D N^{o}	**Jxxxx**
D N^{o} Description	**Interface Specification**
Document Type[1]	**Design Report**

Document Title

Interface Specification for Gemini Interface DLL

Related Documents

Title	Rev	Reference

Revision History

Rev	Date	Comments	Author	Approved
1.0	12/11/08	Written	SPG	
1.1	26/01/09	Further features added	SPG	
1.2	12/02/09	Continuing Development	SPG	
1.3	16/03/09	Initial release of Gemini Sonar	SPG	
1.4	08/05/09	Release with first units	SPG	
1.5.3	08/06/09	Release with second units	SPG	
1.5.10	18/08/09	Release for SDK Development	SPG	

Note
[1] Document type to be selected from
Design Proposal
Design Report
Design Specification
Test Report

All documents to be saved in \\atlantic\design under relevant D Number
Users Manual

Index

Overview

This document describes the DLL which acts as the interface between the Gemini sonar head and other systems, such as Seanet, Evo (the framework underpinning the Eclipse software), the production software used by the manufacturer of the Gemini PCBs, and as an interface for users to gain programmatic access to the Gemini sonar in their own software. This document will describe all the functionality exposed by the DLL, not all of which will be applicable to specific interfacing tasks.

The DLL is written in C++, using Microsoft Visual Studio.

This document describes version 1.05.10 of the DLL, which returns the version string `"Gemini Comms V1.05.10 SRDUK Ltd."`

In respect of the interface between the Gemini sonar head and the Seanet software, the Gemini head will be made to look like a SeaKing sonar head, which will substantially reduce the changes which need to be made to the Seanet software during the integration of Gemini.

Interface

The following paragraphs describe the DLL interface.

Constants

The following constants are defined in the DLL, and are used to indicate the type of data being passed from the DLL to the user defined data handling callback function.

```
PING_HEAD               0
PING_DATA               1
PING_TAIL               2
GEM_STATUS              3
GEM_ACKNOWLEDGE         4
GEM_SERIAL              5
GEM_FLASH_RESULT        6
GEM_BEARING_DATA        7
GEM_FLASH_READBACK      8
GEM_TEST_RESULT         9
PING_TAIL_EX            10
GEM_IP_CHANGED          11
GEM_UNKNOWN_DATA        12
GEM_VELOCIMETER_DATA    13
GEM_SERIAL_PORT_INPUT   14
GEM_PROD_DATA           15
```

The following constants are used to indicate the type of data being passed from the DLL to the user defined programming callback function when the modifier tables are being written.

```
GEM_MODPROG_START       0
GEM_MODPROG_SUCCESS     1
GEM_MODPROG_FAILURE     2
GEM_MODPROG_COMPLETE    3
GEM_MODPROG_NOT_ALIGNED 4
GEM_MODPROG_BAD_COMPARE 5
GEM_MODPROG_TIMEOUT     6
GEM_MODPROG_CALCULATING 7
GEM_MODPROG_ABORTED     8
GEM_MODPROG_LOW_ADDRESS 9
GEM_MODPROG_FAILED      10
GEM_MODPROG_CALCFAIL    11
```

The following constants are used to indicate the type of data being passed from the DLL to the user defined programming callback function when the FPGA images are being written.

```
GEM_FPGAPROG_START        20
GEM_FPGAPROG_SUCCESS      21
GEM_FPGAPROG_FAILURE      22
GEM_FPGAPROG_COMPLETE     23
GEM_FPGAPROG_NOT_ALIGNED  24
GEM_FPGAPROG_BAD_COMPARE  25
GEM_FPGAPROG_TIMEOUT      26
GEM_FPGAPROG_ABORTED      27
GEM_FPGAPROG_LOW_ADDRESS  28
GEM_FPGAPROG_NO_FILE      29
GEM_FPGAPROG_NO_ADDR      30
GEM_FPGAPROG_FAILED       31
GEM_FPGAPROG_FILESIZE     32
```

The following constants are used to indicate the result of the functions which read and write modifiers to file.

```
GEM_FILE_FAILED           0
GEM_FILE_SUCCESS          1
GEM_FILE_OPEN_FAILED      2
GEM_FILE_WRITE_FAILED     3
GEM_FILE_READ_FAILED      4
GEM_CALC_FAILED           5
```

The following constants are used by the retry mechanism which allows the calling program to re-request data which may have been lost in transmission.

```
GEM_RETRY_REQUEST_HEAD    0
GEM_RETRY_REQUEST_LINE    1
GEM_RETRY_REQUEST_TAIL    2
```

Structures

The following structures are used in the interface of the DLL. The C++ class definitions for these are contained in the file GeminiStructures.h. Pointers to these structures are returned to the parent code through the callback function (see below).

CGemHdr

The CGemHdr structure is a common structure which is returned at the start of all data sent from the DLL to the parent software.

Members

```
unsigned char  m_type;
unsigned char  m_version;
unsigned short m_deviceID;
unsigned short m_packetLatency;
unsigned short m_spare;
```

The fields within the CGemHdr structure are defined below

Field	Definition
m_type	Identifies the type of data contained in the structure (see descriptions of individual structures for values)
m_version	Version number
m_deviceID	The unique ID of the sonar
m_packetLatency	Not currently implemented
m_spare	Not used

CGemStatusPacket

The CGemStatusPacket is broadcast once per second by the Gemini Sonar head

Members

```
CGemHdr         m_head;
unsigned short m_firmwareVer;
unsigned short m_sonarId;
unsigned int   m_sonarFixIp;
unsigned int   m_sonarAltIp;
unsigned int   m_surfaceIp;
unsigned short m_flags;
unsigned short m_vccInt;
unsigned short m_vccAux;
unsigned short m_dcVolt;
unsigned short m_dieTemp;
unsigned short m_unusedTemp;
unsigned short m_vga1aTemp;
unsigned short m_vga1bTemp;
unsigned short m_vga2aTemp;
unsigned short m_vga2bTemp;
unsigned short m_psu1Temp;
unsigned short m_psu2Temp;
unsigned int   m_currentTimestampL;
unsigned int   m_currentTimestampH;
unsigned short m_transducerFrequency;
unsigned int   m_subnetMask;
unsigned short m_TX1Temp;
unsigned short m_TX2Temp;
unsigned short m_TX3Temp;
unsigned int   m_BOOTSTSRegister;
unsigned short m_shutdownStatus;
unsigned short m_dieOverTemp;
unsigned short m_vga1aShutdownTemp;
unsigned short m_vga1bShutdownTemp;
unsigned short m_vga2aShutdownTemp;
unsigned short m_vga2bShutdownTemp;
unsigned short m_psu1ShutdownTemp;
unsigned short m_psu2ShutdownTemp;
unsigned short m_TX1ShutdownTemp;
unsigned short m_TX2ShutdownTemp;
unsigned short m_TX3ShutdownTemp;
unsigned short m_linkType;
unsigned short m_VDSLDownstreamSpeed1;
unsigned short m_VDSLDownstreamSpeed2;
unsigned short m_macAddress1;
unsigned short m_macAddress2;
unsigned short m_macAddress3;
unsigned short m_VDSLUpstreamSpeed1;
unsigned short m_VDSLUpstreamSpeed2;
```

The fields within the CGemStatusPacket structure are defined below

Field	Definition
m_head	A CGemHdr structure – see definition of CGemHdr
m_firmwareVer	The version of firmware in the Gemini sonar head
m_sonarId	The sonar ID of the Gemini sonar head
m_sonarFixIp	Fixed IP address of the sonar (10.61.19.200)
m_sonarAltIp	Alternate IP address of the sonar (user programmed)
m_surfaceIp	IP address of surface system currently connected to sonar
m_flags	Not currently implemented
m_vccInt	Sonar FPGA VCC internal
m_vccAux	Sonar FPGA VCC auxilary
m_dcVolt	Sonar incoming DC voltage
m_dieTemp	FPGA Die Temperature
m_unusedTemp	Not used
m_vga1aTemp	VGA Board Temperature Sensor 1A
m_vga1bTemp	VGA Board Temperature Sensor 1B
m_vga2aTemp	VGA Board Temperature Sensor 2A
m_vga2bTemp	VGA Board Temperature Sensor 2B
m_psu1Temp	PSU Temperature 1 - Not used
m_psu2Temp	PSU Temperature 2
m_currentTimestampL	FPGA timestamp value – lower 32 bits
m_currentTimestampH	FPGA timestamp value – upper 32 bits
m_transducerFrequency	0 means the transducer operates at 868kHz 1 means the transducer operates at 723kHz
m_subnetMask	The subnet mask applied to IP addresses which try to connect to the Gemini sonar head
m_TXTemp1	Transmitter Temperature Sensor 1
m_TXTemp2	Transmitter Temperature Sensor 2
m_TXTemp3	Transmitter Temperature Sensor 3
m_BOOTSTSRegister	The value of the BOOTSTS register which contains error codes from the last two boot attempts.
m_shutdownStatus	Bit 0 indicates the Sonar has shutdown due to overtemperature Bit 1 indicates the Sonar has shutdown due to being out of the water
m_dieOverTemp	Maximum acceptable die temperature
m_vga1aShutdownTemp	VGA Board Shutdown Temperature – Sensor 1A
m_vga1bShutdownTemp	VGA Board Shutdown Temperature – Sensor 1B
m_vga2aShutdownTemp	VGA Board Shutdown Temperature – Sensor 2A
m_vga2bShutdownTemp	VGA Board Shutdown Temperature – Sensor 2B
m_psu1ShutdownTemp	PSU Shutdown Temperature – PSU 1
m_psu2ShutdownTemp	PSU Shutdown Temperature – PSU 2
m_TX1ShutdownTemp	Transmitter Shutdown Temperature – Sensor 1
m_TX2ShutdownTemp	Transmitter Shutdown Temperature – Sensor 2
m_TX3ShutdownTemp	Transmitter Shutdown Temperature – Sensor 3

Field	Definition
m_linkType	Bit 0 indicates VDSL is connected Bit 1 indicates Ethernet connection at 10Mbps Bit 2 indicates Ethernet connection at 100Mbps Bit 3 indicates Ethernet connection at 1000Mbps
m_VDSLDownstreamSpeed1	When connected via VDSL: Bits 7 to 0 Downstream 1 constellation Bits 15 to 8 Downstream 1 symbol rate
m_VDSLDownstreamSpeed2	When connected via VDSL: Bits 7 to 0 Downstream 2 constellation Bits 15 to 8 Downstream 2 symbol rate
m_macAddress1	Least significant two bytes of Sonar head MAC address
m_macAddress2	Middle two bytes of Sonar head MAC address
m_macAddress3	Most significant two bytes of Sonar head MAC address
m_VDSLUpstreamSpeed1	When connected via VDSL: Bits 7 to 0 Upstream 1 constellation Bits 15 to 8 Upstream 1 symbol rate
m_VDSLUpstreamSpeed2	When connected via VDSL: Bits 7 to 0 Upstream 2 constellation Bits 15 to 8 Upstream 2 symbol rate

For the CGemStatusPacket structure, the value of m_head.m_type is 0x40.

Due to the layout of the Gemini Status Packet, the C code for the structure aligns it on a two byte boundary rather than the normal four byte boundary. This will also need to be implemented in any code which maps onto the structure returned by the DLL.

The following conversion gives the FPGA die temperature value in °C.

```
FPGA_Die_Temp = ((m_dieTemp * 503.975) / 1024.0) – 273.15
```

The upstream and downstream data rates (measured in bits per second) are given by

```
Current data rate downstream =
(Downstream 1 constellation *
 Downstream 1 symbol rate * 67500) +
(Downstream 2 constellation *
 Downstream 2 symbol rate * 67500)

Current data rate upstream =
(Upstream 1 constellation *
 Upstream 1 symbol rate * 67500) +
(Upstream 2 constellation *
 Upstream 2 symbol rate * 67500)
```

The following description of the m_BOOTSTSRegister field is taken from the Gemini Sonar Hardware Interface specification.

BOOTSTS register read from ICAP module. This contains the error codes for the last two boot attempts with 0 being the most recent.

Bit
0 – Boot attempt 0_Valid
1 – Boot attempt 0_Fallback
2 – Boot attempt 0_IPROG
3 – Boot attempt 0_WTO_ERROR
4 – Boot attempt 0_ID_ERROR
5 – Boot attempt 0_CRD Error
6 – Boot attempt 0_BPI Address wraparound error
7 – Boot attempt 0_RBCRC error caused reconfiguration

8 – Boot attempt 1_Valid
9 – Boot attempt 1_Fallback
10 – Boot attempt 1_IPROG
11 – Boot attempt 1_WTO_ERROR
12 – Boot attempt 1_ID_ERROR
13 – Boot attempt 1_CRD Error
14 – Boot attempt 1_BPI Address wraparound error
15 – Boot attempt 1_RBCRC error caused reconfiguration

For the PSU temperature, the four VGA temperatures, the three transmitter temperatures and the matching shutdown temperatures, the value returned is made up as follows

bit 15	If set, the temperature sensor is present
bit 10	If set, the temperature is negative
bits 9 to 0	The temperature value

The following code converts a temperature value to a string, taking the above into account (the function StringCbPrintf is like the normal C printf function, apart from it produces wide format output strings)

```
void TempStr(unsigned short temperature, TCHAR *outStr)
{
  signed short tempTemp;
  double       finalTemp;

  // If sixteenth bit is set, sensor is present
  bool present = ((temperature & 0x8000) == 0x8000);

  if (present)
  {
    // If tenth bit is set, value is negative
    bool negFlag = ((temperature & 0x0200) == 0x0200);

    // Temperature is contained in 10 bits of value
    tempTemp = temperature & 0x3ff;

    // Sign extend value so that it is a C style signed number
    if (negFlag)
    {
      tempTemp |= 0xfc00;
    }

    // Convert to floating point
    finalTemp = (double)tempTemp / 4.0;

    // And convert to a wide string
    StringCbPrintf(outStr, 20, _T("%1.2lf"), finalTemp);
  }
  else
  {
    // Indicate sensor not present
    StringCbPrintf(outStr, 20, _T(" N/P "));
  }
}

void DisplayTemperatureFunction(unsigned short vga1aTemp)
{
  TCHAR tStr[20];

  TempStr(vga1aTemp, tStr);

  // Do something to display the string returned by TempStr here
}
```

CGemPingHead

Members

```
CGemHdr        m_head;
unsigned short m_pingID;
unsigned short m_extMode;
unsigned int   m_transmitTimestampL;
unsigned int   m_transmitTimestampH;
unsigned short m_startRange;
unsigned short m_endRange;
unsigned int   m_lineTime;
unsigned short m_numBeams;
unsigned short m_numChans;
unsigned char  m_sampChan;
unsigned char  m_baseGain;
unsigned short m_spdSndVel;
unsigned short m_velEchoTime;
unsigned short m_velEntries;
unsigned short m_txAngle;
unsigned short m_sosUsed;
unsigned char  m_RLEThresholdUsed;
unsigned char  m_rangeCompressionUsed;
```

The fields within the CGemPingHead structure are defined below

Field	Definition
m_head	A CGemHdr structure – see definition of CGemHdr
m_pingID	A 16 bit ping id which ties together the ping header, data and tail. Wraps round.
m_extMode	The value of the extMode field sent in the ping configuration.
m_transmitTimestampL	Transmit time in microseconds – lower 32 bits
m_transmitTimestampH	Transmit time in microseconds – upper 32 bits
m_startRange	Start range (in lines) of the data being returned
m_endRange	End range (in lines) of the data being returned
m_lineTime	Not currently implemented
m_numBeams	Number of beams formed in the data being returned
m_numChans	Number of channels in the data being returned
m_sampChan	Value of sample channel sent in the ping configuration
m_baseGain	Value of base gain sent in the ping configuration
m_spdSndVel	Speed of sound being measured by the velocimeter in units of 0.1m/s
m_velEchoTime	Time for velocimeter echo – used for calibration only
m_velEntries	Number of velocimeter readings used in speed of sound calculation
m_txAngle	Not currently implemented

Field	Definition
m_sosUsed	Speed of sound used in the beamforming. Bit 15 is used to indicate the source of the speed of sound Bit 15 = 0, the speed of sound came from the velocimeter Bit 15 = 1, the speed of sound came from the ping config Bits 14 to 0 carry the speed of sound in units of 0.1m/s
m_RLEThresholdUsed	Threshold value applied to Run Length Encoding – zero implies no RLE applied. The DLL will remove any RLE applied before the data is passed to the parent application.
m_rangeCompression-Used	Value indicating what range compression, if any, was applied to the data. The DLL may remove any range compression applied before the data is passed to the parent application. This is dependant on the mode the DLL is set to.

For the CGemPingHead structure, the value of m_head.m_type is 0x41.

CGemPingLine

Members

```
CGemHdr        m_head;
unsigned char  m_gain;
unsigned char  m_pingID;
unsigned short m_lineID;
unsigned short m_scale;
unsigned short m_lineInfo;
unsigned char  m_startOfData;
```

The fields within the CGemPingLine structure are defined below

Field	Definition
m_head	A CGemHdr structure – see definition of CGemHdr
m_gain	Not currently implemented
m_pingID	Lower 8 bits of the ping id which ties together the ping header, data and tail. Wraps round.
m_lineID	The id of the line within the ping data (value ranges from 0 to (CGemPingHead::m_endRange – CGemPingHead::m_startRange) – 1. Bit 15 is used to indicate that this is resent data (part of the retry mechanism implemented by the DLL).
m_scale	Value of software gain applied to this line
m_lineInfo	Bits 15..7 Number of DWORDs in data Bit 6 Last Line Flag (1 indicates last line) Bits 5..0 Ext_mode from ping config bits 5..0
m_startOfData	Field which gives start address of data within the structure.

For the CGemPingLine structure, the value of m_head.m_type is 0x42. m_startOfData is the first byte of the data being returned to the parent process, and therefore is the first byte of an array of CGemPingHead::m_numBeams of data.

The DLL implements a retry mechanism to re-request data lost due to dropped packets. Bit 15 (MSB) of the line ID is used to indicate if a packet is carrying data as a result of a retry request. The DLL will strip this bit out before the data is passed to the calling program, and so will always be seen as zero at the calling program.

CGemPingTail

Members

```
CGemHdr         m_head;
unsigned char   m_pingID;
unsigned char   m_flags;
unsigned short  m_spare;
```

The fields within the CGemPingTail structure are defined below

Field	Definition
m_head	A CGemHdr structure – see definition of CGemHdr
m_pingID	Lower 8 bits of the ping id which ties together the ping header, data and tail. Wraps round.
m_flags	Bit 7 is used to indicate that this is resent data (part of the retry mechanism implemented by the DLL).
m_spare	Not used.

For the CGemPingTail structure, the value of m_head.m_type is 0x43.

The DLL implements a retry mechanism to re-request data lost due to dropped packets. Bit 7 (MSB) of m_flags is used to indicate if a packet is carrying data as a result of a retry request. The DLL will strip this bit out before the data is passed to the calling program, and so will always be seen as zero at the calling program.

CGemPingTailExtended

Members

```
CGemHdr         m_head;
unsigned char   m_pingID;
unsigned char   m_flags;
unsigned short  m_spare;
unsigned short  m_firstPassRetries;
unsigned short  m_secondPassRetries;
unsigned short  m_tailRetries;
unsigned short  m_interMessageGap;
unsigned long   m_packetCount;
unsigned long   m_recvErrorCount;
unsigned long   m_linesLostThisPing;
unsigned long   m_generalCount;
```

The CGemPingTailExended structure is generated within the DLL when a ping tail message is received from the Gemini Sonar. As well as the fields received from the Sonar (which would be returned in a CGemPingTail structure) this structure also contains extra data generated by the DLL during the receipt of the ping data. This data typically contains the values of counters which help to evaluate the quality of the link to the Sonar, and is described in more detail in the table below.

The fields within the CGemPingTailExtended structure are defined below

Field	Definition
m_head	A CGemHdr structure – see definition of CGemHdr
m_pingID	Lower 8 bits of the ping id which ties together the ping header, data and tail. Wraps round.
m_flags	Bit 7 is used to indicate that this is resent data (part of the retry mechanism implemented by the DLL).
m_spare	Not used.
m_firstPassRetries	The number of first retries of data lines attempted
m_secondPassRetries	The number of second (or more) retries of data lines attempted
m_tailRetries	The number of retries of the ping tail attempted
m_interMessageGap	The value of the intermessage gap being set by the DLL when setting up communications with the sonar.
m_packetCount	The number of packets received since the DLL started receiving data. Excludes status packets.
m_recvErrorCount	The number of times the “recv” function in the DLL has returned an error.
m_linesLostThisPing	The number of lines of data which were expected but have not been received in the ping which has just occurred.
m_generalCount	A general count used to convey information from the DLL to the calling program. No specific use in this version of the DLL.

For the CGemPingTailExtended structure, the value of m_head.m_type is 0x61.

The DLL implements a retry mechanism to re-request data lost due to dropped packets. Bit 7 (MSB) of m_flags is used to indicate if a packet is carrying data as a result of a retry request. The DLL will strip this bit out before the data is passed to the calling program, and so will always be seen as zero at the calling program.

CGemFlashResult

Members

```
CGemHdr        m_head;
unsigned int   m_address;
unsigned short m_result;
unsigned short m_spare;
```

The fields within the CGemFlashResult structure are defined below

Field	Definition
m_head	A CGemHdr structure – see definition of CGemHdr
m_address	The address in flash which this result refers to
m_result	The result code for the flash operation 0 means no errors occurred
m_spare	Unused

For the CGemFlashResult structure, the value of m_head.m_type is 0x44.

The following description of the m_result field is taken from the Gemini Sonar Hardware Interface specification.

Bits 2..0 Abort Flags
001 – Erase Timeout
010 – Unable to unlock block
011 – Write Buffer Unavailable
100 – Block Program Timeout

Bit 7 – Completion Lock Result
0 – Flash locked OK on completion
1 – Flash not locked on completion

Bit 8 Flash Status Register bit0
0 – BEFP complete
1 – BEFP in progress

Bit 9 Flash Status Register bit 1
0 – Block not locked during program/erase
1 – Block locked during program or erase; operation aborted

Bit 10 Flash Status Register bit 2
0 – Program suspend not in effect
1 – Program suspend in effect

Bit 11 Flash Status Register bit 3
0 – Program / Erase Voltage OK
1 – Program / Erase Voltage Error

Bit 12 Flash Status Register bit 4
0 – Program OK
1 – Program Error

Bit 13 Flash Status Register bit 5
0 – Erase OK
1 – Erase Error

Bit 14 Flash Status Register bit 6
0 – Erase suspend not in effect
1 – Erase suspend in effect

Bit 15 Flash Status Register bit 7
0 – Device is busy; program or erase cycle in progress; SR[0] valid.
1 - Device is ready; SR[6:1] are valid.

CGemFlashReadback

Members

```
CGemHdr         m_head;
unsigned int    m_address;
unsigned int    m_spare;
unsigned char   m_data[1024];
```

The fields within the CGemFlashReadback structure are defined below

Field	Definition
m_head	A CGemHdr structure – see definition of CGemHdr
m_address	Flash address
m_spare	Not used
m_data	1kb (1024 bytes) of flash data, starting at m_address

For the CGemFlashReadback structure, the value of m_head.m_type is 0x45.

CGemVelocimeterData

Members

```
CGemHdr         m_head;
unsigned int    m_blockCount;
unsigned int    m_spare;
unsigned char   m_data[1024];
```

The fields within the CGemVelocimeterData structure are defined below

Field	Definition
m_head	A CGemHdr structure – see definition of CGemHdr
m_blockCount	The block count of this block of data
m_spare	Not used
m_data	1kb (1024 bytes) block of velocimeter data

For the CGemVelocimeterData structure, the value of m_head.m_type is 0x47.

CGemRS232RXPacket

Members

```
CGemHdr         m_head;
unsigned char   m_startOfData;
```

The fields within the CGemRS232RXPacket structure are defined below

Field	Definition
m_head	A CGemHdr structure – see definition of CGemHdr
m_startOfData	First byte of data

For the CGemRS232RXPacket structure, the value of m_head.m_type is 0x48. m_data1 is the first byte of the data being returned to the parent process, and therefore is the first byte of an array of data.

The data for a CGemRS232RXPacket is made up of a series of 8 byte blocks. The number of blocks in the packet needs to be determined from the size of the structure passed to the calling program by the DLL. Each of the eight byte blocks has the following structure, which conveys a single byte of data received on the Sonar's RS232 port and the timestamp when it was received.

Byte No	Definition
0 (Least Significant)	Timestamp when data received – Least significant byte
1	Timestamp when data received
2	Timestamp when data received
3	Timestamp when data received
4	Timestamp when data received
5	Timestamp when data received
6	Timestamp when data received – Most significant byte
7 (Most significant)	Data received

CGemAcknowledge

Members

```
CGemHdr         m_head;
unsigned int    m_receiptTimestampL;
unsigned int    m_receiptTimestampH;
unsigned int    m_replyTimestampL;
unsigned int    m_replyTimestampH;
```

The fields within the CGemAcknowledge structure are defined below

Field	Definition
m_head	A CGemHdr structure – see definition of CGemHdr
m_receiptTimestampL	The time of receipt of the stay alive packet by the Sonar – Lower 32 bits
m_receiptTimestampH	The time of receipt of the stay alive packet by the Sonar – Upper 32 bits
m_replyTimestampL	Time of transmission – Lower 32 bits
m_replyTimestampH	Time of transmission – Upper 32 bits

For the CGemAcknowledge structure, the value of m_head.m_type is 0x49.

CGemTestResult

Members

```
CGemHdr             m_head;
unsigned short      m_packing1;
unsigned short      m_packing2;
unsigned short      m_packing3;
unsigned short      m_packing4;
unsigned int        m_SRAM0Result;
unsigned int        m_SRAM1Result;
unsigned short      m_flashPostResult;
unsigned int        m_flashProdResultL;
unsigned int        m_flashProdResultH;
unsigned int        m_firstErrorAddress;
unsigned long long  m_seedValueWritten;
unsigned long long  m_seedValueUsed;
```

The fields within the CGemTestResult structure are defined below

Field	Definition
m_head	A CGemHdr structure – see definition of CGemHdr
m_packing1	Unused – Always set to 0xA000
m_packing2	Unused – Always set to 0xA001
m_packing3	Unused – Always set to 0xA002
m_packing4	Unused – Always set to 0xA003
m_SRAM0Result	Result of testing SRAM0 on the FPGA board
m_SRAM1Result	Result of testing SRAM1 on the FPGA board
m_flashPostResult	Result of running the Flash POST
m_flashProdResultL	The lower 32 bits of the Flash production test result
m_flashProdResultH	The upper 32 bits of the Flash production test result
m_firstErrorAddress	Address of First Error, 0xffffffff if no errors
m_seedValueWritten	Bits 0 to 62 Most recent PRBS register value Bit 63 – 1=use this prbs value, 0=use sonar's prbs value
m_seedValueUsed	Bits 0 to 62 PRBS register Value used for the last production test Bit 63 – 1=use this prbs value, 0=use sonar's prbs value

For the CGemAcknowledge structure, the value of m_head.m_type is 0x4A.

Due to the layout of the Gemini Test Result Packet, the C code for the structure aligns it on a two byte boundary rather than the normal four byte boundary. This will also need to be implemented in any code which maps onto the structure returned by the DLL.

This structure is intended to be used by the Gemini Production Test, and the description of the packet is provided for information only. The following descriptions are taken from the Gemini Sonar Hardware Interface specification.

m_SRAM0Result and m_SRAM1Result

```
Bits (20 downto 0) – error count
Bits (23 downto 21) – unused
Bits (27 downto 24) – post run count
Bit 28 – test_complete
Bit 29 – prodtest_npost_flag, '1'=production test, '0'=post test
```

m_flashPostResult

```
Both Flash chips are tested together.
Bits (5 downto 0) – error count
Bits (7 downto 6) - unused
Bits (11 downto 8) – post run count
Bit 12 – test_complete
Bit 13 – prodtest_npost_flag, '1'=production test, '0'=post test
```

m_flashProdResult

```
Bits (27 downto 0) – error count
Bits (31 downto 28) - unused
Bits (35 downto 32) – post run count
Bit 36 – test_complete
Bit 37 – prodtest_npost_flag, '1'=production test, '0'=post test
```

The PBRS register is a 63 bit Pseudo Random Binary Sequence which is used during the production test. It can either be written to the Sonar head (bit 63 = 1) or the Sonar head be requested to generate its own value to use (bit 62 = 0). If the calling program writes a value to the head it can be retrieved and checked by requesting a set of test results. The m_seedValueWritten will have the PBRS value which was written to the head. The value of PBRS which is actually used is returned in every set of test results in m_seedValueUsed.

CGemProdData

Members

```
CGemHdr         m_head;
unsigned short m_boardIdentData1;
unsigned short m_boardIdentData2;
unsigned short m_boardIdentData3;
unsigned short m_boardIdentData4;
unsigned short m_boardIdentData5;
unsigned short m_boardIdentData6;
```

The fields within the CGemProdData structure are defined below

Field	Definition
m_head	A CGemHdr structure – see definition of CGemHdr
m_boardIdentData1	TBC
m_boardIdentData2	TBC
m_boardIdentData3	TBC
m_boardIdentData4	TBC
m_boardIdentData5	TBC
m_boardIdentData6	TBC

For the CGemProdData structure, the value of m_head.m_type is 0x4B.

CGemBearingData

Members

```
CGemHdr         m_head;
unsigned short m_bearingLineNo;
unsigned short m_noSamples;
unsigned char  *m_pData;
```

The fields within the CGemBearingData structure are defined below

Field	Definition
m_head	A CGemHdr structure – see definition of CGemHdr
m_bearingLineNo	The bearing line number for this block of data (between 0 and 255)
m_noSamples	The number of samples in this block of data.
m_pData	Address of the data – Block of separately allocated memory of size m_noSamples, which will be freed by the DLL.

For the CGemBearingData structure, the value of m_head.m_type is 0x60. m_data1 is a pointer to the first byte of the data being returned to the parent process, and therefore is a pointer to the first byte of an array of m_noSamples of data.

Data is returned to the calling program via Gemini Bearing Data packets when the software is put into SeaNet or SeaNetC mode.

See the note about memory allocation and the CGemBearingStructure in the next section in the description of the main callback function.

Callback Functions

The DLL requires a callback function to be defined for it, so that it can inform the calling process when data has been received from the sonar head. A second callback function can optionally be defined for the code which programs the modifier tables or the FPGA images into the head (only done during production or repair).

Main Callback Function

The main callback function has to have the following definition

```
void CallBackFn(int eType, int len, char *dataBlock)
```

eType is the type of data being passed from the DLL to the calling program. It is one of the following (previously defined) values

PING_HEAD	PING_DATA
PING_TAIL	PING_TAIL_EX
GEM_STATUS	GEM_ACKNOWLEDGE
GEM_SERIAL	GEM_FLASH_RESULT
GEM_BEARING_DATA	GEM_FLASH_READBACK
GEM_TEST_RESULT	GEM_IP_CHANGED
GEM_UNKNOWN_DATA	GEM_VELOCIMETER_DARA
GEM_SERIAL_PORT_DATA	GEM_PROD_DATA

len is the length of the data block being passed to the calling program, For any blocks defined as fixed length in the table below, the value of len returned by the DLL is -1. For the variable length blocks, length is defined as follows.

Structure type	Value of len
CGemPingHead	-1
CGemPingLine	Number of beams in the record
CGemPingTail	-1
CGemPingTailExtended	-1
CGemStatusPacket	-1
CGemAcknowledge	-1
CGemRS232RXPacket	Size of packet, including header
CGemFlashResult	-1
CGemBearingData	Number of lines in the record
CGemFlashReadback	-1
CGemTestResult	-1
CGemVelocimeterData	-1
CGemProdData	-1

dataBlock is the actual data being passed to the calling program, and is a pointer to one of the structures defined previously.

eType	dataBlock is a	Fixed Length?
PING_HEAD	CGemPingHead	✓
PING_DATA	CGemPingLine	×
PING_TAIL	CGemPingTail	✓
PING_TAIL_EX	CGemPingTailExtended	✓
GEM_STATUS	CGemStatusPacket	✓
GEM_ACKNOWLEDGE	CGemAcknowledge	✓
GEM_SERIAL	CGemRS232RXPacket	×
GEM_FLASH_RESULT	CGemFlashResult	✓
GEM_BEARING_DATA	CGemBearingData	×
GEM_FLASH_READBACK	CGemFlashReadback	✓
GEM_TEST_RESULT	CGemTestResult	✓
GEM_UNKNOWN_DATA	*Unknown*	×
GEM_IP_CHANGED	*Empty*	✓
GEM_VELOCIMETER_DATA	CGemVelocimeterData	✓
GEM_SERIAL_PORT_INPUT	char*	×
GEM_PROD_DATA	CGemProdData	✓

The callback function is set up by calling the DLL function GEM_SetHandlerFunction as follows

```
GEM_SetHandlerFunction(CallBackFn);
```

The design of the DLL requires that the callback function copies the data from dataBlock into local storage under its own control before returning, as the buffer holding the data in the DLL may be overwritten by other data coming from the sonar head if the block is used by the parent process after the callback function has returned.

For all structures where the length of the structure is not known at design time, the len field contains the length information, allowing the parent code to allocate the correct amount of memory to copy all the data.

For a GEM_BEARING_DATA message, the actual data contained in the message is passed in a block of memory allocated by the DLL. For each bearing line in a ping, the same block of memory is used for the data, and this memory is freed by the DLL after the last line of data has been sent. The calling program will need to copy the data out of this block of memory into memory allocated by the calling process (instead of just copying and retaining the value of the pointer).

For a GEM_IP_CHANGED message, there is no associated data. The DLL has a task which is notified if the IP table of the PC changes. The expectation is if this message is received, the calling program will shutdown and restart the communications with the Gemini sonar, thus allowing the change in the IP address of the computer to be correctly actioned.

The following snippet of code shows the minimum code expected to be executed when this message is received.

```
GEM_StopGeminiNetwork();
GEM_StartGeminiNetwork(0);
GEM_SetGeminiSoftwareMode("RequiredMode");
GEM_SetHandlerFunction(&GemDatahandler);
```

For a GEM_UNKNOWN_DATA message, there is a message block attached whose total length is returned in the len field. The Gemini DLL is not aware of what the packet is, so has passed the packet untouched to calling program.

For a GEM_SERIAL_PORT_INPUT message, the DLL has been requested to open the serial port and return the data to the calling program rather than send it down to the Gemini sonar. Every 50mS (approximately) the serial port is checked to see if any data has been received, if so the data is sent to the calling program as a C `char` array using a GEM_SERIAL_PORT_INPUT message.

Modifier Programming Callback Function

The modifier programming callback function has to have the following definition

```
void ModifierCallback(int sType, int retryCount, int
packet, int block, int totalBlocks, int otherValue)
```

eType is the type of data being passed from the DLL to the calling program. It is one of the following (previously defined) values

If programming modifier tables into the head

GEM_MODPROG_START	GEM_MODPROG_SUCCESS
GEM_MODPROG_FAILURE	GEM_MODPROG_COMPLETE
GEM_MODPROG_NOT_ALIGNED	GEM_MODPROG_BAD_COMPARE
GEM_MODPROG_TIMEOUT	GEM_MODPROG_CALCULATING
GEM_MODPROG_ABORTED	GEM_MODPROG_LOW_ADDRESS
GEM_MODPROG_FAILED	GEM_MODPROG_CALCFAIL

If programming FPGA images into the head

GEM_FPGAPROG_START	GEM_FPGAPROG_SUCCESS
GEM_FPGAPROG_FAILURE	GEM_FPGAPROG_COMPLETE
GEM_FPGAPROG_NOT_ALIGNED	GEM_FPGAPROG_BAD_COMPARE
GEM_FPGAPROG_TIMEOUT	GEM_FPGAPROG_ABORTED
GEM_FPGAPROG_LOW_ADDRESS	GEM_FPGAPROG_NO_FILE
GEM_FPGAPROG_NO_ADDR	GEM_FPGAPROG_FAILED
GEM_FPGAPROG_FILE_SIZE	

retryCount is the number of times programming the current packet of data has been attempted.

packet is the number of the 1k chunk of data from the current block which is being programmed.

block is the number of the current block being programmed.

totalBlocks is the total number of blocks to be programmed.

otherValue is a message specific value.

The documentation of this function is deliberately vague as programming the modifiers in a Gemini Sonar head will only be done by the SRD Production Tool at either manufacture of the head, or when it has been returned to SRD for repair.

Functions

The following functions are defined as the interface of the DLL. The C definitions for these are contained in the file GeminiComms.h. The DLL itself is known as GeminiComms.dll.

For each of the functions exposed by the DLL, the following paragraphs give the name of the function, the C definition of the function, a description of the function, its parameters and return value (if any), an example of calling the function in C, and a summary of the default values and limits for the parameters (where applicable).

An example of using the DLL is attached as an appendix.

GEM_StartGeminiNetwork

```
void GEM_StartGeminiNetwork(unsigned short sonarID);
```

This function has to be the first function called when talking to the GeminiComms DLL. Apart from initialising all the communications with the head and getting the DLL into a state ready to operate, it can also set the ID which will be used in all communications with the sonar. Each Gemini sonar has a unique ID and will only accept commands which have a matching ID to its own.

sonarID is the unique ID of the Gemini sonar to be used in all transmissions. Can be zero, in which case no transmission will take place until GEM_SetSonarID has been called to set a non-zero sonarID

```
GEM_StartGeminiNetwork(513);
```

GEM_StartGeminiNetworkWithResult

```
int GEM_StartGeminiNetworkWithResult(unsigned short
sonarID);
```

This function is an expanded version of the GEM_StartGeminiNetwork function. It is recommended that this version is used, though the original has been retained for backwards compatibility with code that has already been written. Either this function or GEM_StartGeminiNetwork has to be the first function called when talking to the GeminiComms DLL. Apart from initialising all the communications with the head and getting the DLL into a state ready to operate, it can also set the ID which will be used in all communications with the sonar. Each Gemini sonar has a unique ID and will only accept commands which have a matching ID to its own.

sonarID is the unique ID of the Gemini sonar to be used in all transmissions. Can be zero, in which case no transmission will take place until GEM_SetSonarID has been called to set a non-zero sonarID.

The return value is defined in the following table

Return Value	Meaning
0	A failure occurred initialising the DLL and communications with the Sonar is not possible. The most likely cause is that another piece of software which communicates with the Sonar is already running, and so the port could not be opened.
1	The DLL initialised correctly and is ready to communicate with the Sonar.

```
int success = 0;
success = GEM_StartGeminiNetwork(513);

if (success == 0)
{
    // Tell the user that something went wrong
    // Either shutdown the program or
    // enter an offline mode
}
else
{
    // Hooray! We can talk to the Sonar
}
```

GEM_SetGeminiSoftwareMode

```
void GEM_SetGeminiSoftwareMode(char *softwareMode);
```

This function sets the operating mode of the software. Setting a value other than those listed in the table below will result in the software mode being set to the default mode. The software has a production programming mode which is only available to SRD.

softwareMode	Meaning
"Evo"	DLL sends the data from the Gemini sonar head to the calling program unprocessed as CGemPingHead, CGemPingLine, and CGemPingTailExtended messages using the callback function.
"EvoC"	DLL sends the data from the Gemini sonar head to the calling program unprocessed as CGemPingHead, CGemPingLine, and CGemPingTailExtended messages using the callback function. The DLL makes use of the range compression feature of the sonar head firmware to ensure that the head does not return more range lines than is appropriate for the size and quality of the display being used by the calling program.
"SeaNet"	DLL processes the data and sends it to the calling program as CGemPingHead and CGemBearingData messages using the callback function.
"SeaNetC"	DLL processes the data and sends it to the calling program as CGemPingHead and CGemBearingData messages using the callback function. The DLL makes use of the range compression feature of the sonar head firmware to ensure that the head does not return 1500 or more range lines to the Seanet software.

```
GEM_SetGeminiSoftwareMode("SeaNet");
```

Default value: "Evo"

GEM_SetHandlerFunction

```
void GEM_SetHandlerFunction(void (cdecl *FnPtr)(int
eType, int len, char *dataBlock));
```

This function allows the parent code to specify the callback function which will be used by the DLL to return data from the sonar head to the parent code.

FnPtr is the address of the callback function in the parent code which the DLL will use.

```
void CallBackFn(int eType, int len, char *dataBlock)
{
  CGemStatusPacket *pStat;
  CGemStatusPacket *copyStatus;
  CGemBearingData  *pBearing;
  CGemBearingData  *copyBearing;

  switch (eType)
  {
    case PING_HEAD :
    case PING_DATA :
    case PING_TAIL :
      ...
    case GEM_STATUS :
      // Copy the data, so that we do something with it

      pStat = (CGemStatusPacket *)dataBlock;

      copyStatus = new CGemStatusPacket;

      *copyStatus = *pStat;

      // Do something with the message, remember to delete copyStatus

      break;

    case GEM_ACKNOWLEDGE :
      ...
    case GEM_BEARING_DATA :
      // Copy the data, so that we can post a message

      pBearing = (CGemBearingData *)dataBlock;

      copyBearing = new CGemBearingData;

      *copyBearing = *pBearing;

      // The bearing data structure contains a pointer to a block of memory which is
      // allocated in the DLL and which is filled with the data we want to display.
      // Therefore we cannot just take a copy of the structure, as this contains the
      // pointer to the memory allocated by the DLL.

      // Attempt to allocate the memory for the copied data
      copyBearing->m_pData = (unsigned char *)malloc(copyBearing->m_noSamples);

      // Did we allocate the memory?
      if (copyBearing->m_pData)
      {
        // Copy the data from the original structure to the new structure
        memcpy(copyBearing->m_pData, pBearing->m_pData, copyBearing->m_noSamples);
      }
      else
      {
        // Failed to allocate memory? Indicate by setting number of samples to zero
        copyBearing->m_noSamples = 0;
      }

      // Do something with the message, remember to free copyBearing->m_pData and copyBearing

      break;
  }
}

GEM_SetHandlerFunction(CallBackFn);
```

Default value: null

GEM_SetProgressFunction

```
void GEM_SetProgressFunction(void (cdecl *FnPtr)(int
sType, int retryCount, int packet, int block, int
totalBlocks, int otherValue));
```

This function allows the parent code to specify the callback function which will be used by the DLL to return information about the progress of programming the modifier tables, the firmware, or other flash based data to the parent code.

FnPtr is the address of the callback function in the parent code which the DLL will use.

```
void ProgressFn(int sType, int retryCount, int packet,
int block, int totalBlocks, int otherValue)
{
    switch (sType)
    {
        case GEM_MODPROG_START:
            ...
            break;
        ...
    }
}

GEM_SetProgressFunction(ProgressFn);
```

Default value: null

GEM_ResetInternalCounters

```
void GEM_ResetInternalCounters(void);
```

This function resets some internal counters held by the GeminiComms DLL, which are returned in the CGemPingTailExtended structure (a PING_TAIL_EX message).

The DLL keeps a number of counters, such as the packet received and retry counts that only it is aware of. These counters are from the time the DLL started running. There are a number of reasons why these counters may need to be reset, such as changing the sonar you are talking to in a multi-sonar environment. This function allows the calling program to reset the counters when it needs to.

```
GEM_ResetInternalCounters();
```

GEM_StopGeminiNetwork

```
void GEM_StopGeminiNetwork(void);
```

This function stops the functionality of the GeminiComms DLL, which among other things will stop the tasks running which are handling the network and serial data, abort any modifier or FPGA image programming, and free any allocated memory. This function will sleep for a short period to allow the tasks running in the DLL to finish in an orderly fashion.

```
GEM_StopGeminiNetwork();
```

GEM_GetSonarID

```
unsigned short GEM_GetSonarID(void);
```

This function gets the current sonar ID which is used in all communications with the sonar. Each Gemini sonar has a unique ID and will only accept commands which have a matching ID to its own.

The return value is the unique ID of the Gemini sonar which is used in all transmissions. If this value is zero or one, no transmission will take place until GEM_SetSonarID has been called to set a sonarID greater than one.

```
unsigned short sonarID = GEM_GetSonarID();
```

GEM_SetDLLSonarID

```
void GEM_SetDLLSonarID(unsigned short sonarID);
```

This function sets the ID which the DLL will use in all communications with the sonar. Each Gemini sonar has a unique ID and will only accept commands which have a matching ID to its own.

sonarID is the unique ID of the Gemini sonar to be used in all transmissions. Can be zero, in which case no transmission will take place until GEM_SetDLLSonarID has been called to set a non-zero sonarID. The value of one is reserved for use during Gemini production and should not be used, as again no transmission will take place.

```
GEM_SetDLLSonarID(513);
```

Default value: 0

GEM_ProgramSonarID

```
void GEM_ProgramSonarID(unsigned short sonarID, int
useHighFreq);
```

This function programs the Gemini Head with the ID which it will respond to in communications with the DLL. Each Gemini sonar has a unique ID and will only accept commands which have a matching ID to its own.

sonarID is the unique ID of the Gemini sonar to be used in all transmissions. Can be zero, in which case no transmission will take place until GEM_SetDLLSonarID has been called to set a non-zero sonarID. The value of one is reserved for use during Gemini production and should not be used, as again no transmission will take place. Once programmed, the DLL will use this sonarID in all transmissions.

useHighFreq is a value which configures the Sonar for operation at either 723 kHz or 868 kHz.

useHighFreq	Meaning
0	The Sonar operates at 868kHz
1	The Sonar operates at 723kHz

```
GEM_ProgramSonarID(513, 0);
```

Default value: sonarID 0
useHighFreq 0

This function is only available during Gemini production. Programming the sonarID also requires a programming enable switch in the Gemini hardware to be in the appropriate position.

GEM_ProgramVelocimeterCoeffs

```
void GEM_ProgramVelocimeterCoeffs(unsigned int CCoeff,
unsigned int MCoeff);
```

This function programs the Gemini Head with the coefficients needed to calibrate the velocimeter on the front face of the sonar. The C and M coefficients are normally calculated by code in the Gemini Production Tool, based on values read from the sonar and the actual velocity of sound measured.

```
GEM_ProgramVelocimeterCoeffs(40328, 737663106);
```

GEM_GetAltSonarIPAddress

```
void GEM_GetAltSonarIPAddress(unsigned char *a1, unsigned
char *a2, unsigned char *a3, unsigned char *a4, unsigned
char *s1, unsigned char *s2, unsigned char *s3, unsigned
char *s4);
```

This function gets the value the DLL holds for the alternative IP address and the subnet mask that the Gemini sonar can have. A Gemini sonar head will always respond to the IP address 10.61.19.200, but to allow easy integration onto other networks an alternative IP address can be specified which the Gemini head will also respond to. A subnet mask can also be specified to limit the interaction between multiple units on the same physical network. This function returns the alternative IP address which the DLL may use to communicate with the head.

```
unsigned char a1, a2, a3, a4; // IP Address, A1 is MSB
unsigned char s1, s2, s3, s4; // Subnet mask, S1 is MSB

GEM_GetAltSonarIPAddress(&a1, &a2, &a3, &a4,
                         &s1, &s2, &s3, &s4);
```

GEM_SetAltSonarIPAddress

```
void GEM_SetAltSonarIPAddress(unsigned char a1, unsigned
char a2, unsigned char a3, unsigned char a4 unsigned char
*s1, unsigned char *s2, unsigned char *s3, unsigned char
*s4);
```

This function sets the alternative IP address and the subnet mask that the Gemini sonar can have. A Gemini sonar head will always respond to the IP address 10.61.19.200, but to allow easy integration onto other networks an alternative IP address can be specified which the Gemini head will also respond to. This function programs the alternative IP address into the head.

The alternate address used by the DLL is not changed as the Sonar head will not use the newly programmed address until it is rebooted. After issuing this command, the calling program should either wait until a GEM_FLASH_RESULT message is received indicating that the command has finished, or for a fixed (worst case) period of 5 seconds before issuing a command to reboot the sonar. Rebooting the sonar before this command has completed will result in corrupted flash in the Sonar head, and an unusable Sonar.

```
// Set address 192.169.1.10, subnet mask 255.255.255.0
GEM_SetAltSonarIPAddress(192, 168, 1, 10,
                         255, 255, 255, 0);
```

Default value: 0, 0, 0, 0, 0, 0, 0, 0

GEM_UseAltSonarIPAddress

```
void GEM_UseAltSonarIPAddress(unsigned char a1, unsigned
char a2, unsigned char a3, unsigned char a4 unsigned char
*s1, unsigned char *s2, unsigned char *s3, unsigned char
*s4);
```

This function informs the DLL of the alternative IP address and subnet mask that the Gemini sonar can have. A Gemini sonar head will always respond to the IP address 10.61.19.200, but to allow easy integration onto other networks an alternative IP address can be specified which the Gemini head will also respond to. This function informs the DLL of the alternative address which can be used to talk to the head.

```
// Use address 192.168.1.10, subnet mask 255.255.255.0
GEM_UseAltSonarIPAddress(192, 168, 1, 10,
                         255, 255, 255, 0);
```

GEM_TxToAltIPAddress

```
void GEM_TxToAltIPAddress(int useAltIPAddress);
```

This function tells the DLL to use either the main (default) or the alternate IP address when talking to the Gemini sonar.

useAltIPAddress	Meaning
0	Use main IP address to communicate with head (10.61.19.200)
1	Use alternative IP address to communicate with head

```
GEM_TxToAltIPAddress(0);
```

GEM_SetGeminiModFrequency

```
void GEM_SetGeminiModFrequency(int modFrequency);
```

This function is used to tell the DLL the modulation frequency of the Gemini Sonar. This value is used in calculating the number of range lines which represent a particular range in m. This calculation is used as part of the GEM_AutoPingConfig functionality. The code has the current modulation frequency of the Gemini Sonar as its default, but if a new design of sonar with a different modulation frequency is produced, this function is available to set the new value for the DLL to use.

```
GEM_SetGeminiModFrequency(90422);
```

Default value: 90422

GEM_AutoPingConfig

```
void GEM_AutoPingConfig(float range, unsigned short gain,
float sos);
```

This function tells the DLL to build and send a ping configuration packet to the Gemini sonar head using the range, gain, and speed of sound specified in the function call, and calculating all other values for the ping from the range, the gain, and its internal defaults. The ping will be built so that the sonar head pings once in response.

range is the range (in m) which the sonar is expected to return data in this ping.

gain is the percentage gain which the sonar is to apply in this ping.

sos is the speed of sound (in m/s) to be used when calculating values for this ping. As with all speed of sound values associated with the Gemini sonar, this value will be rounded to the nearest 3.2m/s step when it is used.

```
GEM_AutoPingConfig(11, 50, 1473.6);
```

Default value:	range	0
	gain	0
	sos	1499.2
Minimum:	range	0m
	gain	0%
	sos	1400
Maximum:	range	50m
	gain	100%
	sos	1588

GEM_SetGeminiEvoQuality

```
void GEM_SetGeminiEvoQuality(unsigned char
evoQualitySetting);
```

This function sets the compression factor applied when the DLL is running in 'EvoC' mode. The Gemini sonar can range compress the data, so that less bandwidth is used in sending the data from the sonar head to the surface computer. This function matches the range compression applied to the quality setting of the Evo system. The Evo system quality setting defines how many lines are displayed by the Evo system. There is little point in sending up more data than the display system can cope with, as the data will be thrown away by the display system in producing its images.

The evoQualitySetting can have the following values

evoQualitySetting	Meaning
0	The Evo system is displaying 32 lines of data, and the sonar will compress the data so that less than this number of lines is returned.
1	The Evo system is displaying 64 lines of data, and the sonar will compress the data so that less than this number of lines is returned.
2	The Evo system is displaying 128 lines of data, and the sonar will compress the data so that less than this number of lines is returned.
3	The Evo system is displaying 256 lines of data, and the sonar will compress the data so that less than this number of lines is returned.
4	The Evo system is displaying 512 lines of data, and the sonar will compress the data so that less than this number of lines is returned.
5	The Evo system is displaying 1024 lines of data, and the sonar will compress the data so that less than this number of lines is returned.
6	The Evo system is displaying 2048 lines of data, and the sonar will compress the data so that less than this number of lines is returned.
7	The Evo system is displaying 4096 lines of data, and the sonar will compress the data so that less than this number of lines is returned.

```
GEM_SetGeminiEvoQuality(5);
```

GEM_GetRequestedCompressionFactor

```
unsigned short GEM_GetRequestedCompressionFactor(void);
```

This function returns the compression factor applied when a ping was requested in 'EvoC' mode. When running in 'EvoC' mode, the compression factor is optimised for the Evo quality setting applied and the number of range lines requested. This function returns the actual compression factor used for the particular Evo quality setting and range requested.

The value returned by this function can have the following values.

Return Value	Meaning
1	The data will not have been compressed.
2	The data will have been compressed by a factor of 2 to 1.
4	The data will have been compressed by a factor of 4 to 1.
8	The data will have been compressed by a factor of 8 to 1.
16	The data will have been compressed by a factor of 16 to 1.

```
unsigned short compressionFactor = 0;
compressionFactor = GEM_GetRequestedCompressionFactor();
```

GEM_GetRXAddress1

```
unsigned short GEM_GetRXAddress1(void);
```

This function returns the value of the RX_Address1 field in the ping configuration.

```
unsigned short addr = GEM_GetRXAddress1();
```

This field is currently unused by the Gemini sonar.

GEM_SetRXAddress1

```
void GEM_SetRXAddress1(unsigned short rxAddress1);
```

This function sets the RX_Address1 field in the ping configuration which will be sent to the head when a ping is requested.

```
GEM_SetRXAddress1(0);
```

Default value: 0

This field is currently unused by the Gemini sonar.

GEM_GetRXAddress2

```
unsigned short GEM_GetRXAddress2(void);
```

This function returns the value of the RX_Address2 field in the ping configuration.

```
unsigned short addr = GEM_GetRXAddress2();
```

This field is currently unused by the Gemini sonar.

GEM_SetRXAddress2

```
void GEM_SetRXAddress2(unsigned short rxAddress2);
```

This function sets the RX_Address2 field in the ping configuration which will be sent to the head when a ping is requested.

```
GEM_SetRXAddress2(0);
```

Default value: 0

This field is currently unused by the Gemini sonar.

GEM_GetTXAddress1

```
unsigned short GEM_GetTXAddress1(void);
```

This function returns the value of the TX_Address1 field in the ping configuration.

```
unsigned short addr = GEM_GetTXAddress1();
```

This field is currently unused by the Gemini sonar.

GEM_SetTXAddress1

```
void GEM_SetTXAddress1(unsigned short txAddress1);
```

This function sets the TX_Address1 field in the ping configuration which will be sent to the head when a ping is requested.

```
GEM_SetTXAddress1(0);
```

Default value: 0

This field is currently unused by the Gemini sonar.

GEM_GetTXAddress2

```
unsigned short GEM_GetTXAddress2(void);
```

This function returns the value of the TX_Address2 field in the ping configuration.

```
unsigned short addr = GEM_GetTXAddress2();
```

This field is currently unused by the Gemini sonar.

GEM_SetTXAddress2

```
void GEM_SetTXAddress2(unsigned short txAddress2);
```

This function sets the TX_Address2 field in the ping configuration which will be sent to the head when a ping is requested.

```
GEM_SetTXAddress2(0);
```

Default value: 0

This field is currently unused by the Gemini sonar.

GEM_SetRunMode

```
void GEM_SetRunMode(unsigned short triggerEdge, unsigned
short pingMethod);
```

This function sets the Run_mode field in the ping configuration which will be sent to the head when a ping is requested.

triggerEdge	Meaning
0	Use positive edge external TLL trigger (when pingMethod = 2)
1	Use negative edge external TLL trigger (when pingMethod = 2)

pingMethod	Meaning
0	Ping once on receipt of ping configuration message
1	Ping repeatedly at interval fixed by GEM_SetInterPingPeriod
2	Ping on external TTL edge trigger
3	Reserved for future use

```
GEM_SetRunMode(0, 1);
```

Default value: triggerEdge 0
pingMethod 0

GEM_SetExtModeDataType

```
void GEM_SetExtModeDataType(unsigned short dataType);
```

This function sets the Data Type sub-field of the Ext_mode field in the ping configuration which will be sent to the head when a ping is requested.

dataType	Meaning
0	Beamformed magnitude data (8 or 16 bit)
1	Beamformed QI data (16 bit only)
2	Channel QI data (16 bit only)
3	Channel sample data (16 bit only)

```
GEM_SetExtModeDataType(0);
```

Default value: dataType 0

GEM_SetExtModeDataSize

```
void GEM_SetExtModeDataSize(unsigned short dataSize);
```

This function sets the Data Size sub-field of the Ext_Mode field in the ping configuration which will be sent to the head when a ping is requested. This field only has an effect when beamformed magnitude data has been requested from the head.

dataSize	Meaning
0	16 bit data
1	8 bit data

```
GEM_SetExtModeDataSize(1);
```

Default value: dataSize 1

GEM_SetExtModeFocusMode

```
void GEM_SetExtModeFocusMode(unsigned short focusMode);
```

This function sets the Focus Mode sub-field of the Ext_Mode field in the ping configuration which will be sent to the head when a ping is requested.

focusMode	Meaning
0	Two way focus
1	One way focus
2	Disable focus
3	Reserved for future use

```
GEM_SetExtModeFocusMode(0);
```

Default value: focusMode 0

GEM_SetExtModeVelDataEnable

```
void GEM_SetExtModeVelDataEnable(unsigned short
velDataEnable);
```

This function sets the Velocimeter data enable sub-field of the Ext_Mode field in the ping configuration which will be sent to the head when a ping is requested.

velDataEnable	Meaning
0	Do not send velocimeter data
1	Send velocimeter data immediately after ping

```
GEM_SetExtModeVelDataEnable(0);
```

Default value: velDataEnable 0

GEM_SetExtModeOutOfWaterOverride

```
void GEM_SetExtModeOutOfWaterOverride(unsigned short
outOfWaterOverride);
```

This function sets the Out of Water Override sub-field of the Ext_Mode field in the ping configuration which will be sent to the head when a ping is requested.

outOfWaterOverride	Meaning
0	Do not ping when out of water
1	Ping regardless of out of water indicator

```
GEM_SetExtModeOutOfWaterOverride(0);
```

Default value: outOfWaterOverride 0

GEM_SetExtModeScopeEnable

```
void GEM_SetExtModeScopeEnable(unsigned short
scopeEnable);
```

This function sets the Scope Enable sub-field of the Ext_Mode field in the ping configuration which will be sent to the head when a ping is requested.

scopeEnable	Meaning
0	Do not send scope data
1	Send scope data immediately after ping

```
GEM_SetExtModeScopeEnable(0);
```

Default value: scopeEnable 0

This function is for development use only and should not be enabled for production code.

GEM_SetExtModeTDBFlag

```
void GEM_SetExtModeTDBFlag(unsigned short tdbFlag);
```

This function sets the TDB Flag sub-field of the Ext_Mode field in the ping configuration which will be sent to the head when a ping is requested. If this flag is set, then time delay beamforming in the head is disabled. The default state is for time delay beamforming in the head to be enabled.

tdbFlag	Meaning
0	Enable time delay beamforming
1	Disable time delay beamforming

```
GEM_SetExtModeTDBFlag(0);
```

Default value: tdbFlag 0

GEM_SetExtModeSimCQIFlag

```
void GEM_SetExtModeSimCQIFlag(unsigned short simCQIFlag);
```

This function sets the Sim CQI Flag sub-field of the Ext_Mode field in the ping configuration which will be sent to the head when a ping is requested.

simCQIFlag	Meaning
0	Disable CQI simulation
1	Enable CQI simulation

```
GEM_SetExtModeSimCQIFlag(0);
```

Default value: simCQIFlag 0

This function is for development use only and should not be enabled for production code.

GEM_SetExtModeADCFlag

```
void GEM_SetExtModeSimADCFlag(unsigned short simADCFlag);
```

This function sets the Sim ADC Flag sub-field of the Ext_Mode field in the ping configuration which will be sent to the head when a ping is requested.

simADCFlag	Meaning
0	Disable ADC simulation
1	Enable ADC simulation

```
GEM_SetExtModeSimADCFlag(0);
```

Default value: simADCFlag 0

This function is for development use only and should not be enabled for production code.

GEM_SetExtModeVelDataEnableOnce

```
void GEM_SetExtModeVelDataEnableOnce(unsigned short
velDataEnableOnce);
```

This function sets the Velocimeter data enable once sub-field of the Ext_Mode field in the ping configuration which will be sent to the head when a ping is requested.

velDataEnableOnce	Meaning
0	Do not send velocimeter data
1	Send velocimeter data once immediately after ping

```
GEM_SetExtModeVelDataEnable(0);
```

Default value: velDataEnableOnce 0

GEM_SetStartRange

```
void GEM_SetStartRange(unsigned short rangeInLines);
```

This function sets the Start_range field in the ping configuration which will be sent to the head when a ping is requested. This field specifies the starting range (in range lines) of the data to be returned when the ping completes.

```
GEM_SetStartRange(0);
```

Default value: rangeInLines 0
Minimum value: 0
Maximum value:

This functionality has yet to be implemented in the Gemini Sonar Head, and so the sonar will ignore any value set in this field and behave as though it was always set to zero.

GEM_SetEndRange

```
void GEM_SetEndRange(unsigned short rangeInLines);
```

This function sets the End_range field in the ping configuration which will be sent to the head when a ping is requested. This field specifies the end range (in range lines) of the data to be returned when the ping completes.

```
GEM_SetEndRange(0);
```

Default value: rangeInLines 0
Minimum value: 33
Maximum value:

The Gemini Sonar head reacts badly to small values of end range, so that DLL ensures that a minimum value of 33 is sent to the sonar head, even if a value less than that is requested.

GEM_SetInterPingPeriod

```
void GEM_SetInterPingPeriod(unsigned int
periodInMicroSeconds);
```

This function sets the Inter_ping field in the ping configuration which will be sent to the head when a ping is requested. This field specifies the time delay between the start of one ping and the start of the next ping when the sonar head is pinging continuously.

```
GEM_SetInterPingPeriod(200000);
```

Default value: periodInMicroSeconds 0
Minimum value:
Maximum value:

GEM_SetTXLength

```
void GEM_SetTXLength(unsigned short txLength);
```

This function sets the Mb_tx_length field in the ping configuration which will be sent to the head when a ping is requested. This field specifies the length of the multibeam transmit pulse length in cycles.

Large values for txLength should be avoided as these can potentially damage the sonar at high ping rates. In an automatically configured ping (GEM_AutoPingConfig), the maximum value of txLength used is 32.

A value of zero for txLength has a special meaning to the code when it will stop the sonar's transmitter and velocimeter transmitting so the sonar emits no sonar power but receives as normal (for use in calibration, for example).

```
GEM_SetTXLength(4);
```

Default value: txLength 4
Minimum value: 4
Maximum value: 255

GEM_SetTXAngle

```
void GEM_SetTXAngle(unsigned short txAngle);
```

This function sets the Mb_tx_angle field in the ping configuration which will be sent to the head when a ping is requested. This field is reserved for future use.

```
GEM_SetTXAngle(0);
```

Default value: txAngle 0
Minimum value:
Maximum value:

This functionality has yet to be implemented in the Gemini Sonar Head, and so the sonar will ignore any value set in this field and behave as though it was always set to zero.

GEM_SetMainGain

```
void GEM_SetMainGain(unsigned short baseGain, unsigned
short variableGain);
```

This function sets the Main_gain field in the ping configuration which will be sent to the head when a ping is requested. The gain actually applied is the sum of the base gain and the variable gain. The maximum gain which can be applied in the system is 66.5dB. As there are many ways of getting the lower gain values (e.g. 40dB could be 40dB of variable gain with 0dB base gain, or 23.5dB of main gain and 16.5dB of base gain), the aim in setting the gain values should be to use the lowest value of base gain to get the desired gain value, as this minimises the noise from the amplifiers.

When considering the Main_gain field of the ping configuration, the gain response curve of the amplifiers needs to be considered. This is roughly summarised as follows, for variableGain values between 0 and 1000, the gain is 8dB; the gain is linear between variableGain values from 1000 to 3700, rising from 8dB to 50dB; and after a variableGain value of 3700, the gain remains 50dB.

baseGain	Meaning
0	0dB VGA base gain
1	5.5dB VGA base gain
2	11dB VGA base gain
3	16.5dB VGA base gain

variableGain	Meaning
0 to 1000	8dB VGA variable gain
1000 to 3700	8db to 50dB VGA variable gain (linear relationship)
3700 to 4095	50dB VGA variable gain

```
GEM_SetMainGain(0, 1);
```

Default value: baseGain 0
variableGain 0

GEM_SetMainGainPerCent

```
void GEM_SetMainGainPerCent(unsigned short perCentGain);
```

This function sets the Main_gain field in the ping configuration which will be sent to the head when a ping is requested. This function is a wrapper function to ease the implementation of the gain setting for the Gemini sonar. The function calculates the variable gain and base gain setting required for the percentage gain requested in perCentGain, and then calls GEM_SetMainGain with these values to set the gain. These calculations take into account the gain response curve of the amplifiers (as discussed under GEM_SetMainGain).

```
GEM_SetMainGainPerCent(50);
```

Default value: perCentGain 0
Minimum value: 0
Maximum value: 100

GEM_SetMainGainUnits

```
void GEM_SetMainGainUnits(unsigned short dBGain);
```

This function sets the Main_gain field in the ping configuration which will be sent to the head when a ping is requested. This function is a wrapper function to ease the implementation of the gain setting for the Gemini sonar. The function calculates the variable gain and base gain setting required for the gain value requested in dBGain, and then calls GEM_SetMainGain with these values to set the gain. dBGain expresses the gain required in dB * 10 (e.g. 40.5dB would be sent as 405). These calculations take into account the gain response curve of the amplifiers (as discussed under GEM_SetMainGain).

```
GEM_SetMainGainUnits(405);
```

Default value: dBGain 0
Minimum value: 0
Maximum value: 665

GEM_SetAbsorbGain

```
void GEM_SetAbsorbGain(unsigned short absorbtionGain);
```

This function sets the Absorb_gain field in the ping configuration which will be sent to the head when a ping is requested. This field specifies the absorption gain.

```
GEM_SetAbsorbGain(0);
```

Default value: absorbtionGain 0
Minimum value:
Maximum value:

GEM_SetSoftGain

```
void GEM_SetSoftGain(unsigned int softwareGain);
```

This function sets the Soft_gain field in the ping configuration which will be sent to the head when a ping is requested. This field specifies the software gain.

```
GEM_SetSoftGain(0);
```

Default value: softwareGain 0x00001E00
Minimum value:
Maximum value:

GEM_SetSpreadGain

```
void GEM_SetSpreadGain(unsigned short spreadingGain);
```

This function sets the Spreading_gain field in the ping configuration which will be sent to the head when a ping is requested. This field specifies the spreading gain.

```
GEM_SetSpreadGain(0);
```

Default value: spreadingGain 0
Minimum value:
Maximum value:

GEM_SetAutoGain

```
void GEM_SetAutoGain(unsigned short autoGainFlag,
unsigned short targetGain);
```

This function sets the Auto_gain field in the ping configuration which will be sent to the head when a ping is requested.

autoGainFlag	Meaning
0	Disable auto gain
1	Enable auto gain

targetGain	Meaning
0	50% target gain
1	56% target gain
2	63% target gain
3	69% target gain
4	76% target gain
5	82% target gain
6	89% target gain
7	96% target gain

```
GEM_SetAutoGain(0, 0);
```

Default value: autoGainFlag 0
targetGain 0

This functionality has yet to be implemented in the Gemini Sonar Head, and so the sonar will ignore any value set in this field and behave as though it was always disabled.

GEM_SetSoftwareGainRamp

```
void GEM_SetSoftwareGainRamp(unsigned short
softwareGainRamp);
```

This function sets the SW_gain_ramp field in the ping configuration which will be sent to the head when a ping is requested. This field specifies the how the software gain increases with range.

```
GEM_SetSoftwareGainRamp(0);
```

Default value: spreadingGain 0
Minimum value:
Maximum value:

GEM_SetVelocimeterTXPeriod

```
void GEM_SetVelocimeterTXPeriod(unsigned short
velocimeterTXPeriod);
```

This function sets the Vel_tx_period field in the ping configuration which will be sent to the head when a ping is requested. This field specifies the velocimeter transmission period to be used.

```
GEM_SetVelocimeterTXPeriod(0);
```

Default value: velocimeterTXPeriod
Minimum value:
Maximum value:

The DLL is configured with the correct value of velocimeter transmission period to send to the Gemini Sonar head. Using this function to set others values (unless instructed by SRD) will have unexpected and unwanted effects on the operation of the sonar.

GEM_SetVelocimeterTXMark

```
void GEM_SetVelocimeterTXMark(unsigned char
velocimeterTXMark);
```

This function sets the Vel_tx_mark field in the ping configuration which will be sent to the head when a ping is requested. This field specifies the velocimeter transmit mark period to be used.

```
GEM_SetVelocimeterTXMark(0);
```

Default value: velocimeterTXMark
Minimum value:
Maximum value:

The DLL is configured with the correct value of velocimeter transmit mark period to send to the Gemini Sonar head. Using this function to set others values (unless instructed by SRD) will have unexpected and unwanted effects on the operation of the sonar.

GEM_SetFilterBank

```
void GEM_SetFilterBank(unsigned short
channelFIRFilterBandwidth);
```

This function sets the Filter_bank field in the ping configuration which will be sent to the head when a ping is requested.

channelFIRFilterBandwidth	Meaning
0	30kHz channel FIR filter bandwidth
1	40kHz channel FIR filter bandwidth
2	50kHz channel FIR filter bandwidth
3	60kHz channel FIR filter bandwidth
4	70kHz channel FIR filter bandwidth
5	80kHz channel FIR filter bandwidth
6	90kHz channel FIR filter bandwidth
7	100kHz channel FIR filter bandwidth

```
GEM_SetFilterBank(0);
```

Default value: channelFIRFilterBandwidth 0

GEM_SetShadeBank

```
void GEM_SetShadeBank(unsigned short
beamformerShadingLevel);
```

This function sets the Shade_bank field in the ping configuration which will be sent to the head when a ping is requested.

beamformerShadingLevel	Meaning
0	None
1	15dB beamformer shading level
2	20dB beamformer shading level
3	25dB beamformer shading level
4	30dB beamformer shading level
5	35dB beamformer shading level
6	40dB beamformer shading level
7	45dB beamformer shading level

```
GEM_SetShadeBank(0);
```

Default value: beamformerShadingLevel 0

GEM_SetSpeedOfSound

```
void GEM_SetSpeedOfSound(unsigned short speedOfSound);
```

This function sets the Spd_snd field in the ping configuration which will be sent to the head when a ping is requested. This field specifies the speed of sound to be used by the sonar head if it is not using the value determined by the internal velocimeter. The value sent to this function is the speed of sound measured in units of 0.1m/s. The sonar will round the speed of sound to the nearest 3.2m/s step before using the value. Thus 1500m/s will be rounded to 1499.2m/s.

```
GEM_SetSpeedOfSound(15000); // Speed of sound 1500.0 m/s
                            // Value used will be 1499.2
```

Default value: speedOfSound 15000
Minimum value: 1400m/s
Maximum value: 1588m/s

GEM_SetVelocimeterMode

```
void GEM_SetVelocimeterMode(unsigned short gainMode,
unsigned short outputMode);
```

This function sets the Vel_mode field in the ping configuration which will be sent to the head when a ping is requested.

gainMode	Meaning
0	Use auto gain
1	Use manual gain

outputMode	Meaning
0	Use velocimeter calculated speed of sound
1	Use speed of sound specified in this ping configuration message

```
GEM_SetVelocimeterMode(0, 0);
```

Default value: gainMode 0
outputMode 0

The gain mode functionality will not be implemented in the Gemini Sonar Head, as the head automatically calculates the gain required by the velocimeter. Thus the sonar will ignore any value set in the gain mode field and behave as though it was always set to zero.

GEM_SetRangeCompression

```
void GEM_SetRangeCompression(unsigned short
compressionLevel, unsigned short compressionType);
```

This function sets the Rng_comp field in the ping configuration which will be sent to the head when a ping is requested.

compressionLevel	Meaning
0	No range compression
1	2 * range compression
2	4 * range compression
3	8 * range compression
4	16 * range compression

compressionType	Meaning
0	Use average compression
1	Use peak compression

```
GEM_SetRangeCompression(0, 0);
```

Default value: compressionLevel 0
compressionType 0

Only average compression has been implemented in the Gemini sonar, and so setting the compressionType will have no effect.

The range compression is used by the DLL's SeaNetC mode to reduce the amount of data being passed from the DLL to SeaNet. In the SeaNetC mode, the DLL looks at the range requested in the ping and selects a suitable value for range compression so that no more than 1500 lines are passed to SeaNet. Normally if the range compression has been set, the DLL will remove the compression before passing the data to the calling program, in SeaNetC mode, the compression is not removed and the compressed data is passed to SeaNet.

The range compression is used by the DLL's EvoC mode to optimise the bandwidth used to transfer the data from the sonar to the surface computer, whilst considering the performance of the Evo display being used. In the EvoC mode, the DLL looks at the range requested in the ping and Evo quality setting that has been set; and then selects a suitable value for range compression so an optimum number lines are passed. Normally if the range compression has been set, the DLL will remove the compression before passing the data to the calling program, in EvoC mode, the compression is not removed and the compressed data is passed to Evo.

GEM_SetRLEThreshold

```
void GEM_SetRLEThreshold(unsigned short threshold);
```

This function sets the Rle_threshold field in the ping configuration which will be sent to the head when a ping is requested. The DLL removes the run length encoding from the data before it is presented to the calling program, so the calling program does not need to be aware of the run length encoding algorithm used. Using run length encoding may reduce the bandwidth required to get the data from the Gemini sonar head to the hardware running the calling software. Values of 1 and 2 are reserved for use within the software, and so will increased to 3 if selected.

compressionLevel	Meaning
0	No run length encoding
1	Not available for use, will be increased to 3
2	Not available for use, will be increased to 3
3	
...	
255	Maximum run length encoding

```
GEM_SetRLEThreshold(0);
```

Default value: compressionLevel 0
Minimum value: 0
Maximum value: 255

GEM_SetChannelSample

```
void GEM_SetChannelSample(unsigned short channelSample);
```

This function sets the Chan_samp field in the ping configuration which will be sent to the head when a ping is requested. This field only has any effect when the Data Type subfield of the Ext_mode field is set to three. In that case, this field specifies the channel which will be used.

```
GEM_SetChannelSample(0);
```

Default value: channelSample 0
Minimum value: 0
Maximum value: 95

GEM_SetVelocimeterGain

```
void GEM_SetVelocimeterGain(unsigned short gain);
```

This function sets the Vel_gain field in the ping configuration which will be sent to the head when a ping is requested. This field specifies the velocimeter gain to be used.

```
GEM_SetVelocimeterGain(0);
```

Default value: gain 0
Minimum value: 0
Maximum value: 4095

The velocimeter gain in the Gemini Sonar head is automatically calculated by the algorithms in the head, this means that there is no requirement for the user to set the velocimeter gain. Setting the velocimeter gain has no effect on the Gemini sonar.

GEM_SetVelocimeterTXLength

```
void GEM_SetVelocimeterTXLength(unsigned char txLength);
```

This function sets the Vel_tx_length field in the ping configuration which will be sent to the head when a ping is requested. This field specifies the length of the velocimeter transmit pulse (in cycle) to be used.

```
GEM_SetVelocimeterTXLength(0);
```

Default value: txLength 0
Minimum value: 0
Maximum value: 255

The DLL is configured with the correct value of velocimeter transmission length to send to the Gemini Sonar head. Using this function to set others values (unless instructed by SRD) will have unexpected and unwanted effects on the operation of the sonar.

GEM_SetPingToDefaults

```
void GEM_SetPingToDefaults(void);
```

This function returns all the fields of the ping configuration which will be sent to the head when a ping is requested back to their default values.

```
GEM_SetPingToDefaults();
```

GEM_SetStartBeam

```
void GEM_SetStartBeam(unsigned short startBeam);
```

This function specifies the starting beam number of the data to be returned when the ping completes. This function is only applicable when the DLL has been set into "Seanet" or "SeanetC" mode. The processing which removes the unwanted beams from the data being received is applied after a full ping has been received, and as the data is passed back using the CGemBearingData messages.

```
GEM_SetStartBeam(0);
```

Default value: startBeam 0
Minimum value: 0
Maximum value: 255

GEM_SetEndBeam

```
void GEM_SetEndBeam(unsigned short endBeam);
```

This function specifies the end beam number of the data to be returned when the ping completes. This function is only applicable when the DLL has been set into "Seanet" or "SeanetC" mode. The processing which removes the unwanted beams from the data being received is applied after a full ping has been received, and as the data is passed back using the CGemBearingData messages.

If the value of the end beam is set to be less than the start beam value (set by GEM_SetStartBeam), no data will be returned to the calling program.

```
GEM_SetEndBeam(255);
```

Default value: endBeam 255
Minimum value: 0
Maximum value: 255

GEM_SendGeminiStayAlive

```
void GEM_SendGeminiStayAlive(void);
```

This function sends a stay-alive message to the Gemini sonar head. If the head does not receive any communications data from the surface PC for 3 seconds it will stop sending data back to the surface PC. The head will stop pinging if it is in continuous ping mode until further communications are received. The stay-alive message is a simple way of keeping the communications alive between surface PC and the head.

The Gemini sonar head will respond with an acknowledge message which will result in a CGemAcknowledge packet being passed to the data handler callback function (with the type identifier of GEM_ACKNOWLEDGE). This packet contains timestamp information which may or may not be useful to the calling application.

```
GEM_SendGeminiStayAlive();
```

GEM_SendGeminiPingConfig

```
void GEM_SendGeminiPingConfig(void);
```

This function sends a ping configuration message to the Gemini sonar head, which will cause the head to issue one or more pings (depending on how the ping configuration has been set).

```
GEM_SendGiminiPingConfig();
```

GEM_RebootSonar

```
void GEM_RebootSonar(void);
```

This function issues a command to the Gemini sonar head to command it to reboot into the main software image in the Sonar head.

```
GEM_RebootSonar();
```

GEM_InhibitSonarReboot

```
void GEM_InhibitSonarReboot(void);
```

This function issues a command to the Gemini sonar head to command it to inhibit the automatic reboot of the Gemini sonar 10 seconds after power up. This command is available for SRD Production software only.

```
GEM_InhibitSonarReboot();
```

GEM_SendRS232

```
void GEM_SendRS232(int len, char *data);
```

This function sends a block of data to the Gemini sonar head, which the head will copy of its auxiliary communications (RS232) port.

len specifies the amount of data which will be sent to the head.

data is a pointer to the data to be transmitted.

```
GEM_SendRS232(13, "Hello, world!");
```

GEM_CommandPOST

```
void GEM_CommandPOST(void);
```

This function commands the Gemini Sonar Head to perform a Power On Self Test (POST). The results of the POST are available to the calling software using the GEM_RequestTestResult function.

```
GEM_CommandPOST();
```

GEM_CommandProductionTest

```
void GEM_CommandProductionTest(void);
```

This function commands the Gemini Sonar Head to perform a production test. The results of the production test are available to the calling software using the GEM_RequestTestResult function.

The production test is a lengthy and destructive test which will render the sonar head unusable if the head is not reprogrammed once the test has been performed. The test will take some 15 minutes to perform. Because of these two considerations, the production test is only available to the SRD Gemini Production Software.

```
GEM_CommandProductionTest();
```

GEM_AbortProductionTest

```
void GEM_AbortProductionTest(void);
```

This function commands the Gemini Sonar Head to abort a production test it is already performing.

The production test is a lengthy and destructive test which will render the sonar head unusable if the head is not reprogrammed once the test has been performed. The test will take some 15 minutes to perform. Because of these two considerations, the production test is only available to the SRD Gemini Production Software.

Depending on when the abort command is issued relative to the command to start the test, the Gemini Sonar head may still have some functionality when the test has been aborted – this should not be assumed though.

```
GEM_AbortProductionTest();
```

GEM_RequestTestResult

```
void GEM_RequestTestResult(void);
```

This function requests the Gemini Sonar Head to send up the results of the last test performed, either a POST or a production test.

The Gemini sonar head will respond with a test result message which will result in a CGemTestResult packet being passed to the data handler callback function (with the type identifier of GEM_TEST_RESULT). The contents of this packet are only of interest to the SRD Production Software, and are not detailed in this document.

The production test is a lengthy and destructive test which will render the sonar head unusable if the head is not reprogrammed once the test has been performed. The test will take some 15 minutes to perform. GEM_RequestTestResult can be called while the production test is running to ascertain the progress of the test, as well as to see how many (if any) errors have been detected.

```
GEM_RequestTestResult();
```

GEM_RequestRetry

```
void GEM_RequestRetry(unsigned short type, unsigned short
lineNo, unsigned short noOfLines);
```

This function requests the Gemini Sonar Head to resend part of the data from the last ping performed by the sonar.

The Gemini sonar head uses UDP as the basis for its communication with the surface system. As UDP does not guarantee the delivery of any particular packet, a retry mechanism has been built into the DLL. This mechanism will attempt one retry for any data that it notices is missing. This function (GEM_RequestRetry) allows the calling software to request further retries beyond the ones attempted by the DLL.

type is the particular type of retry required, defined as follows

type	meaning
GEM_RETRY_REQUEST_HEAD	The last sent Ping Head packet is re-sent
GEM_RETRY_REQUEST_LINE	The Ping Line specified by lineNo is re-sent
GEM_RETRY_REQUEST_TAIL	The last sent Ping Tail packet is re-sent

lineNo is the first line number that is required to be resent, when a Ping Line resend is being requested.

noOfLines is the number of lines that are required to be resent, when a Ping Line resend is being requested.

```
GEM_RequestRetry(GEM_RETRY_REQUEST_LINE, 20, 1);
```

GEM_SendNetworkConfig

```
void GEM_SendGeminiNetworkConfig(void);
```

The DLL holds a network configuration structure within it which has various settings for the network configuration of the Gemini sonar head. After any of these settings has been changed, the structure needs to be sent to the head. The GEM_SendNetworkConfig command sends the structure to the head, and so implementing the changes made upto that point.

The following commands change the network settings, and so need to be followed by a call to GEM_SendNetworkConfig to commit the changes made.

GEM_SetNetworkIFGs
GEM_SetNetworkAdvertiseGigaBit
GEM_SetNetworkRetriggerVDSLRateAdaption
GEM_SetNetworkInhibitRecommendedRateAdaption
GEM_SetNetworkEnableAllRateAdaption
GEM_SetNetworkEnableRAMonitoring
GEM_SetNetworkRS232BaudRate
GEM_SetNetworkConfigToDefaults

As GEM_SetNetworkRetriggerVDSLRateAdaption is used to trigger an action rather than just change a setting, after the message has been sent to the Sonar head, GEM_SendGeminiNetworkConfig will reset the rate adaption retrigger bit so that the same action will not be triggered the next time the network configuration is sent to the head.

```
GEM_SendGeminiNetworkConfig();
```

GEM_SetNetworkIFGs

```
void GEM_SetNetworkIFGs(unsigned char ethernetIFG,
unsigned char VDSLIFG);
```

This command sets the value of the interframe gap (IFG) in clock cycles for the ethernet and VDSL links. The minimum value for either IFG is 15. The values are stored in the network configuration structure held in the DLL and sent to the Sonar head when the GEM_SendGeminiNetworkConfig command is called.

```
GEM_SetNetworkIFGs(15, 15);
```

Default value: ethernetIFG 0
VDSLIFG 0
Minimum value: 15 (in both cases)
Maximum value: 255 (in both cases)

GEM_GetNetworkIFGs

```
void GEM_GetNetworkIFGs(unsigned char *ethernetIFG,
unsigned char *VDSLIFG);
```

This command retrieves the value of the interframe gap (IFG) in clock cycles for the ethernet and VDSL links. The DLL adjusts the interframe gap in the network message to attempt to improve the quality of the communications, so this function allows the current values being used by the DLL to be retrieved by the calling program.

```
unsigned char ethernetIFG;
unsigned char VDSLIFG;

GEM_GetNetworkIFGs(&ethernetIFG, &VDSLIFG);
```

GEM_SetNetworkAdvertiseGigaBit

```
void GEM_SetNetworkAdvertiseGigaBit(unsigned short
advertiseGigaBit);
```

This command controls whether the Gemini sonar advertises that it is capable of running over GigaBit ethernet when auto-negotiating the ethernet link to the unit. When a standard Gemini is connected to an ethernet network (as opposed to VDSL) there are not enough connections available for a GigaBit link to be made. Unfortunately this does not stop the unit attempting to connect at a GigaBit rate (and then not being able to communicate). By default the Gemini sonar does not advertise the fact that it is GigaBit capable, and hence it does not attempt GigaBit connections. Setting this flag, and sending the resultant network configuration to the Sonar head will cause the unit to advertise that it is GigaBit capable.

This command should be used with extreme care, as a standard Gemini sonar cannot support GigaBit communications electrically, and enabling this functionality by calling GEM_SetNetworkAdvertiseGigaBit(1) will stop the unit being able to communicate until it is power cycled.

The value is stored in the network configuration structure held in the DLL and sent to the Sonar head when the GEM_SendGeminiNetworkConfig command is called.

advertiseGigaBit	Meaning
0	Do not advertise that Gigabit connections are available
1	Advertise that Gigabit connections are available

```
GEM_SetNetworkAdvertiseGigaBit(0);
```

Default value: 0

GEM_SetNetworkRetriggerVDSLRateAdaption

```
void GEM_SetNetworkRetriggerVDSLRateAdaption(unsigned
short retriggerVDSLadaption);
```

This command sets a value in the network configuration, which when sent to the Gemini sonar head causes the head to renegotiate the speed of the VDSL link (assuming the head is connected via VDSL and not directly via ethernet). During this renegotiation process no communications with the head can take place.

The value is stored in the network configuration structure held in the DLL and sent to the Sonar head when the GEM_SendGeminiNetworkConfig command is called. As this is a trigger for an action rather than an ongoing value, the code in the DLL resets this value so that further commands to the head will not result in the VDSL link speed being renegotiated.

retriggerVDSLAdaption	Meaning
0	Do not retrigger a VDSL rate adaption
1	Retrigger a VDSL rate adaption

```
GEM_SetNetworkRetriggerVDSLRateAdaption(1);
```

Default value: 0

GEM_SetNetworkIgnoreRecommendedRateAdaption

```
void GEM_SetNetworkInhibitRecommendedRateAdaption(
unsigned short inhibitRecommendedRA);
```

This command sets a value in the network configuration, which when sent to the Gemini sonar head causes the head to ignore any recommended VDSL link rate adaptions (assuming the head is connected via VDSL and not directly via ethernet).

The value is stored in the network configuration structure held in the DLL and sent to the Sonar head when the GEM_SendGeminiNetworkConfig command is called.

inhibitRecommendedRA	Meaning
0	Ignore rate adaption recommendations from the VDSL chipset.
1	Act on rate adaption recommendations from the VDSL chipset.

```
GEM_SetNetworkInhibitRecommendedRateAdaption(1);
```

Default value: 0

GEM_SetNetworkEnableAllRateAdaption

```
void GEM_SetNetworkEnableAllRateAdaption(unsigned short
enableAllRA);
```

This command sets a value in the network configuration, which when sent to the Gemini sonar head causes the head to enable all VDSL link rate adaptions (assuming the head is connected via VDSL and not directly via ethernet).

The value is stored in the network configuration structure held in the DLL and sent to the Sonar head when the GEM_SendGeminiNetworkConfig command is called.

enableAllRA	Meaning
0	Disable all VDSL rate adaptions
1	Enable all VDSL rate adaptions

```
GEM_SetNetworkEnableAllRateAdaption(1);
```

Default value: 0

GEM_SetNetworkEnableRAMonitoring

```
void GEM_SetNetworkEnableAllRateAdaption(unsigned short
enableRAMonitoring);
```

This command sets a value in the network configuration, which when sent to the Gemini sonar head causes the head to enable VDSL link rate monitoring (assuming the head is connected via VDSL and not directly via ethernet).

The value is stored in the network configuration structure held in the DLL and sent to the Sonar head when the GEM_SendGeminiNetworkConfig command is called.

enableRAMontoring	Meaning
0	Disable VDSL rate adaption monitoring
1	Enable VDSL rate adaption monitoring

```
GEM_SetNetworkEnableRAMonitoring(1);
```

Default value: 0

GEM_SetNetworkRS232BaudRate

```
void GEM_SetNetworkRS232BaudRate(unsigned short
baudRate);
```

This command sets the baud rate of the onboard serial port of the Gemini sonar head. The possible baud rates are 2,400 baud, 4,800 baud, 9,600 baud, 19,200 baud, 38,400 baud, 57,600 baud, and 115,200 baud. The value of baudRate is the baud rate required (e.g. 19200 for 19,200 baud). If the value is not valid, the baud rate will default to 9600 baud.

The value is stored in the network configuration structure held in the DLL and sent to the Sonar head when the GEM_SendGeminiNetworkConfig command is called.

```
GEM_SetNetworkRS232BaudRate(9600);
```

Default value: 9600
Possible values: 2400, 4800, 9600, 19200, 38400, 57600, 115200

GEM_SetNetworkConfigToDefaults

```
void GEM_SetNetworkConfigToDefaults(void);
```

This function will return all the values stored in the network configuration structure of the DLL to their default settings.

The values are stored in the network configuration structure held in the DLL and sent to the Sonar head when the GEM_SendGeminiNetworkConfig command is called.

```
GEM_SetNetworkConfigToDefaults();
```

GEM_SetNetworkInterPacketGap

```
void GEM_SetNetworkInterPacketGap(unsigned short
interPacketGap);
```

This function sets the interpacket gap used by the Gemini sonar when transmitting data. The interpacket gap is the time inserted between packets sent by the Gemini sonar, and allows the packet rate on the network to be reduced so improving the quality of the link at the cost of reducing the update rate of images received from the sonar. The interpacket gap is specified in units of 8nS. A value of zero is permitted which means that the network runs at the highest possible rate. The maximum value which can be sent is 65335, corresponding to an interpacket gap of 524.28uS.

The value is stored in the network configuration structure held in the DLL and sent to the Sonar head when the GEM_SendGeminiNetworkConfig command is called.

```
GEM_SetNetworkInterPacketGap(10);
```

Default value: 0
Minimum value: 0
Maximum value: 65535 (equates to 524.280 uS between packets)

GEM_GetNetworkInterPacketGap

```
unsigned short GEM_GetNetworkInterPacketGap(void);
```

This function returns the value of interpacket gap currently that was last sent to the Gemini sonar. The interpacket gap is the time inserted between packets sent by the Gemini sonar, and allows the packet rate on the network to be reduced so improving the quality of the link at the cost of reducing the update rate of images received from the sonar. The interpacket gap is specified in units of 8nS.

```
unsigned short ipg = 0;

ipg = GEM_GetNetworkInterPacketGap()
```

GEM_MDIOInit

```
void GEM_MDIOInit(void);
```

The DLL holds an MDIO (Management Data Input/Output) configuration structure within it which has various settings for the MDIO serial link between the MAC and the PHY in the electronics of the Gemini sonar head. This command clears the structure held in the DLL and initialises it ready for commands to be added. When all the required commands have been added, the list of commands is sent by calling GEM_MDIOSend.

```
GEM_MDIOInit();
```

The MDIO interface is a crucial part of the communications circuitry of the sonar head, therefore these commands should be used with great care, and generally under the direction of SRD. Changes can be made which are potentially permanaent and could stop the Sonar communicating.

GEM_MDIOAddVDSLReg

```
void GEM_MDIOAddVDSLReg(unsigned short regAddr, unsigned
short value);
```

This command adds a register address and value pair to the MDIO (Management Data Input/Output) configuration structure held within the DLL. This structure contains various settings for the MDIO serial link between the MAC and the PHY in the electronics of the Gemini sonar head. When all the required commands have been added, the list of commands is sent by calling GEM_MDIOSend.

```
GEM_MDIOAddVDSLReg(usRegAddr, usValue);
```

The MDIO interface is a crucial part of the communications circuitry of the sonar head, therefore these commands should be used with great care, and generally under the direction of SRD. Changes can be made which are potentially permanaent and could stop the Sonar communicating. Register addresses and values will only be provided by SRD when specific changes are required.

GEM_MDIOAddSwitchToEthernet

```
void GEM_MDIOAddSwitchToEthernet(void);
```

This command adds a command to switch to the Ethernet PHY to the MDIO (Management Data Input/Output) configuration structure held within the DLL. This structure contains various settings for the MDIO serial link between the MAC and the PHY in the electronics of the Gemini sonar head. When all the required commands have been added, the list of commands is sent by calling GEM_MDIOSend.

```
GEM_MDIOAddSwitchToEthernet();
```

The MDIO interface is a crucial part of the communications circuitry of the sonar head, therefore these commands should be used with great care, and generally under the direction of SRD. Changes can be made which are potentially permanaent and could stop the Sonar communicating.

GEM_MDIOAddSwitchToVDSL

```
void GEM_MDIOAddSwitchToVDSL(void);
```

This command adds a command to switch to the VDSL PHY to the MDIO (Management Data Input/Output) configuration structure held within the DLL. This structure contains various settings for the MDIO serial link between the MAC and the PHY in the electronics of the Gemini sonar head. When all the required commands have been added, the list of commands is sent by calling GEM_MDIOSend.

```
GEM_MDIOAddSwitchToVDSL();
```

The MDIO interface is a crucial part of the communications circuitry of the sonar head, therefore these commands should be used with great care, and generally under the direction of SRD. Changes can be made which are potentially permanaent and could stop the Sonar communicating.

GEM_MDIOSend

```
void GEM_MDIOSend(void);
```

The DLL holds an MDIO (Management Data Input/Output) configuration structure within it which has various settings for the MDIO serial link between the MAC and the PHY in the electronics of the Gemini sonar head. After commands have been added, the structure needs to be sent to the head. The GEM_MDIOSend command sends the list of commands in the structure to the head, which then passes them onto the MDIO interface to be executed.

The following commands add instructions to the MDIO list of commands, and so need to be followed by a call to GEM_MDIOSend to send the changes made.

GEM_MDIOInit
GEM_MDIOAddVDSLReg
GEM_MDIOAddSwitchToEthernet
GEM_MDIOAddSwitchToVDSL

```
GEM_MDIOSend();
```

The MDIO interface is a crucial part of the communications circuitry of the sonar head, therefore these commands should be used with great care, and generally under the direction of SRD. Changes can be made which are potentially permanaent and could stop the Sonar communicating.

GEM_SetVDSLSetting

```
void GEM_SetVDSLSetting(unsigned short level);
```

The link between the sonar head and the surface is an ethetnet link. In some configurations, the link may be carried over VDSL for part of its journey. In order to optimise the speed of the VDSL connection depending on the amount of electrical noise the link was subjected to, three different settings are provided. This function allows the calling program to change the settings of the VDSL link and then retrigger a VDSL rate adaption to make use of the new settings.

level signifies which set of VDSL settings to use, as defined below.

level	Meaning
0	Normal electrical noise environment
1	Medium electrical noise environment
2	High electrical noise environment

```
GEM_SetVDSLSetting(0);
```

Calling this function will cause a VDSL rate adaption, which will interrupt communications with the sonar head for a short period.

GEM_ProgramFPGA

```
void GEM_ProgramFPGA(char *fileName, char *destination);
```

This function launches a new task which programs an FPGA image into the Gemini Sonar head. The progress of the programming is monitored using the progress callback function (as previously described), if one has been specified. Programming the FPGA image takes a number of minutes to perform, the use of a separate task allows the calling code to continue operating, including the option to abort the programming of the FPGA image part way through.

There are two images stored in the sonar head, one is a boot image which is programmed during production of the unit and is unchangeable during normal operation of the unit. The main image is changeable during normal operation of the unit, but because this is undesirable, this function is only available to the SRD Production Software.

destination	meaning
"BOOT"	The contents of fileName are written to the Boot program area of the Sonar
"MAIN"	The contents of fileName are written to the Main program area of the Sonar

```
GEM_ProgramFPGA("GeminiMainFile.bit", "MAIN");
```

GEM_AbortFPGAProgramming

```
void GEM_AbortFPGAProgramming(void);
```

This function aborts the programming of the FPGA image previously started by a call to GEM_ProgramFPGA. The programming will stop as soon as practicably possible once this function has been called. This will generally leave an incomplete FPGA image programmed into the Gemini sonar head, and hence leave the head unuseable.

```
GEM_AbortFPGAProgramming();
```

GEM_InitCalibrationValues

```
void GEM_InitCalibrationValues(bool useFiles, char
*txFile, char *rxFile);
```

This function is used to add individual calibration data to the modifiers calculated and programmed into the Gemini sonar head. If the useFiles flag is set to a value other 0, the contents of the two files are read and used in the calculation of the modifier tables which are written to the Gemini sonar.

One file is used to carry the transmitter calibration characteristics. The name of this file is passed in through the txFile parameter. The file contains 256 floating point values, which are applied to the beams generated by the Gemini firmware.

The second file is used to carry the receiver calibration characteristics. The name of this file is passed in through the rxFile parameter. The file contains 96 pairs of floating point values. Each pair represents the real and imaginary parts of the factor which is applied to the receive channels of the Gemini sonar.

```
GEM_InitCalibrationValues(1, "C:\\Gemini\\txCalFile.txt",
"C:\\Gemini\\rxCalFile.txt");
```

GEM_CalculateModifiers

```
void GEM_CalculateModifiers(void);
```

This function calculates the modifier tables, using constants stored in the code, ready for them to be programmed into the Gemini sonar head during production and production testing.

```
GEM_CalculateModifiers();
```

GEM_ProgramModifiers

```
void GEM_ProgramModifiers(void);
```

This function launches a new task which programs the modifier tables into the Gemini Sonar head. The progress of the programming is monitored using the progress callback function (as previously described), if one has been specified. Programming the modifiers takes a significant time to perform, the use of a separate task allows the calling code to continue operating, including the option to abort the programming of the modifiers part way through.

If the modifiers have not been calculated already, nor loaded from file, the code will call the function to calculate the modifiers prior to programming.

```
GEM_ProgramModifiers();
```

GEM_AbortModifierProgramming

```
void GEM_AbortModifierProgramming(void);
```

This function aborts the programming of the modifier tables previously started by a call to GEM_ProgramModifiers. The programming will stop as soon as practicably possible once this function has been called. This will generally leave an incomplete set of modifier tables programmed into the Gemini sonar head, and hence leave the head potentially unuseable.

```
GEM_AbortModifierProgramming();
```

GEM_WriteModifiersToFile

```
int GEM_WriteModifiersToFile(char *fileName);
```

This function writes the modifier tables calculated from constants in the code to a file name, which is specified in the call to the function. The return value of the function indicates whether or not this was successful, and if not gives an indication of why the failure occured.

```
int retValue =
GEM_WriteModifiersToFile("c:\\temp\\modFile.mod");
```

retValue	meaning
GEM_FILE_FAILED	A general failure occured
GEM_FILE_SUCCESS	The file was written successfully
GEM_FILE_OPEN_FAILED	The file failed to open for writing
GEM_FILE_WRITE_FAILED	A write of data to the file failed

GEM_ReadModifiersFromFile

```
int GEM_ReadModifiersFromFile(char *fileName);
```

This function reads a set of modifier tables from a file, which is specified in the call to the function, and makes them available to be written to the Gemini sonar head. The return value of the function indicates whether or not this was successful, and if not gives an indication of why the failure occured.

```
int retValue =
GEM_ReadModifiersFromFile("c:\\temp\\modFile.mod");
```

retValue	meaning
GEM_FILE_FAILED	A general failure occured
GEM_FILE_SUCCESS	The file was written successfully
GEM_FILE_OPEN_FAILED	The file failed to open for reading
GEM_FILE_READ_FAILED	A read of data from the file failed

GEM_ReadFromFlash

```
void GEM_ReadFromFlash(unsigned int addressToRead)
```

This function requests a read from the flash in the Sonar head. Issuing this command will cause a message to be sent from the head which will be processed by the DLL and result in the main callback function being called to handle a GEM_FLASH_READBACK message. This message will have 1024 bytes of data read from the flash.

It should be noted that the Gemini firmware will truncate the address value so that it starts reading from the nearest word boundary below the requested address. For example if a request was made to read from address 1045, the resulting data would be 1024 bytes of data which started at address 1044.

```
GEM_ReadFromFlash(0x00000000);
```

GEM_WriteToFlash

```
void GEM_WriteToFlash(unsigned int addressToWrite,
unsigned short readbackFlag, unsigned char *data)
```

This function writes a 1024 bytes block of data to the flash in the Gemini sonar head. The address for the data has to be aligned to a 1k boundary for the Gemini firmware to handle it correctly, so the DLL will ensure this by setting the lowest 10 bits of the addressToWrite to zero before it is sent to sonar. Setting the readbackFlag to 1 will result in the sonar firmware reading the data back after it has been written and sending it back to the DLL. data is a pointer to the 1024 byte block of data to be written to the flash.

As a result of issuing this command, the DLL will receive a message from the sonar with the result of the write process, which will result in the main callback function being called to handle a GEM_FLASH_RESULT message. If the readbackFlag is set the sonar will also send a message with the data read back from the flash, which will result in the main callback function being called to handle a GEM_FLASH_READBACK message.

```
unsigned char data[1024];

// Write something into data to go in the flash.

GEM_WriteToFlash(0x00000000, 1, data);
```

The addresses of the flash info blocks (see GEM_ReadFlashInfoBlock for more information) are specifically trapped by the DLL so that the calling program cannot use GEM_WriteToFlash to write to the flash info blocks.

GEM_ReadFlashInfoBlock

```
void GEM_ReadFlashInfoBlock(unsigned short blockToRead)
```

This function requests a read of one of the info blocks from the flash in the Sonar head. Issuing this command will cause a message to be sent from the head which will be processed by the DLL and result in the main callback function being called to handle a GEM_FLASH_READBACK message. This message will have 1024 bytes of data read from the flash.

blockToRead	Contents
0	Manufacturer's data – sonar ID, etc
1	Velocimeter calibration coefficients
2	User data – custom IP address, etc

As the GEM_FLASH_READBACK mechanism has no method of identifying whether the data is due to GEM_ReadFromFlash or a GEM_ReadFlashInfoBlock command, the following are the addresses in the GEM_FLASH_READBACK message which correspond to the flash info block numbers.

blockToRead	Address
0	0x3E0000
1	0x7C0000
2	0x7E0000

```
GEM_ReadFlashInfoBlock(0);
```

GEM_SendProductionTestRS232Command

```
void GEM_SendProductionTestRS232Command(unsigned short
command);
```

This function sends a command to the RS232 logic of the Gemini sonar.

This command is part of the Gemini production test.

```
GEM_SendProductionTestRS232Command(0);
```

GEM_SendProductionTestGPIOCommand

```
void GEM_SendProductionTestGPIOCommand(unsigned short
command);
```

This function sends a command to the GPIO logic of the Gemini sonar.

This command is part of the Gemini production test.

```
GEM_SendProductionTestGPIO(0);
```

GEM_RequestBoardIDs

```
void GEM_RequestBoardIDs(void);
```

This function requests the Gemini sonar to send the identification codes of the electronic assemblies which make up the Gemini sonar. This command results in the Gemini sonar sending a CGemProdData packet, which is returned to the calling program, as a GEM_PROD_DATA message.

```
GEM_RequestBoardIDs();
```

GEM_GetVStringLen

```
int GEM_GetVStringLen(void);
```

This function returns the length of the version string which identifies the DLL.

```
int i = GEM_GetVStringLen();
```

GEM_GetVString

```
void GEM_GetVString(char *data, int len);
```

This function returns the version string which identifies the DLL. The function will return a maximum of len characters in the buffer pointed to by data.

```
int i = GEM_GetVStringLen();
char *p = malloc(i + 1);
GEM_GetVString(p, i);

// p points to memory containing the string
// 'Gemini Comms V1.05.10 SRDUK Ltd.'
```

GEM_GetVNumStringLen

```
int GEM_GetVNumStringLen(void);
```

This function returns the length of the version number string which identifies the DLL.

```
int i = GEM_GetVNumStringLen();
```

GEM_GetVNumString

```
void GEM_GetVNumString(char *data, int len);
```

This function returns the version number string which identifies the DLL. The function will return a maximum of len characters in the buffer pointed to by data.

```
int i = GEM_GetVNumStringLen();
char *p = malloc(i + 1);
GEM_GetVNumString(p, i);

// p points to memory containing the string
// '1.05.10'
```

GEM_InitComPort

```
int GEM_InitComPort(int comPortNo, unsigned short
echoData, unsigned short sendToCalling);
```

This function opens the COM port (specified by comPortNo) on the PC for use by the Gemini software. Any data received from the serial port on the Gemini sonar is received by the code in the DLL, it is sent out through the specified serial port on the PC.

The echoData flag specifies whether or not data received by the Gemini DLL from the Gemini sonar is echoed back to the Gemini sonar by the code in the DLL.

The sendToCalling flag specifies the routing of serial data received from on the serial port of the PC. If the flag is 0, the code in the DLL will send the data received on the serial port to the Gemini sonar. If the flag is any value other than zero, the code in the DLL will send the data up to the calling program using GEM_SERIAL_PORT_INPUT messages into the callback function specified by the calling program.

```
int returnValue;

returnValue = GEM_InitComPort(3, 0, 0);
```

GEM_WriteToComPort

```
int GEM_WriteToComPort(char *data, unsigned short len);
```

This function writes the string of character data (of length len) to the PC's COM port (as opened by the GEM_InitiComPort command).

The value returned by the function is the Error Code (if any) returned by the library call which writes the data to the serial port.

```
int returnValue;

returnValue = GEM_WriteToComPort("Hello", 5);
```

or

```
(void)GEM_WriteToComPort("Hello", 5);
```

Appendix A

The following is the source code for a simple test program which uses the GeminiComms DLL to communicate with a Gemini sonar head. It is beyond the scope of a simple example like this to use the data returned by the head, but it gives a framework which can be used as the basis for software which does look at the sonar data returned.