Dearborn Group™
Technology

GRYPHON C++ CLASS LIBRARY

Release 4.5

USER'S MANUAL

33604 West Eight Mile
Farmington Hills, MI 48335-5202
Phone (248) 888-2000 • Fax (248) 888-9977
http://www.dgtech.com

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++ Class 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.

Table of Contents

Chapter

1 INTRODUCTION

1.1 Document organization and format

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

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

Chapter 3 – Class Library Reference – Describes in detail each class and its member functions in the Gryphon C++ Class Library, as well as non-class functions.

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) 888-2000
Fax: (248) 888-9977
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	http://<Gryphon IP address>/html/

Chapter

2 Class 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:\gryphdll.exe (If "D" is your PC's CD-ROM label; otherwise replace "D" with the correct label for your CD-ROM drive).
- Follow the instructions on the screen.

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.

3 Class Library Reference

The Gryphon C++ Class Library was developed by the Dearborn Group to provide a cross-platform library supplying the Gryphon's services for C++ application development. Current platforms supported include Linux (and most Unix variants) with pthread support, and all 32-bit versions of MS Windows.

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.
- Class declarations are in file "dggryph.h".

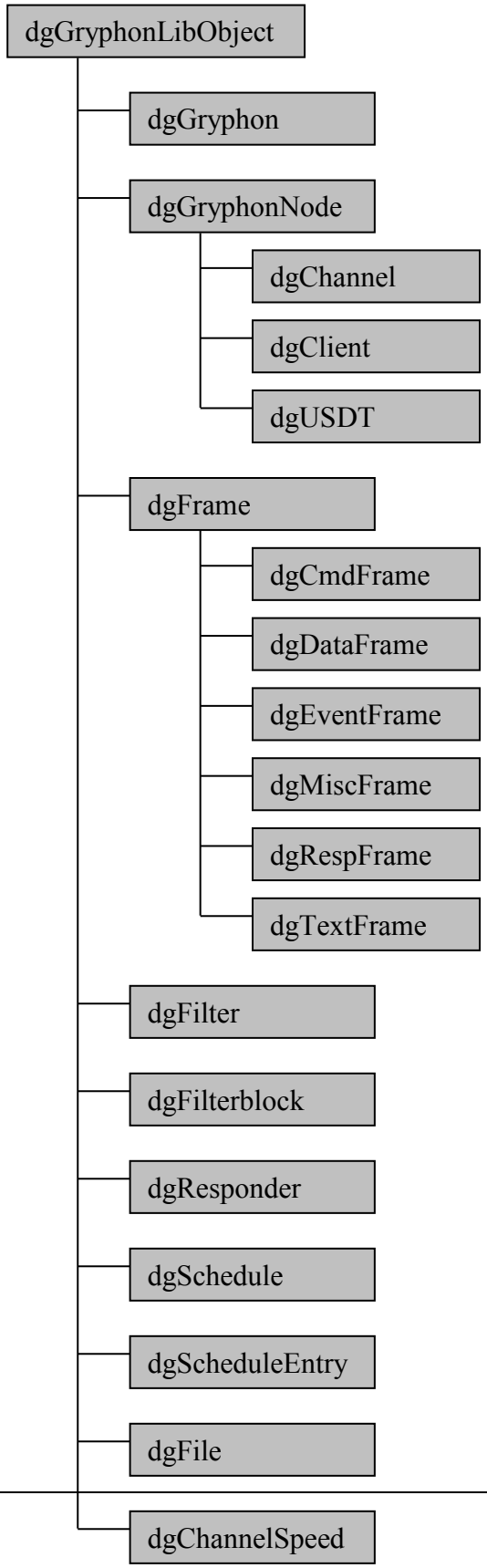
3.1 Class Overviews

dgGryphon	Represents a connected Gryphon device.
dgGryphonNode	Generic representation of a node in the Gryphon network which can act as a source and destination for GC frames; the base class for **dgChannel**, **dgClient** and **dgUSDT**.
dgChannel	A daughter card channel on the Gryphon.
dgClient	A client of the Gryphon server.
dgUSDT	An ISO 15765-2 "network layer" node.
dgFrame	The base class for the various GC Frame classes.
dgCmdFrame	A GC Command Frame, which is used to carry commands to be executed at the destination.
dgRespFrame	A GC Response Frame, which is issued in response to a command frame.
dgDataFrame	A GC Data Frame, which encapsulates and describes a vehicle network message. Use dgDataFrame objects to transmit and receive messages over connected vehicle network channels.
dgEventFrame	A GC Event notification Frame, which is used by the Gryphon to inform connected clients about certain events and error conditions.
dgMiscFrame	A GC Miscellaneous Frame, which may be used to transfer arbitrary data between clients.
dgTextFrame	A GC Text Frame, which may be used to send text messages between clients.
dgFilter	A specification of a message filter, which can be applied to a **dgChannel** object, giving the capability to apply filter conditions to messages received over the network channel.
dgFilterblock	Used in building a **dgFilter**.
dgChannelSpeed	Used in setting a channel speed.
dgEventType	Used in managing and interpreting channel event notifications.

dgSchedule	A list of messages to be transmitted over a specified channel, with iteration and interval capabilities.
dgScheduleEntry	Used in building a **dgSchedule** object.
dgResponder	A Gryphon message responder.
dgFile	Represents a file in the local file system, which can be uploaded to the Gryphon, optionally as an executable program.
[dgUSDTFrame	This class is now deprecated.]

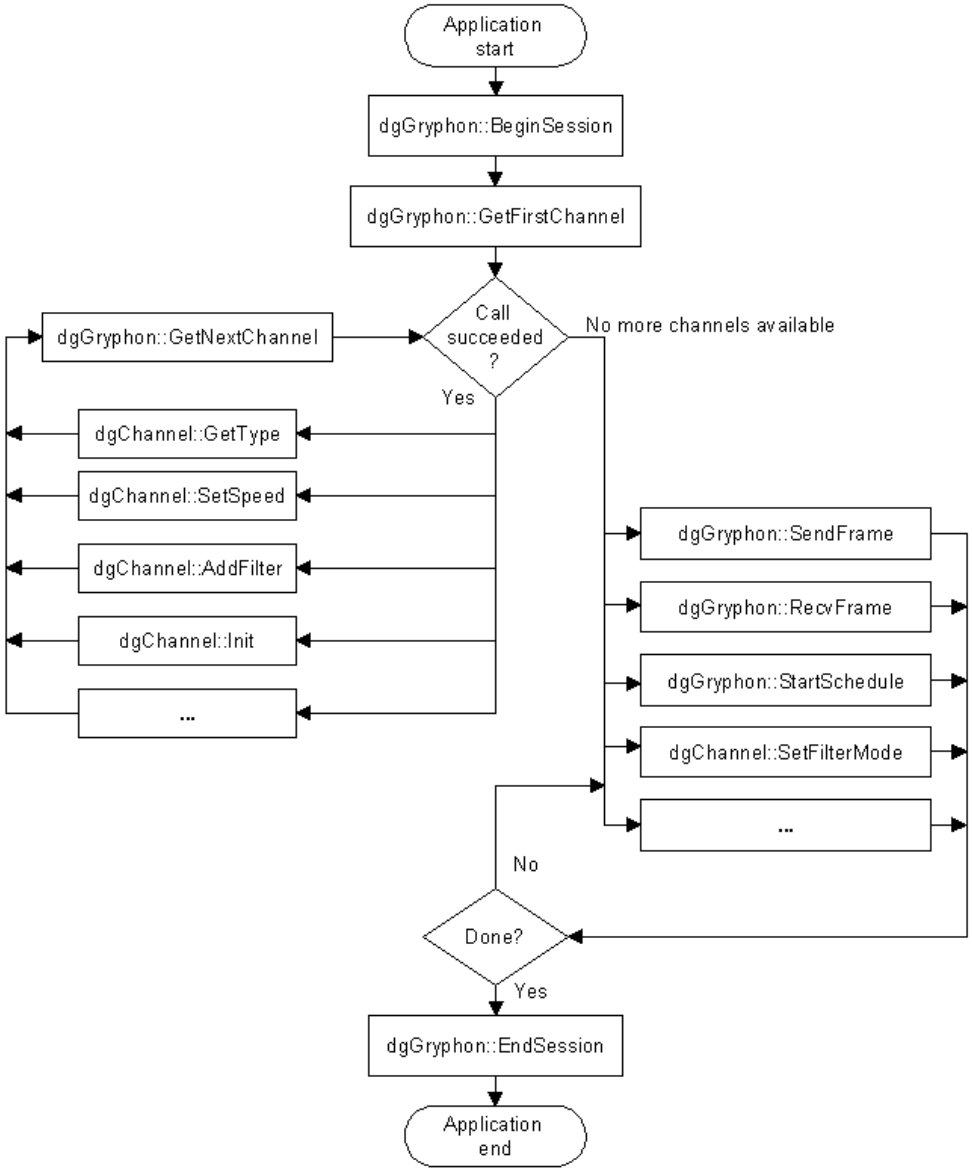
All the above classes are derived from a common base class, **dgGryphonLibObject**, as illustrated in the Class Hierarchy Chart following. This base class is used to ensure a common error reporting model throughout the library, represented in particular by the public member function **GetLastOpStatus**().

3.1.1 Class Hierarchy



3.1.2 Flowchart

An overview of the use of the major library classes can also be demonstrated by flowchart:



3 Class dgGryphonLibObject

This is the base class for (almost) all of the other classes in the library. This class is a convenient way to specify a set of operations common to all derived classes. Currently, the 'public' portion of this set contains only the error reporting function.

3.2.1 GetLastOpStatus

Function Get information about the last error condition encountered by the derived object.

Syntax void **GetLastOpStatus**(dgUint32*, char* ***buffer***)

void **GetLastOpStatus**(dgUint32*,
char* ***buffer***,
size_t ***max***) const

Remarks **GetLastOpStatus** () returns the GC response frame return code, where applicable, and a text description of the error.

Most class member functions return a boolean value: TRUE if the operation completed successfully, FALSE if an error or other anomaly occurred. Use **GetLastOpStatus** () to obtain more information on the error condition. Functions that do not behave in this way will be documented individually.

The second form of the function copies the initial ***max*** characters of the library error string to memory pointed to by ***buffer***. If ***max*** is less than or equal to the length of the source string, a null character is not appended automatically to the buffer. If ***max*** is greater than the length of the source string, the destination buffer is padded with null characters up to length ***max***.

Example:

```
#define BUFF_SIZE 128
dgUint32 errnum;
char errstr[BUFF_SIZE];
if( ! gryph_object-> KillSchedules() )
        gryph_object->GetLastOpStatus( &errnum, errstr );
```

3.3 Class dgGryphon

Base class: **dgGryphonLibObject**

The class dgGryphon represents the Gryphon hardware device and the Gryphon server process within. It is used to configure and control a session with the Gryphon, obtain information about the Gryphon server and installed channels, control Gryphon schedules and responders, and send and receive messages to and from daughter card (hardware) channels, higher layer protocol nodes, and other clients.

3.3.1 BeginSession

Function Initiate the TCP session and register with the Gryphon server.

Syntax

bool **BeginSession** (const dgString& ***addr***,
const dgString& ***username***,
const dgString& ***password***,
dgUint8 ***srcType***,
dgUint16 ***port***)

bool **BeginSession** (const char* ***addr,***
const char* ***username***,
const char* ***password***,
dgUint8 ***srcType***,
dgUint16 ***port***)

Remarks

addr	Hostname or IP address of Gryphon
username	Currently not implemented on Gryphon.
password	Currently not implemented on Gryphon
srcType	The source/destination type to register as.
port	Port number of the server application on Gryphon.

Default values for these parameters:

IP address: 192.168.1.1
Username: ""
Password: ""
SrcType: SD_CLIENT
Port: 7000

In the second form of the function, ***addr*** points to a null-terminated character string.

Example:

Example using default values:
result = gryph->BeginSession();

Example changing the IP address:
result = gryph->BeginSession("192.168.1.100");

3.3.2 EndSession

Function End the TCP session with the Gryphon server.

Syntax bool **EndSession** (bool ***flush*** = false)

Remarks If the parameter ***flush*** is supplied as TRUE, any incoming frames previously enqueued by the library, and not yet retrieved by **RecvFrame()**, are deleted. Otherwise, enqueued frames can be retrieved after the call to **EndSession()**

Example:

result = gryph->EndSession();

3.3.3 SessionActive

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

Syntax bool **SessionActive** () const

Remarks None.

Example:

```
if(gryph->SessionActive()) {
        // Session is active
}
else {
        // Session is NOT active
}
```

3.3.4 GetId

Function Get the identifier assigned by the Gryphon server for the session.

Syntax bool **GetId**(dgUint8*) const

Remarks Normally, the user registers with the Gryphon server as a client (SD_CLIENT); this is the default value of the ***srcType*** parameter of **BeginSession**(). In this case, this function will retrieve the "Client ID" which will be valid for the duration of the session.

3.3.5 SetTimeout

Function Set the timeout period for all of the Gryphon library functions.

Syntax bool **SetTimeout** (dgUint32 ***ms***)

Remarks If the operation of a library function did not complete within the timeout period, the function will return FALSE and the member function GetLastOpStatus() will return a RESP_TIMEOUT value.

The default timeout period is 500 ms.

Example:

Example using a numerical value:
result = gryph->SetTimeout(1000); //Sets timeout to 1000 ms or 1 s.

Example using a variable:
dgUint32 m_timeout=0;
result = gryph->SetTimeout(m_timeout); //Sets timeout to 0 ms.

3.3.6 EnableBroadcastRx

Function Enable or disable reception of broadcast messages for the current session.

Syntax bool **EnableBroadcastRx** (bool)

Remarks None.

Example:

```
result = gryph->EnableBroadcastRx(TRUE); // Enables reception of
                                         // broadcast messages

result = gryph->EnableBroadcastRx(FALSE); // Disables reception
                                          // of broadcast messages
```

3.3.7 KillSchedules

Function Remove all existing schedules on a Gryphon.

Syntax bool **KillSchedules**()

Remarks None.

Example:

```
result = gryph->KillSchedules();
```

3.3.8 KillResponders

Function Remove all existing responders on a single or on all channels

of a Gryphon.

Syntax bool **KillResponders**(dgChannel*)

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.

If no responders are deleted, the function will fail.

Example:

```
result = gryph->KillResponders(NULL);
// Kills all responders on all channels

dgChannel m_channel;
// m_channel is initialized and one or more responders have been
// started on it.
result = gryph->KillResponders( &m_channel );
```

3.3.9 SetCommOpt

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

Syntax bool **SetCommOpt** (dgUint8 ***flags***)

Remarks Optimization can be set to minimize latency or maximize throughput.

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 = session1->SetCommOpt( COMMOPT_LATENCY_SRVR +
        COMMOPT_LATENCY_CLNT );
```

3.3.10 SetSortMode

Function Set the server's data message sorting behavior.

Syntax bool **SetSortMode** (dgUint8 ***mode***)

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

Example:

```
Example using a numerical value;
result = gryph->SetSortMode(0);              // Turn sorting off

Example using a variable:
dgUint8 m_mode=1;
result = gryph->SetSortMode(m_mode);         // Turn sorting on
```

3.3.11 GetName

Function Get the name of the Gryphon server.

Syntax bool **GetName** (dgString*)
bool **GetName** (char* ***buff***, size_t ***max***)

Remarks The second form of the function copies the initial ***max*** characters of the source string to memory pointed to by ***buff***. If ***max*** is less than or equal to the length of the source string, a null character is not appended automatically to the buffer. If ***max*** is greater than the length of the source string, the destination buffer is padded with null characters up to length

max.

Example:

```
dgString m_tempstr;
result = gryph->GetName (&m_tempstr);
```

3.3.12 GetSerial

Function Get the serial number (Ethernet MAC address) of the Gryphon.

Syntax bool **GetSerial** (dgString*)
bool **GetSerial** (char* ***buff***, size_t ***max***)

Remarks See **Remarks** in **GetName**(), above.

Example:

```
dgString m_tempstr;
result = gryph->GetSerial (&m_tempstr);
```

3.3.13 GetVersion

Function Get the version number of the Gryphon server.

Syntax bool **GetVersion** (dgString*)
bool **GetVersion** (char* ***buff***, size_t ***max***)

Remarks See **Remarks** in **GetName**(), above.

Example:

```
dgString m_tempstr;
result = gryph->GetVersion (&m_tempstr);
```

3.3.14 GetTime

Function Get time values from the Gryphon.

Syntax bool **GetTime** (dgGryphonTime*)
bool **GetTime** (time_t*)
bool **GetTime** (dgUint32*, dgUint32*)

Remarks Differences in syntax:

GetTime(dgGryphonTime*) will retrieve the 64-bit Gryphon time. The units are tens of microseconds.

GetTime(time_t*) will retrieve the 32-bit time in seconds (standard C time).

GetTime (dgUint32*, dgUint32*) will retrieve 64-bit Gryphon time in two 32-bit variables. The first variable will contain the high 32-bit value and the second variable the low 32-bit value.

Example:

Example using dgGryphonTime:
```
dgGryphonTime m_gryphtime;
result = gryph->GetTime(&m_gryphtime);    // 64-bit time
```

Example using time_t:
```
time_t m_time;
result = gryph->GetTime(&m_time);                    // 32-bit time (seconds)
```

Example using two dgUint32 variables:
```
dgUint32 m_high, m_low;
result = gryph->GetTime(&m_high, &m_low);  // 64-bit time in 2 variables
```

3.3.15 SetTime

Function Set the time value(s) for the Gryphon.

Syntax bool **SetTime** (dgGryphonTime)
bool **SetTime** (time_t)
bool **SetTime** (dgUnit32, dgUint32)

Remarks Setting the time with a 32- bit value will set the fractional

portion of the Gryphon's time to zero.

Example:

```
dgGryphonTime m_gryphtime;
//  Initialize m_gryphtime
result = gryph->SetTime( m_gryphtime);      // Sets time
                                            // (10 microseconds)

time_t m_time;
//  Initialize m_time                       // Sets time (seconds)
result = gryph->SetTime( m_time );          // fractional time set to 0

dgUint32 m_low, m_high;
//  Initialize m_low and m_high             // Sets time
result = gryph->SetTime( m_high, m_low );   // (10 microseconds)
```

3.3.16 GetTimestamp

Function Get the 32- bit timestamp from Gryphon.

Syntax bool **GetTimestamp** (dgUint32*)

Remarks None.

Example:

```
dgUint32 m_timestamp;
result = gryph->GetTimestamp(&m_timestamp);
```

3.3.17 SetTimestamp

Function Replace the timestamp element of the Gryphon's time with the value supplied.

Syntax bool **SetTimestamp** (dgUint32)

Remarks None.

Example:

```
dgUint32 m_timestamp;
// Initialize m_timestamp
```

```
result = gryph->SetTimestamp( m_timestamp );
```

3.3.18 GetFirstChannel

Function Get the information for the first channel of the installed Gryphon daughter cards.

Syntax bool **GetFirstChannel** (dgChannel*)

Remarks This function retrieves all the information needed to utilize the channel. After this function returns TRUE, the dgChannel object referenced in the parameter list can be used to access that channel's information and services (see section 3.3).

Example:

```
dgChannel m_channel;
result = gryph->GetFirstChannel( &m_channel );
```

3.3.19 GetNextChannel

Function Get the information on the next available channel on a Gryphon.

Syntax bool **GetNextChannel** (dgChannel*)

Remarks This function should only be issued after the **GetFirstChannel**() function.

Example:

```
dgChannel m_channel;
//  GetFirstChannel() is called successfully
while (gryph->GetNextChannel(&m_channel)) {
        // Handling for each valid channel
}
// All valid channels have been retrieved
```

3.3.20 SendFrame

Function Send frame to the target node.

Syntax bool **SendFrame** (dgFrame* ***frame***,
const dgGryphonNode& ***target***)

Remarks dgFrame is the base class for all the specific frame type classes (i.e., dgDataFrame, dgCmdFrame, dgEventFrame, etc.), so a pointer to an object of one of the derived classes would typically be passed as the parameter.

dgChannel, dgClient, and dgUSDT are derived from dgGryphonNode, so references to objects of these types are suitable to pass as the ***target***.

To broadcast a frame to all clients (which have enabled broadcast reception; see **EnableBroadcastRx**()), use a dgClient object with a Node Id of CH_BROADCAST for the ***target***.

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

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 **dgGryphon::BeginSession**().

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 **dgGryphonNode::GetNodeType()**.

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 **dgGryphonNode::GetNodeId()**.

Example:

```
dgDataFrame my_dataframe;
// my_dataframe is initialized
result = gryph->SendFrame( &my_dataframe, my_channel );
```

3.3.21 RecvFrame

Function Receive a GC frame.

Syntax

bool **RecvFrame**(dgFrame* ***frame***, bool ***noWait*** = 0);

bool **RecvFrame**(dgFrame* ***frame***,
const dgGryphonNode& ***source,***
dgUint8 ***cmdID*** = 0,
bool ***noWait*** = 0);

bool **RecvFrame**(dgFrame* ***frame***,
const dgGryphonNode& ***source***,
dgUint8* ***hdr***,
dgUint8 ***len***,
bool ***noWait*** = 0);

Remarks 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.

The appropriateness of the frame to be retrieved is determined by the data type of the ***frame*** argument, as well as the number of arguments used.

If the data type of ***frame*** is "dgFrame*", 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.)

If ***frame*** points to an object of a class derived from class dgFrame, then only a GC frame corresponding to that data

type will be appropriate; e.g., if a pointer to an object of type "dgEventFrame" is supplied, then only a GC frame with a frame type of FT_EVENT is appropriate.

If the first form of this function is used, then the source of the GC frame is not relevant to determining the appropriateness of the received frame; e.g., use this form to receive broadcast frames or a frame from any source.

The second and third forms of this function are used to receive a frame from a specific source.

The parameter ***cmdID*** is used only when receiving a frame of type FT_RESP, in which case ***cmdID*** must match the Id of the command frame that was sent.

hdr and ***len*** allow a retrieval of frames based on matching the contents of the GC frame body of the incoming frame with the buffer contents beginning at ***hdr*** for length ***len***.

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: **RecvFrame**() returns FALSE, and a descriptive message is obtained from **GetLastOpStatus**().

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

The parameter ***noWait*** allows for a convenient override of the library timeout period (for the current invocation of this function only). When the supplied value is TRUE: if the queue is empty, the function returns immediately; if the queue is not empty, but no appropriate frame is found, the function

returns immediately.

Example:

```
dgDataFrame my_dataframe;
result = gryph->RecvFrame( &my_dataframe, my_channel );
```

3.3.22 FlushQueue

Function Flush the receive frame queue.

Syntax void **FlushQueue**(void);

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 **RecvFrame**().

3.3.23 SetOnRx

Function Register an event handler function, to be called upon the receipt of a frame of the specified type and, optionally, from the specified source.

Syntax

bool **SetOnRx**(PFV,
dgUint8 ***frametype,***
const dgGryphonNode& ***source***)
bool **SetOnRx**(PFV,
dgUint8 ***frametype***)

Remarks The first form registers a callback function to be called when a frame of ***frametype*** is received from the specified ***source***.

The second form registers a function to be called when a frame of ***frametype*** is received from any source.

PFV is a typedef for a pointer to a function returning a void.

Setting the first parameter to 0 will undo registering of the event handler function.

The callback function will execute in a new thread, so the

library is not blocked during callback execution.

Example:

```
Void OnRx()
{
        gryph.RecvFrame(&my_dataframe);
}
gryph.SetOnRx(OnRx, FT_DATA, my_channel);
```

3.3.24 StartSchedule

Function Start the schedule process represented by the first parameter.

Syntax bool **StartSchedule**(dgSchedule*, PFV ***p_pfv = 0***)

Remarks ***p_pfv*** is a pointer to the function to be called when the schedule finishes. This is an optional parameter.

For more information on creation of schedules please see the class **dgSchedule**.

Example:

```
dgSchedule m_mysched;
// Initialize m_mysched
result = gryph->StartSchedule(&m_mysched);
```

3.3.25 ScheduleActive

Function Indicates if the schedule process corresponding to the parameter dgSchedule object is currently active in the Gryphon.

Syntax bool **ScheduleActive**(dgSchedule*) const

Remarks NB: This function is accurate as long as there is only one client, i.e., the user of this library, 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

dgGryphon::KillSchedules()) before the schedule runs to completion.

3.3.26 ModifyScheduleEntry

Function Replace the data frame associated with a schedule entry of a currently active dgSchedule.

Syntax bool **ModifyScheduleEntry**(dgSchedule*, dgUint8 ***entry_index***, const dgDataFrame&, dgUint8 ucFlags, dgUint16 usSize)

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

UcFlags is optional, defaults to 0. If ucFlags is 1, flush the messages in the delay queue keeping only those that will be output within the time in milliseconds present in the usSize field. If ucFlags is 0, keep all current message in the Delay queue.

usSize is optional, defaults to 0. usSize is the maximum amount of time in milliseconds that the previously scheduled message(s) will be output.

3.3.27 StopSchedule

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

Syntax bool **StopSchedule**(dgSchedule*)

Remarks None.

Example:

```
dgSchedule m_mysched;
// m_mysched is initialized and started
```

result = gryph->StopSchedule(&m_mysched);

3.3.28 AddResponder

Function Add a message responder process to the Gryphon.

Syntax bool **AddResponder** (dgResponder*)

Remarks For more information on creation of responders please see the class dgResponder (section 3.16).

Example:

```
dgResponder my_resp;
// Initialize my_resp ...
result = gryph->AddResponder (&my_resp);
```

3.3.29 SetResponderActive

Function Set active/inactive the message responder process represented by the dgResponder object.

Syntax bool **SetResponderActive** (dgResponder*, bool=TRUE)

Remarks While inactive, no response messages are issued by the responder process.

Examples:

```
result = gryph->SetResponderActive( &my_resp );
result = gryph->SetResponderActive(&my_resp, FALSE );
```

3.3.30 ResponderPresent

Function Determines if the message responder process is present on the Gryphon (i.e. has been downloaded and has not been deleted).

Syntax bool **ResponderPresent** (dgResponder*, bool* ***ispresent***)

Remarks If the function succeeds ***ispresent*** indicates whether or not the responder is present.

Example:

```
bool m_present;
result = gryph->ResponderPresent(&my_resp, &m_present );
```

3.3.31 DeleteResponder

Function Delete the message responder process from the Gryphon.

Syntax bool **DeleteResponder** (dgResponder*)

Remarks None.

Example:

```
result = gryph->DeleteResponder(&my_resp);
```

3.3.32 AddFile

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

Syntax bool **AddFile** (dgFile*, int flags = 0)

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 default value 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.

3.3.33 DeleteFile

Function Delete the previously uploaded file from the Gryphon.

Syntax bool **DeleteFile** (dgFile*)

Remarks None.

3.3.34 StartProgram

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

Syntax bool **StartProgram** (dgFile*, const dgString& ***args*** = "")
bool **StartProgram** (dgFile*, const 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.

3.3.35 StopProgram

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

Syntax bool **StopProgram** (dgFile*)

Remarks None.

3.4 Class dgChannel

Base class: **dgGryphonNode**

The class dgChannel represents a channel of a daughter card installed in a Gryphon. It is used to access information and services available on the channel, and to configure the channel.

3.4 GetDataMaxLen

Function Get the maximum valid length of a data field of a channel's network frame.

Syntax bool **GetDataMaxLen** (dgUint16*) const

Remarks None.

Example:

```
dgUint16 m_maxlen;
result = m_channel->GetDataMaxLen(&m_maxlen);
```

3.4.2 GetDataMinLen

Function Get the valid minimum length of the data field of a channel's network frame.

Syntax bool **GetDataMinLen** (dgUint16*) const

Remarks None.

Example:

```
dgUint16 m_minlen;
result = m_channel->GetDataMinLen(&m_minlen);
```

3.4.3 GetExtraMaxLen

Function Get the valid maximum length of an extra data field of a

channel's network frame.

Syntax bool **GetExtraMaxLen** (dgUint16*) const

Remarks None.

Example:

```
dgUint16 m_extramaxlen;
result = m_channel->GetExtraMaxLen(&m_extramaxlen);
```

3.4.4 GetExtraMinLen

Function Get the valid minimum length of an extra data field of a channel's network frame.

Syntax bool **GetExtraMinLen** (dgUint16*) const

Remarks None.

Example:

```
dgUint16 m_extraminlen;
result = m_channel->GetExtraMinLen(&m_extraminlen);
```

3.4.5 GetFirstValidHeaderLength

Function Get the first valid header length for channel's network frame format.

Syntax bool **GetFirstValidHeaderLength** (size_t*)

Remarks None.

Example:

```
size_t  m_headerlength;
result = m_channel->GetFirstValidHeaderLength(&m_headerlength);
```

3.4.6 GetNextValidHeaderLength

Function Get subsequent valid header length(s) for channel's network frame format.

Syntax bool **GetNextValidHeaderLength** (size_t*)

Remarks None.

Return Value Returns FALSE when the end of the list of valid header lengths has been reached.

Example:

```
size_t  m_headerlength;
// GetFirstValidHeaderLength is called successfully
while(m_channel->GetNextValidHeaderLength(&m_headerlength)) {
        // Handling for each valid header length
}
// All valid header lengths have been retrieved
```

3.4.7 GetName

Function Get the channel device name.

Syntax bool **GetName** (dgString*) const

bool **GetName**(char* ***buffer***, size_t ***max***) const

Remarks The second form of the function copies the initial ***max*** characters of the source string to memory pointed to by ***buffer***. If ***max*** is less than or equal to the length of the source string, a null character is not appended automatically to the buffer. If ***max*** is greater than the length of the source string, the destination buffer is padded with null characters up to length ***max***.

Example:

```
dgString m_channelname;
result = m_channel->GetName(&m_channelname);
```

3.4.8 GetSecurity

Function Get the encrypted device security string of the channels.

Syntax bool **GetSecurity** (dgString*) const

bool **GetSecurity**(char* ***buffer***, size_t ***max***) const

Remarks The second form of the function copies the initial ***max*** characters of the source string to memory pointed to by ***buffer***. If ***max*** is less than or equal to the length of the source string, a null character is not appended automatically to the buffer. If ***max*** is greater than the length of the source string, the destination buffer is padded with null characters up to length ***max***.

Example:

```
dgString m_securitystring;
result = m_channel->GetSecurity(&m_securitystring);
```

3.4.9 GetSerial

Function Get the serial number for the card of the specified channel.

Syntax bool **GetSerial** (dgString*) const

bool **GetSerial** (char* ***buffer***, size_t ***max***) const

Remarks The second form of the function copies the initial ***max*** characters of the source string to memory pointed to by ***buffer***. If ***max*** is less than or equal to the length of the source string, a null character is not appended automatically to the buffer. If ***max*** is greater than the length of the source string, the destination buffer is padded with null characters up to length ***max***.

Example:

```
dgString m_serialnum;
result = m_channel->GetSerial(&m_serialnum);
```

3.4.10 GetVersion

Function Get the device version.

Syntax bool **GetVersion** (dgString*) const

bool **GetVersion** (char* ***buffer***, size_t ***max***) const

Remarks The second form of the function copies the initial ***max*** characters of the source string to memory pointed to by ***buffer***. If ***max*** is less than or equal to the length of the source string, a null character is not appended automatically to the buffer. If ***max*** is greater than the length of the source string, the destination buffer is padded with null characters up to length ***max***.

Example:

```
dgString m_version;
result = m_channel->GetVersion(&m_version);
```

3.4.11 GetSlot

Function Get the slot number of the channel.

Syntax bool **GetSlot** (dgUint8*) const

Remarks None.

Example:

```
dgUint8 m_slotnum;
result = m_channel->GetSlot(&m_slotnum);
```

3.4.12 GetType

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

Syntax bool **GetType** (dgUint8*) const

Remarks None.

Example:

```
dgUint8 m_type;
result = m_channel->GetType(&m_type);
```

3.4.13 GetSubtype

Function Get the subtype of the channel.

Syntax bool **GetSubtype** (dgUint8*) const

Remarks None.

Example:

```
dgUint8 m_subtype;
result = m_channel->GetSubtype(&m_subtype);
```

3.4.14 GetFirstPresetSpeed

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

Syntax bool **GetFirstPresetSpeed** (dgChannelSpeed*)

Remarks Please see the class dgChannelSpeed for more on channel speed information.

Example:

```
dgChannelSpeed m_chanspeed;
result = m_channel->GetFirstPresetSpeed(&m_chanspeed);
```

3.4.15 GetNextPresetSpeed

Function Get information on each of the remaining preset speeds

available for a channel.

Syntax bool **GetNextPresetSpeed** (dgChannelSpeed*)

Remarks Use this function after the function **GetFirstPresetSpeed()**. Please see the class dgChannelSpeed for more on channel speed information.

Example:

```
dgChannelSpeed m_chanspeed;
// GetFirstPresetSpeed is called successfully
while(m_channel->GetNextPresetSpeed(&m_chanspeed)) {
        //Handling for each preset speed
}
// All preset speeds have been retrieved
```

3.4.16 GetGetSpeedIOCtl

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 bool **GetGetSpeedIOCtl** (dgUint32*)

Remarks None.

Example:

```
dgUint32 m_num;
if(m_channel->GetGetSpeedIOCtl(&m_num))
{
        result=m_channel->IOCtl(m_num, ...);
}
```

3.4.17 GetSetSpeedIOCtl

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 this channel.

Syntax bool **GetSetSpeedIOCtl** (dgUint32*)

Remarks None.

Example:

```
dgUint32 m_num;
if(m_channel->GetSetSpeedIOCtl(&m_num))
        result=m_channel->IOCtl(m_num, ...);
```

3.4.18 GetSpeedDataSize

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

Syntax size_t **GetSpeedDataSize** ( )

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=m_channel->GetSpeedDataSize();
```

3.4.19 GetSpeed

Function Get the current speed characteristics of the specified channel.

Syntax bool **GetSpeed** (dgChannelSpeed*)

Remarks None.

Example:

```
dgChannelSpeed m_chanspeed;
result=m_channel->GetSpeed(&m_chanspeed);
```

3.4.20 SetSpeed

Function Set the current speed characteristics for this channel.

Syntax bool **SetSpeed** (const dgChannelSpeed&)

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

See section 3.3 dgChannelSpeed for more information on setting the speed information of the channel.

Example:

```
dgChannelSpeed m_chanspeed;
// m_chanspeed is initialized
result=m_channel->SetSpeed(m_chanspeed);
```

3.4.21 Init

Function Initialize the channel for the most recent initialization parameters.

Syntax bool **Init** (dgUint8 ***mode***)

Remarks ***mode***:

0: Always initialize
1: Initialize if not previously initialized

Example:

```
dgUint8 init_arg=0;                    // Will initialize even if
result = m_channel->Init(init_arg);    // initialized previously

result = m_channel->Init( 1 );         // Will not initialize if already initialized
```

3.4.22 EnableLoopback

Function Enable or disable the echo back to the client application of transmitted messages.

Syntax bool **EnableLoopback** (bool)

Remarks None.

Example:

```
result = m_channel->EnableLoopback( TRUE );
        // Enables the echo back to the client of transmitted messages.

result = m_channel->EnableLoopback( FALSE );
        // Disables the echo back to the client of transmitted messages.
```

3.4.23 GetFirstEventType

Function Get information on the first event type that may be returned by the channel.

Syntax bool **GetFirstEventType** (dgEventType*)

Remarks None.

Example:

```
dgEventType m_event;
result = m_channel->GetFirstEventType(&m_event);
```

3.4.24 GetNextEventType

Function Get information on the remaining event types that may be returned by the channel.

Syntax bool **GetNextEventType** (dgEventType*)

Remarks Use this command after the **GetFirstEventType ()** command.

Refer to dgEventType.

Return Value Returns **false** when the list of possible events has been traversed.

Example:

```
dgEventType m_event;
// GetFirstEventType() is called successfully
while (m_channel->GetNextEventType(&m_event)) {
        // Handling of each successive event type
}
// All event type has been retrieved
```

3.4.25 ReportEventType

Function Specifies if an event type is to be delivered to the client application, or if it should be filtered out.

Syntax bool **ReportEventType** (dgEventType, bool)

Remarks The default state is to not report the event.

Example:

```
dgEventType m_event;
//m_event is initialized
result = m_channel->ReportEventType( m_event, TRUE )
        // Events of type "m_event" will be reported

result = m_channel->ReportEventType( m_event, FALSE );
        // Events of type "m_event" will NOT be reported
```

3.4.26 ReportAllEvents

Function Enable or disable reporting of all events for the channel to the client application.

Syntax bool **ReportAllEvents** (bool)

Remarks Parameter value:

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

Example:

```
result = m_channel->ReportAllEvents ( 1 );
result = m_channel->ReportAllEvents ( 0 );
```

3.4.27 AddFilter

Function Add the filter, represented by the dgFilter object, to the channel.

Syntax bool **AddFilter** (dgFilter*)

Remarks Please see the dgFilter class for more information.

Example:

```
dgFilter myfilter;
// myfilter is initialized and set with a dgFilterBlock
result = m_channel->AddFilter(&myfilter);
```

3.4.28 ModifyFilter

Function Activate or deactivate the previously added filter.

Syntax bool **ModifyFilter** (dgFilter*, dgUint8 ***action***)

Remarks ***action*** values: ACTIVATE_FILTER,
DEACTIVATE_FILTER

Example:

```
dgFilter myfilter;
// myfilter has already been added with AddFilter()
result = m_channel->ModifyFilter(&myfilter, DEACTIVATE_ FILTER );
```

3.4.29 DeleteFilter

Function Delete the filter from the channel.

Syntax bool **DeleteFilter** (dgFilter*)

Remarks Please see the dgFilter class for more information.

Example:

```
dgFilter myfilter;
// myfilter is initialized and added to the channel using AddFilter()
result = m_channel->DeleteFilter(&myfilter);
```

3.4.30 SetFilterMode

Function Specify whether the data from the channel is to be blocked, filtered or passed.

Syntax bool **SetFilterMode** (dgUint8 ***mode***)

Remarks ***mode***:

FILTER_OFF_BLOCK_ALL : No messages are to be passed to the client (default state).

FILTER_ON : Only those messages that are passed by an active filter or the default filter are passed to the client.

FILTER_OFF_PASS_ALL : All messages are passed to the client.

Example:

```
result = m_channel->SetFilterMode (FILTER_OFF_PASS_ALL);
result = m_channel->SetFilterMode (FILTER_ON);
//This must be set for any filters added with AddFilter() to work
```

3.4.31 SetDefaultFilterAction

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

Syntax bool **SetDefaultFilterAction** (dgUint8 ***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 = m_channel->SetDefaultFilterAction(DEFAULT_FILTER_PASS);
```

3.4.32 IOCtl

Function This function gives access to generic I/O functions and channel (expansion module)-specific I/O functions.

Syntax bool **IOCtl** (dgUint32 ***num***, dgUint8* ***data***, size_t ***length***)

Remarks For more information on the **IOCtl** parameters please see the Gryphon module documentation.

Example:

```
dgUint32 m_ionum;
dgUint8 m_iodata;
size_t  m_len;
// m_ionum, m_iodata, m_len are all initialized to valid values
result = m_channel->IOCtl(m_ionum, &m_iodata, m_len);
```

3.4.33 SetBLMParams

Function Specify the busload monitor behavior.

Syntax bool **SetBLMParams** (dgUint32 ***mode***,
dgUint32 ***averaging_period***)

Remarks **mode**: Indicates the desired busload monitor mode.

Valid values are:

0 BLM off
1 BLM average over time
2 BLM average over frame count

averaging_period: Indicates the period over which averages are calculated.

If ***mode***=1 (Time average): ***averaging_period*** is interpreted as a time period in units of milliseconds.

If ***mode***=2 (Frame count average): ***averaging_period*** is interpreted as a message count.

Example:

```
dgUint32 m_avgperiod;

result = m_channel->SetBLMParams(0); // No bus load monitoring done

m_avgperiod=1000;              // Bus load monitoring done over 1s.
result = m_channel->SetBLMParams(1, m_avgperiod);

m_avgperiod=100;     // Bus load monitoring done over next 100 messages.
result = m_channel->SetBLMParams(2, m_avgperiod);
```

3.4.34 GetBLMStats

Function Get the BLM and other statistics.

Syntax bool **GetBLMStats** (dgUint32* ***timestamp***,
dgUint16* ***blAvg***,
dgUint16* ***blNow***,
dgUint16* ***blPeak***,
dgUint16* ***blHistoricPeak***,

dgUint32* ***rxCnt***,
dgUint32* ***txCnt***,
dgUint32* ***rxDropCnt***,
dgUint32* ***txDropCnt***,
dgUint32* ***rxErrCnt***,
dgUint32* ***txErrCnt***)

Remarks

The statistics received will include busload, total frames received, total frames transmitted, total frames dropped by the device driver and average busload.

timestamp	System time at which the last bus load reading occurred
blAvg	Average bus load on the channel
blNow	Current bus load on the channel
blPeak	Highest bus load during the averaging period.
blHistoricPeak	Highest busload since busload monitoring was enabled.
rxCnt	Total frames received on a channel since the bus load monitor was enabled
txCnt	Total frames transmitted on a channel since the bus load monitor was enabled
rxDropCnt	Total received frames dropped since the bus load monitor was enabled
txDropCnt	Total transmitted frames dropped since the bus load monitor was enabled
rxErrCnt	Total received frame errors since the bus load monitor was enabled
txErrCnt	Total transmitted frame errors since the bus load monitor was enabled

Example:

```
dgUint32 m_timestamp, m_rxCnt, m_txCnt, m_rxDropCnt, m_txDropCnt,
         m_rxErrCnt, m_txErrCnt;
dgUint16 m_blAvg, m_blNow, m_blPeak, m_blHistoricPeak;

result = m_channel->GetBLMStats (&m_timestamp, &m_blAvg, &m_blNow,
&m_blPeak, &m_blHistoricPeak, &m_rxCnt, &m_txCnt, &m_rxDropCnt,
&m_txDropCnt, &m_rxErrCnt,  &m_txErrCnt);
```

3.5 Class dgChannelSpeed

Base class: **dgGryphonLibObject**

The dgChannelSpeed class can be used to obtain information on the available speeds for a channel, and for setting the speed of a channel. This class is used with the functions **dgChannel::GetFirstPresetSpeed**(), **dgChannel::GetNextPresetSpeed**(), **dgChannel::GetSpeed**() and **dgChannel::SetSpeed**(). Information retrieved via this class can also be used in calls to **dgChannel::IOCtl**().

Operators defined for this class: ==, !=, <, >

3.5.1 GetIOCtlData

Function Get the data that may be used in the data parameter of the **dgChannel::IOCtl**() function when setting the channel speed.

Syntax
bool **GetIOCtlData** (dgUint8* ***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:

```
dgUint8 *m_databuffer;
size_t  numread, total_size;
// initialize m_databuffer to an array of type dgUint8 and set total_size to the
// size of m_databuffer
result = dgspeed->GetIOCtlData( m_databuffer, total_size, &numread );
```

3.5.2 GetIOCtlDataSize

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

Syntax bool **GetIOCtlDataSize** (size_t*)

Remarks None.

Example:

```
size_t  m_datasize;
result = dgspeed->GetIOCtlDataSize( &m_datasize );
```

3.6 Class dgEventType

The class dgEventType represents an event that can occur for a channel, or other entity in a Gryphon. A dgEventType object may be used to access the numeric ID and a text string description of the event. dgEventType objects may be compared for equality or inequality. Objects of this type can also be passed to the function **dgChannel**::**ReportEventType**() to control which events will be passed to the client.

3.6.1 GetId

Function Get the numeric value for an event.

Syntax dgEventId **GetId** () const

Remarks None.

Example:

```
dgEventId  m_eventID;
                                        // EType is a pointer to an
m_eventID = EType->Getid();          // instance of dgEventType
```

3.6.2 GetMeaning

Function Get the text string description of the event.

Syntax void **GetMeaning** (dgString*) const
void **GetMeaning** (char* ***buffer***, size_t ***max***) const

[dgString **GetMeaning** () const – is now deprecated.]

Remarks The second form of the function copies the initial ***max*** characters of the source string to memory pointed to by ***buffer***. If ***max*** is less than or equal to the length of the source string, a null character is not appended automatically to the buffer. If ***max*** is greater than the length of the source string, the destination buffer is padded with null characters up to length ***max***.

Example:

```
dgString  evt_meaning;
                                                    // EType is a pointer to an
                                                    // instance of dgEventType
EType->GetMeaning( &evt_meaning );
```

3.7 Class dgFrame

Base class: **dgGryphonLibObject**

Class **dgFrame** represents a generic GC frame, and is the base class for classes representing specific frame types: dgCmdFrame, dgDataFrame, dgEventFrame, etc.

3.7.1 GetSrc

Function Get the value of the Source field of the GC frame header.

Syntax void **GetSrc** (dgUint8*) const

Remarks The Source field identifies what type of Gryphon source sent the GC frame. The following is a list of the valid types for the Source/Destination fields:

SD_SERVER	Server program
SD_CARD	Hardware Channels
SD_CLIENT	Client connection
SD_SCHED	Message Scheduler program
SD_SCRIPT	Script Processor program
SD_PGM	Program Loader
SD_BLM	Bus load monitor program
SD_USDT	ISO 15765-2 "network layer" node
SD_FLIGHT	Flight recorder application
SD_SCHED	Message responder program

Example:

```
dgUint8 src;
// myframe is a pointer to an instance of dgFrame or a derived class
myframe->GetSrc( &src );
switch (src) {
        case SD_SERVER:         // Server Program
            ...
        case SD_CARD:           // Hardware Channels
            .
            .
            .
```

```
            case SD_SCHED:          // Message Responder Program
                ...
}
```

3.7.2 SetSrc

Function Set the value of the Source field of the GC frame header.

Syntax void S**etSrc** (dgUint8)

Remarks Cf. GetSrc()

3.7.3 GetSrcChannel

Function Get the value of the Source Channel field of the GC frame.

Syntax void **GetSrcChannel** (dgUint8*) const

Remarks The Source Channel field further defines the source of the GC frame. For example, if the Source is an SD_CLIENT, the Source Channel field will contain the 'client ID'. Similarly, if the source is an expansion module channel, the Source Channel field will contain the 'channel ID'.

Example:

```
dgUint8  m_srcchan;
myframe->GetSrcChannel( &m_srcchan );
```

3.7.4 SetSrcChannel

Function Set the value of the Source Channel field of the GC frame header.

Syntax void S**etSrcChannel** (dgUint8)

Remarks Cf. GetSrcChannel()

3.7.5 GetDest

Function Get the value of the Destination field of a GC frame.

Syntax void **GetDest** (dgUint8*) const

Remarks For a listing of the Source/Destinations available please see the remarks for **GetSrc**.

Example:

```
dgUint8 dest;
// myframe is a pointer to an instance of dgFrame or another frame class
myframe->GetSrc( &dest );
switch ( dest ) {
        case SD_SERVER:         // Server Program
                ...
        case SD_CARD:           // Hardware Channels
                .
                .
        case SD_SCHED:          // Message Responder Program
                ...
}
```

3.7.6 SetDest

Function Set the value of the Destination field of a GC frame.

Syntax void **SetDest** (dgUint8)

Remarks Cf. GetDest().

3.7.7 GetDestChannel

Function Get the value of the Destination Channel field of the GC frame.

Syntax void **GetDestChannel** (dgUint8*) const

Remarks The destination channel further defines the destination. FT_MISC, FT_EVENT or FT_TEXT frames may use the

special destination channel CH_BROADCAST, which indicates a broadcast. Any such broadcast messages sent to this channel will be forwarded to all clients that have their broadcast flags set.

Example

```
dgUint8  m_destchan;
myframe->GetDestChannel( &m_destchan );
```

3.7.8 SetDestChannel

Function Set the value of the Destination Channel field of a GC frame.

Syntax void **SetDestChannel** (dgUint8)

Remarks Cf. GetDestChannel().

3.7.9 GetDataLength

Function Get the data length of a GC frame.

Syntax void **GetDataLength** (dgUint16*) const

Remarks Data length is the number of bytes of frame data that follow the frame header. This is an unsigned short value. The valid values range from zero to 7168 (1C00 hex).

Example:

```
dgUint16 datalen;
myframe->GetDataLength ( &datalen );
```

3.7.10 SetDataLength

Function Set the data length of a GC frame.

Syntax void **SetDataLength** (dgUint16)

Remarks Data length is the number of bytes of frame data that follow the frame header. This is an unsigned short value. The valid values range from zero to 7168 (1C00 hex).

When classes derived from the dgFrame class, it will not be necessary to use this function, as data length will be set automatically.

Example:

```
dgUint16 datalen;
// datalen is initialized
myframe->SetDataLength ( datalen );
```

3.7.11 GetFrameType

Function Get the frame type of the GC frame.

Syntax void **GetFrameType** (dgUint8*) const

Remarks The frame type field identifies the frame body format.

There are several types of GC frames:

Command Frames
Response Frames
Event Frames
Data Frames
Miscellaneous Frames
Text Frames
Broadcast Frames

When classes derived from the dgFrame class, it will not be necessary to use this function, as the class type will imply the frame type.

Example:

```
dgUint8 m_ftype;
myframe->GetFrameType( &m_ftype );
```

3.7.12 SetFrameType

Function Set the frame type of a GC frame.

Syntax void **SetFrameType** (dgUint8)

Remarks The frame type field identifies the frame body format.

There are five types of GC frames:

Command Frames
Response Frames
Event Frames
Data Frames
Miscellaneous Frames
Text Frames
Broadcast Frames

When classes derived from the dgFrame class, it will not be necessary to use this function, as frame type will be set automatically.

Example:

```
dgUint8 m_ftype;
// m_ftype is initialized
myframe->SetFrameType( m_ftype );
```

3.8 Class dgDataFrame

Base class: **dgFrame**

The class dgDataFrame provides services to help create well-formed GC data frames, as well as services to access information in data frames that were received from another source.

3.8.1 GetHeaderLength

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

Syntax void **GetHeaderLength** (dgUint8*) const

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

Example:

```
dgUint8 m_len;
dataframe->GetHeaderLength( &m_len );
```

3.8.2 SetHeaderLength

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

Syntax bool **SetHeaderLength** (dgUint8)

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

Example:

```
dgUint8 m_len;
// m_len is initialized to desired header length
dataframe->SetHeaderLength( m_len );
```

3.8.3 GetHeaderBits

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

Syntax void **GetHeaderBits** (dgUint8*) const

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 a multiple of eight (such as in the case for CAN, with 11- and 29-bit headers).

Example:

```
dgUint8 m_numbits;
dataframe->GetHeaderBits( &m_numbits );
```

3.8.4 SetHeaderBits

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

Syntax void **SetHeaderBits** (dgUint8)

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 a multiple of eight (such as in the case for CAN, with 11- and 29-bit headers). The default value is zero.

Example:

```
dgUint8 m_numbits;
// m_numbits is initialized to desired header length
dataframe->SetHeaderBits( m_numbits );
```

3.8.5 GetDataLength

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

Syntax void **GetDataLength** (dgUint16*) const

Remarks The Data Length field specifies the length in bytes of the data

segment of the payload message.

Example:

```
dgUint16 m_datalength;
dataframe->GetDataLength ( &m_datalength );
```

3.8.6 SetDataLength

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

Syntax bool **SetDataLength** (dgUint16)

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

Example:

```
dgUint16 m_datalength;
// m_datalength is initialized
dataframe->SetDataLength ( m_datalength );
```

3.8.7 GetExtraLength

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

Syntax void **GetExtraLength** (dgUint8*) const

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

Example:

```
dgUint8 m_extralen;
dataframe->GetExtraLength( &m_extralen );
```

3.8.8 SetExtraLength

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

Syntax bool **SetExtraLength** (dgUint8)

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:

```
dgUint8 m_extralen;
// m_extralen is initialized
dataframe->SetExtraLength( m_extralen );
```

3.8 GetMode

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

Syntax void **GetMode** (dgUint8*) const

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:

```
dgUint8 m_mode;
dataframe->GetMode ( &m_mode );
```

3.8.10 SetMode

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

Syntax void S**etMode** (dgUint8)

Remarks See the "Gryphon Hardware Information" web pages for supported "mode" values.

Example:

dataframe->SetMode (MODE_REMOTE);

3.8.11 GetErrorStatus

Function Get the value of the Status field of a GC data frame.

Syntax void **GetErrorStatus** (dgUint8*) const

Remarks The Status byte contains network-specific flags and status information.

Example:

```
dgUint8 m_status;
dataframe->GetErrorStatus ( &m_status );
```

3.8.12 SetErrorStatus

Function Set the value of the Status field of a GC data frame.

Syntax void **SetErrorStatus** (dgUint8)

Remarks The Status byte contains network-specific flags and status information.

Example:

```
dgUint8 m_status;
dataframe->SetErrorStatus ( m_status );
```

3.8.13 GetPriority

Function Get the value of the Priority field of a GC data frame.

Syntax void **GetPriority** (dgUint8*) const

Remarks Priority is not currently used and should be treated as a

reserved byte.

Example:

```
dgUint8 m_priority;
dataframe->GetPriority ( &m_priority );
```

3.8 SetPriority

Function Set the value of the Priority field of a GC data frame.

Syntax void **SetPriority** (dgUint8)

Remarks Priority is not currently used and should be treated as a reserved byte.

Example:

```
dgUint8 m_priority;
// m_priority is initialized
dataframe->SetPriority ( m_priority );
```

3.8.15 GetTimestamp

Function Get the timestamp of a GC data frame.

Syntax void **GetTimestamp**(dgUint32*) const

Remarks

Example:

```
dgUint32 ts;
dataframe->GetTimestamp( &ts );
```

3.8.16 GetPayload

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

Syntax void **GetPayload** (dgUint8*) const

Remarks The parameter 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.

Example:

```
dgUint8  *m_payload;
// m_payload is initialized to an array of dgUint8 equal to the current
// message's header length, data length, and extra length
dataframe->GetPayload ( m_payload );
```

3.8.17 SetPayload

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

Syntax void **SetPayload** (const dgUint8* ***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:

```
dgUint8 *m_payload;
// SetHeaderLength() is called and m_payload is initialized to an array of
// dgUint8 with the message's payload
dataframe->SetPayload( m_payload );
```

3.9 Class dgEventFrame

Base class: **dgFrame**

This class provides services to access information in event frames.

Various entities in a Gryphon network, such as network channel drivers or well-known clients, can send GC event notification frames to clients to inform them of certain 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).

3.9.1 GetEventId

Function Get the numeric identifier of the event.

Syntax void **GetEventId** (dgEventId*) const

Remarks Please see the module documentation for card specific events.

Example:

```
dgEventId id;
eventframe->GetEventId ( &id );
```

3.9.2 GetContext

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

Syntax void **GetContext** (dgUint8*) const

Remarks None.

Example:

```
dgUint8 my_context;
eventframe->GetContext( &my_context );
```

3.9.3 GetTimestamp

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

Syntax void **GetTimestamp** (dgUint32*) const

Remarks The timestamp is in units of 10 microseconds.

Example:

```
dgUint32 my_timestamp;
eventframe->GetTimestamp( &my_timestamp );
```

3.9.4 GetData

Function Get the data specific to the event.

Syntax
```
void GetData ( dgUint8* evtdata,
               size_t maxlen,
               size_t* actual_len ) const
```

Remarks User sets ***maxlen*** to max desired length, (e.g. buffer size); ***actual_len*** is guaranteed <=maxlen. Returned data is the 'Event Frame Data'.

Example:

```
dgUint8* my_data;
size_t maxlen, len;
// my_data is initialized to an array of dgUint8 and maxlen is set to its length
eventframe->GetData( my_data, maxlen, &len );
```

3.10 Class dgCmdFrame

Base class: **dgFrame**

The class dgCmdFrame provides services to help create well-formed GC command frames.

Most of the Gryphon commands are encapsulated in library member functions; e.g., the function **dgGryphon::SetSortMode**() sends the command "CMD_SERVER_SET_SORT" to the Gryphon server, and waits for the corresponding response frame. If the available member functions are not adequate for a user's needs, the dgCmdFrame class should be helpful.

3.10.1 SetCmdID

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

Syntax void **SetCmdID** (dgUint8)

Remarks None.

Example:

```
dgUint8 m_cmd = CMD_SERVER_SET_SORT;
cmdframe->SetCmdID( m_cmd );
```

3.10.2 SetContext

Function Set the user-specified data that will be returned with the response to this command.

Syntax void **SetContext** (dgUint8)

Remarks This data field is optional and does not modify the command in any way. Clients can use this field to identify the responses to specific commands.

Example:

```
dgUint8 m_context;
```

```
// m_context is initialized
cmdframe->SetContext( m_initialized );
```

3.10.3 SetCmdData

Function Set the command-specific data.

Syntax void **SetCmdData** (const dgUint8* ***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:

```
dgUint8 m_buffer[4], m_len;
// m_buffer is initialized with data and m_len is initialized with the
// length of m_buffer
cmdframe.SetCmdData( &m_buffer, sizeof(m_buffer) );
```

3.11 Class dgRespFrame

Base class: **dgFrame**

The dgRespFrame class provides services to retrieve information from a GC response frame.

Most of the Gryphon's response frames are consumed and interpreted by library member functions. If the available member functions are not adequate for a user's needs, the dgRespFrame class should be helpful.

3.11.1 GetCmd

Function Get the command ID of the command frame in response to which this frame was sent.

Syntax bool **GetCmd** (dgUint8*) const

Remarks None.

Example:

```
dgUint8 m_cmd;
result = respframe->GetCmd( &m_cmd );
```

3.11.2 GetReturnCode

Function Get the return code of the response frame.

Syntax bool **GetReturnCode** (dgUint32*) const

Remarks 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:

```
dgUint32 m_retval;
result = respframe->GetReturnCode( &m_retval );
```

3.11.3 GetRespData

Function Get the response data that is generated by the execution of the original command.

Syntax bool **GetRespData** (dgUint8* ***respdata***,
size_t ***maxlen***,
size_t* ***actual_len***) const

Remarks Parameter ***respdata*** points to the user's buffer; data bytes will be copied to this buffer. Parameter ***maxlen*** 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_len***.

Example:

```
dgUint8 *m_resp;
size_t  m_max, m_len;
// m_resp is initialized as an array of dgUint8 and m_max set to its length
result = respframe->GetRespData( m_resp, m_max, &m_len);
```

3.11.4 GetContext

Function Get the value of the context field of the original command frame.

Syntax bool **GetContext** (dgUint8*) const

Remarks None.

Example:

```
dgUint8 m_context;
result = respframe->GetContext( &m_context );
```

3.12 Class dgMiscFrame

Base class: **dgFrame**

This class represents GC frames of type FT_MISC that can be used to transfer arbitrary data between Gryphon nodes, such as between Gryphon clients. Frames of this type have an undefined message body format, so they are suitable for carrying any user-defined protocol messages.

3.12.1 GetData

Function	Get the user data that is encapsulated by the FT_MISC frame.
Syntax	void **GetData**(dgUint8 * ***buffer***, size_t ***maxlen***, size_t* ***actual_len***) const;
Remarks	Set ***maxlen*** to the max desired length, e.g., the user buffer size. ***actual_len*** is guaranteed <= ***maxlen***. The contents of the "Data" field (w/o padding) are copied to ***buffer***.

3.12.2 SetData

Function	Set the user data to be encapsulated by the FT_MISC frame.
Syntax	bool **SetData**(const dgUint8 * ***buffer***, size_t ***length***);
Remarks	Set ***length*** to the size of ***buffer***. The ***buffer*** is then copied into the “Data” field of the frame.

3.13 Class dgText Frame

Base class: **dgFrame**

This class represents GC frames of type FT_TEXT. This frame type is similar to the FT_MISC frame type, but instead of arbitrary data, its message body format is null-terminated ASCII text.

3.13.1 GetData

Function	Get the text data that is encapsulated by the FT_TEXT frame.
Syntax	void **GetData**(dgString *) const; void **GetData**(char * ***buffer***, size_t ***max***) const;
Remarks	The second form of the function copies the initial ***max*** characters of the source string to ***buffer***. If ***max*** is less than or equal to the length of the source string, a null character is not appended automatically to the buffer. If ***max*** is greater than the length of the source string, the destination buffer is padded with null characters up to length ***max***.

3.13.2 SetData

Function	Set the text data to be encapsulated by the FT_TEXT frame.
Syntax	bool **SetData**(const dgString&); bool **SetData**(char* ***str***);
Remarks	In the second form of this function, ***str*** is assumed to be null-terminated.

3.14 Class dgFilter

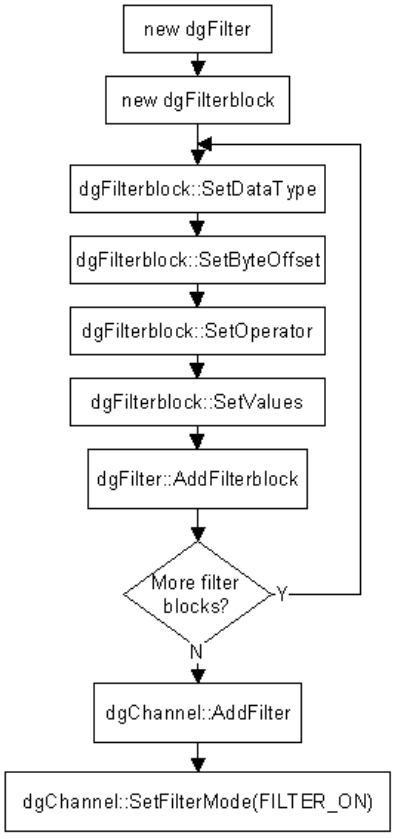
Base class: **dgGryphonLibObject**

The dgFilter class is used to create and manage a set of filters, associated with a dgChannel object, which can be used to filter messages directed to the client. Incoming messages are checked against each filter in the order that they were added to the set. The "action" (pass or block) of the first conforming filter (set by **SetAction**()) will cause that message to be passed to the client or blocked from it. If the message does not conform to any of the filters, the default action, as specified by **dgChannel::SetDefaultFilterAction**(), will determine its fate.

A message conforms to a filter only if all the filter's *Filter Blocks* conform to the message (see Class dgFilterblock).

See also: **dgChannel::AddFilter**(), **dgChannel::DeleteFilter**(), **dgChannel::SetFilterMode**().

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



3.14.1 SetActive

Function Set the filter on or off.

Syntax bool **SetActive** (bool)

Remarks If the parameter is set to true, the filter will be applied to all messages; otherwise the filter will not be applied to any messages. The default state of a dgFilter object is "on".

Example:

result = myfilter->SetActive(TRUE);

result = myfilter->SetActive(FALSE);

3.14.2 SetAction

Function Specify whether messages conforming to the filter will be blocked or passed.

Syntax bool **SetAction** (dgUint8 ***action***)

Remarks ***action***:

FILTER_FLAG_PASS Messages conforming to the filter will be passed to the client.

FILTER_FLAG_BLOCK Messages conforming to the filter will not be passed to the client.

The default state of this attribute is FILTER_FLAG_PASS.

Example:

result = myfilter->SetAction (FILTER_FLAG_BLOCK);

result = myfilter->SetAction (FILTER_FLAG_PASS);

3.14.3 SetLogicalOp

Function Specify whether the filter blocks are to be ANDed or ORed together.

Syntax bool **SetLogicalOp** (dgUint8 ***op***)

Remarks ***op***:

FILTER_FLAG_AND_BLOCKS All of the filter blocks must be true in order for a message to conform to the filter.

FILTER_FLAG_OR_BLOCKS Only one of the filter blocks must be true in order for a message to conform to the filter.

The default state of this attribute is FILTER_FLAG_AND_BLOCKS.

Example:

```
result = myfilter->SetLogicalOp ( FILTER_FLAG_OR_BLOCKS );
```

3.14.4 AddFilterblock

Function Add a filter block to a filter.

Syntax bool **AddFilterblock** (const dgFilterblock&)

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 the entire defined filter blocks associated with that filter. Please see section 3.14 for more information on creating filter blocks.

This is a copy operation on the supplied dgFilterBlock object parameter. For example, it can be modified and/or reused in the same or another dgFilter object.

This function involves a copy operation performed on the dgFilterBlock object. Once the copy is made, the filter block might be modified and reused in the same or another dgFilter

object. Thus, changes made to the filter block after the call to **dgFilter::AddFilterblock()** have no effect on the referenced filter object.

Example:

```
dgFilterblock myblock;
// myblock is initialized using dgFilterblock functions
result = myfilter->AddFilterblock(myblock);
```

3.15 Class dgFilterblock

Base class: **dgGryphonLibObject**

The dgFilterblock class is used to create filter blocks for a filter. Filter blocks specify a section of the message to be checked.

3.15.1 SetDataType

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

Syntax bool **SetDataType** (dgUint8 ***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 = myblock->SetDataType( FILTER_DATA_TYPE_DATA );

result = myblock->SetDataType( FILTER_DATA_TYPE_HEADER_FRAME );
```

3.15.2 SetByteOffset

Function Identify the location of the section of the message to be checked.

Syntax bool **SetByteOffset** (dgUint16 ***offset***)

Remarks 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.

The default value is zero.

Example:

```
dgUint16 m_offset;
// m_offset is initialized
result = myblock->SetByteOffset ( m_offset );
```

3.15.3 SetOperator

Function Specify the type of comparison to be done in this filter block.

Syntax bool **SetOperator** (dgUint8 ***op***)

Remarks

Operator	Value 1	Value 2	Notes
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 = myblock->SetOperator(VALUE_EQ);

result = myblock->SetOperator(BIT_FIELD_CHECK);

3.15.4 SetValues

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

Syntax

bool **SetValues** (dgUint16 ***length***,
dgUint8* ***value1***,
dgUint8* ***value2***=0)

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 **SetOperator()** must be called before

this function can succeed.

For more information about **SetOperator()** and the available operators please see section 3.14.3.

Example:

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

dgUint8 m_val1, m_val2;
dgUint16 m_num;

// m_num and m_val1 are initialized
result = myblock->SetValues ( m_num, &m_val1 );

// m_num, m_val1 and m_val2 are initialized and SetOperator used
// BIT_FIELD_CHECK
result = myblock->SetValues ( m_num, &m_val1, &m_val2 );
```

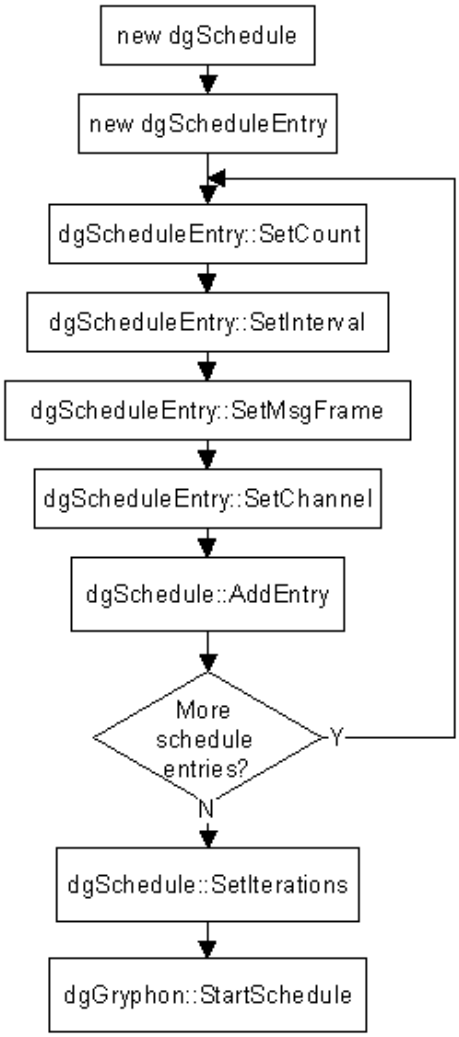
3.16 Class dgSchedule

Base class: **dgGryphonLibObject**

Clients use the dgSchedule class to transmit a set of messages over a specified channel (see **dgScheduleEntry::SetChannel**()). This class will allow users to specify additional transmit options (e.g. iterations, transmit intervals, etc).

See also **dgGryphon::StartSchedule**() and **dgGryphon::StopSchedule**().

As an overview, the following flowchart illustrates one approach to setting up a Gryphon schedule:



3.16.1 SetCritical

Function Specify whether the Gryphon schedule object is to be time-critical or not.

Syntax bool **SetCritical** (bool)

Remarks Time deviations in non-critical schedules will be dictated by bus activity and other processes running and may vary from situation to situation.

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 = mysched->SetCritical( TRUE );

result = mysched->SetCritical( FALSE );
```

3.16.2 SetIterations

Function Set the number of iterations of the list of schedule entries.

Syntax bool **SetIterations** (dgUint32)

Remarks A value of 0xFFFFFF indicates an indefinite number of iterations. Entries in the list are processed serially.

Example:

```
result = mysched->SetIterations ( 100 );

dgUint32 m_iterations;
// m_iterations is initialized
result = mysched->SetIterations ( m_iterations );
```

3.16.3 AddEntry

Function Add an entry to a list to be scheduled.

Syntax bool **AddEntry** (const dgScheduleEntry&)

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

Example:

```
dgScheduleEntry m_schentry;
// m_schentry is initialized using functions in dgScheduleEntry
result = mysched->AddEntry( m_schentry );
```

3.17 Class dgScheduleEntry

Base class: **dgGryphonLibObject**

The dgScheduleEntry class creates and configures a schedule entry. See class dgSchedule for more information on creating schedules.

3.17.1 SetMsgFrame

Function Set the message frame to be transmitted.

Syntax bool **SetMsgFrame** (const dgDataFrame&)

Remarks This function performs a copy operation on the data frame.

Example:

```
dgDataFrame m_frame;
// m_frame is initialized using the functions in dgDataFrame
result = m_schentry->SetMsgFrame( m_frame );
```

3.17.2 SetChannel

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

Syntax bool **SetChannel** (const dgChannel&)

Remarks None.

Example:

```
dgChannel m_channel;
// m_channel is initialized using the functions in dgChannel
result = m_schentry->SetChannel ( m_channel );
```

3.17.3 SetInterval

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

Syntax bool **SetInterval** (dgUint32 ***interval***)

Remarks None.

Example:

```
result = m_schentry->SetInterval( 1000 );  // 1 sec interval

dgUint32 m_interval;
// m_interval is initialized;
result = m_schentry->SetInterval( m_interval );
```

3.17.4 SetCount

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

Syntax bool **SetCount** (dgUint32 ***count***)

Remarks None.

Example:

```
result = m_schentry->SetCount( 100 );  // message will be sent 100 times

dgUint32 m_num;
// m_num is initialized
result = m_schentry->SetCount( m_num );
```

3.17.5 SetDelay

Function Set the number of milliseconds to wait before the initial transmit of the message owned by this object within each iteration of the dgSchedule object.

Syntax bool **SetDelay** (dgUint32 ***delay***)

Remarks The number of iterations of the dgSchedule object is set by **dgSchedule::SetIterations().**

Example:

```
result = m_schentry->SetDelay( 100 );  // 100ms delay before first message
                                       //  is sent

dgUint32 m_delay;
// m_delay is initialized
result = m_schentry->SetDelay( m_delay );
```

3.18 Class dgResponder

Base class: **dgGryphonLibObject**

The dgResponder class is used to create and manage Gryphon Message Responders. A Gryphon Message Responder consists of a set of filter conditions together with a list of GC frames. When an active Responder is added to the Gryphon, its filter criteria are applied to incoming Data and Event frames from the selected Gryphon channel. When an incoming frame conforms to the filter criteria, the associated GC frames are transmitted. These frames may be of any type, and transmitted to any destination.

See also: **dgGryphon::AddResponder**(), et al.

For details of filter operation, see classes dgFilter and dgFilterblock.

As an overview, the following flowchart illustrates one way to set up a message responder:

3.18.1 SetChannel

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

Syntax bool **SetChannel** (const dgChannel&)

Remarks Note: this syntax replaces **SetChannel** (dgChannel*) which is now deprecated.

Example:

```
dgChannel m_channel;
//m_channel is initialized using the functions in dgChannel
result = m_responder->SetChannel( m_channel );
```

3.18.2 SetAction

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

Syntax bool **SetAction** (dgUint8 ***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 = m_responder->SetAction( FR_RESP_AFTER_PERIOD );

dgUint8 m_action;
//m_action is initialized
result = m_responder->SetAction( m_action );
```

3.18.3 SetActionValue

Function Set the amount of time in tens of milliseconds, or the number of conforming messages to use for the period specified by the **SetAction()** entry.

Syntax bool **SetActionValue** (dgUint16 ***value***)

Remarks None.

Example:

```
//dependant on SetAction()
dgUint16 m_actionval;
// m_actionval is initialized
result = m_responder->SetActionValue( m_actionval );
```

3.18.4 AddFilterblock

Function Add filter information for the message responder.

Syntax bool **AddFilterblock** (const dgFilterblock&)

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. For more information on creation of the filter block, please see class dgFilterblock.

This is a copy operation.

Example:

```
dgFilterblock m_filterblock;
// m_filterblock is initialized using the functions in dgFilterblock
result = m_responder->AddFilterBlock( m_filterblock );
```

3.18.5 AddResponseFrame

Function Add a response frame for the message responder.

Syntax bool **AddResponseFrame** (const dgFrame&)

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

Each response message is a complete GC frame.

This is a copy operation.

Example:

```
dgFrame m_frame;
//m_frame is initialized
result = m_responder->AddResponseFrame( m_frame );
```

3.18.6 SetReplacement

Function Specify that a previously created responder is to be replaced by this responder.

Syntax bool **SetReplacement** ()

Remarks After calling this function, the dgResponder object may be used to replace the existing corresponding Gryphon entity via a call to dgGryphon::AddResponder(). The filters and responses of the dgResponder object are removed in order for new ones may be added. None of the object's other attributes are modified.

Example:

```
result = m_responder->SetReplacement();
        // m_responder will replace any responder on the channel it's added to
```

3.19 Class dgFile

Base class: **dgGryphonLibObject**

An object of this class represents a file that can be copied onto the Gryphon. It may either be a program that is executable on the Gryphon, or arbitrary data. The contents of the file may be encoded in ASCII, with end-of-line characters, or it may consist of any binary data.

A dgFile object can be copied to the Gryphon with the function **dgGryphon::AddFile**(). It can be removed from the Gryphon with **dgGryphon::DeleteFile**(). If the file contains an executable program, execution is started with **dgGryphon::StartProgram**(), and stopped with **dgGryphon::StopProgram**().

3.19.1 SetName

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

Syntax
bool **SetName** (const dgString &)
bool **SetName** (const char* ***str***)

Remarks Only the first 32 characters of the parameter string will be used.

In the second form of this function, ***str*** is assumed to be null-terminated.

3.19.2 SetPath

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

Syntax
bool **SetPath** (const dgString &)
bool **SetPath** (const char* ***str***)

Remarks Do not include a trailing directory separator character.

If this member function is not invoked, the default path to the file is the root directory.

In the second form of this function, ***str*** is assumed to be null-terminated.

3.19.3 SetDescription

Function Assign a text description of the dgFile object.

Syntax bool **SetDescription** (const dgString &)
bool **SetDescription** (const char* ***str***)

Remarks Only the first 80 characters of the parameter string will be used.

In the second form of this function, ***str*** is assumed to be null-terminated.

3.19.4 SetExecutable

Function Specify whether the uploaded file should be set as "executable" in the Gryphon.

Syntax bool **SetExecutable** (bool)

Remarks None.

3.20 Class dgGryphonNode

Base class: **dgGryphonLibObject**

The **dgGryphonNode** class is the base class for the **dgChannel, dgClient** and **dgUSDT** classes. A **dgGryphonNode** object represents a node in the Gryphon network which can act as a source and destination for GC frames.

3.20.1 GetNodeType

Function Get the node type.

Syntax void **GetNodeType**(dgUint8* ***type***) const

Remarks ***type*** may be **SD_CLIENT** for a Gryphon client, **SD_CARD** for a hardware channel, or **SD_USDT** for an ISO 15765-2 "network layer" node.

3.20.2 SetNodeType

Function Set the node type.

Syntax void **SetNodeType**(const dgUint8 ***type***)

Remarks See previous function description for values of ***type.*** This member is private in the derived classes.

3.20.3 GetNodeId

Function Get the numeric identifier of the node.

Syntax void **GetNodeId**(dgUint8* ***id***) const

Remarks Within each node "type", each instance will have a unique identifier.

3.20.4 SetNodeId

Function Set the numeric identifier of the node.

Syntax void **SetNodeId**(const dgUint8 ***id***)

Remarks Within each node "type", each instance will have a unique identifier.

Alternatively, the reserved value CH_BROADCAST can be used as the ***id*** value with a node type of SD_CLIENT to specify a broadcast target of all clients when used with **dgGryphon::SendFrame()**.

3.21 Class dgClient

Base class: **dgGryphonNode**

The class dgClient represents a client of the Gryphon server, i.e., a process that has opened a TCP/IP session with the Gryphon server program. An object of this class may be used to identify a source or destination for GC frame transfers; see **dgGryphon::SendFrame(), dgGryphon::RecvFrame()** and **dgGryphon::SetOnRx()**.

Aside from the constructor/destructor, there are currently no member functions specific to this class.

3.22 Class dgUSDT

Base class: **dgGryphonNode**

The class dgUSDT provides the ISO 15765-2 "network layer" service, along with additional services currently supplied by the Gryphon "USDT" server.

3.22.1 SetActive

Function Activate or deactivate the USDT service.

Syntax bool **SetActive** (bool)

Remarks When activating the service, the service parameters specified by all the other member functions (loopback, event reporting, etc.) are used. For this reason, the other member 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 irrevelant.

3.22.2 EnableLoopback

Function Enable or disable the echo back to the client application of transmitted ISO 15765-2 messages.

Syntax bool **EnableLoopback** (bool)

Remarks Default = no loopback.

3.22.3 ReportEvent

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

Syntax bool **ReportEvent** (dgEventId , bool)

Remarks The default state is to report all USDT events. Some

dgEventIds which are configurable here are:

USDT_FIRSTFRAME
USDT_LASTFRAME
USDT_DONE
USDT_ERROR

Example:

```
result = flowctrl_node->ReportEvent( USDT_LASTFRAME, FALSE )
                    // USDT_LASTFRAME events will NOT be reported
```

3.22.4 ReportAllEvents

Function Enable or disable reporting of all USDT events detected by the node to the client application.

Syntax void **ReportAllEvents** (bool)

Remarks Parameter value:

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

3.22.5 SetFilterMode

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

Syntax bool **SetFilterMode** (dgUint8 ***mode***)

Remarks ***mode***:

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 to be passed to the client (default). This option also enables timeout and sequence error event reporting.

Example:

result = flowctrl_node->SetFilterMode (FILTER_OFF_BLOCK_ALL);

3.22.6 SetDataLink

Function Specify which Gryphon CAN channel to use for data link services.

Syntax bool **SetDataLink** (const dgChannel&)

Remarks The dgChannel object must have previously been associated with a Gryphon hardware channel; e.g., via a successful call to **dgGryphon::GetNextChannel**().

Example:

```
result = gryphon->GetNextChannel( my_channel );
// determine if this is the desired CAN channel ...
// Init() the channel, and so on ...
// & use it as the data link channel for ISO 15765-2:
result = flowctl_node->SetDataLink( my_channel ) ;
```

3.22.7 SetPad

Function Specify desired message padding.

Syntax void **SetPad** (dgUint8)

Remarks Parameter value:

0 = Pad messages with 0x00's.
1 = Pad messages with 0xFF's.
2 = Do not pad messages (default).

3.22.8 SetIdSize

Function Specify CAN identifier size(s) to work with.

Syntax void **SetIdSize** (dgUint8)

Remarks Parameter value:

0 = Use 11-bit identifiers only (default).
1 = Use 29-bit identifiers only.
2 = Use both 11- and 29-bit identifiers.

3.22.9 SetExtAddrIds

Function Specify a set of CAN identifiers for which to use extended addressing, replacing the default value(s).

Syntax void **SetExtAddrIds** (const std::vector<unsigned>&)

Remarks 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:
std::vector<unsigned> eaIds;
eaIds.push_back( 0x42 );
eaIds.push_back( 0x44 );
eaIds.push_back( 0x46 );
flowctrl_node->SetExtAddrIds( eaIds );
```

3.22.10 SetIdBlock

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

Syntax bool SetIdBlock (dgUint8 ***blkIndex***,
dgUint32 ***blkSize***,
dgUint32 ***target_addr***,
dgUint32 ***source_addr***)

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 member 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:

```
result = flowctrl_node->SetIdBlock( 0, 32, 0x0640, 0x0240 );
```

3.23 Non-Class Functions

3.23.1 dgLibGetVersion

Function Get the version string of the library.

Syntax void **dgLibGetVersion** (dgString* ***str***)

void **dgLibGetVersion** (char* ***str***, size_t ***max***)

Remarks ***str*** points to the library version string returned.

The second form of the function copies the initial ***max*** characters of the source string to memory pointed to by ***str***. If ***max*** is less than or equal to the length of the source string, a null character is not appended automatically to the buffer. If ***max*** is greater than the length of the source string, the destination buffer is padded with null characters up to length ***max***.

Example:

```
dgString tempstr;
dgLibGetVersion ( &tempstr );
```