



**Dearborn Group**  
***Technology***

## **GRYPHON C LIBRARY**

Release 4.2

### **USER'S MANUAL**

©2005: Dearborn Group Inc.  
27007 Hills Tech Court  
Farmington Hills, MI 48331  
Phone (248) 488-2080 • Fax (248) 488-2082  
*<http://www.dgtech.com>*

This document is copyrighted by the Dearborn Group, Inc. Permission is granted to copy any or all portions of this manual, provided that such copies are for use with the product provided by the Dearborn Group, and that the name “Dearborn Group, Inc.” remain on all copies as on the original.

## IMPORTANT NOTICE

When using this manual, please remember the following:

- This manual may be changed, in whole or in part, without notice.
- Dearborn Group Inc. assumes no responsibility for damage resulting from any accident—or for any other reason—which occurs while the *Gryphon C Library* is in use.
- No license is granted—by implication or otherwise—for any patents or other rights of Dearborn Group Inc., or of any third party.

*Gryphon* is a trademark of Dearborn Group Inc. Other products are trademarks of their respective manufacturers.

---

## Table of Contents

<b>1 INTRODUCTION .....</b>	<b>1</b>
1.1 Document organization and format .....	1
1.2 Technical support .....	1
1.3 References .....	1
<b>2 C LIBRARY INSTALLATION .....</b>	<b>2</b>
2.1 Installation for Windows Platforms .....	2
2.2 Installation for Unix Platforms .....	2
<b>3 C LIBRARY OVERVIEW .....</b>	<b>3</b>
3.1 General Notes .....	3
3.2 Library Structure .....	4
3.2.1 Flowchart of Typical Use .....	4
3.2.2 Handles to Objects .....	5
<b>4 C LIBRARY REFERENCE .....</b>	<b>7</b>
4.1 Library Information and Management .....	7
4.1.1 dgGetLastOpStatus .....	7
4.1.2 dgLibGetVersion .....	8
4.1.3 dgFreeHandle .....	8
4.2 Session Management .....	9
4.2.1 dgBeginSession .....	9
4.2.2 dgEndSession .....	10
4.2.3 dgSessionActive .....	10
4.2.4 dgGryphonGetName .....	11
4.2.5 dgGryphonGetSerial .....	11
4.2.6 dgGryphonGetVersion .....	12
4.2.7 dgGetId .....	12
4.2.8 dgGetChannel .....	13
4.2.9 dgSetTimeout .....	13
4.2.10 dgEnableBroadcastRx .....	14
4.2.11 dgSetCommOpt .....	14
4.2.12 dgSetSortMode .....	15
4.2.13 dgFlushQueue .....	15
4.2.14 dgGryphonGetTime .....	16
4.2.15 dgGryphonSetTime .....	16
4.2.16 dgGryphonGetTimestamp .....	16
4.2.17 dgGryphonSetTimestamp .....	17
4.3 Data Link Management .....	18
4.3.1 Device Identification .....	18
4.3.1.1 dgChannelGetName .....	18
4.3.1.2 dgChannelGetSecurity .....	18
4.3.1.3 dgChannelGetSerial .....	19

4.3.1.4	dgChannelGetVersion .....	19
4.3.1.5	dgChannelGetSlot .....	19
4.3.1.6	dgChannelGetType .....	20
4.3.1.7	dgChannelGetSubtype .....	20
4.3.1.8	dgGetDataMaxLen .....	20
4.3.1.9	dgGetDataMinLen .....	21
4.3.1.10	dgGetExtraMaxLen .....	21
4.3.1.11	dgGetExtraMinLen .....	21
4.3.1.12	dgGetValidHeaderLength .....	22
4.3.1.13	dgGetNodeId .....	22
4.3.2	<b>General Configuration .....</b>	<b>22</b>
4.3.2.1	dgSetFilterMode .....	22
4.3.2.2	dgInit .....	23
4.3.2.3	dgEnableLoopback .....	23
4.3.2.4	dgIOctl .....	24
4.3.3	<b>Event Management .....</b>	<b>24</b>
4.3.3.1	dgReportAllEvents .....	24
4.3.3.2	dgReportEventType .....	25
4.3.3.3	dgGetEventType .....	25
4.3.3.4	dgGetEventTypeId .....	26
4.3.3.5	dgGetEventTypeMeaning .....	27
4.3.4	<b>Message Filter Management .....</b>	<b>28</b>
4.3.4.1	dgCreateFilterHandle .....	29
4.3.4.2	dgAddFilter .....	29
4.3.4.3	dgModifyFilter .....	29
4.3.4.4	dgDeleteFilter .....	29
4.3.4.5	dgFilterSetActive .....	30
4.3.4.6	dgFilterSetAction .....	30
4.3.4.7	dgSetDefaultFilterAction .....	31
4.3.4.8	dgCreateFilterBlockHandle .....	31
4.3.4.9	dgFilterAddFilterBlock .....	31
4.3.4.10	dgSetDataType .....	32
4.3.4.11	dgSetByteOffset .....	33
4.3.4.12	dgSetOperator .....	33
4.3.4.13	dgSetValues .....	34
4.3.5	<b>Channel Speed Management .....</b>	<b>36</b>
4.3.5.1	dgGetSpeed .....	36
4.3.5.2	dgSetSpeed .....	36
4.3.5.3	dgGetPresetSpeed .....	37
4.3.5.4	dgGetGetSpeedIOctl .....	37
4.3.5.5	dgGetSetSpeedIOctl .....	38
4.3.5.6	dgGetIOctlData .....	38
4.3.5.7	dgGetIOctlDataSize .....	39
4.3.5.8	dgGetSpeedDataSize .....	39

4.3.6	Bus Load Monitoring.....	40
4.3.6.1	dgSetBLMParams.....	40
4.3.6.2	dgGetBLMStats.....	40
4.4	Network Services - Primary.....	42
4.4.1	Principal Functions.....	42
4.4.1.1	dgSendFrame.....	42
4.4.1.2	dgRecvFrame.....	43
4.4.1.3	dgSetOnRx.....	45
4.4.2	GC Frame Functions.....	46
4.4.2.1	GENERIC FRAME FUNCTIONS.....	47
4.4.2.1.1	dgCreateFrameHandle.....	47
4.4.2.1.2	dgGetSrc.....	47
4.4.2.1.3	dgGetSrcChannel.....	48
4.4.2.1.4	dgSetDest.....	48
4.4.2.1.5	dgSetDestChannel.....	48
4.4.2.1.6	dgFrameGetType.....	49
4.4.2.1.7	dgFrameGetData.....	49
4.4.2.1.8	dgFrameSetData.....	50
4.4.2.1.9	dgFrameGetTimestamp.....	50
4.4.2.1.10	dgGetContext.....	50
4.4.2.1.11	dgSetContext.....	51
	DATA FRAME-SPECIFIC FUNCTIONS.....	52
4.4.2.2.1	dgGetHeaderLength.....	52
4.4.2.2.2	dgSetHeaderLength.....	52
4.4.2.2.3	dgGetHeaderBits.....	52
4.4.2.2.4	dgSetHeaderBits.....	53
4.4.2.2.5	dgGetDataLength.....	53
4.4.2.2.6	dgSetDataLength.....	53
4.4.2.2.7	dgGetExtraLength.....	54
4.4.2.2.8	dgSetExtraLength.....	54
4.4.2.2.9	dgGetMode.....	54
4.4.2.2.10	dgSetMode.....	55
4.4.2.2.11	dgGetErrorStatus.....	55
4.4.2.2.12	dgSetErrorStatus.....	55
4.4.2.2.13	dgGetPayload.....	56
4.4.2.2.14	dgSetPayload.....	56
4.4.2.3	EVENT FRAME-SPECIFIC FUNCTIONS.....	57
4.4.2.3.1	dgGetEventId.....	57
4.4.2.4	COMMAND FRAME-SPECIFIC FUNCTIONS.....	57
4.4.2.4.1	dgGetCmd.....	57
4.4.2.4.2	dgSetCmd.....	57
4.4.2.4.3	dgSetCmdData.....	58
4.4.2.5	RESPONSE FRAME-SPECIFIC FUNCTIONS.....	59
4.4.2.5.1	dgGetRespCmd.....	59

4.4.2.5.2	dgGetReturnCode.....	59
<b>4.5</b>	<b>Network Services - Advanced.....</b>	<b>60</b>
4.5.1	Scheduler Functions.....	60
4.5.1.1	dgCreateScheduleHandle.....	61
4.5.1.2	dgStartSchedule.....	61
4.5.1.3	dgModifyScheduleEntry.....	61
4.5.1.4	dgStopSchedule.....	62
4.5.1.5	dgScheduleActive.....	62
4.5.1.6	dgKillSchedules.....	63
4.5.1.7	dgSetCritical.....	63
4.5.1.8	dgSetIterations.....	63
4.5.1.9	dgAddEntry.....	64
4.5.1.10	dgCreateScheduleEntryHandle.....	64
4.5.1.11	dgSetMsgFrame.....	64
4.5.1.12	dgScheduleEntrySetChannel.....	65
4.5.1.13	dgSetInterval.....	65
4.5.1.14	dgSetCount.....	65
4.5.1.15	dgSetDelay.....	66
4.5.2	Responder Functions.....	67
4.5.2.1	dgCreateResponderHandle.....	67
4.5.2.2	dgAddResponder.....	68
4.5.2.3	dgDeleteResponder.....	68
4.5.2.4	dgKillResponders.....	68
4.5.2.5	dgResponderSetChannel.....	69
4.5.2.6	dgResponderSetActive.....	69
4.5.2.7	dgResponderSetAction.....	70
4.5.2.8	dgResponderSetActionValue.....	71
4.5.2.9	dgResponderAddFilterblock.....	72
4.5.2.10	dgResponderAddResponseFrame.....	72
4.5.2.11	dgResponderSetReplacement.....	73
4.5.2.12	dgResponderPresent.....	73
4.5.3	USDT Functions.....	74
4.5.3.1	dgCreateUSDTHandle.....	74
4.5.3.2	dgSetActive.....	74
4.5.3.3	dgReportEventId.....	74
4.5.3.4	dgSetDataLink.....	75
4.5.3.5	dgSetPad.....	76
4.5.3.6	dgSetIdSize.....	76
4.5.3.7	dgSetExtAddrIds.....	76
4.5.3.8	dgSetIdBlock.....	77
4.5.3.9	dgSetFilterMode.....	78
4.5.3.10	dgEnableLoopback.....	78
4.5.3.11	dgReportAllEvents.....	78
<b>4.6</b>	<b>Miscellaneous Services.....</b>	<b>79</b>

<b>4.6.1</b>	<b>Program Loader Functions</b> .....	<b>79</b>
4.6.1.1	dgAddFile .....	79
4.6.1.2	dgDeleteFile .....	80
4.6.1.3	dgStartProgram .....	80
4.6.1.4	dgStopProgram .....	80
<b>4.6.2</b>	<b>dgFile Functions</b> .....	<b>81</b>
4.6.2.1	dgCreateFileHandle .....	81
4.6.2.2	dgSetName.....	81
4.6.2.3	dgSetPath.....	81
4.6.2.4	dgSetDescription .....	82
4.6.2.5	dgSetExecutable .....	82
<b>4.6.3</b>	<b>dgClient Functions</b> .....	<b>83</b>
4.6.3.1	dgCreateClientHandle .....	83
<b>4.6.4</b>	<b>dgGryphonNode Functions</b> .....	<b>84</b>
4.6.4.1	dgCreateGryphonNodeHandle .....	84
4.6.4.2	dgGetNodeType.....	84
4.6.4.3	dgSetNodeType .....	84
4.6.4.4	dgGetNodeId.....	85
4.6.4.5	dgSetNodeId .....	85
<b>APPENDIX A: VISUAL BASIC NOTES.....</b>		<b>86</b>





# 1 INTRODUCTION

## 1.1 Document organization and format

**Chapter 1 – Introduction** - Provides an overview of the manual. This section of the introduction summarizes the remaining chapters and appendices.

**Chapter 2 – C Library Installation** - Describes the procedures necessary for successful installation and operation of the Gryphon C Library.

**Chapter 3 – C Library Overview** – Describes the structure and concepts of the Gryphon C Library.

**Chapter 4 – C Library Reference** – Details the syntax and semantics of each function in the Gryphon C Library.

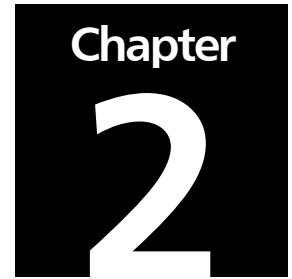
## 1.2 Technical support

In the U.S., technical support representatives are available to answer your questions between 9 a.m. and 5 p.m. EST. You may also fax or e-mail your questions to us. Please include your [voice] telephone number, for prompt assistance. Non-U.S. users may choose to contact their local representatives.

Phone: (248) 488-2080  
 Fax: (248) 488-2082  
 e-mail: techsupp@dgtech.com  
 web site: <http://www.dgtech.com>

## 1.3 References

Ref.	Title	Rev/Date/URL
1	Gryphon Installation Manual	Version 1.0
2	Gryphon Hardware & Communication Manual	<a href="http://&lt;Gryphon IP address&gt;/html/">http://&lt;Gryphon IP address&gt;/html/</a>



## 2 C Library Installation

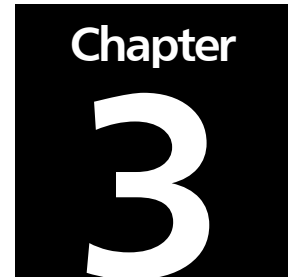
Please complete the registration card, and return it via fax or mail. You may also log on to our Web site to register this product on-line. As a registered user, you will receive technical support and important product upgrade information.

### 2.1 Installation for Windows Platforms

- Insert the Setup CD into the CD-ROM drive.
- Click on Start, Run...
- Type D:\setup.exe (If "D" is your PC's CD-ROM label; otherwise replace "D" with the correct label for your CD-ROM drive); then choose OK.
- Follow the instructions on the screen.

### 2.2 Installation for Unix Platforms

A source code distribution is available for Unix (and other) platforms, which includes suggested Makefiles. Please refer to the UNIX\_README file in the distribution for details.

A black square with the word "Chapter" in white at the top and a large white number "3" in the center.

## 3 C Library Overview

The Gryphon C Library was developed by the Dearborn Group to provide a high level interface to Gryphon services for application development using the C language, and other C-interface compatible languages like Visual BASIC and LabWindows/CVI. Current platforms supported include Linux (and most Unix variants) with pthread support, and all 32-bit versions of MS Windows with threading support.

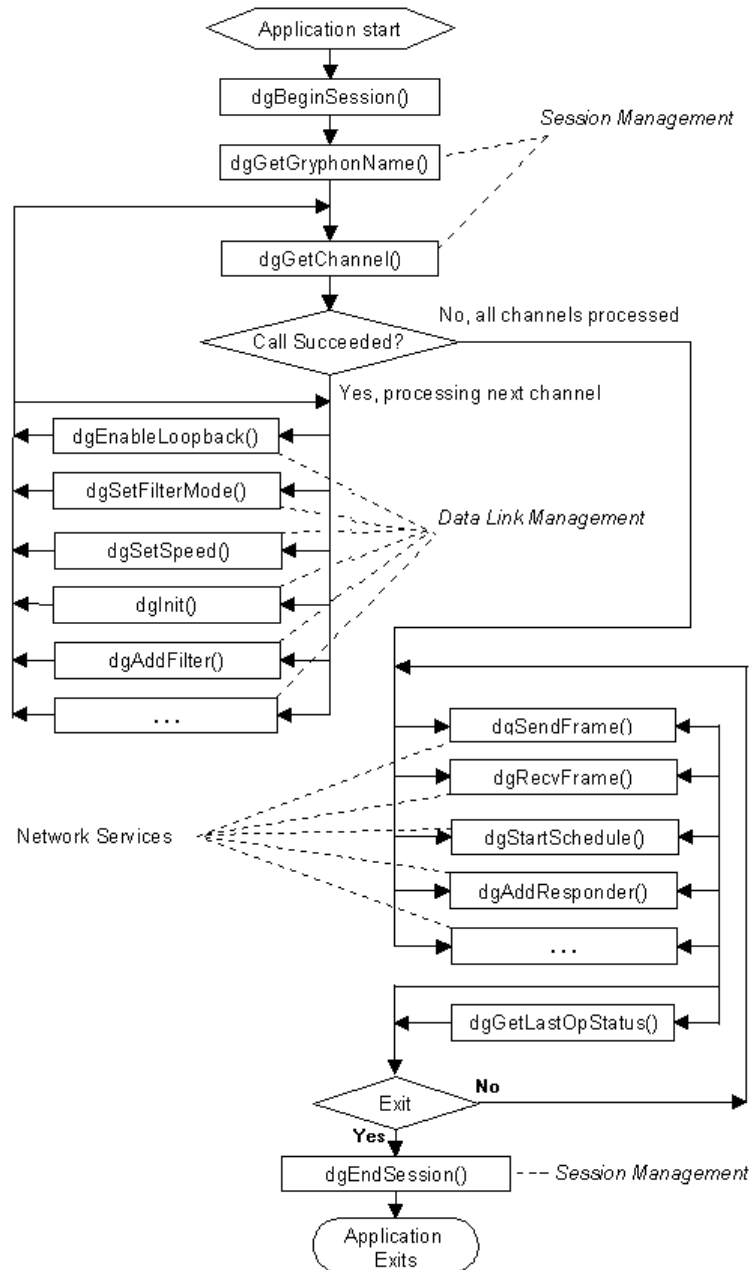
The Gryphon C Library is a thin wrapper over the Gryphon C++ Library. The main motivation for its creation was the wider compatibility with other languages that the C-interface provides. Because of its dependency on the C++ Library, the reader is often referred in this manual to the C++ Library manual for further functional details.

### 3.1 General Notes

- All parameter data are in host byte order; transformations to and from network byte order are handled by the library.
- "GC frame" is shorthand for "Gryphon Communication Protocol message frame".
- All GC frame padding is handled by the library.
- "Payload" or "payload message" refers to a message sent or received over the network connected to a Gryphon channel; e.g., a CAN message.
- Function declarations are in file "dggryphC.h".
- Buffers returned from functions are always null-terminated and truncated to the length passed into parameter `max_len`, with the exception of `dgFrameGetData`.
- Unless otherwise specified, all functions return a boolean value to determine if the function succeeded or failed.

## 3.2 Library Structure

### 3.2.1 Flowchart of Typical Use



### 3.2.2 Handles to Objects

The library makes extensive use of different custom handles. Each handle is used to reference some type of library object, e.g. a Gryphon session, a Gryphon channel, a channel filter, etc.

All handles, except for `dgSessionHandle`, `dgEventTypeHandle`, and `dgChannelHandle`, are created by explicitly calling a create function; e.g. **`dgCreateScheduleHandle()`**. To create a `dgSessionHandle`, `dgEventTypeHandle`, or `dgChannelHandle` call the following functions:

**`dgBeginSession()`** for `dgSessionHandle`  
**`dgGetEventType()`** for `dgEventTypeHandle`  
**`dgGetChannel()`** for `dgChannelHandle`

To free any handle, call the function **`dgFreeHandle()`**.

The first parameter in most functions in this library requires a handle. If an invalid handle or a handle of the wrong type is passed to a function, it will fail and `dgGetLastOpStatus()` will return `RESP_INVALID_PARAM`.

Please see the documentation below for more information on each function and handle type.

#### Handle Types:

The following are the handles that are used in this library.

<b><code>dgSessionHandle</code></b>	Represents the client/server session with the Gryphon device.
<b><code>dgClientHandle</code></b>	A client process of the Gryphon server.
<b><code>dgChannelHandle</code></b>	A vehicle network channel on the Gryphon.
<b><code>dgChannelSpeedHandle</code></b>	Used in setting a channel speed.
<b><code>dgFrameHandle</code></b>	A GC frame.
<b><code>dgFilterHandle</code></b>	A specification of a Gryphon message filter, which provides the capability to filter messages received over a vehicle network channel.
<b><code>dgFilterBlockHandle</code></b>	Used in building a message filter.
<b><code>dgScheduleHandle</code></b>	A specification of a Gryphon message schedule, which is a list of messages to be transmitted over a channel, with iteration and interval capabilities.
<b><code>dgScheduleEntryHandle</code></b>	Used in building a Gryphon message schedule.

<b>dgResponderHandle</b>	A Gryphon message responder.
<b>dgEventTypeHandle</b>	Used in managing and interpreting channel event notifications.
<b>dgFileHandle</b>	Represents a file in the local file system, which can be uploaded to the Gryphon.
<b>dgGryphonNodeHandle</b>	Generic representation of any network endpoint on the Gryphon Network.
<b>dgUSDTHandle</b>	A reference to a USDT node. Stated another way, it is used to access the services provided by the Gryphon USDT server, which are essentially ISO 15765-2 "Network Layer" services.



### 4.1.2 dgLibGetVersion

<b>Function</b>	Get the version string of the library.
<b>Syntax</b>	void <b>dgLibGetVersion</b> ( unsigned char * <b>str</b> , size_t <i>max_len</i> )
<b>Remarks</b>	<i>str</i> points to the library version string after function return.

**Example:**

```
size_t max_len = 1024;
unsigned char tempstr[max_len];
dgLibGetVersion ( tempstr, max_len );
```

### 4.1.3 dgFreeHandle

<b>Function</b>	Frees any handle created by this library and returns the resources to the system.
<b>Syntax</b>	int <b>dgFreeHandle</b> ( dgHandle )
<b>Remarks</b>	If this function is called passing a dgSessionHandle, the function will free the dgSessionHandle as well as all handles of type dgChannelHandle associated with it.

**Example:**

```
dgSessionHandle mygryphon;
result = dgFreeHandle( mygryphon );
```



**Example:**

Example changing the IP address:

```
strcpy( dgDefaultSession.gryphon_addr, "192.168.1.100" );
gryph_session = dgBeginSession( &dgDefaultSession );
```

**4.2.2 dgEndSession**

**Function** End the TCP session with the Gryphon server.

**Syntax** int **dgEndSession** ( dgSessionHandle, int *flush* )

**Remarks** Parameter *flush* is a boolean value: if the value supplied is TRUE, any incoming frames previously enqueued by the library, and not yet retrieved by **dgRecvFrame()**, are deleted. Otherwise, enqueued frames can be retrieved after the call to **dgEndSession()**

**Example:**

```
result = dgEndSession(my_session, TRUE);
```

**4.2.3 dgSessionActive**

**Function** Check if the TCP session is still valid and the client is still connected to the Gryphon.

**Syntax** int **dgSessionActive** ( dgSessionHandle )

**Remarks** None.

**Example:**

```
if ( dgSessionActive( my_session ) ) {
    // Session is active
}
else {
    // Session is NOT active
}
```

#### 4.2.4 dgGryphonGetName

<b>Function</b>	Get the name of the connected Gryphon server.
<b>Syntax</b>	int <b>dgGryphonGetName</b> ( dgSessionHandle, unsigned char * <i>str</i> , size_t <i>max_len</i> )
<b>Remarks</b>	If this function returns TRUE, then <i>str</i> contains the Gryphon Name.

##### Example:

```
unsigned char tempstr[1024];
size_t max_len;
max_len=1024;
result = dgGryphonGetName (my_session, tempstr, max_len);
```

#### 4.2.5 dgGryphonGetSerial

<b>Function</b>	Get the serial number (Ethernet MAC address) of the Gryphon.
<b>Syntax</b>	int <b>dgGryphonGetSerial</b> (dgSessionHandle, unsigned char * <i>str</i> , size_t <i>max_len</i> )
<b>Remarks</b>	If this function returns TRUE, then <i>str</i> contains the Gryphon serial number.

##### Example:

```
unsigned char m_tempstr[1024];
size_t max_len;
max_len=1024;
result = dgGryphonGetSerial (my_session, m_tempstr, max_len);
```

## 4.2.6 dgGryphonGetVersion

<b>Function</b>	Get the version number of the Gryphon server.
<b>Syntax</b>	int <b>dgGryphonGetVersion</b> (dgSessionHandle, unsigned char * <b>str</b> , size_t <b>max_len</b> )
<b>Remarks</b>	If this function returns TRUE, then <i>str</i> contains the Gryphon version.

### Example:

```
unsigned char tempstr[1024];
size_t max_len;
max_len=1024;
result = dgGryphonGetVersion (my_session, tempstr, max_len);
```

## 4.2.7 dgGetId

<b>Function</b>	Get the identifier assigned by the Gryphon server for the session.
<b>Syntax</b>	int <b>dgGetId</b> ( dgSessionHandle, unsigned char *)
<b>Remarks</b>	Normally, the user registers with the Gryphon server as a client (SD_CLIENT); this is the default value of the <i>srcType</i> parameter of <b>BeginSession</b> (). In this case, this function will retrieve the "client Id" which will be valid for the duration of the session.

### Example:

```
unsigned char my_clientId;
result = dgGetId( my_session, &my_clientId );
```

### 4.2.8 dgGetChannel

<b>Function</b>	Get a handle to a channel on the connected Gryphon.
<b>Syntax</b>	dgChannelHandle <b>dgGetChannel</b> ( dgSessionHandle, unsigned char <i>index</i> )
<b>Remarks</b>	<p>The parameter <i>index</i> is a zero-based integer index into the array of hardware channels installed on the Gryphon.</p> <p>This function creates a <b>dgChannelHandle</b> and links it to the associated hardware channel. It returns 0 on error. If the function returns a non-zero handle, the handle can be used to access the hardware channel's information and services.</p>

#### Example:

```
dgChannelHandle my_channel;
my_channel = dgGetChannel( my_session, 0 );
```

### 4.2.9 dgSetTimeout

<b>Function</b>	Set the timeout period for all of the Gryphon library functions.
<b>Syntax</b>	int <b>dgSetTimeout</b> ( dgSessionHandle, unsigned int <i>ms</i> )
<b>Remarks</b>	<p>If set to zero, the function will return immediately. If the operation did not complete, the member function dgGetLastOpStatus() will return a RESP_TIMEOUT value.</p> <p>The specified timeout is in effect for the duration of the session, or until the function is called again with a different timeout value.</p> <p>The library's default timeout value is 500 ms.</p>

#### Example:

```
Example using a numeric value:
result = dgSetTimeout(my_session, 1000); //timeout = 1000 ms = 1 s.
```

Example using a variable:  
 unsigned int m\_timeout=0;  
 result = dgSetTimeout(my\_session, m\_timeout); //timeout = 0 ms.

#### 4.2.10 dgEnableBroadcastRx

**Function** Enable or disable reception of broadcast messages.

**Syntax** int **dgEnableBroadcastRx** ( dgSessionHandle, int *flag* )

**Remarks** The parameter *flag* is a boolean value which enables or disables reception of broadcast messages.

**Example:**

```
result = dgEnableBroadcastRx(my_session, TRUE);
// Enable reception of broadcast
// messages
```

#### 4.2.11 dgSetCommOpt

**Function** Set the optimization of the TCP link between the Gryphon server and this client.

**Syntax** int **dgSetCommOpt** ( dgSessionHandle, unsigned char *flags* )

**Remarks** Optimization can be set to minimize latency (Nagle algorithm disabled) or maximize throughput (Nagle algorithm enabled).

*flags* is a bitfield which can represent these values:

COMMOPT\_THROUGHPUT: Maximize throughput (default)

COMMOPT\_LATENCY\_SRVR: Minimize latency of server-to-client communication (disable Nagle algorithm at the server socket)

COMMOPT\_LATENCY\_CLNT: Minimize latency of client-to-server communication (disable Nagle algorithm at the client)

socket)

The use of a "latency" flag will override the "throughput" optimization in the selected direction of communication.

**Example:**

```
// optimize latency in both directions:
result = dgSetCommOpt( my_session, COMMOPT_LATENCY_SRVR +
                        COMMOPT_LATENCY_CLNT );
```

### 4.2.12 dgSetSortMode

**Function** Set the server's data message sorting behavior.

**Syntax** int **dgSetSortMode** ( dgSessionHandle,  
unsigned char *mode* )

**Remarks** *mode:*  
0 No sorting performed (default)  
1 Block of up to 16 messages sorted.

**Example:**

```
result = dgSetSortMode(my_session, 1); // Turn sorting on
```

### 4.2.13 dgFlushQueue

**Function** Empty the received frame queue.

**Syntax** int dgFlushQueue ( dgSessionHandle )

**Remarks** Empties the queue of GC frames received by the library from the connected Gryphon, but not yet retrieved by the application via calls to **dgRecvFrame()**.

**Example:**

```
if (dgFlushQueue( my_session ) ) {
    // queue is emptied
}
```

#### 4.2.14 dgGryphonGetTime

**Function** Get the Gryphon time value, in seconds.

**Syntax** int **dgGryphonGetTime** (dgSessionHandle, time\_t\*)

**Remarks** time\_t will retrieve the 32- bit time in seconds (standard C time).

**Example:**

```
time_t m_time;
result = dgGryphonGetTime(my_session, &m_time);
// 32-bit time (seconds)
```

#### 4.2.15 dgGryphonSetTime

**Function** Set the Gryphon time value, in seconds.

**Syntax** int **dgGryphonSetTime** ( dgSessionHandle, time\_t )

**Remarks** Setting the Gryphon time with this function sets the fractional portion of the Gryphon time to zero.

**Example:**

```
time_t m_time;
// Initialize m_time
result = dgGryphonSetTime(my_session, m_time);
```

#### 4.2.16 dgGryphonGetTimestamp

**Function** Get the 32- bit Gryphon timestamp.

**Syntax** int **dgGryphonGetTimestamp**( dgSessionHandle, unsigned int\*)

**Remarks** None.

**Example:**

```
unsigned int m_timestamp;  
result = dgGryphonGetTimestamp(my_session, &m_timestamp);
```

**4.2.17 dgGryphonSetTimestamp**

**Function** Set the 32-bit Gryphon timestamp.

**Syntax** int **dgGryphonSetTimestamp** (dgSessionHandle, unsigned  
int )

**Remarks** None.

**Example:**

```
unsigned int m_timestamp;  
// Initialize m_timestamp  
result = dgGryphonSetTimestamp( my_session, m_timestamp );
```

## 4.3 Data Link Management

Functions in this section provide information about a Gryphon's installed data link hardware, and control of the data link operating parameters.

### 4.3.1 Device Identification

#### 4.3.1.1 dgChannelGetName

<b>Function</b>	Get the name of the channel device.
<b>Syntax</b>	int <b>dgChannelGetName</b> ( dgChannelHandle, unsigned char * <b>str</b> , size_t <b>max_len</b> )
<b>Remarks</b>	If this function returns TRUE, then <b>str</b> points to the name of the channel.

**Example:**

```
unsigned char tempstr[1024];
size_t max_len;
max_len=1024;
result = dgChannelGetName (my_channel, tempstr, 1024 );
```

#### 4.3.1.2 dgChannelGetSecurity

<b>Function</b>	Get the encrypted device security string of the channel device.
<b>Syntax</b>	int <b>dgChannelGetSecurity</b> ( dgChannelHandle, unsigned char * <b>str</b> , size_t <b>max_len</b> )
<b>Remarks</b>	If this function returns TRUE, then <b>str</b> points to the security string.

**Example:**

```
unsigned char tempstr[1024];
size_t max_len;
max_len=1024;
result = dgChannelGetSecurity (my_channel, tempstr, 1024 );
```

### 4.3.1.3 dgChannelGetSerial

<b>Function</b>	Get the serial number of the channel device.
<b>Syntax</b>	int <b>dgChannelGetSerial</b> ( dgChannelHandle, unsigned char * <i>str</i> , size_t <i>max_len</i> )
<b>Remarks</b>	If this function returns TRUE, then <i>str</i> contains the serial number.

#### Example:

```
unsigned char tempstr[1024];
size_t max_len;
max_len=1024;
result = dgChannelGetSerial (my_channel, tempstr, 1024 );
```

### 4.3.1.4 dgChannelGetVersion

<b>Function</b>	Get version identification of the channel device.
<b>Syntax</b>	int <b>dgChannelGetVersion</b> ( dgChannelHandle, unsigned char * <i>str</i> , size_t <i>max_len</i> )
<b>Remarks</b>	If this function returns TRUE, then <i>str</i> contains the version.

#### Example:

```
unsigned char tempstr[1024];
size_t max_len;
max_len=1024;
result = dgChannelGetVersion (my_channel, tempstr, 1024 );
```

### 4.3.1.5 dgChannelGetSlot

<b>Function</b>	Get the slot number of the channel device.
<b>Syntax</b>	int <b>dgChannelGetSlot</b> ( dgChannelHandle, unsigned char* <i>slot</i> )
<b>Remarks</b>	None.

**Example:**

```
unsigned char m_slotnum;
result = dgChannelGetSlot(my_channel, &m_slotnum);
```

**4.3.1.6 dgChannelGetType**

**Function** Get the channel type (e.g. CAN, J1850, etc)

**Syntax** int **dgChannelGetType** ( dgChannelHandle,  
unsigned char\* *type* )

**Remarks** None.

**Example:**

```
unsigned char m_type;
result = dgChannelGetType(my_channel, &m_type);
```

**4.3.1.7 dgChannelGetSubtype**

**Function** Get the subtype of the channel.

**Syntax** int **dgChannelGetSubtype**( dgChannelHandle,  
unsigned char\* *subtype* )

**Remarks** None.

**Example:**

```
unsigned char m_subtype;
result = dgChannelGetSubtype(my_channel, &m_subtype);
```

**4.3.1.8 dgGetDataMaxLen**

**Function** Get the maximum length, in bytes, of the "data" portion of the channel's network frame format.

**Syntax** int **dgGetDataMaxLen** ( dgChannelHandle,  
unsigned short\* *length* )

**Remarks** None.

**Example:**

```
unsigned short m_maxlen;
result = dgGetDataMaxLen(my_channel, &m_maxlen);
```

**4.3.1.9 dgGetDataMinLen**

**Function** Get the minimum length, in bytes, of the "data" portion of the channel's network frame format.

**Syntax** int **dgGetDataMinLen** ( dgChannelhandle,  
unsigned short\* *length* )

**Remarks** None.

**Example:**

```
unsigned short m_minlen;
result = dgGetDataMinLen(my_channel, &m_minlen);
```

**4.3.1.10 dgGetExtraMaxLen**

**Function** Get the maximum length, in bytes, of the "extra" portion of the channel's network frame format.

**Syntax** int **dgGetExtraMaxLen** ( dgChannelHandle,  
unsigned short\* *length* )

**Remarks** None.

**Example:**

```
unsigned short m_extramaxlen;
result = dgGetExtraMaxLen(my_channel, &m_extramaxlen);
```

**4.3.1.11 dgGetExtraMinLen**

**Function** Get the minimum length, in bytes, of the "extra" portion of the channel's network frame format.

**Syntax** int **dgGetExtraMinLen** ( dgChannelHandle,  
unsigned short\* *length* )

unsigned short\* *length* )

**Remarks** None.

**Example:**

```
unsigned short m_extraminlen;
result = dgGetExtraMinLen(my_channel, &m_extraminlen);
```

#### 4.3.1.12 dgGetValidHeaderLength

**Function** Get the valid header lengths for the channel's network frame format.

**Syntax** int **dgGetValidHeaderLength**( dgChannelHandle,  
unsigned char *index*,  
size\_t\* *length* )

**Remarks** Through iterated calls, starting with *index* set to zero, this function returns all the valid header lengths for the channel's network frame format. When zero return value is obtained, the list has been exhausted.

**Example:**

```
size_t firstheaderlength;
result = dgGetValidHeaderLength( my_channel, 0, &firstheaderlength);
```

#### 4.3.1.13 dgGetNodeId

**Function** Get the numeric identifier of the channel node.

**Syntax** int **dgGetNodeId**( dgChannelHandle,  
unsigned char \**id* );

**Remarks** *id* can be used to determine the source channel of a GC frame received via **dgRecvFrame**(). If the frame originated from this channel, the value of *id* will match the value returned by **dgGetSrcChannel**() for that frame handle.

### 4.3.2 General Configuration

#### 4.3.2.1 dgSetFilterMode

**Function** Specify whether messages received by the node are to be blocked, filtered or passed to the client application.





***boole:***

0 = Reports no events to the client (default).

1 = Reports all events to the client.

**Example:**

```
result = dgReportAllEvents ( my_channel, 1 );
```

**4.3.3.2 dgReportEventType**

<b>Function</b>	Specifies if an event type is to be delivered to the client application, or if it should be filtered out.
<b>Syntax</b>	int <b>dgReportEventType</b> ( dgChannelHandle, dgEventTypeHandle, int <b><i>flag</i></b> )
<b>Remarks</b>	The parameter <b><i>flag</i></b> is a boolean value which determines if the Event Type should be filtered out or not.  <b><i>flag</i></b> value: FALSE = Do <b>NOT</b> Report the Event Type to the client. TRUE = Report the Event Type to the client.

**Example:**

```
dgEventTypeHandle m_event;
//m_event is initialized
result = dgReportEventType( my_channel, m_event, TRUE )
// Events of type "m_event" will be reported

result = dgReportEventType( my_channel, m_event, FALSE );
// Events of type "m_event" will NOT be reported
```

**4.3.3.3 dgGetEventType**

<b>Function</b>	Create a dgEventTypeHandle in order to get information on the type of events that may be signaled by the channel.
<b>Syntax</b>	dgEventTypeHandle <b>dgGetEventType</b> (dgChannelHandle, unsigned char <b><i>index</i></b> )

**Remarks** The function returns 0 on error, or a dgEventTypeHandle on success. If a non-zero handle is returned it can be used in other functions to represent the event type.

**Example:**

```
dgEventTypeHandle m_event;
m_event = dgGetEventType(my_channel, 0);
```

#### 4.3.3.4 dgGetEventTypeId

**Function** Get the numeric identifier for the event type.

**Syntax** int **dgGetEventTypeId**( dgEventTypeHandle, dgEventId\* )

**Remarks** None.

**Example:**

```
dgEventId my_eventId;
result = dgGetEventTypeId( my_event, &my_eventId);
```

#### 4.3.3.5 dgGetEventTypeMeaning

**Function** Get the text string description of the event.

**Syntax** int **dgGetEventTypeMeaning**( dgEventTypeHandle,  
unsigned char \**str*,  
size\_t *max\_len* )

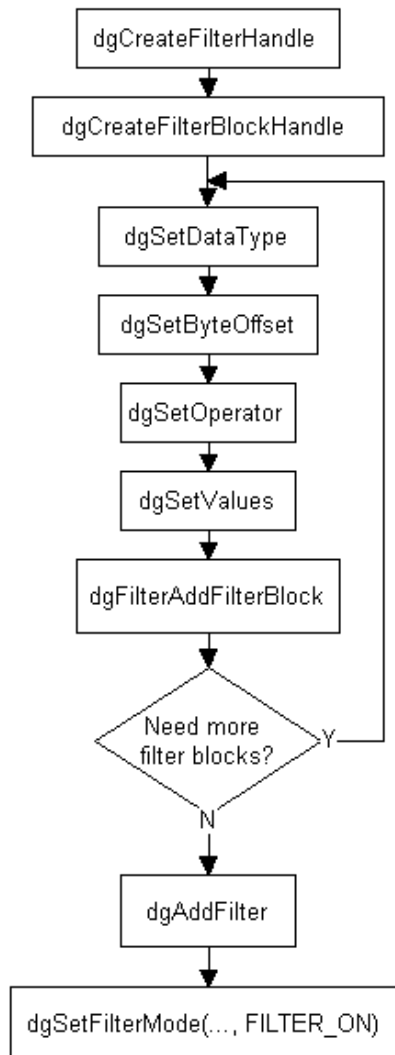
**Remarks** None.

**Example:**

```
unsigned char meaning[1024];  
size_t max_len = 1024;  
result = dgGetEventTypeMeaning( my_event, meaning, max_len );
```

### 4.3.4 Message Filter Management

For an overview, the following flowchart illustrates a typical way to set up a message filter:



#### 4.3.4.1 dgCreateFilterHandle

**Function** Create and return a dgFilterHandle.

**Syntax** dgFilterHandle **dgCreateFilterHandle()**

**Remarks** None.

**Example:**

```
dgFilterHandle myfilter = dgCreateFilterHandle();
```

#### 4.3.4.2 dgAddFilter

**Function** Add the filter represented by the dgFilterHandle, to the channel.

**Syntax** int **dgAddFilter** ( dgChannelHandle, dgFilterHandle )

**Remarks** None.

**Example:**

```
dgFilterHandle myfilter;
// myfilter is initialized and set with a dgFilterBlockHandle
result = dgAddFilter(my_channel, myfilter);
```

#### 4.3.4.3 dgModifyFilter

**Function** Modify the previously added filter.

**Syntax** int **dgModifyFilter** ( dgChannelHandle, dgFilterHandle, unsigned char *action* )

**Remarks** *action* values: ACTIVATE\_FILTER, DEACTIVATE\_FILTER

#### 4.3.4.4 dgDeleteFilter

**Function** Delete the filter from the channel.



```
result = dgFilterSetAction ( my_filter, FILTER_FLAG_PASS );
```

#### 4.3.4.7 dgSetDefaultFilterAction

**Function** Specify the action to be taken when a message fails to conform to all of the active filters for the channel.

**Syntax** int **dgSetDefaultFilterAction**( dgChannelHandle,  
unsigned char *action* )

**Remarks** *action*:

DEFAULT\_FILTER\_BLOCK : messages not conforming to any of the active filters will not be passed to the client. This is the default state.

DEFAULT\_FILTER\_PASS : messages not conforming to any of the active filters will be passed to the client.

#### Example:

```
result = dgSetDefaultFilterAction(my_channel, DEFAULT_FILTER_PASS);
```

#### 4.3.4.8 dgCreateFilterBlockHandle

**Function** Create and return a dgFilterBlockHandle.

**Syntax** dgFilterBlockHandle **dgCreateFilterBlockHandle**()

**Remarks**

#### Example:

```
dgFilterHandle myfilter;  
myfilter = dgCreateFilterBlockHandle();
```

#### 4.3.4.9 dgFilterAddFilterBlock

**Function** Add a filter block to a filter.

**Syntax** int **dgFilterAddFilterBlock** ( dgFilterHandle,  
dgFilterBlockHandle )

**Remarks** Filter blocks specify a section of the message to be checked. In order for a message to conform to a filter, it must conform to all of the defined filter blocks associated with that filter.

This function involves a copy operation performed on the filter block object referenced by the second parameter. Once the copy is made, the filter block might be modified and reused in the same or another filter object. Thus, changes made to the filter block after the call to **dgFilterAddFilterBlock()** have no effect on the referenced filter object.

**Example:**

```
dgFilterBlockHandle my_block;
// my_block is initialized
result = dgFilterAddFilterblock(my_filter, my_block);
```

#### 4.3.4.10 dgSetDataType

**Function** Specify which portion of the message the Byte Offset field references.

**Syntax** int **dgSetDataType** ( dgFilterBlockHandle,  
unsigned char *datatype* )

**Remarks** *datatype*:

FILTER\_DATA\_TYPE\_HEADER\_FRAME – Message Header Frame (prepended to the message by the Gryphon)  
 FILTER\_DATA\_TYPE\_HEADER -- Message Header  
 FILTER\_DATA\_TYPE\_DATA -- Message Data  
 FILTER\_DATA\_TYPE\_EXTRA\_DATA – Message extra data

**Example:**

```
result = dgSetDataType( my_filterblock, FILTER_DATA_TYPE_DATA );
```

#### 4.3.4.11 dgSetByteOffset

<b>Function</b>	Identify the location of the section of the message to be checked.
<b>Syntax</b>	int <b>dgSetByteOffset</b> ( dgFilterBlockHandle, unsigned short <i>offset</i> )
<b>Remarks</b>	The Byte Offset is from the beginning of the portion of the message specified in the Data Type Index. Note that the message header is right justified in its byte field.

#### Example:

```
unsigned short m_offset;
result = dgSetByteOffset ( my_filterblock, 0 );

// m_offset is initialized
result = dgSetByteOffset ( my_filterblock, m_offset );
```

#### 4.3.4.12 dgSetOperator

<b>Function</b>	Specify the type of comparison to be done in this filter block.			
<b>Syntax</b>	int <b>dgSetOperator</b> ( dgFilterBlockHandle, unsigned char <i>operator</i> )			
<b>Remarks</b>	<i>operator</i>	<b>Value 1</b>	<b>Value 2</b>	<b>Notes</b>
	BIT_FIELD_CHECK	Pattern	Mask	Bit check with 0, 1 and don't care
	SVALUE_GT	Value	Not used	Signed value compare
	SVALUE_GE	Value	Not used	Signed value compare
	SVALUE_LT	Value	Not used	Signed value compare
	SVALUE_LE	Value	Not used	Signed value compare
	VALUE_EQ	Value	Not used	Check for value equality
	VALUE_NE	Value	Not used	Check for value inequality
	UVALUE_GT	Value	Not used	Unsigned value compare
	UVALUE_GE	Value	Not used	Unsigned value compare

UVALUE_LT	Value	Not used	Unsigned value compare
UVALUE_LE	Value	Not used	Unsigned value compare
DIG_LOW_TO_HIGH	Bit Mask	Not used	Check for 0 to 1
DIG_HIGH_TO_LOW	Bit Mask	Not used	Check for 1 to 0
DIG_TRANSITION	Bit Mask	Not used	Check for change

**BIT\_FIELD\_CHECK**

A message conforms to a BIT\_FIELD\_CHECK Filter Block when the specified message byte(s) are identical those in the Pattern. The Mask allows portions of one or more message bytes to be ignored. Each bit set in the Mask indicates that the corresponding bit in the message must match the corresponding Pattern bit. Each bit cleared in the Mask indicates that the corresponding bit in the message may be either set or reset.

#### **SVALUE, VALUE and UVALUE comparisons**

A message conforms to one of these Filter Blocks when the specified message byte(s) is taken as a signed or unsigned character, or the short or long value (8, 16 or 32 bits) is greater than, etc. the value present in Value1.

#### **DIG comparisons**

A message conforms to one of these Filter Blocks when the specified bit makes a low to high or a high to low transition or changes state. The Bit Mask is used to isolate a single bit to be checked in a byte. If bit 0, the significant bit, is to be checked, the Bit Mask should be set to 1.

The filter mask qualifies the filter pattern by specifying which bits are required and which bits are to be ignored. A bit value of 1 indicates that the corresponding bit in the pattern needs to be matched. A bit value of 0 indicates the corresponding bit in the pattern is not checked (always matches).

#### **Example:**

```
result = dgSetOperator( my_filterblock, VALUE_EQ );
```

#### **4.3.4.13 dgSetValues**

**Function** Set the object's "First Value" attribute, and optionally the "Second Value" attribute.

**Syntax** int **dgSetValues**( dgFilterBlockHandle,  
unsigned short *length*,

```
unsigned char* value1,
unsigned char* value2 )
```

**Remarks**

This command sets the values by reading the number of bytes indicated by the length parameter from memory pointed to by the ‘value1’ parameter.

If the operator of the object has been set to “BIT\_FIELD\_CHECK”, then the ‘Second Value’ attribute is also set , likewise, by reading the number of bytes indicated by the length parameter from memory pointed to by the ‘value2’ parameter.

This rule implies that **dgSetOperator()** must be called before this function can succeed.

**Example:**

```
// dependant on what was used in SetOperator()

unsigned char m_val1, m_val2;
unsigned short m_num;

// m_num and m_val1 are initialized
result = dgSetValues ( my_filterblock, m_num, &m_val1 );

// m_num, m_val1 and m_val2 are initialized and SetOperator used
// BIT_FIELD_CHECK
result = dgSetValues ( my_filterblock, m_num, &m_val1, &m_val2 );
```

## 4.3.5 Channel Speed Management

### 4.3.5.1 dgGetSpeed

**Function** Get the current speed characteristics of the specified channel.

**Syntax** int **dgGetSpeed** (dgChannelHandle,  
dgChannelSpeedHandle *speed*)

**Remarks** If the function returns TRUE, then the Channel Speed information is stored in *speed* for use in other function calls.

**Example:**

```
dgChannelSpeedHandle my_chanspeed;
result=dgGetSpeed(my_channel, my_chanspeed);
```

### 4.3.5.2 dgSetSpeed

**Function** Set the current speed characteristics for the specified channel.

**Syntax** int **dgSetSpeed** ( dgChannelHandle,  
dgChannelSpeedHandle *speed*)

**Remarks** The bus speed, when modified with this command, takes effect only with a subsequent call to **dgInit()**

**Example:**

```
dgChannelSpeedHandle my_chanspeed;
result=dgGetSpeed(my_channel(0), my_chanspeed);
result=dgSetSpeed(my_channel(1), my_chanspeed);
```

#### 4.3.5.3 dgGetPresetSpeed

**Function** Get information on the set of predefined speeds for a channel.

**Syntax** dgChannelSpeedHandle **dgGetPresetSpeed**  
(dgChannelHandle,  
unsigned char *index* )

**Remarks** The parameter *index* is a zero-based reference to the preset speed to be retrieved.

The function returns 0 on error.

**Example:**

```
dgChannelSpeedHandle my_chanspeed;
my_chanspeed = GetPresetSpeed(my_channel, 0);
```

#### 4.3.5.4 dgGetGetSpeedIOctl

**Function** The returned parameter value is the first parameter of the IOctl function that may be used to manually retrieve the current network timing parameters for this channel.

**Syntax** int **dgGetGetSpeedIOctl**( dgChannelHandle,  
unsigned int\* *ioctl\_num* )

**Remarks** None.

**Example:**

```
unsigned int m_num;
if( dgGetGetSpeedIOctl(&m_num) )
    result=dgIOctl(m_num, ...);
```

#### 4.3.5.5 dgGetSetSpeedIOctl

**Function** The returned parameter value is the first parameter of the IOctl function that may be used to manually set the network timing parameters for his channel.

**Syntax** int **dgGetSetSpeedIOctl** ( dgChannelHandle,  
unsigned int\* *ioctl\_num* )

**Remarks** None.

##### Example:

```
unsigned int m_num;
if(dgGetSetSpeedIOctl(my_channel, &m_num))
    result=dgIOctl(my_channel, m_num, ...);
```

#### 4.3.5.6 dgGetIOctlData

**Function** Get the data which may be used in the data parameter of **dgIOctl()** when setting the channel speed.

**Syntax** int **dgGetIOctlData**( dgChannelHandle,  
unsigned char\* *buff*,  
size\_t *max*,  
size\_t\* *actual* )

**Remarks** Parameter *buff* points to the user's buffer; data bytes will be copied to this buffer. Parameter *max* specifies the maximum number of bytes to copy to the buffer. The actual number of bytes copied will be written to the size\_t pointed to by parameter *actual*.

##### Example:

```
unsigned char *m_databuffer;
size_t numread, total_size;
// initialize m_databuffer to an array of type unsigned char and set total_size
// to the size of m_databuffer
result = dgGetIOctlData( my_channel, m_databuffer, total_size, &numread);
```

#### 4.3.5.7 dgGetIOctlDataSize

**Function** Get the number of bytes of data which are expected to be used when calling **dgIOctl()** to set the channel speed.

**Syntax** Int **dgGetIOctlDataSize** (dgChannelHandle, size\_t\* )

**Remarks** None.

##### Example:

```
size_t m_datasize;
result = dgGetIOctlDataSize( my_channel, &m_datasize );
```

#### 4.3.5.8 dgGetSpeedDataSize

**Function** Get the number of bytes of data associated with each speed.

**Syntax** size\_t **dgGetSpeedDataSize** (dgChannelHandle)

**Remarks** None.

**Return Value** Value will return 0 if error occurred, else the number returned is this size of the speed data.

##### Example:

```
size_t m_datasize;
m_datasize=dgGetSpeedDataSize(my_channel);
```

## 4.3.6 Bus Load Monitoring

### 4.3.6.1 dgSetBLMParams

<b>Function</b>	Specify the bus load monitor behavior.
<b>Syntax</b>	int <b>dgSetBLMParams</b> (dgChannelHandle, unsigned int <i>mode</i> , unsigned int <i>averaging_period</i> )
<b>Remarks</b>	<p><b>mode</b>: Indicates the desired bus load monitor mode.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>0 BLM off</li> <li>1 BLM average over time</li> <li>2 BLM average over frame count</li> </ul> <p><i>averaging_period</i>: Indicates the period over which averages are calculated.</p> <p>If <i>mode</i>=1 (Time average): <i>averaging_period</i> is interpreted as a time period in units of milliseconds.</p> <p>If <i>mode</i>=2 (Frame count average): <i>averaging_period</i> is interpreted as a message count.</p>

#### Examples:

```
unsigned int m_avgperiod = 1000; // Bus load monitoring done over 1s.
result = dgSetBLMParams(my_channel, 1, m_avgperiod);
```

```
m_avgperiod=100; // Bus load monitoring done over next 100 messages.
result = dgSetBLMParams(my_channel, 2, m_avgperiod);
```

```
result = dgSetBLMParams( my_channel, 0, 0 ); // turn BLM off
```

### 4.3.6.2 dgGetBLMStats

<b>Function</b>	Get the BLM and other statistics.
<b>Syntax</b>	int <b>dgGetBLMStats</b> ( dgChannelHandle, unsigned int* <i>timestamp</i> , unsigned short* <i>blAvg</i> , unsigned short* <i>blNow</i> ,

	unsigned short* <i>blPeak</i> , unsigned short* <i>blHistoricPeak</i> , unsigned int* <i>rxCnt</i> , unsigned int* <i>txCnt</i> , unsigned int* <i>rxDropCnt</i> , unsigned int* <i>txDropCnt</i> , unsigned int* <i>rxErrCnt</i> , unsigned int* <i>txErrCnt</i> )
<b>Remarks</b>	The statistics received will include bus load, total frames received, total frames transmitted, total frames dropped by the device driver and average bus load.
<i>timestamp</i>	System time at which the last bus load reading occurred
<i>blAvg</i>	Average bus load on the channel
<i>blNow</i>	Current bus load on the channel
<i>blPeak</i>	TBD
<i>blHistoricPeak</i>	Highest bus load during the averaging period
<i>rxCnt</i>	Total frames received on a channel since the bus load monitor was enabled
<i>txCnt</i>	Total frames transmitted on a channel since the bus load monitor was enabled
<i>rxDropCnt</i>	Total received frames dropped since the bus load monitor was enabled
<i>txDropCnt</i>	Total transmitted frames dropped since the bus load monitor was enabled
<i>rxErrCnt</i>	Total received frame errors since the bus load monitor was enabled
<i>txErrCnt</i>	Total transmitted frame errors since the bus load monitor was enabled

## 4.4 Network Services - Primary

Functions in this section provide the primary methods for sending and receiving bus data frames over a Gryphon's installed data link hardware channels. Also included is the primary function to obtain asynchronous notification of bus data frame reception.

### 4.4.1 Principal Functions

#### 4.4.1.1 dgSendFrame

**Function** Transmit a frame via the target channel or to a target client.

**Syntax** `int dgSendFrame ( dgSessionHandle,  
dgGryphonNodeHandle target,  
dgFrameHandle frame)`

**Remarks** The parameter *target* can be a dgChannelHandle, a dgClientHandle, a dgUSDTHandle or a generic dgGryphonNodeHandle.

To broadcast a frame to all clients (which have enabled broadcast reception), use the value CH\_BROADCAST as the Id of the *target* dgClientHandle.

This function changes the 'Source' field of the Frame Header of the GC frame represented by *frame* to the value specified by parameter member *srcType* in the earlier invocation of **dgBeginSession()**.

This function changes the 'Source Channel' field of the Frame Header of the GC frame represented by *frame* to the value of the "Client ID" returned by the Gryphon server in the earlier invocation of **dgBeginSession()**.

This function changes the 'Destination' field of the Frame Header of the GC frame represented by *frame* to the value of the node type of the *target* Gryphon node. See also **dgGetNodeType()**.

This function changes the 'Destination Channel' field of the Frame Header of the GC frame represented by *frame* to the value of the node ID of the *target* Gryphon node. See also

**dgGetNodeId().****Example:**

```

dgDataFrame my_dataframe;
dgChannelHandle my_channel;
// my_dataframe and my_channel are initialized
result = dgSendFrame( my_session, my_channel, my_dataframe );

dgDataFrame my_dataframe;
dgClientHandle my_client;
// my_dataframe and my_client are initialized
result = dgSendFrame( my_session, my_client, my_dataframe );

```

**4.4.1.2 dgRecvFrame**

<b>Function</b>	Receive a GC frame.
<b>Syntax</b>	int <b>dgRecvFrame</b> ( dgSessionHandle, dgGryphonNodeHandle <i>source</i> , dgFrameHandle <i>frame</i> , unsigned char <i>cmdId</i> )
<b>Remarks</b>	<p>This function retrieves the next appropriate GC frame from the library receive frame queue. If no frames are queued, it waits (limited by the library timeout period) for the next appropriate frame to be received from the connected Gryphon.</p> <p>The appropriateness of the frame to be retrieved is determined by the parameters.</p> <p>The parameter <i>source</i> can be a dgChannelHandle, a dgClientHandle, a dgUSDTHandle or a generic dgGryphonNodeHandle, indicating the desired source of the frame. If <i>source</i> is NULL, then a frame from any source will be appropriate.</p> <p>The parameter <i>frame</i> indicates the type of frame desired. If the frame handle was created by the call “<b>dgCreateFrameHandle</b>( FT_DATA )”, for example, then FT_DATA is the appropriate type of frame to be retrieved from the queue. If the frame handle was created by the call</p>

“**dgCreateFrameHandle( 0 )** then a GC frame having *any* frame type -- other than FT\_RESP -- will be appropriate; e.g., FT\_DATA, or FT\_EVENT, or FT\_MISC, etc.

(The exception is made for frames of type FT\_RESP because the library itself, during processing of Gryphon Protocol commands on behalf of the client program, retrieves FT\_RESP frames from the receive frame queue; the client program, in these cases, would have no interest in retrieving these frames.)

The parameter *cmdId* is 0 in all cases except when attempting to receive a response frame. In this case, *cmdId* is the numeric identifier of the command for which you wish to receive a response.

**Receive Frame Queue** The receive frame queue is filled by a secondary thread, the main function of which is to wait for GC frames sent from the connected Gryphon, and enqueue those frames. The queue size is limited only by the size of the PC's virtual memory. Any error conditions detected in the reading thread are reported to the application program in the following way: **dgRecvFrame()** returns FALSE, and a descriptive message is obtained from **dgGetLastOpStatus()**.

**Queue overflow** If frames are not removed from the queue as fast as they are added then eventually the queue will overflow. The user is notified of this occurrence by a return value of FALSE from **dgRecvFrame()**, followed by the code RESP\_RX\_FAIL and the string "memory allocation failed" from **dgGetLastOpStatus()**. When the overflow occurs the oldest 10% of the queued frames are deleted in order to restore some virtual memory.

### Example:

```
dgDataFrame my_dataframe;
dgChannelHandle my_channel;
// my_dataframe and my_channel are initialized
result = dgRecvFrame( my_session, my_channel, my_dataframe, 0 );
```

```
dgDataFrame my_dataframe;
dgClientHandle my_client;
```

```
// my_dataframe and my_client are initialized
result = dgRecvFrame( my_session, my_client, my_dataframe, 0 );

dgRespFrame my_dataframe;
dgChannelHandle my_channel;
unsigned char cmdID;
// cmdID, my_dataframe, and my_channel are initialized
result = dgRecvFrame( my_session, my_channel, my_dataframe, cmdID );
```

#### 4.4.1.3 dgSetOnRx

**Function** Register a callback function, which will be called upon the receipt of a frame of the specified type from the specified source.

**Syntax** int **dgSetOnRx** ( dgSessionHandle,  
const dgGryphonNodeHandle *source*,  
PFV *callback*,  
dgFrameType *frametype* )

**Remarks** Specify the source by setting parameter *source* to a dgChannelHandle, dgClientHandle, dgUSDTHandle or dgGryphonNodeHandle. Set *source* to zero if any source is appropriate.

Set *callback* to point to your function serving as the callback.

Set *callback* to 0 to unregister a previously registered callback.

In the Win32 library, a callback will execute in a new thread, so the library is not blocked during callback execution.

#### Example:

```
dgSetOnRx( gryph_session, my_channel, OnEvent, FT_EVENT );

void OnEvent()
{
    dgRecvFrame(...); // retrieve the FT_EVENT frame
}
```

## 4.4.2 GC Frame Functions

A Gryphon Communication (GC) Frame is the unit of communication between user applications (clients) and the Gryphon (the server).

The following frame types are defined:

➤ Data Frame (frametype = FT\_DATA)

An FT\_DATA frame encapsulates a message sent over an automotive / industrial automation network. Such an encapsulated message may be a CAN or J1850 frame, for example.

The Gryphon server sends Data Frames to clients. The Data Frames contain the messages the Gryphon received over network channels.

Clients send Data Frames to the Gryphon server in order to transmit the encapsulated messages over network channels.

➤ Event Frame (FT\_EVENT)

The Gryphon sends out Event Frames to inform connected clients of certain events and error conditions, e.g., when a service requested by the client has been completed, or when an error condition has been detected on one of the channels.

➤ Command Frame (FT\_CMD)

Commands defined in the GC Protocol can be sent to the Gryphon in Command Frames.

Most of these Gryphon commands are encapsulated in library functions. E.g., the function **dgSetSortMode()** sends the command "CMD\_SERVER\_SET\_SORT" to the Gryphon server, and waits for the corresponding response. For this reason, most users of the library will probably never need to explicitly use FT\_CMD frames.

➤ Response Frame (FT\_RESP)

Responses to Command Frames are sent via Response Frames.

As in the case of Command Frames, most users of the library will probably never need to use FT\_RESP frames.

➤ Miscellaneous Frame (FT\_MISC)

Frames of type FT\_MISC may be used to transfer data of any format between Gryphon clients.

➤ Text Frame (FT\_TEXT)

An FT\_TEXT frame is similar to a FT\_MISC frame, but its message body is a null-terminated ASCII string.

In the library, GC frames are created and managed with the functions described in this section.

#### 4.4.2.1 GENERIC FRAME FUNCTIONS

##### 4.4.2.1.1 dgCreateFrameHandle

<b>Function</b>	Create and return a dgFrameHandle.
<b>Syntax</b>	dgFrameHandle <b>dgCreateFrameHandle</b> ( unsigned char <i>ftype</i> )
<b>Remarks</b>	This function can be called passing one of the following values, or alternatively passing 0 to create a generic GC Frame:  FT_DATA FT_EVENT FT_MISC FT_TEXT FT_CMD FT_RESP

##### Example:

```
dgFrameHandle myframe = dgCreateFrameHandle(FT_DATA);
```

##### 4.4.2.1.2 dgGetSrc

<b>Function</b>	Get the source type of the GC frame.
<b>Syntax</b>	int <b>dgGetSrc</b> ( dgFrameHandle, unsigned char* <i>src</i> )
<b>Remarks</b>	The source type field identifies the Gryphon source type of the sender of the frame.  Gryphon source (and destination) types include the following: SD_CARD indicates a (vehicle) network interface

SD\_SERVER indicates the Gryphon server  
 SD\_CLIENT indicates a client of the Gryphon server  
 SD\_SCHED indicates the Gryphon scheduler process

**Example:**

```
unsigned char src;
result=dgGetSrc( my_frame, &src );
```

**4.4.2.1.3 dgGetSrcChannel**

**Function** Get the source channel field of the GC frame.

**Syntax** int **dgGetSrcChannel**( dgFrameHandle,  
 unsigned char\* *srcchan* )

**Remarks** The source channel field of the GC frame is used by the Gryphon to uniquely identify a source/destination node, within a group of possible nodes having the same source type.

**Example:**

```
unsigned char src_chan;
result=dgGetSrcChannel( my_frame, &src_chan );
```

**4.4.2.1.4 dgSetDest**

**Function** Set the destination field of the GC frame.

**Syntax** int **dgSetDest**( dgFrameHandle, unsigned char *dest* )

**Remarks** Cf. **dgGetSrc()** for discussion of source/destination types.

**4.4.2.1.5 dgSetDestChannel**

**Function** Set the destination channel field of the GC frame.

**Syntax** int **dgSetDestChannel**( dgFrameHandle, unsigned char *destchan* )

**Remarks** Cf. `dgGetSrcChannel()` for discussion of source/destination channels.

#### 4.4.2.1.6 `dgFrameGetType`

**Function** Get the frame type of the GC frame.

**Syntax** `int dgFrameGetType ( dgFrameHandle,  
unsigned char* type )`

**Remarks** None.

#### Example:

```
unsigned char m_ftype;
result=dgFrameGetType ( my_frame, &m_ftype );
```

#### 4.4.2.1.7 `dgFrameGetData`

**Function** Get the data from a frame.

**Syntax** `int dgFrameGetData( dgFrameHandle,  
unsigned char* data,  
size_t maxlen,  
size_t* actual_len )`

**Remarks** This function applies to frames of type FT\_RESP, FT\_TEXT, and FT\_EVENT.

User sets *maxlen* to max desired length, ( e.g. buffer size); *actual\_len* is guaranteed  $\leq$  maxlen.

For FT\_TEXT frames, the buffer is returned with the null-terminated data from the frame.

For FT\_EVENT and FT\_RESP frames, the buffer is returned with the data from the frame, but it is **NOT** null-terminated.

#### Example:

```
unsigned char my_text[100];
size_t len;
```

```
result = dgFrameGetData( my_TEXTframe, my_text, 100, &len );
```

#### 4.4.2.1.8 dgFrameSetData

**Function** Set the user data to be encapsulated by the frame.

**Syntax** int **dgFrameSetData**( dgFrameHandle,  
unsigned char \* *buffer*,  
size\_t *length* )

**Remarks** This function applies to frames of type FT\_CMD, FT\_TEXT, and FT\_MISC.

Set *length* to the length of the data in *buffer*. The contents of *buffer* are copied to the "Data" field (w/o padding).

#### Example:

```
size_t len;
unsigned char* my_data;
// my_data is initialized and len is set to length of relevant data
result = dgFrameSetData( my_frame, m_data, len );
```

#### 4.4.2.1.9 dgFrameGetTimestamp

**Function** Get the value of the Timestamp field of the GC event frame.

**Syntax** int **dgFrameGetTimestamp**( dgFrameHandle,  
unsigned int\* )

**Remarks** This function applies to frames of type FT\_DATA and FT\_EVENT.

The timestamp is in units of 10 microseconds.

#### Example:

```
unsigned int m_timestamp;
result=dgFrameGetTimestamp( my_frame, &m_timestamp );
```

#### 4.4.2.1.10 dgGetContext

**Function** Get the value of the Context field of the GC frame.

**Syntax**            int **dgGetContext** ( dgFrameHandle, unsigned char\* )

**Remarks**        This function applies to frames of type FT\_RESP and FT\_EVENT.

**Example:**

```
unsigned char my_context;
result = dgGetContext( my_frame, &my_context );
```

#### 4.4.2.1.11 **dgSetContext**

**Function**        Set the value of the Context field of the frame.

**Syntax**            int **dgSetContext** ( dgFrameHandle, unsigned char )

**Remarks**        This function applies to frames of type FT\_CMD.

**Example:**

```
unsigned char m_context;
dgFrameHandle my_frame;
// my_frame is initialized
result = dgSetContext( my_frame, m_context );
```

## DATA FRAME-SPECIFIC FUNCTIONS

### 4.4.2.2.1 dgGetHeaderLength

**Function** Get the value of the Header Length field of a GC data frame.

**Syntax** int **dgGetHeaderLength**( dgFrameHandle,  
unsigned char\* )

**Remarks** The Header Length field specifies the length in bytes of the header segment of the payload message.

**Example:**

```
unsigned char m_len;
result = dgGetHeaderLength( my_dataframe, &m_len );
```

### 4.4.2.2.2 dgSetHeaderLength

**Function** Set the value of the Header Length field of the GC data frame.

**Syntax** int **dgSetHeaderLength**( dgFrameHandle,  
unsigned char )

**Remarks** The Header Length field specifies the length of the header segment of the payload message. The default value is zero.

**Example:**

```
result = dgSetHeaderLength( my_CANframe, 2 );
```

### 4.4.2.2.3 dgGetHeaderBits

**Function** Get the value of the Header Bits field of a GC data frame.

**Syntax** int **dgGetHeaderBits**( dgFrameHandle, unsigned char\* )

**Remarks** The Header Bits field specifies the length in bits of the actual payload message header for cases where the number of bits in the header is not an even multiple of eight (such as in the case for CAN, with 11 and 29 bit headers).

**Example:**

```
unsigned char m_numbits;
result = dgGetHeaderBits( dgFrameHandle, &m_numbits );
```

**4.4.2.2.4 dgSetHeaderBits**

**Function** Set the value of the Header Bits field of a GC data frame.

**Syntax** int **dgSetHeaderBits** ( dgFrameHandle, unsigned char )

**Remarks** The Header Bits field specifies the length in bits of the payload message header for cases where the number of bits in the header is not an even multiple of eight (such as in the case for CAN, with 11 and 29 bit headers). The default value is zero.

**Example:**

```
result = dgSetHeaderBits( my_CANframe, 11 );
```

**4.4.2.2.5 dgGetDataLength**

**Function** Get the value of the Data Length field of a GC data frame.

**Syntax** int **dgGetDataLength** ( dgFrameHandle, unsigned short\*)

**Remarks** The Data Length field specifies the length in bytes of the data segment of the payload message.

**Example:**

```
unsigned short m_datalength;
result = dgGetDataLength ( my_dataframe, m_datalength );
```

**4.4.2.2.6 dgSetDataLength**

**Function** Set the value of the Data Length field of a GC data frame.

**Syntax** int **dgSetDataLength** ( dgFrameHandle, unsigned short )

**Remarks** The Data Length field specifies the length in bytes of the data segment of the payload message. The default value is zero.

**Example:**

```
result=dgSetDataLength ( my_dataframe, 5 );
```

**4.4.2.2.7 dgGetExtraLength**

**Function** Get the value of the Extra Length field of a GC data frame.

**Syntax** int **dgGetExtraLength** ( dgFrameHandle, unsigned char\* )

**Remarks** The Extra Length field specifies the length in bytes of the extra information segment of the payload message.

**Example:**

```
unsigned char m_extralen;
result = dgGetExtraLength( my_dataframe, &m_extralen );
```

**4.4.2.2.8 dgSetExtraLength**

**Function** Set the value of the Extra Length field of a GC data frame.

**Syntax** int **dgSetExtraLength** ( dgFrameHandle, unsigned char )

**Remarks** The Extra Length field specifies the length in bytes of the extra information segment of the payload message. The default value is zero.

**Example:**

```
result = dgSetExtraLength( my_dataframe, 2 );
```

**4.4.2.2.9 dgGetMode**

**Function** Get the value of the mode field of a GC data frame.

**Syntax** int **dgGetMode** ( dgFrameHandle, unsigned char\* )

**Remarks** The mode field contains the following flags:

MODE\_RX – Indicates that the Gryphon received the message.

MODE\_TX – Indicates that the message was transmitted by the Gryphon

For a listing of additional hardware specific modes, please see the “Gryphon Hardware Information” web pages for supported mode values.

**Example:**

```
unsigned char m_mode;
result = dgGetMode ( my_dataframe, &m_mode );
```

**4.4.2.2.10 dgSetMode**

**Function** Set the value of the mode field of a GC data frame.

**Syntax** int **dgSetMode** ( dgFrameHandle, unsigned char )

**Remarks** Refer to **dgDataFrame::SetMode()** in the Gryphon C++ User Manual for details.

**Example:**

```
result = dgSetMode ( my_dataframe, MODE_REMOTE );
```

**4.4.2.2.11 dgGetErrorStatus**

**Function** Get the value of the status field of a GC data frame.

**Syntax** int **dgGetErrorStatus**( dgFrameHandle, unsigned char\* )

**Remarks** The status field contains network-specific flags and status information.

**Example:**

```
unsigned char stat;
result = dgGetErrorStatus( my_dataframe, &stat );
```

**4.4.2.2.12 dgSetErrorStatus**

**Function** Set the value of the status field of a GC data frame.

**Syntax** int **dgSetErrorStatus**( dgFrameHandle, unsigned char )

**Remarks** The status field contains network-specific flags and status information.

**Example:**

```
unsigned char UBP_nak = 1;
result = dgSetErrorStatus( my_dataframe, UBP_nak );
```

**4.4.2.2.13 dgGetPayload**

**Function** Get the data that has been sent across the channel's network (e.g., data sent on a J1850 network)

**Syntax** int **dgGetPayload** ( dgFrameHandle,  
unsigned char\* *buffer*,  
unsigned int *max\_len* )

**Remarks** The parameter *buffer* is a pointer to a buffer in the user's program. The number of bytes copied into the user's buffer is the sum of the data frame's Header Length, Data Length, and Extra Length fields, up to but not exceeding *max\_len*.

**Example:**

```
unsigned char buffer[MAX];
result = dgGetPayload ( dgFrameHandle, buffer, MAX );
```

**4.4.2.2.14 dgSetPayload**

**Function** Set the data to be transmitted across the channel's network (e.g.. data to be transmitted on a J1850 network)

**Syntax** int **dgSetPayload** ( dgFrameHandle,  
unsigned char\* *buffer*)

**Remarks** *buffer* points to a buffer in the user's program. The number of bytes copied from the buffer is the sum of the data frame's Header Length, Data Length, and Extra Length fields. This implies that the proper lengths must be set by calling **SetHeaderLength()**, etc., prior to calling this function.

**Example:**

```
unsigned char *m_payload;
// SetHeaderLength() is called and m_payload is initialized to an array of
// unsigned char with the message's payload
result = dgSetPayload( my_dataframe, m_payload );
```

### 4.4.2.3 EVENT FRAME-SPECIFIC FUNCTIONS

#### 4.4.2.3.1 dgGetEventId

**Function** Get the Event identification number of the event.

**Syntax** int **dgGetEventId** ( dgFrameHandle, dgEventId\* )

**Remarks** Please see the module documentation for card specific events.

**Example:**

```
dgEventId my_id;
dgFrameHandle my_eventframe;
// my_eventframe is initialized
result = dgGetEventId ( my_eventframe, &my_id );
```

### 4.4.2.4 COMMAND FRAME-SPECIFIC FUNCTIONS

#### 4.4.2.4.1 dgGetCmd

**Function** Get the identifier of the command to be executed at the destination.

**Syntax** int **dgGetCmd**( dgFrameHandle, unsigned char\* )

**Remarks** None.

**Example:**

```
unsigned char my_cmd;
result = dgGetCmd( my_frame, &my_cmd );
```

#### 4.4.2.4.2 dgSetCmd

**Function** Set the identifier of the command to be executed at the destination.

**Syntax** int **dgSetCmd**( dgFrameHandle, unsigned char )

**Remarks** None.

**Example:**

```
result = dgSetCmd( my_frame, CMD_CARD_IOCTL );
```

**4.4.2.4.3 dgSetCmdData**

**Function** Set the command-specific data.

**Syntax** int **dgSetCmdData** ( dgFrameHandle,  
unsigned char\* *buffer*,  
size\_t *count*)

**Remarks** *count* number of bytes is copied from the user's buffer pointed to by *buffer* to the command frame.

**Example:**

```
unsigned char *m_buffer, m_len;
// m_buffer is initialized with data and m_len is initialized with the
// length of m_buffer
result = dgSetCmdData( my_frame, m_buffer, m_len );
```

#### 4.4.2.5 RESPONSE FRAME-SPECIFIC FUNCTIONS

##### 4.4.2.5.1 dgGetRespCmd

<b>Function</b>	Get the command Id number of the original command frame in response to which the response frame was sent.
<b>Syntax</b>	int <b>dgGetRespCmd</b> ( dgFrameHandle, unsigned char* )
<b>Remarks</b>	None.

**Example:**

```
unsigned char my_cmd;
result = dgGetRespCmd( my_frame, &my_cmd );
```

##### 4.4.2.5.2 dgGetReturnCode

<b>Function</b>	Get the return code of the response frame.
<b>Syntax</b>	int <b>dgGetReturnCode</b> ( dgFrameHandle, unsigned int* )
<b>Remarks</b>	A return code is a four-byte value that indicates the result of the command execution. Please see command specific documentation for response data for a particular command.

**Example:**

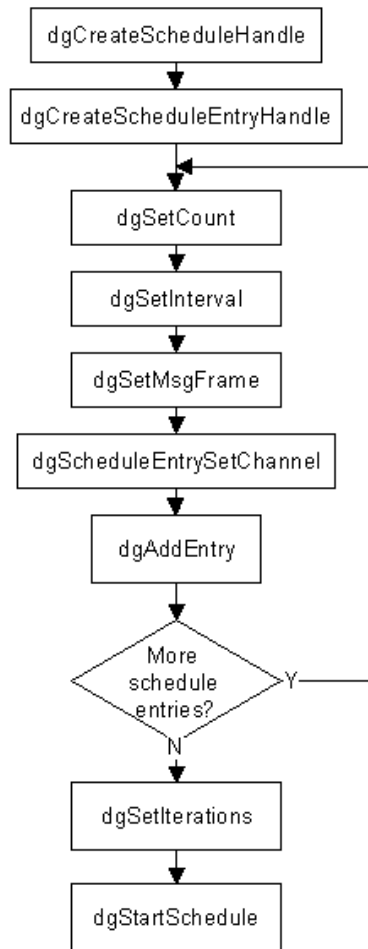
```
unsigned int m_retval;
result = dgGetReturnCode( my_frame, &m_retval );
```

## 4.5 Network Services - Advanced

Functions in this section provide advanced methods for sending and receiving bus data frames over a Gryphon's installed data link channels (Scheduler and Responder), as well as configuring higher layer protocols (USDT).

### 4.5.1 Scheduler Functions

The following flowchart shows an example sequence for the setting up of a Gryphon scheduler:



#### 4.5.1.1 dgCreateScheduleHandle

<b>Function</b>	Create and return a dgScheduleHandle.
<b>Syntax</b>	dgScheduleHandle <b>dgCreateScheduleHandle()</b>
<b>Remarks</b>	None.

##### Example:

```
dgScheduleHandle mySched = dgCreateScheduleHandle();
```

#### 4.5.1.2 dgStartSchedule

<b>Function</b>	Start the schedule process represented by the second parameter.
<b>Syntax</b>	int <b>dgStartSchedule</b> ( dgSessionHandle, dgScheduleHandle, PFV <i>p_pfv</i> )
<b>Remarks</b>	<i>p_pfv</i> is a pointer to the function to be called when the schedule finishes. Alternately, if no callback is desired, supply a zero value in place of the pointer.  PFV is a typedef for a pointer to a function returning a void.

##### Example:

```
dgScheduleHandle m_mysched;  
// Initialize m_mysched  
result = dgStartSchedule(my_session, m_mysched, NULL);
```

#### 4.5.1.3 dgModifyScheduleEntry

<b>Function</b>	Replace the data frame associated with a schedule entry of a currently active dgSchedule.
<b>Syntax</b>	int <b>dgModifyScheduleEntry</b> (dgSessionHandle, dgScheduleHandle, unsigned char <i>index</i> , dgFrameHandle )

**Remarks** The first dgScheduleEntry added to a dgSchedule (cf. **dgAddEntry ()**) is referenced by the *index* value of 0, the second by the value of 1, and so on.

#### 4.5.1.4 dgStopSchedule

**Function** Stop the Gryphon's schedule process represented by the parameter.

**Syntax** int **dgStopSchedule**( dgSessionHandle, dgScheduleHandle )

**Remarks** None.

#### Example:

```
dgScheduleHandle m_mysched;
// m_mysched is initialized and started
result = dgStopSchedule(my_session, m_mysched);
```

#### 4.5.1.5 dgScheduleActive

**Function** Indicates if the schedule process corresponding to the dgScheduleHandle object is currently active in this Gryphon session.

**Syntax** int **dgScheduleActive** (dgSessionHandle, dgScheduleHandle)

**Remarks** This function is accurate as long as there is only one client issuing Scheduler commands to the Gryphon device. For example, this would be inaccurate in the case where the current client starts a schedule that a second client then kills (e.g., using **dgKillSchedules()**) before the schedule runs to completion.

#### Example:

```
if ( dgScheduleActive( my_session, my_sched ) )
    // my_sched is active
else
    // my_sched is inactive
```

#### 4.5.1.6 dgKillSchedules

<b>Function</b>	Remove all existing schedules on a Gryphon.
<b>Syntax</b>	int <b>dgKillSchedules</b> ( dgSessionHandle )
<b>Remarks</b>	None.

##### Example:

```
result = dgKillSchedules( my_session );
```

#### 4.5.1.7 dgSetCritical

<b>Function</b>	Specify whether the Gryphon schedule object is to be time-critical or not.
<b>Syntax</b>	int <b>dgSetCritical</b> ( dgScheduleHandle, int <i>boole</i> )
<b>Remarks</b>	Time deviations in non-critical schedules will be dictated by bus activity and other processes running and may vary from situation to situation.  <i>boole</i> values:  False : “not critical” : The time intervals specified may not be strictly honored.  True : “critical” : Efforts will be made to honor the time intervals.

##### Example:

```
result = dgSetCritical( my_schedule, TRUE );
```

#### 4.5.1.8 dgSetIterations

<b>Function</b>	Set the number of iterations of the list of schedule entries.
<b>Syntax</b>	int <b>dgSetIterations</b> ( dgScheduleHandle, unsigned int )
<b>Remarks</b>	A value of 0xFFFFFFFF indicates an indefinite number of iterations. Entries in the list are processed serially.

**Example:**

```
result = dgSetIterations ( my_schedule, 100 );
```

**4.5.1.9 dgAddEntry**

**Function** Add an entry to a list to be scheduled.

**Syntax** int **dgAddEntry** (dgScheduleHandle,  
dgScheduleEntryHandle)

**Remarks** Please see dgScheduleEntry class functions for information on setting up the messages to be added.

**Example:**

```
dgScheduleEntryHandle m_schentry;
// m_schentry is initialized
result = dgAddEntry( m_schedule, m_schentry );
```

**4.5.1.10 dgCreateScheduleEntryHandle**

**Function** Create and return a dgScheduleEntryHandle.

**Syntax** dgScheduleEntryHandle **dgCreateScheduleEntryHandle**()

**Remarks** None.

**Example:**

```
dgScheduleEntryHandle myentry = dgCreateScheduleEntryHandle();
```

**4.5.1.11 dgSetMsgFrame**

**Function** Set the message frame to be transmitted.

**Syntax** int **dgSetMsgFrame** ( dgScheduleEntryHandle,  
dgFrameHandle)

**Remarks** This is a copy operation for the frame.

**Example:**

```
dgFrameHandle my_dataframe;
// my_dataframe is initialized ...
result = dgSetMsgFrame( my_schedentry, my_dataframe );
```

#### 4.5.1.12 dgScheduleEntrySetChannel

**Function** Specify the channel over which the data frame will be transmitted.

**Syntax** int **dgSetScheduleChannel** (dgScheduleEntryHandle, dgChannelHandle)

**Remarks** None.

##### Example:

```
dgChannel m_channel;
// m_channel is initialized using the functions in dgChannel
result = dgSetScheduleChannel ( my_schedentry, m_channel );
```

#### 4.5.1.13 dgSetInterval

**Function** Set the interval, in milliseconds, between successive transmissions of a message.

**Syntax** int **dgSetInterval** (dgScheduleEntryHandle, unsigned int *interval*)

**Remarks** None.

##### Example:

```
result = dgSetInterval( my_schedentry, 1000 ); // 1 sec interval
```

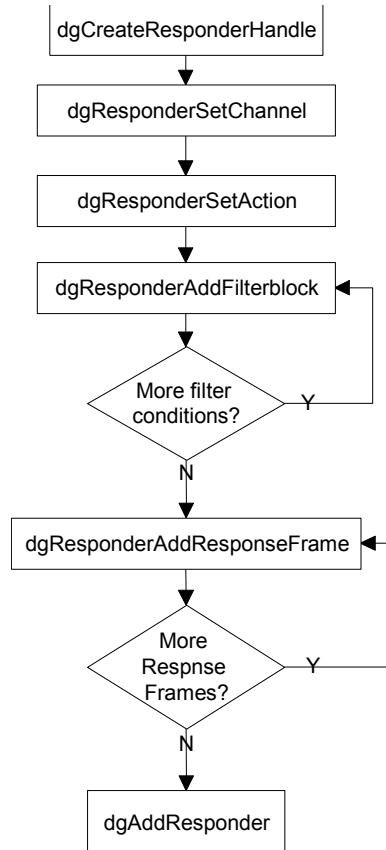
#### 4.5.1.14 dgSetCount

**Function** Set the number of times that the message is to be transmitted.



## 4.5.2 Responder Functions

As an overview, the following flowchart illustrates a typical way of setting up a message responder:



### 4.5.2.1 dgCreateResponderHandle

**Function** Create and return a dgResponderHandle.

**Syntax** dgResponderHandle **dgCreateResponderHandle()**

**Remarks** None.

#### Example:

```
dgResponderHandle myresponder = dgCreateResponderHandle();
```

#### 4.5.2.2 dgAddResponder

**Function** Add a message responder client to the Gryphon.

**Syntax** int **dgAddResponder** (dgSessionHandle,  
dgResponderHandle)

**Remarks** None.

**Example:**

```
dgResponderHandle m_myresp;
// Initialize m_myresp
result = dgAddResponder(my_session, m_myresp);
```

#### 4.5.2.3 dgDeleteResponder

**Function** Delete the message responder client from the Gryphon.

**Syntax** int **dgDeleteResponder** (dgSessionHandle,  
dgResponderHandle)

**Remarks** None.

**Example:**

```
dgResponderHandle m_myresp;
// m_myresp is initialized and started
result = dgDeleteResponder(my_session, m_myresp);
```

#### 4.5.2.4 dgKillResponders

**Function** Remove all existing responders on a single or on all channels of a Gryphon.

**Syntax** int **dgKillResponders**( dgSessionHandle, dgChannelHandle )

**Remarks** If a pointer to a channel object is passed to the function, all of the responders on that channel are removed. If a NULL is passed to the function, all of the responders on all of the channels are removed.

An error code of 6 (Invalid parameter(s)) will return if no

responders are deleted.

**Example:**

```
result = dgKillResponders(my_session, NULL);
// Kills all responders on all channels

dgChannelHandle m_channel;
// m_channel is initialized and one or more responders have been
// started on it.
result = dgKillResponders( my_session, m_channel );
```

**4.5.2.5 dgResponderSetChannel**

**Function** Specify the channel to monitor for data and event messages.

**Syntax** int **dgResponderSetChannel** (dgResponderHandle,  
dgChannelHandle)

**Remarks** None.

**Example:**

```
dgChannelHandle m_channel;
//m_channel is initialized using the functions in dgChannel
result = dgResponderSetChannel(my_responder, m_channel);
```

**4.5.2.6 dgResponderSetActive**

**Function** Set the responder object active or inactive.

**Syntax** int **dgResponderSetActive** ( dgSessionHandle,  
dgResponderHandle,  
int *flag* )

**Remarks** While inactive, no response messages are issued by the responder object.  
*flag* is a boolean.

**Example:**

```
result = dgResponderSetActive( my_session, my_responder, TRUE );
```

### 4.5.2.7 dgResponderSetAction

**Function** Specify the action to be performed upon receipt of conforming data frames.

**Syntax** int dgResponderSetAction( dgResponderHandle,  
unsigned char *action*)

**Remarks** The first three entries to follow are action codes; only one action code may be present. The remaining are action flags that may be ORed with any of the three action codes.

#### Action Codes

FR\_RESP\_AFTER\_EVENT Response messages are sent upon receipt of a message that conforms to the filter conditions.

FR\_RESP\_AFTER\_PERIOD Response messages are sent after the specified time has expired, following the receipt of a message that conforms to the filter conditions. If conforming messages continue to arrive before the specified time has elapsed, no messages are sent. If no conforming message is received, the response messages are sent out periodically.

FR\_IGNORE\_DURING\_PER The response messages are not sent for the specified time following the receipt of a message that conforms to the filter conditions. This effectively causes incoming messages to be ignored for the specified time after a response has been sent.

#### Action Flags

FR\_PERIOD\_MSGS This is a flag that can be ORed with one of the above values. When present, the time values referred to can be changed into message counters.

FR_DEACT_AFTER_PER	This is a flag that can be ORed with one of the above values. When present, the response will deactivate itself after the specified time, following the receipt of a message that conforms to the filter. This option may be used to limit the amount of time that a given response is valid.
FR_DEACT_AFTER_PERIOD	This is a flag that can be ORed with one of the above values. When present, the response will deactivate itself upon the receipt of a conforming message.
FR_DELETE	This is a flag that can be ORed with either of the two previous flags, in order to change the action from deactivating a response to deleting it.

Default value: FR\_RESP\_AFTER\_EVENT

**Example:**

```
result = dgResponderSetAction(my_responder, FR_RESP_AFTER_PERIOD );

unsigned char m_action;
//m_action is initialized
result = dgResponderSetAction( my_responder, m_action );
```

#### 4.5.2.8 dgResponderSetActionValue

<b>Function</b>	Set the amount of time in tens of milliseconds, or the number of conforming messages to use for the period specified by the <b>dgResponderSetAction()</b> function.
<b>Syntax</b>	int <b>dgResponderSetActionValue</b> (dgResponderHandle, unsigned short <i>value</i> )
<b>Remarks</b>	None.

**Example:**

```
//dependant on dgResponderSetAction()
unsigned short m_actionval;
// m_actionval is initialized
result = dgResponderSetActionValue( my_responder, m_actionval );
```

#### 4.5.2.9 dgResponderAddFilterblock

**Function** Add filter information for the message responder.

**Syntax** int **dgResponderAddFilterblock** (dgResponderHandle,  
dgFilterBlockHandle)

**Remarks** Each filter block specifies a section of the message to be checked. In order for a message to conform to a condition, it must conform to all of the defined blocks.

This is a copy operation.

##### Example:

```
dgFilterBlockHandle m_filterblock;
// m_filterblock is initialized using the functions in dgFilterblock
result = dgResponderAddFilterBlock( m_filterblock );
```

#### 4.5.2.10 dgResponderAddResponseFrame

**Function** Add a response frame for the message responder.

**Syntax** int **dgResponderAddResponseFrame** (dgResponderHandle,  
dgFrameHandle)

**Remarks** When creating a message responder, all calls to **dgResponderAddFilterblock()** must be made prior to any call to **dgResponderAddResponseFrame()**, for that message responder.

Each response message is a complete GC frame.

This is a copy operation.

##### Example:

```
dgFrameHandle m_frame;
//m_frame is initialized
result = dgResponderAddResponseFrame( my_responder, m_frame );
```

#### 4.5.2.11 dgResponderSetReplacement

- Function** Specify that a previously created responder is to be replaced by this responder.
- Syntax** int **dgResponderSetReplacement** (dgResponderHandle)
- Remarks** After calling this function, the Responder object may be used to replace the existing corresponding Gryphon entity via a call to **dgAddResponder()**. The filters and responses of the Responder object are removed in order for new ones may be added. None of the object's other attributes are modified.

##### Example:

```
result = dgResponderSetReplacement(my_responder);
        // my_responder will replace any responder on the channel it's
        // added to
```

#### 4.5.2.12 dgResponderPresent

- Function** Determines if the responder is present on the Gryphon (i.e. has been downloaded and has not been deleted).
- Syntax** int **dgResponderPresent** ( dgSessionHandle,  
dgResponderHandle,  
int \* *ispresent* )
- Remarks** None.

##### Example:

```
int m_present;
result = dgResponderPresent( my_session, my_responder, &m_present );
// only if result is TRUE: m_present is set to whether or not the responder is
// present
```

### 4.5.3 USDT Functions

The USDT functions provide access to the ISO 15765-2 "network layer" service, along with additional services currently supplied by the Gryphon USDT server.

#### 4.5.3.1 dgCreateUSDTHandle

<b>Function</b>	Create and return a dgUSDTHandle.
<b>Syntax</b>	dgUSDTHandle <b>dgCreateUSDTHandle()</b>
<b>Remarks</b>	The handle is a representation of a USDT (ISO 15765-2 "network layer") node.

**Example:**

```
dgUSDTHandle my_flowcontrol_node = dgCreateUSDTHandle();
```

#### 4.5.3.2 dgSetActive

<b>Function</b>	Activate or deactivate the USDT service.
<b>Syntax</b>	int <b>dgSetActive</b> (dgUSDTHandle, int <i>boole</i> )
<b>Remarks</b>	When activating the service, the service parameters set by all the following USDT functions (loopback, event reporting, etc.) are used. For this reason, the other functions must be called before calling this function in order for those service parameters to be in effect during the activation period.  When deactivating the service, all the service parameters are irrelevant.

**Example:**

```
if( ! dgSetActive(my_flowcontrol_node, 1 ) )
    handle_error();
```

#### 4.5.3.3 dgReportEventId

<b>Function</b>	Determines if a specific event notification is to be delivered to the client application, or if it should be discarded.
-----------------	---

<b>Syntax</b>	<code>int <b>dgReportEventId</b>( dgUSDTHandle,                           dgEventId <i>eventId</i>,                           int <i>boole</i>)</code>
<b>Remarks</b>	The default state is to report all ISO 15765-2 events. Some dgEventIds which are configurable here are:  USDT_FIRSTFRAME USDT_LASTFRAME USDT_DONE USDT_ERROR

**Example:**

```
// specify that USDT_LASTFRAME events will NOT be reported:
if( ! dgReportEventId(my_flowcontrol_node, USDT_LASTFRAME, 0 )
    handle_error());
```

**4.5.3.4 dgSetDataLink**

<b>Function</b>	Specify which Gryphon CAN channel to use for data link services.
<b>Syntax</b>	<code>int <b>dgSetDataLink</b>( dgUSDTHandle, dgChannelHandle )</code>
<b>Remarks</b>	The dgChannelHandle must reference a CAN hardware channel by having previously been returned by a successful call to <b>dgGetChannel</b> ().

**Example:**

```
my_CAN_channel = dgGetChannel( session, idxt );
// ensure the channel handle references the desired CAN channel...

if( ! dgInit( my_CAN_channel, 1 ))
    handle_error();

if( ! dgSetDataLink(my_flowcontrol_node, my_CAN_channel ) )
    handle_error();
```

#### 4.5.3.5 dgSetPad

<b>Function</b>	Specify desired message padding.
<b>Syntax</b>	int <b>dgSetPad</b> (dgUSDTHandle, unsigned char <i>mode</i> )
<b>Remarks</b>	<p><i>mode</i>:</p> <p>0 = Pad messages with 0x00's.  1 = Pad messages with 0xFF's.  2 = Do not pad messages (default).</p>

#### 4.5.3.6 dgSetIdSize

<b>Function</b>	Specify CAN identifier size(s) to work with.
<b>Syntax</b>	int <b>dgSetIdSize</b> (dgUSDTHandle, unsigned char <i>mode</i> )
<b>Remarks</b>	<p><i>mode</i>:</p> <p>0 = Use 11-bit identifiers only (default).  1 = Use 29-bit identifiers only.  2 = Use both 11- and 29-bit identifiers.</p>

#### 4.5.3.7 dgSetExtAddrIds

<b>Function</b>	Specify a set of CAN identifiers for which to use extended addressing, replacing the default value(s).
<b>Syntax</b>	int <b>dgSetExtAddrIds</b> ( dgUSDTHandle, unsigned char <i>arraysize</i> , unsigned* <i>eaIds</i> )
<b>Remarks</b>	See subdirectory "GCprotocol/commands/USDT" of [Ref. 2] for default value(s) and other details.

#### Example:

```
// replace the default CAN identifier(s) with the following three:
unsigned eaIds[3];
eaIds[0] = 0x42;
eaIds[1] = 0x44;
eaIds[2] = 0x46;
```

```
if( ! dgSetExtAddrIds( my_node, 3, eaIds ))
    handle_error();
```

#### 4.5.3.8 dgSetIdBlock

**Function** Specify the correspondence between Network Source Addresses and Network Target Addresses for 11-bit CAN identifiers.

**Syntax** int **dgSetIdBlock**( dgUSDTHandle,  
                           unsigned char *blkIndex*,  
                           unsigned *blkSize*,  
                           unsigned *N\_TA*,  
                           unsigned *N\_SA* )

**Remarks** ISO 15765-2 specifies how (8-bit) Network Source Address (N\_SA) and Network Target Address (N\_TA) values are mapped into 29-bit CAN identifiers, but, curiously, does not specify how to map 16 bits into an 11-bit CAN identifier.

This function allows the application to assign the correspondences between sets of N\_SA and N\_TA values for 11-bit CAN identifiers, to be used by the Gryphon USDT server for segmented message transport.

You can specify up to 64 such Id blocks, using *blkIndex* values from 0 to 63. Specifying an Id block with *blkSize* = 0 removes a previously specified Id block at that index position.

If this function is not called, then default Id blocks are used by the USDT server.

See subdirectory "GCprotocol/commands/USDT" of [Ref. 2] for default block value(s) and other details.

#### Example:

```
if( ! dgSetIdBlock( my_flowctrl_node, 0, 32, 0x0640, 0x0240 ))
    handle_error();
```

#### 4.5.3.9 dgSetFilterMode

<b>Function</b>	Specify whether messages received by the node are to be blocked, filtered or passed to the client application.
<b>Syntax</b>	int <b>dgSetFilterMode</b> ( dgUSDTHandle <i>handle</i> , unsigned char <i>mode</i> )
<b>Remarks</b>	Parameter <i>mode</i> values:  FILTER_OFF_BLOCK_ALL : No messages are to be passed to the client. This option also disables timeout and sequence error event reporting.  FILTER_OFF_PASS_ALL : All messages are passed to the client (default). This option also enables timeout and sequence error event reporting.

#### **Examples:**

```
result = dgSetFilterMode (my_flowcontrol_node, FILTER_OFF_PASS_ALL);

if( ! dgSetFilterMode( my_flowcontrol_node, FILTER_OFF_BLOCK_ALL ) )
    handle_error();
```

#### 4.5.3.10 dgEnableLoopback

<b>Function</b>	Enable or disable the echo back to the client application of transmitted messages.
<b>Syntax</b>	int <b>dgEnableLoopback</b> ( dgUSDTHandle, int <i>boole</i> )
<b>Remarks</b>	Default = no loopback

#### **Example:**

```
result = dgEnableLoopback( my_flowcontrol_node, TRUE );
```

#### 4.5.3.11 dgReportAllEvents

<b>Function</b>	Enable or disable reporting of all node events to the client application.
<b>Syntax</b>	int <b>dgReportAllEvents</b> ( dgUSDTHandle <i>handle</i> , int <i>boole</i> )

**Remarks** The parameter *boole* is a boolean value which determines if all Event types should be filtered out or not.

*boole*:

0 = Reports no events to the client.

1 = Reports all events to the client (default).

**Example:**

```
result = dgReportAllEvents ( my_channel, 1 );
```

## 4.6 Miscellaneous Services

### 4.6.1 Program Loader Functions

#### 4.6.1.1 dgAddFile

**Function** Copy a file from the local file system to the Gryphon.

**Syntax** int **dgAddFile** ( dgSessionHandle,  
dgFileHandle,  
int *flags* )

**Remarks** The parameter *flags* can be used to specify options of the file transfer. It may be a combination of the following constants:

dgOVERWRITE: This flag specifies that the file copy operation is to replace any previously uploaded file having the same name.

dgASCII : Specifies that the file is a text file and that DOS to Linux end-of-line character conversion should be performed (CR-LF to LF character).

The value 0 for *flags* specifies that there is no overwrite of a file with the same name that was previously uploaded to the Gryphon, and no end-of-line character conversion.

**Example:**

```
dgFile myfile;
```

```
// myfile is initialized
result = dgAddFile( my_session, my_file, dgOVERWRITE);
```

#### 4.6.1.2 dgDeleteFile

**Function** Delete the previously uploaded file from the Gryphon.

**Syntax** int **dgDeleteFile** (dgSessionHandle, dgFileHandle)

**Remarks** None.

##### Example:

```
dgFile myfile;
// myfile is initialized
result = dgDeleteFile( my_session, my_file );
```

#### 4.6.1.3 dgStartProgram

**Function** Start the execution on the Gryphon of an uploaded executable file.

**Syntax** int **dgStartProgram** ( dgSessionHandle,  
dgFileHandle,  
unsigned char \* *args* )

**Remarks** The parameter *args* can optionally be used to specify the command line arguments to be supplied to the executable. The first 119 characters of the parameter string will be used.

##### Example:

```
dgFile myfile;
// myfile is initialized
result = dgStartProgram( my_session, my_file, "" );
```

#### 4.6.1.4 dgStopProgram

**Function** Stop the execution on the Gryphon of an uploaded executable.

**Syntax**            int **dgStopProgram** ( dgSessionHandle, dgFileHandle )

**Remarks**        None.

**Example:**

```
dgFile myfile;
// myfile is initialized and started as a program
result = dgStopProgram( my_session, my_file );
```

## 4.6.2 dgFile Functions

### 4.6.2.1 dgCreateFileHandle

**Function**        Create and return a dgFileHandle.

**Syntax**            dgFileHandle **dgCreateFileHandle**()

**Remarks**        None.

**Example:**

```
dgFileHandle myfile = dgCreateFileHandle();
```

### 4.6.2.2 dgSetName

**Function**        Specify the name of the file to be copied to the Gryphon.

**Syntax**            int **dgSetName** ( dgFileHandle, unsigned char \* )

**Remarks**        None.

**Example:**

```
dgFileHandle myfile;
unsigned char fname[256];
// myfile is initialized
strcpy(fname, "test.out");
result = dgSetName( my_file, fname );
```

### 4.6.2.3 dgSetPath

**Function**        Specify the location of the file (on the local file system) to be  
copied to the Gryphon.



**Remarks**           None.

**Example:**

```
dgFileHandle myfile;
// myfile is initialized
result = dgSetExecutable ( my_file, TRUE );
```

### 4.6.3    **dgClient Functions**

dgClientHandles can be used in the functions **dgSendFrame()**, **dgRecvFrame()**, and **dgSetOnRx()**, to send/receive GC frames to/from other Gryphon clients.

#### 4.6.3.1   **dgCreateClientHandle**

**Function**            Create and return a dgClientHandle.

**Syntax**             dgClientHandle **dgCreateClientHandle()**

**Remarks**           None.

**Example:**

```
dgClientHandle other_client = dgCreateClientHandle();
```

## 4.6.4 dgGryphonNode Functions

A dgGryphonNode is a generic representation of a node in the Gryphon network which can act as a source and destination for GC frames. For example, a dgGryphonNodeHandle can be used in the functions **dgSendFrame()**, **dgRecvFrame()**, and **dgSetOnRx()**, to send/receive GC frames to/from other Gryphon clients.

In the C++ library, **dgGryphonNode** is the base class for **dgChannel**, **dgClient** and **dgUSDT** classes.

### 4.6.4.1 dgCreateGryphonNodeHandle

<b>Function</b>	Create and return a dgGryphonNodeHandle.
<b>Syntax</b>	dgGryphonNodeHandle <b>dgCreateGryphonNodeHandle()</b>
<b>Remarks</b>	None.

### 4.6.4.2 dgGetNodeType

<b>Function</b>	Query the type of the Gryphon node.
<b>Syntax</b>	int <b>dgGetNodeType</b> (dgGryphonNodeHandle, unsigned char* <i>type</i> )
<b>Remarks</b>	<i>type</i> may be <b>SD_CLIENT</b> , <b>SD_CARD</b> or <b>SD_USDT</b> .

### 4.6.4.3 dgSetNodeType

<b>Function</b>	Set the type of the Gryphon node.
<b>Syntax</b>	int <b>dgSetNodeType</b> (dgGryphonNodeHandle, unsigned char <i>type</i> )
<b>Remarks</b>	See previous function for valid <i>type</i> values.

#### 4.6.4.4 dgGetNodeId

<b>Function</b>	Get the numeric identifier of the Gryphon node.
<b>Syntax</b>	int <b>dgGetNodeId</b> ( dgGryphonNodeHandle, unsigned char* <i>id</i> );
<b>Remarks</b>	Within each Gryphon node "Type", each instance will have a unique numeric identifier.

#### 4.6.4.5 dgSetNodeId

<b>Function</b>	Set the numeric identifier of the Gryphon node.
<b>Syntax</b>	int <b>dgSetClientId</b> (dgGryphonNodeHandle, unsigned char <i>id</i> )
<b>Remarks</b>	The reserved value CH_BROADCAST can be used as the <i>id</i> value to specify a broadcast target of all clients when the dgClientHandle is passed to <b>dgSendFrame()</b> .

#### Example:

```
dgClientHandle myclient = dgCreateClientHandle();
result = dgSetNodeId ( myclient, CH_BROADCAST );
```

## Appendix A: Visual BASIC Notes

- In the Visual BASIC module, there exists one user-defined type named `dgSessionRequest`. This user-defined type is passed as a parameter to `dgBeginSession()`.
- All other data types convert directly from C data types to Visual BASIC data types. Please see the Gryphon Visual BASIC module for the complete list of the Visual BASIC **Declare** statements.
- Any Visual BASIC String that is passed into a function in the library in order to retrieve information must have space already allocated to it. Furthermore, the string must be 1 byte longer than the length that is passed to the function as the 'max length'. This can be done in two ways:
  - Using fixed-length strings:
    - `dim namestring as String * 1025`
    - `dgGryphonGetName( m_session, namestring, 1024 )`
  - Using variable length strings, assign a value to them before passing them to a function:
    - `dim namestring as String`
    - `namestring = Space$(1025)`
    - `dgGryphonGetName( m_session, namestring, 1024 )`
- All functions, unless otherwise documented, return **dgFail** on failure. Any other return value indicates success. If a function fails, `dgGetLastOpStatus()` may be called to get the error code and error string.