testing

The **gttp** tool has options for transferring data with DMA or PIO, and for transferring with multiple simultaneous threads of execution. It is the perfect tool for diagnosing and optimizing your system performance.

5.2.3 gtnex – Network Exerciser

The **gtnex** application is a generator and verifier of network traffic. This application exercises multiple nodes on the network simultaneously. By using pre-defined GT memory regions for control information, **gtnex** is able to detect other nodes that are running **gtnex** and can display a summary of statistics from all active **gtnex** nodes. For example:

- Evaluate network stability
- Verify network functionality (including data verification)
- Perform a quick memory test

5.2.4 gtmem – Memory/Register Access Utility

The **gtmem** utility provides access to device memory and device registers. For example:

- Display and modify device memory/registers
- Monitor changes in device memory/registers
- And search for data values in device memory/registers

The **gtmem** utility has options for specifying the memory/register offset and length for its operation. Operations such as filling memory with a value are easily performed. Like **gtmon**, **gtmem** is also a powerful tool for use in shell scripting.

5.2.5 gtint – Network Interrupt Utility

The **gtint** application generates and monitors network interrupts. For example:

- Send unicast or broadcast network interrupts
- Monitor the network interrupts received by the device based on the current interrupt mask

5.2.6 gtprog – Firmware Programmer

The **gtprog** utility permits updating the firmware in GT devices in the field.



CAUTION: Proper precautions should be taken to avoid power-failure during programming. Should programming fail for any reason, do the following:

- DO NOT TURN OFF THE COMPUTER. Devices with invalid or incomplete firmware may not function after power-cycle and may require service.
- Verify that you are using the proper firmware-programming file.
- Attempt to program the firmware again. If programming fails, try different firmware versions.

Contact Curtiss-Wright Controls, Inc. Technical Support if problems persist. Contact information can be found in your product manual or on the internet at www.cwcembedded.com.

A CRC for the firmware update file can be obtained with the following options:

```
gtprog -f <firmwarefile.gfw>
```

Once you have verified that the **.gfw** file is the correct file to use for programming, proceed with programming by using the '-B' and '-u' options to **gtprog**:

```
gtprog -f <firmwarefile.gfw> -B -u <unit_number>
```

The status of the firmware update is displayed as **gtprog** processes the file. Once complete, power-cycle the computer for the update to take effect.

This page intentionally left blank

APPENDIX A

PRIMITIVES

TABLE OF CONTENTS

A.1 Basic Data Types	A-1
----------------------------	-----

A.1 Basic Data Types

The SCRAMNet GT API defines a set of base data types that are used throughout the API. This ensures compatibility across platforms and compilers. These data types are described below.

int64	The type <i>int64</i> is a signed 64-bit integer.
uint64	The type <i>uint64</i> is an unsigned 64-bit integer.
int32	The type <i>int32</i> is a signed 32-bit integer.
uint32	The type <i>uint32</i> is an unsigned 32-bit integer.
int16	The type <i>int16</i> is a signed 16-bit integer.
uint16	The type <i>uint16</i> is an unsigned 16-bit integer.
int8	The type <i>int8</i> is a signed 8-bit integer.
uint8	The type <i>uint8</i> is an unsigned 8-bit integer.
scgtHandle	The type <i>scgtHandle</i> is an abstraction of the file descriptor in UNIX, the <i>HANDLE</i> in Windows, and other similar descriptors used by other operating systems. The SCRAMNet GT API uses a <i>scgtHandle</i> to determine on which device the request should be executed. Consult the example applications for more information on using the type <i>scgtHandle</i> .

This page intentionally left blank

APPENDIX B

INSTALLATION

TABLE OF CONTENTS

B.1 Overview	B-1
B.2 Install the Hardware	B-1
B.3 Windows Software Installation	B-2
B.3.1 Install the Device Driver.....	B-2
B.3.2 Install the API Library and Utility Applications.....	B-5
B.3.3 Rebuilding Applications	B-5
B.3.4 Linking with the API library	B-6
B.4 UNIX-like OS Software Installation	B-6
B.4.1 Install the Software Onto the Host Computer (UNIX).....	B-6
B.4.2 Loading/Unloading SCRAMNet GT Device Driver.....	B-7
B.4.3 Rebuilding Applications	B-7
B.5 VxWorks Software Installation	B-8
B.5.1 Solaris Host Loading Procedure	B-8
B.5.2 Microsoft Windows Host Loading Procedure.....	B-8
B.5.3 Loading the Driver Module	B-8
B.5.4 Ensure proper memory mappings	B-8
B.5.5 Installing the GT Device Driver	B-8
B.5.6 Uninstalling the GT Device Driver.....	B-10
B.5.7 Unloading the Driver Module.....	B-10
B.6 RTX Software Installation.....	B-11
B.6.1 Install the Software.....	B-11
B.6.2 Loading/Unloading SCRAMNet GT Device Driver.....	B-11
B.6.3 Rebuilding Applications	B-11

TABLES

Table B-1 Error Codes	B-9
Table B-2 Driver Install Flags.....	B-10

B.1 Overview

To install the SCRAMNet GT software, complete the following steps:

- Install the hardware.
- Install the software. See the installation section for your operating system.



NOTE: This SCRAMNet GT device driver can support between one and sixteen SCRAMNet GT PCI cards on each host computer.

B.2 Install the Hardware

Install the SCRAMNet GT hardware before installing the GT software. See the *SCRAMNet GT Hardware Reference Manual* for details on installing GT hardware in the host system.



NOTE: SCRAMNet GT cards do not communicate with older generations (SCRAMNet+ SC150 and SC150e) of SCRAMNet cards.

B.3 Windows Software Installation

This section describes how to install the software on Windows operating systems. Supported Windows platforms include:

- Windows 2000
- Windows XP
- Windows Server 2003

B.3.1 Install the Device Driver

On the first boot after the hardware is installed, Windows will detect the GT card as shown below.



NOTE: To upgrade the Windows device driver from an existing installation, execute the win2kUpdateDriver.exe program provided on the CD-ROM and proceed to section B.3.2.

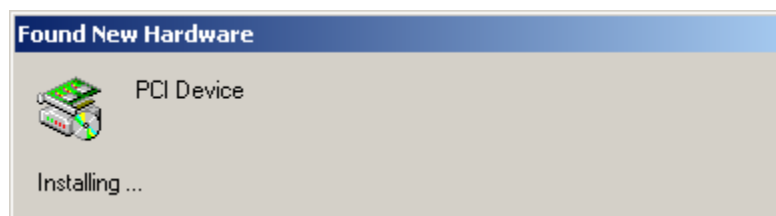


Figure B-1 Found New Hardware

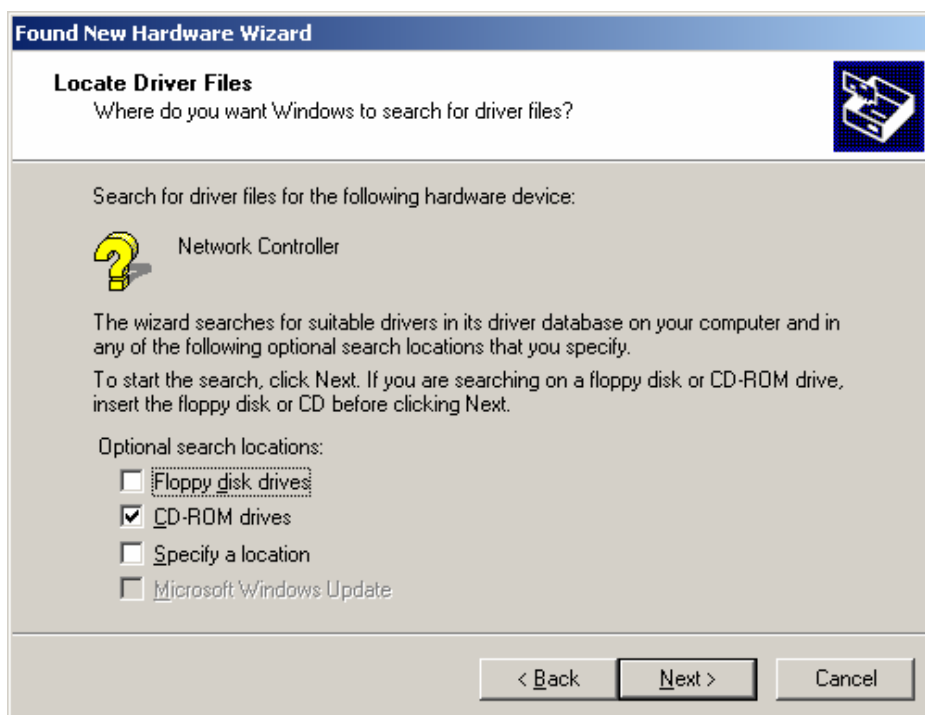


Figure B-2 Found New Hardware Wizard

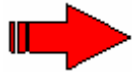
Click the **Next** button to begin the install.

**Figure B-3 Search for Driver**

Click the **Next** button to have Windows search for the driver.

**Figure B-4 Search the CD-ROM**

Make sure the SCRAMNet GT Software CD is in the drive, and the CD-ROM drives box is checked, then click the **Next** button.



NOTE: At this time the SCRAMNet GT driver has not received a digital signature. A dialog may appear warning you of this fact. Click the **Yes** button to install the driver anyway.



NOTE: If you have set your Local Security Settings to prohibit installing unsigned drivers, you will not be able to install the driver without changing these security settings.

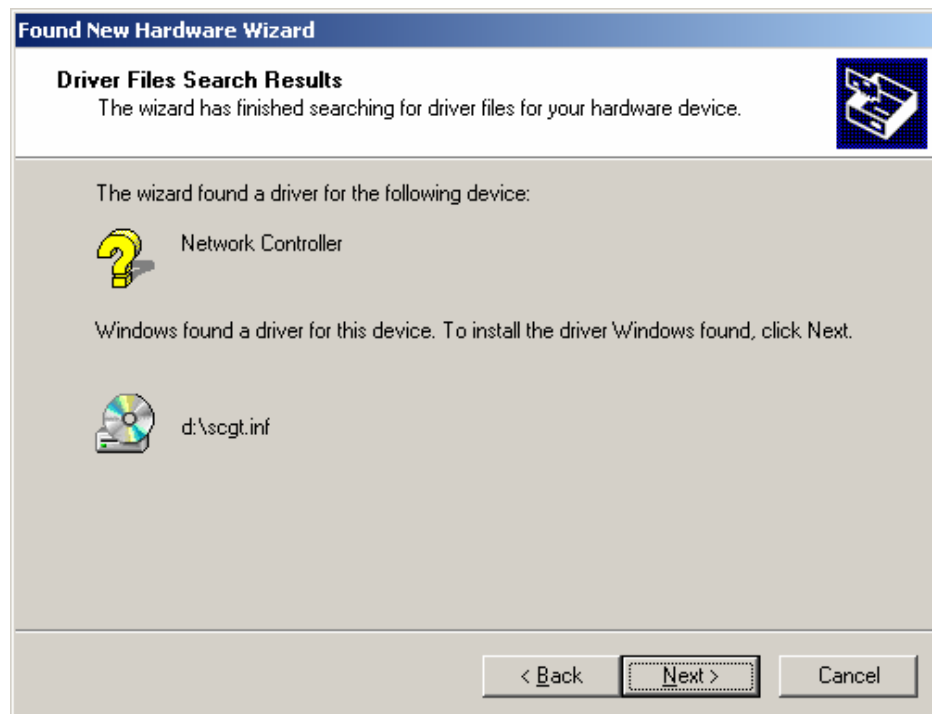


Figure B-5 Find and Install INF

Windows will find the **INF** file for the driver on the CD-ROM. Click the **Next** button to install the driver.

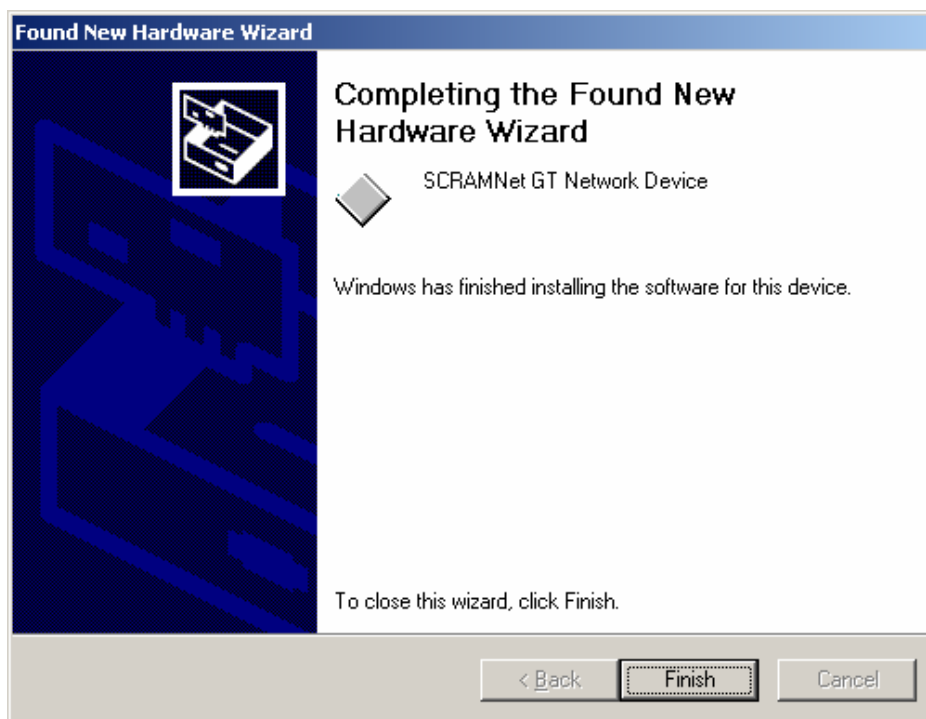


Figure B-6 Driver Installation Completed

After Windows has finished installing the driver click the **Finish** button. The driver should now be running, and will restart automatically whenever the computer is rebooted.

B.3.2 Install the API Library and Utility Applications

The SCRAMNet GT API and utility applications are distributed as a self-extracting zip file. The file is located in the software directory on the CD-ROM. Install the software by simply running the executable and choosing a location to extract the files. The default location is *c:\scgt*.

B.3.3 Rebuilding Applications

Application executable files are supplied with the software and are located in the *win2k/bin* directory. The applications were built with Microsoft Visual C++ version 6.0. If it is necessary to modify the source files, you can rebuild the applications using the provided makefile and the **nmake** Visual C++ utility. For example:

```
cd apps\gtutils
nmake -f egt.win2k.mak
```

You may need to execute the **vcvars32.bat** file distributed with Visual C++ to set the appropriate environment variables in order to execute **nmake**. See the Visual C++ documentation for more information.

B.3.4 Linking with the API library

The SCRAMNet GT API is distributed in a static library format under Windows. User applications that wish to use the GT API calls must link in the library. The following instructions will show how to link a user application with the SCGT API using Microsoft Visual C++ 6.0 project files.

- Make sure that the GT API header files can be found. Add the SCGT inc directory to the 'Additional include directories:' list. The 'Additional include directories:' list is located in Project Settings under the Visual C++ tab when the Category list box is set to 'Preprocessor.' For example, add *C:\scgt\inc* to this list.
- Add the static library (**scgtapi.lib**) located in the *win2k\lib* directory to the project's file list.

Once these steps are completed, the application can be compiled and linked in the usual fashion.

B.4 UNIX-like OS Software Installation

This section describes how to install the software on UNIX-like operating systems. Supported platforms include:

- Linux
- Solaris
- IRIX

Contact Curtiss-Wright Controls Technical Support for specific versions supported.

B.4.1 Install the Software onto the Host Computer (UNIX)

The following sections describe how to install the software. Software is distributed on a CD-ROM as a UNIX **.tar.gz** file. The instructions below assume that the login session will be using a terminal window.

To install the SCRAMNet GT software on your system:

- Place the CD-ROM in the drive.
- Login to the system as root (superuser).
- Mount the CD-ROM if it isn't already mounted. For example:

```
mount /mnt/cdrom
```

- Change to the desired destination directory. Here we use */usr/local*.

```
cd /usr/local/
```

- Extract the distribution **tar.gz** file. For example:

```
gzip -dc /mnt/cdrom/software/scgt-X.tar.gz | tar xf -
```

Replace 'X' in the above file name to match the date in the file name supplied on the CD-ROM.

These procedures will place the contents of the tar file into the */usr/local/scgt* directory.

If you are installing the software for Linux, we will now build the software. The software is pre-built for other operating systems.


```
cd scgt/  
make -f makefile.linux
```

B.4.2 Loading/Unloading SCRAMNet GT Device Driver

You must properly load the SCRAMNet GT device driver before it can be accessed.

First, change to the scgt driver directory for your OS by typing:

```
cd /usr/local/scgt/osname/driver
```

Next, execute the driver load script by typing:

```
./gtload
```

You can unload the driver by executing the unload script:

```
./gtunload
```

B.4.3 Rebuilding Applications

Application executable files are supplied with the software and are located in the *osname/bin* directory. If it is necessary to modify the source files you can rebuild the applications using the provided makefile for your operating system. For example:

```
cd apps/gtutils  
make -f egt.osname.mak
```

Where **osname** is the name of the operating system.

B.5 VxWorks Software Installation

B.5.1 Solaris Host Loading Procedure

To install the SCRAMNet GT software on your system, follow the directions found in Section B.4.1 Install the Software onto the Host Computer (UNIX).

B.5.2 Microsoft Windows Host Loading Procedure

To install the SCRAMNet GT software on your system, follow the directions found in the Windows installation Section B.3.2 Install the API Library and Utility Applications. This will also install the VxWorks software.

Alternatively, decompress and extract the **tar.gz** file in the software directory on the CD-ROM to the directory of your choice using an archiving tool such as **Winzip**.

B.5.3 Loading the Driver Module

The driver module is found in directory `vxworks/driver/manufacturer/sbc`, where - the path is relative to the base of the GT installation - '`manufacturer`' is the single board computer manufacturer - and '`sbc`' describes the single board computer (SBC). There is also a `README.sbc` file in this directory with GT hardware and GT driver module notes that are specific to the SBC and its board support package. Please read this file before loading the module.

Before installing the VxWorks driver, you must first load the driver module into memory. For example,

```
cd vxworks/driver/cwcec/svme182
ld < scgt_driver.o
```

B.5.4 Ensure proper memory mappings

The function `scgtShow()` is used to list all GT devices in the system. It provides information on device location, initialization status, and the PCI base addresses used by the device(s). It is your responsibility to ensure that proper CPU address mappings exist for the PCI base addresses. See your hardware reference manual for information on the size of the required mappings. On systems that do not support PCI auto-configuration, the PCI base addresses will be invalid for uninitialized devices.

B.5.5 Installing the GT Device Driver

To complete the GT device driver installation, `scgtInitDevice()` must be called to initialize each device that will be used. See the README file for a specific example. The first call to `scgtInitDevice()` installs the device driver. Function `scgtInitDevice()` exits with a return value of zero (0) when successful. Below is the calling syntax and description of the parameters for this function.

FUNCTION PROTOTYPE:

```
int scgtInitDevice(uint32 unit,
                  uint32 usePciInfo,
                  uint32 irqLine,
                  uint32 cregAddress,
                  uint32 nmregAddress,
                  uint32 gtmemAddress,
                  uint32 driverFlags)
```

The parameters are:

unit	Number of the unit to initialize. The valid range is 0 to 15.
useConfigInfo	If non-zero, the remaining parameters can be used to override PCI auto-configuration and driver default settings. Those remaining parameters, which are -1 will still be ignored, in case only a partial override is necessary. On computer systems where PCI auto-configuration is unavailable, this option must be used and all values must be specified (excluding driverFlags which may be -1).
irqLine	Interrupt line for the card to use.
cregAddress	Base PCI bus address for configuration registers.
nmregAddress	Base PCI bus address for network management registers.
gtmemAddress	Base PCI bus address for GT memory (used by PIO).
driverFlags	A bit vector which specifies variations from default driver behavior. If -1 or 0, then default driver behavior is applied. The parameter driverFlags is specified on a per device basis. Bit values for the parameter driverFlags are given in file <code>scgtdrv.h</code> , and in table B-2 below. The parameter useConfigInfo must be non-zero for driverFlags to have any effect.

If the GT device driver fails to start correctly, a non-zero error code is returned from `scgtInitDevice()`. Table B-1 lists the error codes. Macro definitions for these error codes are in the header file `vxworks/driver/scgtdrv.h`.

Table B-1 Error Codes

Error Code		Explanation
SCGT_VXW_NO_CARD_FOUND	0xFFFFFFFF	No device found at that unit number.
SCGT_VXW_CAN_NOT_MAP_MEM	0xFFFFFFFFE	Cannot map memory.
SCGT_VXW_INSUFFICIENT_MEMORY	0xFFFFFFFFD	Insufficient Memory.
SCGT_VXW_IOSDRVINSTALL_FAILED	0xFFFFFFFFC	<code>losDrvInstall()</code> Failed.
SCGT_VXW_IOSDEVADD_FAILED	0xFFFFFFFFB	<code>losDevAdd()</code> Failed.
SCGT_VXW_INT_CONNECT_ERROR	0xFFFFFFFFA	Interrupt could not be connected.
SCGT_VXW_INVALID_BUS_ADDRESS	0xFFFFFFFF9	Invalid bus address.
SCGT_VXW_DMA_RESOURCE_FAILURE	0xFFFFFFFF8	Insufficient resources for DMA.
SCGT_VXW_ALREADY_INITIALIZED	0xFFFFFFFF7	The specified unit was already initialized.
SCGT_VXW_DEVICE_INIT_ERROR	0xFFFFFFFF6	Core driver code failed to initialize device.
SCGT_VXW_GENERAL_ERROR	0xFFFFFFFF5	Non-specific failure.
SCGT_VXW_INVALID_PARAMETER	0xFFFFFFFF4	Invalid command line parameter

Table B-2 Driver Install Flags

Flag	Value	Explanation
SCGT_VXW_NO_CACHE_FLUSH	0x00000001	Prevents cache flush operations on user buffers before DMA writes to device memory.
SCGT_VXW_NO_CACHE_INVALIDATE	0x00000002	Prevents cache invalidate operations on user buffers before DMA reads from device memory.
SCGT_VXW_QUIET	0x00000004	Prevents driver initialization messages

B.5.6 Uninstalling the GT Device Driver

The GT device driver can be dynamically uninstalled by using the *scgtRemoveDevice()* command. Since this is not a straightforward procedure, it is generally better to just reboot the system. Below is the calling syntax and description of the parameters for this function.

int scgtRemoveDevice(uint32 unit)

unit Number of the unit to uninitialized. The valid range is 0 to 15.

If unit is the last device presently initialized, then the driver will be uninstalled from the system. Function *scgtRemoveDevice()* returns 0 when it completes successfully. Before calling *scgtRemoveDevice()*, ensure that the device is idle. Verify that no applications are running that use the device(s) or driver, and close all open handles to the device(s).

B.5.7 Unloading the Driver Module

Unloading the driver module fully removes the driver from system memory.

For example:

```
unld "scgt_driver.o"
```

B.6 RTX Software Installation

This section describes how to install the software for RTX.

Contact Curtiss-Wright Controls Technical Support for specific versions supported.

B.6.1 Install the Software

The SCRAMNet GT Software is distributed as a self-extracting zip file located in the software directory on the CD-ROM. Install the software by running the executable and choosing a location to extract the files. The default location is *c:\scgt*. Next, change the default mapping size used in RTX using regedit. Change the following key to the size of the memory on the SCRAMNet GT device:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Rtx\MapMemory  
SizeBound
```

For instance, change it to 0x08000000 for SCRAMNet GT devices with 128MB.

B.6.2 Loading/Unloading SCRAMNet GT Device Driver

You must properly load the SCRAMNet GT device driver before it can be accessed.

First, change to the scgt driver directory by typing:

```
cd \scgt\rtx\driver
```

Next, execute the driver load script by typing:

```
gtload
```

You can unload the driver by executing the unload script:

```
gtunload
```

B.6.3 Rebuilding Applications

Application executable files are supplied with the software and are located in the *rtx\bin* directory. If it is necessary to modify the source files, you can rebuild the applications using the provided makefile. For example:

```
cd apps\gtutils  
make -f egt.rtx.mak
```

You may need to execute the **vcvars32.bat** file distributed with Visual C++ to set the appropriate environment variables in order to execute **nmake**. See the Visual C++ documentation for more information.

In addition to building the SCRAMNet GT utilities, the provided makefile is also a good example of how to build your own applications.

This page intentionally left blank

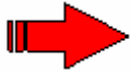
APPENDIX C

UTILITY HELP

TABLE OF CONTENTS

GT Monitor (gtmon)	C-1
GT Memory/Register Access (gtmem)	C-2
GT Throughput Test (gttp).....	C-3
GT Network Exerciser (gtnex).....	C-4
GT Interrupt Test (gtint)	C-4
GT In-Circuit Programming (gtprog).....	C-5

SCRAMNet GT Utility Usage Listings



NOTE: These examples are included for your convenience. See the application help output for an up-to-date listing.

GT Monitor (gtmon)

Monitoring:

```
gtmon [-u unit] [-p msperiod] [-s seconds] [-N | -V | -S] [-h]
gtmon -A [-V | -VV | -N | -S]
```

Inquiry:

```
gtmon [-u unit] [--nodeid] [--interface] [--ringsize] [--d64] [-V]
      [--wlast] [--linkup] [--rx] [--uint] [--laser] [--signal]
      [--ewrap] [--bswap] [--tx] [--sint] [--laser0] [--signal0]
      [--spyid] [--wswap] [--px] [--bint] [--laser1] [--signal1]
```

Configuration:

```
gtmon [-u unit] [-n nodeid] [-i interface] [-y spyid] [-b bcastIntrMask]
      [--wlaston|--wlastoff] [--rxon|--rxoff] [--ewrapon|--ewrapoff]
      [--sinton|--sintoff] [--txon|--txoff] [--laseron|--laseroff]
      [--uinton|--uintoff] [--pxon|--pxoff] [--laser0on|--laser0off]
      [--bswapon|--bswapoff] [--d64on|--d64off] [--laser1on|--laser1off]
      [--wswapon|--wswapoff] [-V]
```

Defaults: gtmon -u 0

Options:

```
-u #      - board/unit number          -h      - show help, use -h 1 for more
-n #      - set node ID                 -b #     - set broadcast interrupt mask
-N        - show network info           -i #     - set receive interface, 0 or 1
-S        - show driver statistics      -A       - show info for all local devices
-V        - verbose
-p #      - period, refresh every # milliseconds, 'q' to stop
-s #      - seconds to run gtmon, defaults -p to 500, 'q' to stop
-y #      - set the spy ID. HW will count (spy) traffic from this node ID.
--reset   - reset the device
```

Inquiry options (value of 1 means "on", 0 means "off"):

```
--linkup - link status          --signal - active interface sig.det.
--nodeid - node ID              --signal0 - interface 0 signal detect
--bint    - broadcast interrupt mask --signal1 - interface 1 signal detect
--spyid   - spy node ID         --interface - active interface
```

Inquiry/assignment options (append "on" or "off" to assign value):

```
--rx      - receive path (R)      --laser   - active interface laser
--px      - retransmit path (P)   --laser0  - interface 0 laser status
--tx      - transmit path (T)    --laser1  - interface 1 laser status
--wlast   - write-last path (W)  --ewrap   - electronic wrap path (E)
--uint    - unicast interrupts    --bswap   - byte swap GT memory
--sint    - self-interrupts      --wswap   - word swap GT memory
--d64     - 64bit PCI data transfer --link    - link error Interrupts
```

Examples:

Show all devices in the computer:

```
gtmon -a
```

Show verbose information about unit 0:
gtmon -V

Show network topology information:
gtmon -N

GT Memory/Register Access (gtmem)

Usage: gtmem [-P|D] [-u unit] [-o offset] [-c count] [-l length]
 [-w data [-i increment]] [-b bytes] [-p msec_period]
 [-A] [-f search_pattern [-E] [-n maxFinds]] [-h]
gtmem -R|N [-u unit] [-o offset] [-c count] [-l length]
 [-A] [-w data [-m mask]] [-b bytes]

Defaults: gtmem -rP -u 0 -o 0x0 -c 1 -b 4 -d 0

Options:

- D - DMA access to GT's memory
- P - PIO access to GT's memory
- R - control register access
- N - network register access
- u # - unit/board number
- r - read memory/registers (default)
- w # - write data pattern (see -m)
- m # - bitmask (applicable to control register writes only, required)
- o # - shared GT memory offset or register offset (in bytes)
- c # - count of words/bytes on which to apply operation
- l # - length in bytes on which to apply operation (overrides -c)
- A - all, operate on full memory/register range (overrides -o,-l)
- b # - size of read/write operations in bytes: 1 - bytes,
 2 - 16bit words, 4 - 32bit words (default), 8 - 64bit words
- p # - periodic, repeat every # milliseconds('q' exits). Read mode only
- i # - increment the written pattern by # for each offset written
- f # - find/search for pattern # in memory
- E - modifies search (-f #) to find all patterns except # in memory
- n # - max number of times to find the search pattern (default is 1)
- d # - display modifier flags (positionally coded), default is 0:
 0x1 - omit offsets, 0x2 - add 0x, 0x4 - one column, 0x8 - one row
 0x10 - relative offsets, 0x20 - word offsets (see -b)
- z - DMA - only swapping flags: 0x1 - swap bytes, 0x2 - swap words

Note: run `gtmem -h 1` for more information.

Examples:

Display values of first 32 32-bit words in GT memory.
gtmem -P -c 32

Write 0 to the 32 words starting at offset 0x100 in GT memory:
gtmem -P -c 32 -o 0x100 -w 0

Run "gtmem -h 1" for more examples.

GT Throughput Test (gttp)

Usage: gttp [-w|-r] [-PEMSV] [-u unit] [-l packetLength]
 [-n iterations] [-t numOfTasks] [-o offset]
 [-s seconds] [-d dataToWrite] [-h]
 gttp -G [-w|-r] [-PEMS] [-u unit] [-o offset]

Defaults: gttp -r -u 0 -l 131072 -n 1000 -t 1 -o 0

Options:

-u # - board/unit number
 -w|r - data transfer direction, -w for write, -r for read DIR
 -P - PIO - use CPU (as opposed to DMA) for data movement P
 -E - do data processing emulation (CPU will touch all data) E
 -M - use system's memory instead of GT's memory M
 -S - run test for about one second S
 -t # - number of threads/tasks to use t
 -n # - number of iterations (per task) to transfer the packet ITER/t
 -l # - packet length in bytes SIZE
 -o # - offset (address) in the shared GT memory
 -s # - seconds to repeatedly run the test (-1 for forever)
 -d # - data value to write to memory (-E overrides)
 -G - graph throughput (many packet sizes exercised)
 -A - graph all, exercises many transfer settings (only valid with -G)
 -V - verbose, print read data)

Other reported values:

- total number of bytes transferred TOT.BYTES
 - test time in seconds TIME
 - number of IO operations per second IO/s
 - number of microseconds per IO us/IO
 - throughput in mega (million) bytes per second MB/s

Example: gttp -u 0 - use defaults on unit 0
 gttp -r -l 8 -n 100 -t 3 - read test
 gttp -w -l 0x100 -n 10000 -t 2 - write test
 gttp -wG - graph write throughput

Notes: Run `gttp -h 1` for more information.
 Press 'q' to stop a running test.

GT Network Exerciser (gtnex)

Usage: gtnex [-PMQTR] [-u unit] [-m memberId] [-d displayMode]
 [-s secondsToRun] [-l packetLength] [-h]

Defaults: gtnex -D -u 0 -d 7 -s -1 (-m -1 -p -1)

Options:

- M - use system's memory instead of GT's memory
- P - PIO - use CPU (do not use DMA) exclusively
- T - terminate the test on all the nodes on the ring
- R - restart for all the nodes on the ring
- Q - run quick memory test only
- u # - board/unit number (default 0)
- m # - use member Id # (mId=#) instead of node Id (mId=nId)
- l # - limit the maximum packet length to # of bytes
- s # - limit the test time to # of seconds
- d # - display mode flags (positionally coded):
 1 - error details, 2 - network layout, 4 - summary line

Example: gtnex -u 0 - use defaults on unit 0
 gtnex -M - use system memory instead of GT's

Notes: Run `gtnex -h 1' for more information.
 Press 'q' to stop a running test.

GT Interrupt Test (gtint)

Usage: gtint [-U | -B] [-Ah] [-u unit] [-s seconds] [-p msPause]
 [-n iterations] [-i dest_or_bcast_ID] [-w value]
 [-d displayMode]

Defaults: gtint -u 0 -i 0 -n 1 -w 0x0 -d 0x0

Options:

- u # - board/unit number
- B - send a broadcast interrupt
- U - send a unicast interrupt
- A - send all broadcast and/or all unicast interrupts
- i # - destination ID (unicast), interrupt ID (broadcast), default is 0
- w # - data value to send with interrupt
- n # - number of times to send the interrupt(s)
- d # - display mode flags (default is 0):
 0x1 - broadcast, 0x2 - unicast, 0x4 - error interrupts
- s # - number of seconds to run test (-1 for infinity)
- p # - pause for # milliseconds b/w interrupt bursts (send modes)
- h - display this help menu

Runtime options:

- q - quit
- r - reset statistics
- b - toggle broadcast display
- e - toggle error interrupt display
- u - toggle unicast display

Notes: Run `gtint -h 1' for more information.

Examples:

Display interrupts received.

gtint -s -1

Send broadcast interrupt 10 with a data value of 0x1000.

gtint -B -i 10 -w 0x1000

Send a unicast interrupt to node ID 12.

gtint -U -i 12 -w 0x1000

Note: Use gtmon to modify the interrupt enable masks. See the '-b', '--uinton' and '--sinton' options.

GT In-Circuit Programming (gtprog)

Usage: gtprog -f file [-u unit] [-B] [-h]

Options:

- u unit : SCRAMNet GT board/unit number
- f file : firmware filename (typically with .gfw extension)
- B : proceed with firmware update (burn)
- X : treat -f file as a raw fw file (typically with .xvf extension)
- h : print this help menu

Examples: gtprog -f abc.gfw verify and print info for file abc.gfw
 gtprog -f abc.gfw -u 2 -B update firmware on unit 2

This page intentionally left blank

INDEX

A

API.i, 1-1, 2-1, 2-2, 3-2, 3-3, 4-1, 4-5, 4-6, 4-15, 4-16, 5-1, A-1, 1, B-5, B-6, B-8

B

base addresses..... B-8
 binaries 2-1, 5-1
 Broadcast 3-3, 4-3, 4-4, 4-13

C

CD-ROM B-2, B-3, B-4, B-5, B-6, B-8, B-11
 CPU B-8, C-3, C-4

D

data types A-1
 device.. 2-1, 3-1, 3-2, 3-3, 4-2, 4-5, 4-6, 4-7, 4-8, 4-9, 4-11, 4-12, 4-13, 5-2, 5-3
 DMA..... 2-1, 3-1, 3-2, 3-3, 4-9, 4-14, 4-16, 5-2
 documentation B-5, B-11
 driver.... B-1, B-2, B-3, B-4, B-5, B-7, B-8, B-9, B-10, B-11, C-1

E

Error..... 4-4, B-9
 executable B-5, B-7, B-11

F

firmware revision..... 4-2
 functions 1-1

G

GT memory ... 3-1, 3-2, 3-3, 4-5, 4-8, 4-9, 4-14, 5-2

I

ID..... i, 3-1, 3-3, 4-2, 4-3, 4-4, 4-13, 4-14
 interrupt 3-3, 4-3, 4-4, 4-9, 4-11, 4-13, 4-14, 5-3
 IRIX..... B-6
 ISO..... 1-2

L

library 2-1, 2-2, 5-1
 LinkXchange 3-1

Linux B-6

M

monitor 1-1, 5-2

P

packet 3-1, 4-14, 5-2
 PCI..... B-1, B-8, B-9, C-1
 pHandle 4-5, 4-6, 4-7, 4-8, 4-9, 4-11, 4-12, 4-13
 PIO 3-1, 3-2, 3-3, 4-2, 4-5, 4-9, 4-14, 5-2
 Pointer 4-6, 4-7, 4-8, 4-9, 4-11, 4-12, 4-13

R

README B-8
 restart B-5, C-4
 RTX..... 1, B-11

S

scripting 5-2, 5-3
 Solaris..... 1, B-6, B-8
 stateID 4-13
 syntax B-8, B-10

T

throughput 3-3, 5-2
 transfers 3-1, 3-2, 3-3, 4-9, 4-14, 4-16

U

Unicast..... 3-3, 4-3, 4-4, 4-13
 unitNum..... 4-6
 UNIX..... A-1, 1, B-6, B-8
 utility 1-1, 2-1, 5-1, 5-3, B-5

V

Visual C++ B-5, B-6, B-11
 VxWorks i, 5-1

W

Windows..... A-1, 1, B-2, B-3, B-4, B-5, B-6, B-8

Z

zip file..... B-5, B-11