SCRAMNet GT Network

Software and API Guide

Part Number: DDOC0197-000-A1
(Formerly G-T-ML-G1AP1###-A-0-A7)

This Page Intentionally Left Blank

Front Matter

Revisions

Document Number	Media	Revision	Date	Description	PCN
DDOC0197-000	PDF	A0	2/14/22	Preliminary draft	NA
DDOC0197-000	PDF	A1	2/15/22	Initial issue	0222-0003

The Curtiss-Wright SCRAMNet GT Network Software and API Guide (DDOC0197-000) is made up of the following individual sections:

Section	Topic	Content Revision
1.0	Introduction	0.0
2.0	Software Overview	0.0
3.0	Network Design / Operation	0.0
4.0	API Description	0.0
5.0	Utility Applications	0.0
A	Primitives	0.0
B	Installation	0.0
C	Utility Help	0.0

NOTE

No change bars are applied to changes from review release to initial release.

Changes to technical content are shown through the use of change bars placed in the left margin next to the changed material (as shown here). Corrections to typographical errors are not noted unless they significantly impact the content.

Proprietary

Copyright

Safety

WARNING

HAZARD. A potential hazard that could result in serious injury or death.

Information contained in WARNINGS applies to dangers and hazards that may result in injury and / or death to personnel. The actual hazard is provided in CAPITALIZED letters and the information that mitigates the danger is provided in sentence case. This information typically precedes procedural steps. It also may be present in narrative text to warn operators or maintenance personnel of dangers present in the equipment.

CAUTION

HAZARD. A potential hazard that could result in equipment damage or improper operation.

Information contained in CAUTIONS applies to dangers and hazards that may result in damage to equipment or improper operation. The actual hazard is provided in CAPITALIZED letters and the information that mitigates the danger is provided in sentence case. This information typically precedes procedural steps. It also may be present in narrative text to warn operators or maintenance personnel of dangers present in the equipment.

NOTE

Amplifying information that helps in making a task or procedure more easily understood.

NOTES are used to supply amplifying information that will result in ease of testing or be beneficial to personnel. This information typically precedes procedural steps. It also may be present in narrative text as well.

Style and Conventions

This Software and API Guide uses the following typographical conventions.

This style	Refers to
`Ready`	Text the software displays.
go	Anything you type, exactly as it appears, whether referenced in text or at a prompt.
ENTER	Special keys on the keyboard, such as enter, alt, and spacebar.
Save	Software command buttons and sections of dialog boxes, such as group boxes, text boxes, and text fields.
File → Open	A menu and a specific menu command.
ALT+F1	Pressing more than one key at the same time.
ALT, TAB	Pressing more than one key in sequence.
xx,yy	Variable in error messages and text.
jobfile.dat	File names.

This style	Refers to
◆	Denotes the result of an action or procedure.
xyz	Hyperlink.
STOP	Controls on equipment.

Table of Contents

Utility Applications

Primitives

Installation

Utility Help

Introduction

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.

1.2 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 **Installation** section.

1.3 Related Information

- SCRAMNet GT Hardware Reference Manual for PCI and PMC cards (Doc. No G-T-MR-G1PCPMCP-A-0-xx)
- LinkXchange GLX4000 Physical Layer Switch User Reference Manual (Doc. No F-T-MR-L5XL144) Curtiss-Wright Controls, Inc.

1.4 Quality

Curtiss-Wright Controls, Inc., Electronic Systems is committed to leveraging our technology leadership to deliver products and services that meet or exceed customer requirements. 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, Inc., Electronic Systems' Quality Management System is accredited to the latest revision of the aerospace standard, AS9100 Quality Management Systems - Requirements for Aviation, Space, and Defense Organizations.

Our Quality System addresses the following basic objectives:

- Achieve, maintain, and continually improve the quality of our products and service through established design, test, production and service 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.

EAGLE Registrations Inc. assessed Curtiss-Wright's Quality Management System and confirmed conformance to AS9100D including ISO 9001:2015 with Certificate No. 5819. The scope of the registration is as follows: “Design, manufacture, test and repair of board level products, electronic sub-systems, related software and services for commercial, aerospace and military applications.”

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.

1.5 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.

While 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.cwcdefense.com

1.6 Ordering Process

To learn more about Curtiss-Wright Defense Solutions' products or to place an order, please use the following contact information.

- E-mail: DTN_info@curtisswright.com
- World Wide Web address: http://www.cwcdefense.com/

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 the user to write effective software with minimal complication. The software consists of a device driver, an API library, and a set of utility applications. See the **Installation** section for the software installation procedure for the associated operating system.

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 **API Description** section.

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

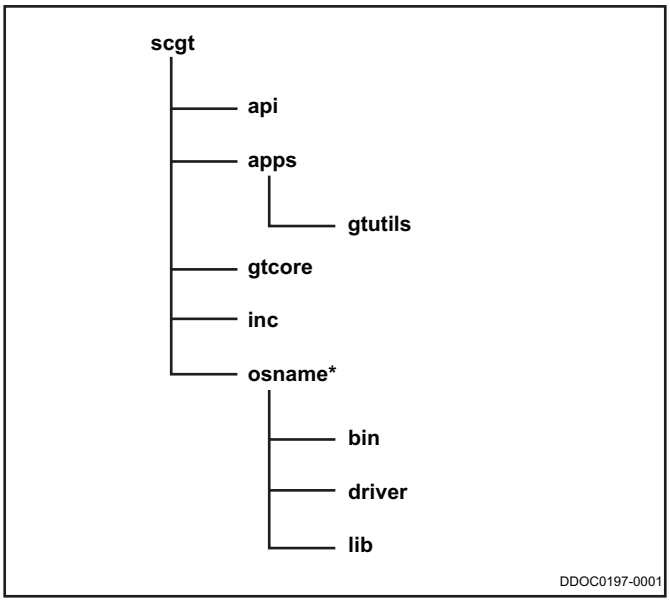

Figure 2.1 Directory Structure

Directory	Function
api	contains API source code
gtutils	contains source code for SCRAMNet GT utilities.
gtcore	contains core driver files.
inc	contains API header files.

Directory	Function
osname*	is the operating-system-specific directory.
bin	contains binaries for the utilities.
driver	contains the driver(s), and where applicable, the gtload and gtunload scripts.
lib	contains the built API library.

* Replace the osname with associated operating system.

Network Design / 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 s**cgtDeviceInfo** 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 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 *

* 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.

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 ‡

‡ 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.

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.

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.

API Description

4.1 Overview

This chapter describes the SCRAMNet GT API in detail. First, the structures defined by the API are described in paragraph 4.2 **Structures**. Then the API functions and their arguments are described in detail in paragraph 4.3 **API Functions**. Lastly, possible return codes of the API functions are described in paragraph 4.4 **Return Codes**. Note that base data types (for example, int32) are used throughout the API. This ensures compatibility across platforms and compilers. See **Primitives** section 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.

Example

```
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.

Example

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

4.2.2.1 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.

4.2.2.2 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

4.2.2.3 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 (Table 4.1). 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 paragraph 4.4 **Return Codes**.

Table 4.1 API Functions

Function	Description	Page
scgtOpen	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.	4-2
scgtClose	Closes a handle to a device.	4-2
scgtMapMem	Map GT memory for PIO	4-2
scgtUnmapMem	Unmap GT memory	4-2
scgtRead	Read data from GT memory.	4-2
scgtWrite	Write data to GT memory.	4-2
scgtGetInterrupt	Retrieve network and error interrupts from the driver.	4-2
scgtGetDeviceInfo	Retrieves information about the device.	4-2
scgtGetState	Get the value of a specified configuration parameter.	4-2
scgtSetState	Set the value of a specified configuration parameter.	4-2
scgtGetErrStr	Returns a string describing a specific error code.	4-2

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

CAUTION

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

Function Prototype

*uint32 scgtClose (scgtHandle *pHandle);*

Description

Closes a handle to a device.

Input

pHandle:.....Pointer to a device handle.

4.3.3 scgtMapMem, scgtUnmapMem

Function Prototype

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

```
void scgtUnmapMem
        (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 s*cgtUnmapMem()* function frees the resources used by the s*cgtMapMem()* function.

Input

pHandle:.....Pointer to a device handle.

4.3.4 scgtRead, scgtWrite

Function Prototype

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

```
uint32 scgtWrite(scgtHandle *pHandle
        uint32 gtMemoryOffset,
        void *pDataBuffer,
        uint32 bytesToTransfer,
        uint32 flags,
        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.

CAUTION
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 paragraph 4.3.7 **scgtGetState, scgtSetState**.

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		T he 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.

Table 4.2 State ID

stateID	Settable	Description
SCGT_LATENCY_TIMER_VAL		Timer that measures transit latency of the network.
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		1 if link is active, 0 if link is inactive
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.

Table 4.3 Return Codes

Name	Description
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 Defense Solutions technical support for more information.

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.

5.2 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.

Example

```
gtmon -h
```

Example

```
gtmon -h 1
```

See **Utility Help** section for a listing of the output.

5.2.1 Device Monitor and Configuration Tool (gtmon)

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 Network Throughput Graphing Tool (gttp)

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 system performance.

5.2.3 Network Exerciser (gtnex)

The **gtnex** application is a generator and verifier of network traffic. This application exercises multiple nodes on the network simultaneously. By using predefined 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 Memory/Register Access Utility (gtmem)

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 Network Interrupt Utility (gtint)

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 Firmware Programmer (gtprog)

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 the proper firmware-programming file is being used.
- Attempt to program the firmware again. If programming fails, try different firmware versions.

NOTE
Contact Curtiss-Wright Defense Solution Technical Support if problems persist

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

Example

```
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**:

Example

```
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.

Primitives

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 *int64* is a signed 64-bit integer.

uint64 ..... The type *uint64* is an unsigned 64-bit integer.

int32 ..... The type *int32* is a signed 32-bit integer.

uint32 ..... The type *uint32* is an unsigned 32-bit integer.

int16 ..... The type *int16* is a signed 16-bit integer.

uint16 ..... The type *uint16* is an unsigned 16-bit integer.

int8 ..... The type *int8* is a signed 8-bit integer.

uint8 ..... The type *uint8* is an unsigned 8-bit integer.

scgtHandle ..... The type *scgtHandle* is an abstraction of the file descriptor in Linux, the HANDLE in Windows, and other similar descriptors used by other operating systems. The SCRAMNet GT API uses a *scgthandle* to determine on which device the request should be executed. Consult the example applications for more information on using the type *scgtHandle*.

Installation

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.

B.2 Install Hardware

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

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

Install the SCRAMNet GT hardware before installing the GT software. See SCRAMNet GT Hardware Reference Manual (Doc. No G-T-MR-G1PCPMCP-A-0-xx) for details on installing GT hardware in the host system.

B.3 Windows Install

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

- Windows 10
- Windows 11
- Windows Server 2016
- Windows Server 2019
- Windows Server 2022

B.3.1 Software Install

CAUTION
IMPROPER OPERATION. Install SCRAMNet GT hardware before installing GT software. Failure to install hardware first may result in improper installation / operation.

To install the SCRAMNet GT software perform the following steps:

1. Navigate to software\A13 Windows update 20220202 folder on CD.
2. Unzip / extract scgt_windows_02Feb2022 file to a location on the computer hard drive.

B.3.2 Device Driver Install

CAUTION
IMPROPER OPERATION. Install SCRAMNet GT hardware before installing GT device driver. Failure to install hardware first may result in improper installation / operation.

To install the SCRAMNet GT device driver perform the following steps:

NOTE
The GT card may show up as a Network Controller under Other devices.

1. Right click Start icon (Figure B.1) and then select Device Manager).
2. Right click on Other devices->Network Controller and select Update Driver Software…

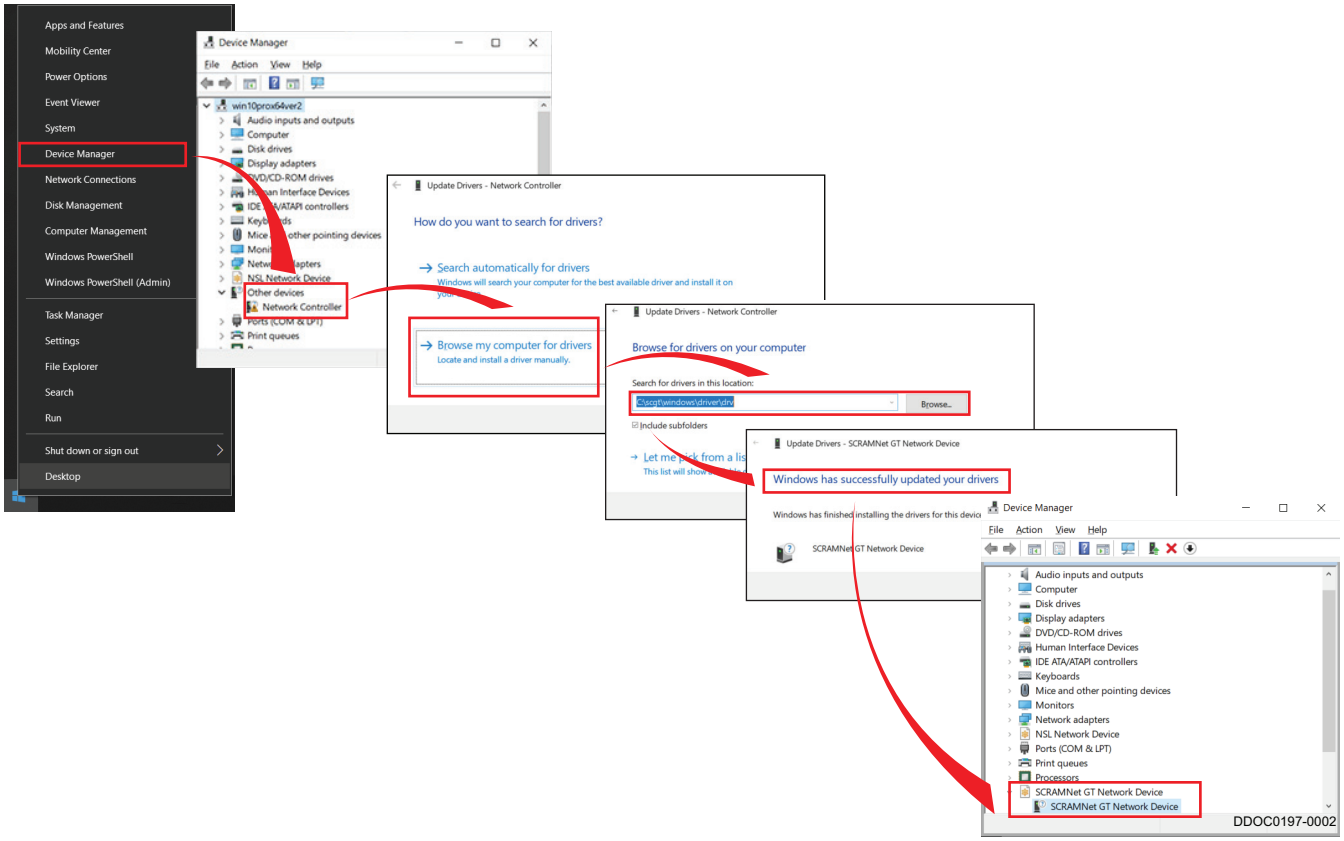

Figure B.1 Device Driver Install

3. Select Browse my computer for driver software. Enter path to scgt/windows/driver/drv folder location.
4. Click on Next.
   - ◆ The driver will be installed.
5. When installation is complete, a message will be displayed saying Windows has successfully updated the driver software.
6. The GT device now appears in the Device Manager.

B.3.3 Visual Studio

The Visual Studio 2019 solution file and project files for the API and utility applications are located in the scgt/windows/gtapps folder.

In order to run the application executables supplied with the software the system needs to have the Microsoft Visual C++ runtime support files installed. These files are installed when Visual Studio 2019 or some other application software requiring these files is installed. If an error message (Figure B.2) appears when the application is run, it indicates the support files are not present.

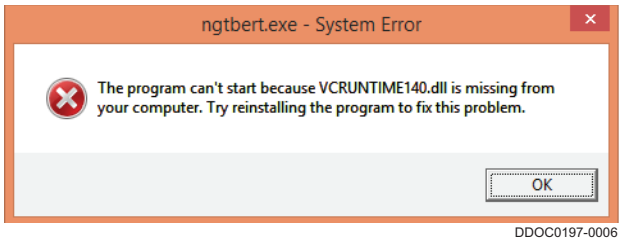

Figure B.2 System Error Message

NOTE

For convenience, a copy of the software from the time of release is included in the **scgt\windows\bin** and **scgt\windows\binx64** directories.

Vcruntime140.dll is one of the Visual Studio runtime support files that is needed to run the precompiled executables. To fix this issue install Visual Studio 2019 or install the Visual Studio Redistributable files. To install the Visual Studio redistributable files run the Microsoft program **vs_redist.x64.exe** to install the 64-bit support files and **vc_redist.x86.exe** to install the 32-bit support files. To get the latest version of these files go to https://support.microsoft.com/ and search for **Microsoft Visual C++ Redistributable**.

B.3.3.1 API

The API source code is in the api folder. The object library is in the scgt/windows/lib folder for 32-bit applications and in the scgt/windows/lib64 folder for 64-bit applications.

B.3.3.2 API - Rebuild as Static Library

NOTE

By default, the latest SCRAMNet GT software library is built as a DLL and the dll file scgtapi.dll needs to be added to the path or the current directory with your executable file.

If a DLL version of the library is not required, the requirement can be eliminated by recompiling the API as a static library.

1. To rebuild the API using visual studio project GUI (Figure B.3).
   a. Open the gtapp.sln file in scgt\windows\gtapp folder
   b. Select scgtapi project.
   c. Open scgtapi property page dialog as follows:
      1) Select **project** from Visual Studio menu bar.
      2) Select **properties** from Visual Studio menu bar.
   d. From project property dialog select **general** tab (Figure B.4).
   e. Change **configuration type** from Dynamic library (.dll) to Static library (.lib).
   f. Click on **apply** and rebuild the solution.

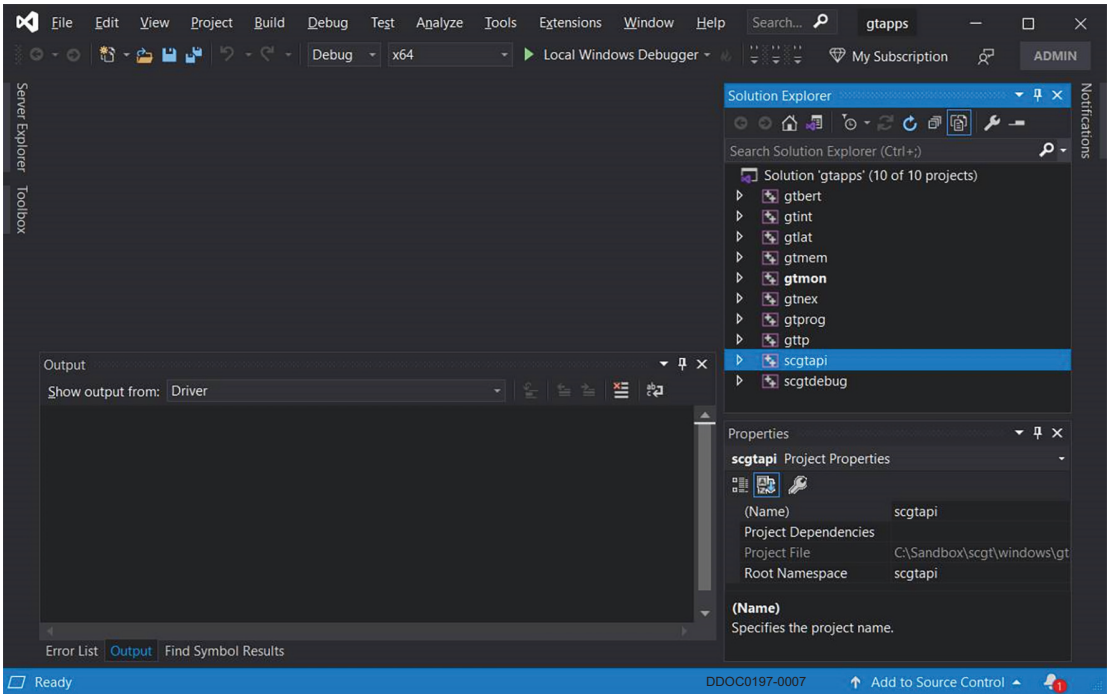

Figure B.3 Visual Studio Project Example

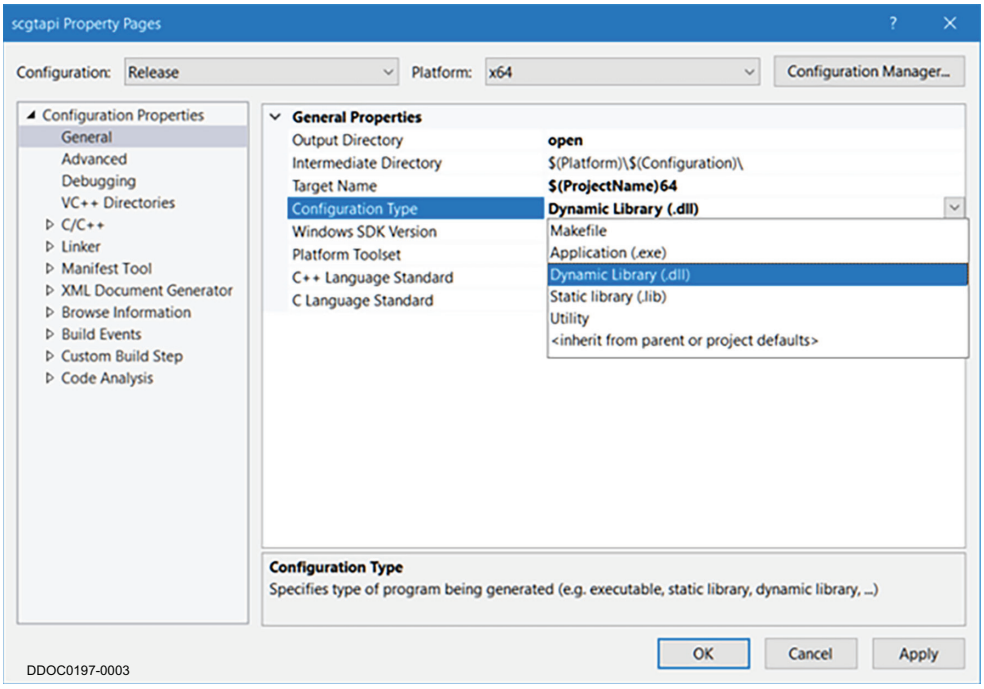

Figure B.4 Rebuild API by GUI

NOTE

If using an older version of visual studio that does not work with the solution files provided the API and applications can be built from the command line.

2. To rebuild the API using command line:
   a. Open compiler developer's window (Figure B.5).
   b. Change directory to scgt\api.

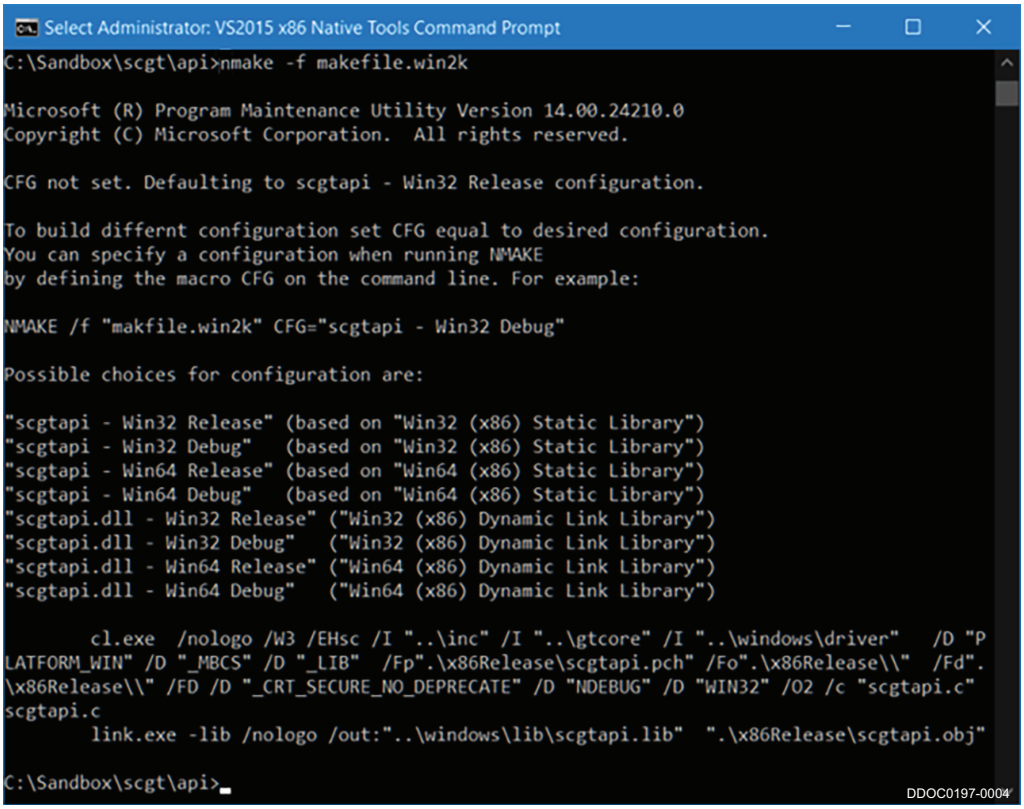

Figure B.5 Rebuild API by Command Line

NOTE
By default the make file will build library as a 32-bit static library and display a list of alternative configuration settings. To change configuration, set CFG flag to desired configuration to build.

c. At the prompt type **`nmake -f makefile.win2k`** and press ENTER key.

d. To build 64-bit versions of the utilities, type **`nmake -f makefile.win2k.make CFG="scgtapps - Win64 Release"`** and press ENTER key.

B.3.3.3 API Library - Link

The SCRAMNet GT API is distributed in a DLL 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 Visual Studio 2019 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 the Configuration Properties->C/C++->General property page for a project.
- Add the static library (scgtapi.lib) located in the scgt/windows/lib (for 32-bit applications) or scgt/windows/lib64 (for 64-bit applications) to the 'Additional Dependencies' list.
- The 'Additional Dependencies' list is located in the Configuration Properties->Linker->Input property page for a project.

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

B.3.3.4 Utility Applications

The source code for the utility programs are located in the scgt\apps\gtutils folder. The 32-bit binaries are placed the scgt\windows\bin folder. The 64-bit binaries are placed in the scgt\windows\bin64 folder. The SCRAMNet GT windows software comes with prebuild utility application binaries in these folders.

B.3.3.5 Utility Applications - Rebuild

NOTE
The gtapp visual studio project can be used to rebuild the sample application. If using an older version of visual studio that does not work with the solution file, the gtapps can be built from the command line.

1. Rebuild utility applications using the Visual Studio Solution GUI as follows:
   a. Open solution file scgt.sln in scgt/windows/gtapps folder
   b. Select desired configuration type x86 or x64
   c. Open build menu and select rebuild solution.
2. Rebuild utility applications using the command line as follows:
   a. Open a compiler developers window.
   b. Change directory to scgt\apps\gtutils

NOTE
By default the make file will build library as a 32-bit static library and display a list of alternative configuration settings. To change configuration, set CFG flag to desired configuration to build.

c. At the prompt type **`nmake -f makefile.win2k`** and press ENTER key.

d. To build 64-bit versions of the utilities, type **`nmake -f egt.win2k.make CFG="scgtapps - Win64 Release"`** and press ENTER key.

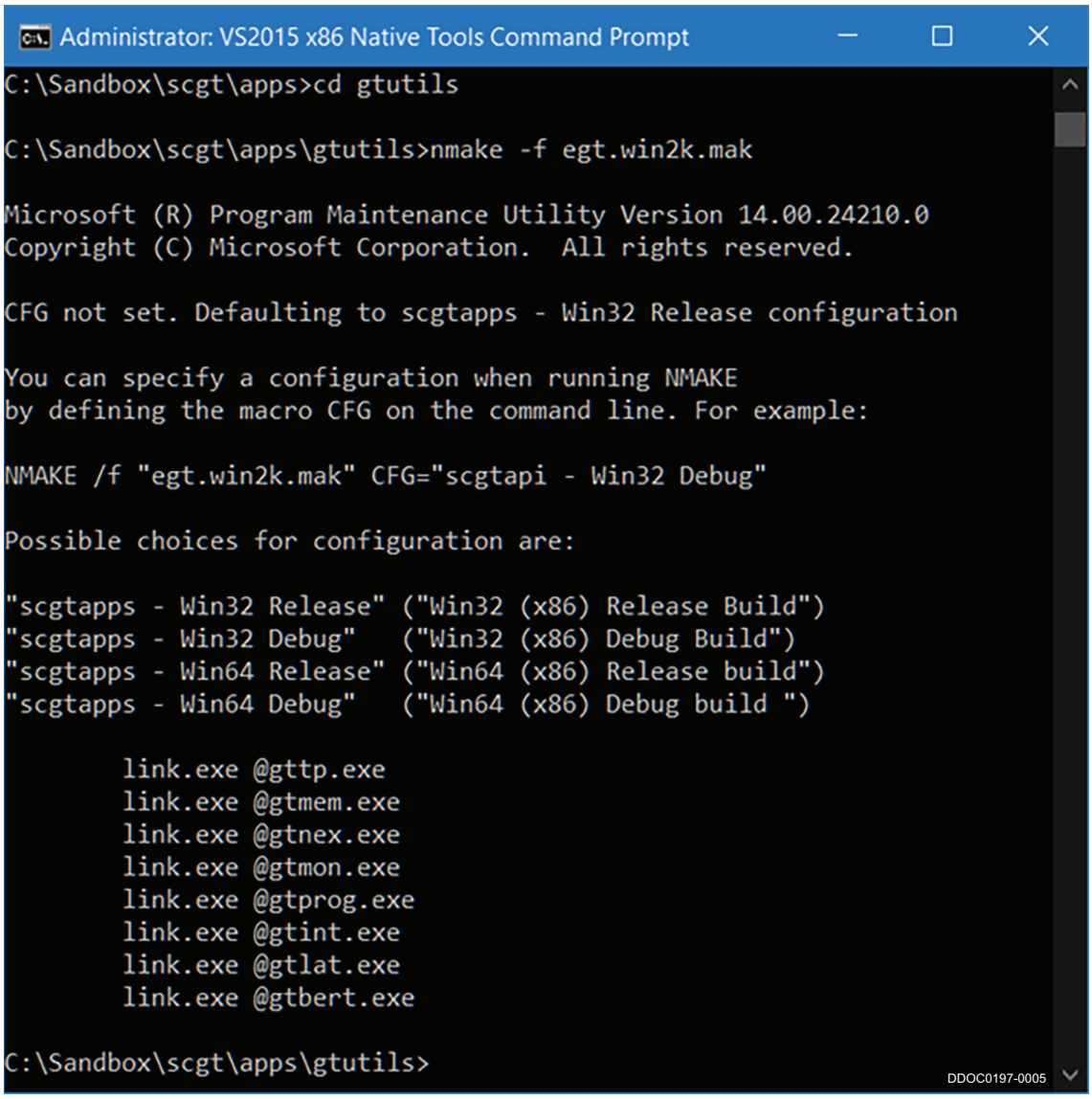

Figure B.6 Rebuild Utilities by Command Line

B.4 Linux Install

NOTE
Contact Curtiss-Wright Defense Solutions for information on which versions of Linux are supported.

This section describes how to install the software on Linux operating systems.

B.4.1 Software Install

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

To install the SCRAMNet GT software perform the following steps:

1. Place the CD-ROM in the drive.
2. Login to the system as root (superuser).
3. Mount the CD-ROM if it isn.t already mounted. Type **`mount /mnt/cdrom`** and press ENTER.

NOTE
The example uses /usr/local directory.

4. Change to the desired destination directory. Type **`cd /usr/local/`** and press ENTER.

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

5. Extract the distribution tar.gz file. Type **`gzip ¡Vdc /mnt/cdrom/software/scgt-X.tar.gz | tar xf -`** and press ENTER. This will place the contents of the tar file into the /usr/local/scgt directory.
6. Build the software as follows:
   a. Type **`cd scgt/`** and press ENTER.
   b. Type make –f makefile.linux and press ENTER.

B.4.2 Device Driver Loading / Unloading

The SCRAMNet GT device driver must be properly loaded before it can be accessed.

1. Change to the scgt driver directory for your OS by typing **`cd /usr/local/scgt/linux/driver a`**nd press ENTER.
2. Execute the driver load script by typing **`./gtload a`**nd press ENTER.
3. Unload the driver by executing the unload script. Type **`./gtunload a`**nd press ENTER.

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 the applications can be rebuilt using the provided makefile for the associated operating system. To rebuild the software as follows:

1. Type **`cd apps/gtutils`** and press ENTER.
2. Type **`make -f egt.linux.mak`** and press ENTER.

Utility Help

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

C.1 GT Monitor

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:

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

Inquiry options (value of 1 = ON, 0 + OFF):

--bint	Broadcast interrupt mask.
--interface	Active interface.
--linkup	Link status.
--nodeid	Node ID.
--spyid	Spy node ID.
--signal	Active interface signal detect.
--signal0	Interface 0 signal detect.
--signal1	Interface 1 signal detect.

Inquiry / Assignment Options (append on or off to assign value):

--bswap	Byte swap GT memory.
--d64	64bit PCI data transfer.
--ewrap	Electronic wrap path (E).
--laser	Active interface laser.
--laser0	Interface 0 laser status.
--laser1	Interface 1 laser status.
--link	Link error Interrupts.
--px	Retransmit path (P).
--rx	Receive path (R).
--tx	Transmit path (T).
--wlast	Write-last path (W).
--uint	Unicast interrupts.
--sint	Self-interrupts.
--wswap	Word swap GT memory.

Examples:

Show all devices in the computer: *gtmon -a*
Show verbose information about unit 0: *gtmon -V*
Show network topology information: *gtmon -N*

C.2 GT Memory / Register Access

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:

-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.
-c #	Count of words/bytes on which to apply operation.
-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).
-D	DMA access to GT's memory.
-E	Modifies search (-f #) to find all patterns except # in memory.
-i #	Increment the written pattern by # for each offset written.
-f #	Find / search for pattern # in memory.
-l #	Length in bytes on which to apply operation (overrides -c).
-m #	Bitmask (applicable to control register writes only, required).

-o # ..... Shared GT memory offset or register offset (in bytes).
-n # ..... Max number of times to find the search pattern (default is 1).
-N ..... Network register access.
-p # ..... Periodic, repeat every # milliseconds (Q exits). Read mode only.
-P ..... PIO access to GT's memory.
-r ..... Read memory/registers (default).
-R ..... Control register access.
-u # ..... Unit / board number.
-w # ..... Write data pattern (see -m).
-z ..... DMA – only swapping flags: 0x1 – swap bytes, 0x2 – swap words.

NOTE
Run *gtmem -h 1* for more examples.

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*

C.3 GT Throughput Test

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:
-A ..... Graph all, exercises many transfer settings (only valid with -G).
-d # ..... Data value to write to memory (-E overrides).
-E ..... Do data processing emulation (CPU will touch all data).
-G ..... Graph throughput (many packet sizes exercised)
-l # ..... Packet length in bytes.
-M ..... Use system's memory instead of GT's memory
-n # ..... Number of iterations (per task) to transfer the packet.
-o # ..... Offset (address) in the shared GT memory.
-P ..... PIO - use CPU (as opposed to DMA) for data movement.
-s # ..... Seconds to repeatedly run the test (-1 for forever).
-S ..... Run test for about one second.
-t # ..... Number of threads/tasks to use.
-u # ..... Board / unit number.
-V ..... Verbose, print read data.
-w|r ..... Data transfer direction, -w for write, -r for read.

Other Reported Values:

TOT.BYTES ..... Total number of bytes transferred.
TIME ..... Test time in seconds.
IO/s ..... Number of IO operations per second.
us/IO ..... Number of microseconds per IO.
MB/s ..... Throughput in mega (million) bytes per second.

NOTE
Run *gttp -h 1* for more information. Press Q to stop a running test.

Example:

Use defaults on unit 0: g*ttp -u* 0
Read test: *gttp -r -l 8 -n 100 -t 3*
Write test: *gttp -w -l 0x100 -n 10000 -t 2*
Graph write throughput: *gttp -wG*

C.4 GT Network Exerciser

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:

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

NOTE
Run *gtnex -h 1* for more information. Press Q to stop a running test.

Example:

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

C.5 GT Interrupt Test

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:

-A ..... Send all broadcast and/or all unicast interrupts.
-B ..... Send a broadcast interrupt.
-d # ..... Display mode flags (default is 0):
0x1 ..... Broadcast.
0x2 ..... Unicast.
0x4 ..... Error interrupts.
-h ..... Display this help menu.
-i # ..... Destination ID (unicast), interrupt ID (broadcast), default is 0.
-n # ..... Number of times to send the interrupt(s).
-p # ..... Pause for # milliseconds b/w interrupt bursts (send modes).
-s # ..... Number of seconds to run test (-1 for infinity).

-u # .....Board / unit number.
-U .....Send a unicast interrupt.
-w # .....Data value to send with interrupt.

Runtime Options:

b .....Toggle broadcast display.
e .....Toggle error interrupt display.
q .....Quit.
r .....Reset statistics.
u .....Toggle unicast display.

NOTE
Run *gtint -h 1* for more information.

NOTE
Use **gtmon** to modify the interrupt enable masks. See the -b, --uinton and --sinton options.

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*

C.6 GT In Circuit Programming

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

Options:

-B.....Proceed with firmware update (burn).
-f fil .....Firmware filename (typically with gfw extension).
-h .....Print this help menu.
-u unit .....SCRAMNet GT board/unit number.
-X.....Treat -f file as a raw fw file (typically with xvf extension).

Examples:

Verify and print info for file abc.gfw:*gtprog -f abc.gfw*
Update firmware on unit 2: *gtprog -f abc.gfw -u 2 -B*