Tritech
International
Limited

Gemini Software Development Kit

Specification for Gemini Interface DLL

(‘GeminiComms.dll’)

Notes:

Applies to Gemini Sonar VDSL/Ethernet connections.
The ‘GeminiComms.dll’ and relevant header files will be provided with this documentation.

Revision History

Document Rev.**1**, 18th August 2009 spg, Release of full interface specification.
Document Rev.**2**, 21st August 2009 km, Edited for SDK Development release.

Index

Overview

This document describes the DLL which acts as the interface between the Gemini sonar head and other systems, such as Seanet and Gemini software packages. It will provide details of all the DLL functions allowing users to gain programmatic access to the Gemini sonar in their own software. This document will describe all the functionality exposed by the DLL that is relevant for users to write their own control and display programs.

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."`

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	*

* Not required by user. Details not provided in this document.

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 'GeminiStructuresPublic.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.

* This field can be ignored for a basic user interface.

```
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.

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.

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.

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_TAIL
GEM_STATUS
GEM_BEARING_DATA
GEM_UNKNOWN_DATA
PING_DATA
PING_TAIL_EX
GEM_ACKNOWLEDGE
GEM_IP_CHANGED

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
CGemBearingData	Number of lines in the record

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_BEARING_DATA	CGemBearingData	✗
GEM_UNKNOWN_DATA	*Unknown*	✗
GEM_IP_CHANGED	*Empty*	✓

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_StartGeminiNetworkWithResult(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.

Functions

The following functions are defined as the interface of the DLL. The C definitions for these are contained in the file ‘GeminiCommsPublic.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_StartGeminiNetworkWithResult

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

This function, apart from initialising all the communications with the head and getting the DLL into a state ready to operate, 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.

Notes:

"EvoC" is the Mode used by the Gemini software Package.
"SeanetC" is the Mode used by the Seanet Pro software Package.

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_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 data 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_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 wait 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 192.168.1.10, 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_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_SetPingMode

```
void GEM_SetPingMode(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.

pingMethod	Meaning
0	Ping once on receipt of ping configuration message
1	Ping repeatedly at interval fixed by GEM_SetInterPingPeriod

```
GEM_SetPingMode(1);
```

Default value: pingMethod 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_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_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_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

Important:

The ‘Gem_SetMainGain’ function or the ‘GEM_SetMainGainPerCent’ function should be used along with ‘GEM_SetAbsorbGain’, ‘GEM_SetSoftGain’, ‘GEM_SetSpreadGain’ & ‘GEM_SetSoftwareGainRamp’ (all described further below) to manually set the overall Sonar Gain control.

Alternatively, if the ‘GEM_SetAutoPingConfig’ is used then a Gain Percentage is set and the DLL will apply optimised values for all the above manual parameters based on the Gain Percentage value.

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_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_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_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_SendGeminiPingConfig();
```

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_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_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_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

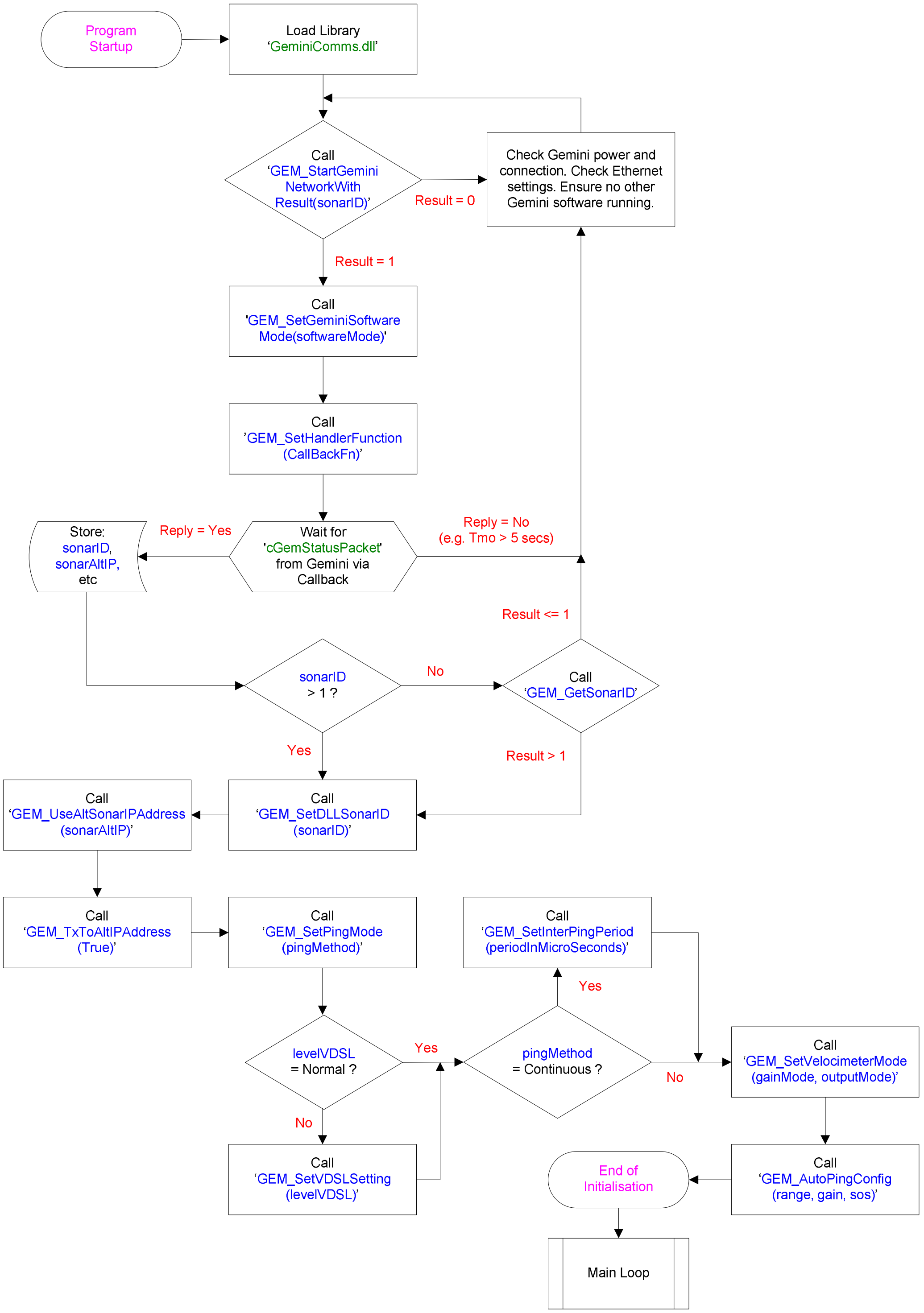
```
void GEM_GetVNumString(char *data, int len);
```

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.

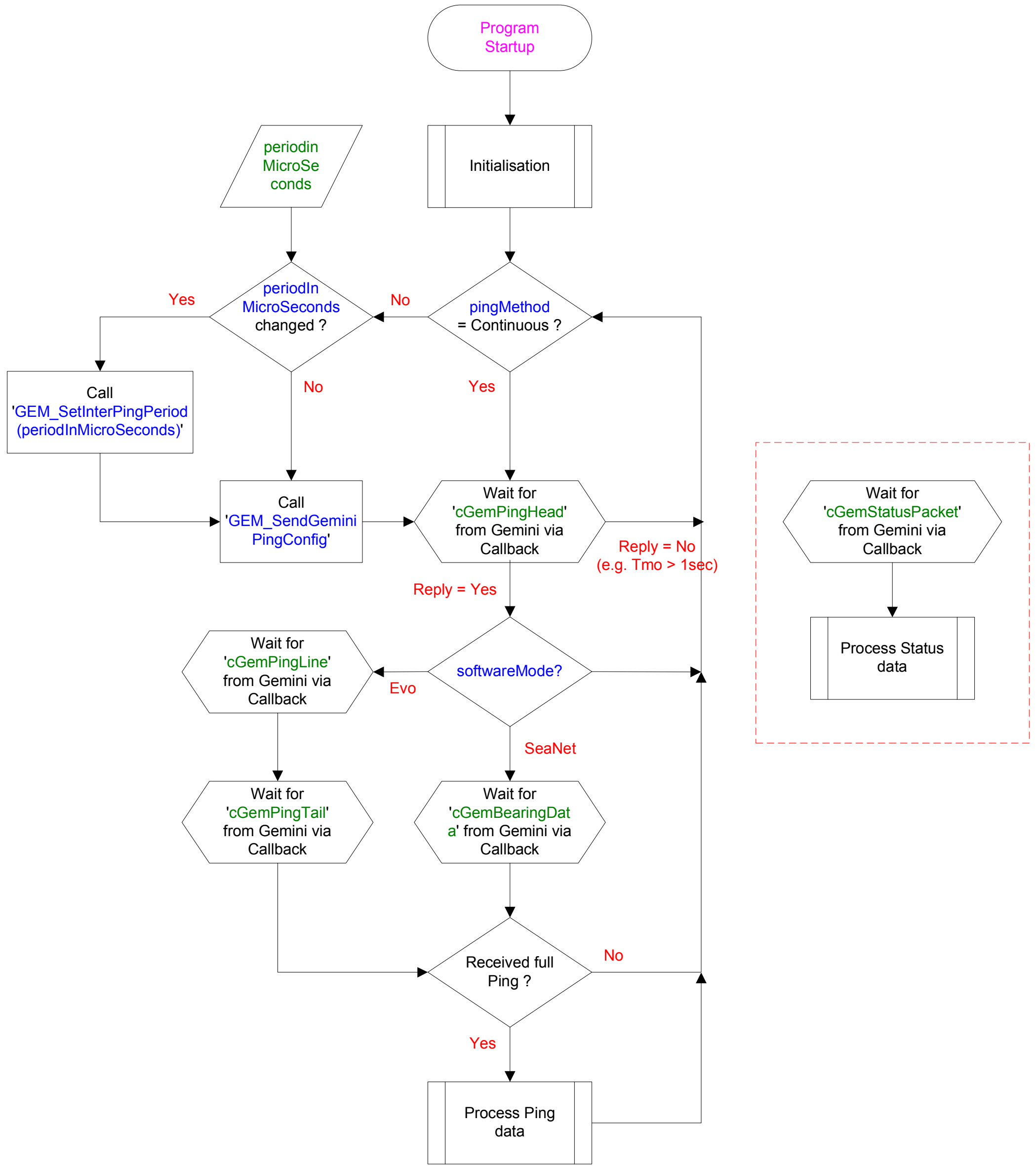
```
int i = GEM_GetVNumStringLen();
char *p = malloc(i + 1);
GEM_GetVNumString(p, i);

// p points to memory containing the string
// ‘1.05.10’
```

Appendix A

Flow Chart – Basic Initialisation routine


Program Startup
Load Library 'GeminiComms.dll'
Call 'GEM_StartGeminiNetworkWithResult(sonarID)'
Result = 0
Check Gemini power and connection. Check Ethernet settings. Ensure no other Gemini software running.
Result = 1
Call 'GEM_SetGeminiSoftwareMode(softwareMode)'
Call 'GEM_SetHandlerFunction(CallBackFn)'
Store: sonarID, sonarAltIP, etc
Reply = Yes
Wait for 'cGemStatusPacket' from Gemini via Callback
Reply = No (e.g. Tmo > 5 secs)
Result <= 1
sonarID > 1 ?
No
Call 'GEM_GetSonarID'
Yes
Result > 1
Call 'GEM_UseAltSonarIPAddress(sonarAltIP)'
Call 'GEM_SetDLLSonarID(sonarID)'
Call 'GEM_TxToAltIPAddress(True)'
Call 'GEM_SetPingMode(pingMethod)'
Call 'GEM_SetInterPingPeriod(periodInMicroSeconds)'
Yes
levelVDSL = Normal ?
Yes
pingMethod = Continuous ?
No
Call 'GEM_SetVelocimeterMode(gainMode, outputMode)'
No
Call 'GEM_SetVDSLSetting(levelVDSL)'
End of Initialisation
Call 'GEM_AutoPingConfig(range, gain, sos)'
Main Loop

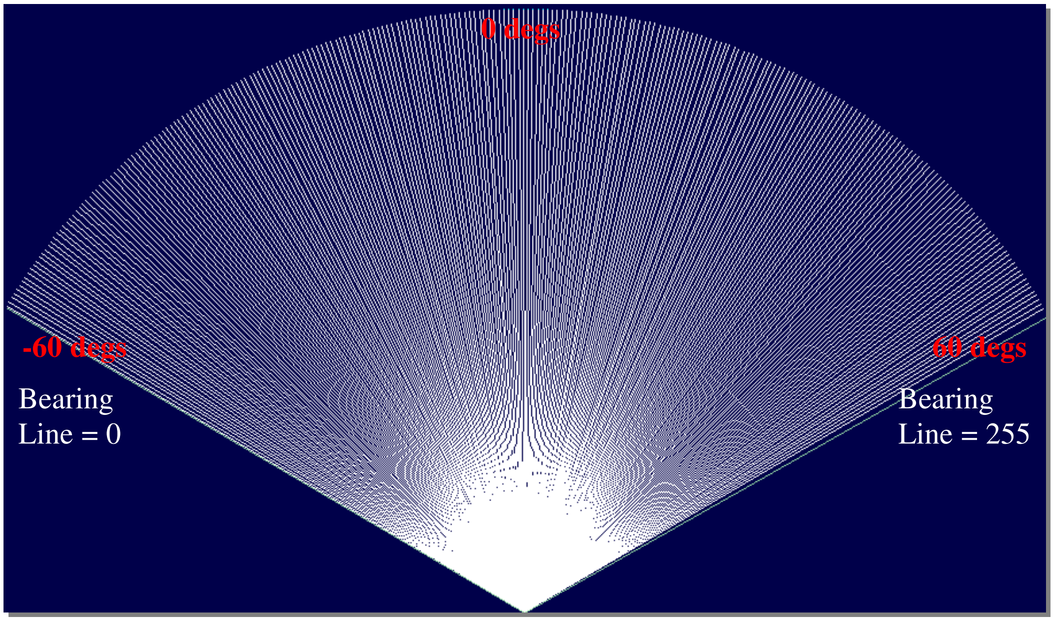
Flow Chart – Main Loop routine


Program Startup
Initialisation
periodin MicroSe conds
periodIn MicroSeconds changed ?
Yes
No
pingMethod = Continuous ?
No
Yes
Call 'GEM_SetInterPingPeriod (periodInMicroSeconds)'
Call 'GEM_SendGemini PingConfig'
Wait for 'cGemPingHead' from Gemini via Callback
Reply = No (e.g. Tmo > 1sec)
Reply = Yes
softwareMode?
Evo
SeaNet
Wait for 'cGemPingLine' from Gemini via Callback
Wait for 'cGemPingTail' from Gemini via Callback
Wait for 'cGemBearingData' from Gemini via Callback
Received full Ping ?
No
Yes
Process Ping data
Wait for 'cGemStatusPacket' from Gemini via Callback
Process Status data

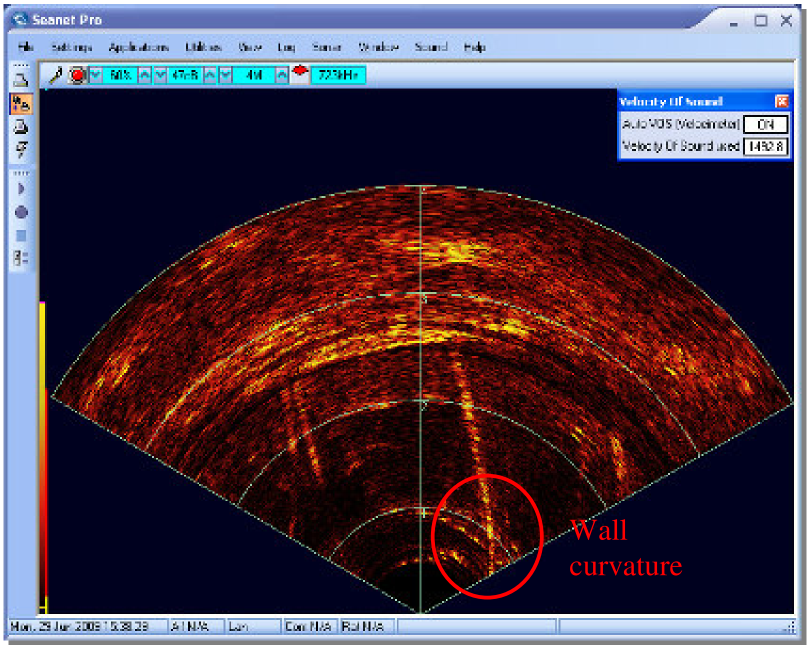
Appendix B

Plotting the Bearing Data (softwareMode = 'SeaNet'/'SeaNetC')

There are 256 bearing lines returned in 'SeaNet'/'SeaNetC' modes. These are plotted over a 120 degree scan sector as follows…



In the above example, the 256 bearing lines are radially spaced equally over the 120 degree sector. This is incorrect and in the test tank example shown below will produce an irregular image with wall curvature in the end beams...



Auto VOS (Velocimeter) ON

Velocity Of Sound used 1482.8

Mon, 29 Jun 2009 15:38:29 | A: N/A | Lan | Con N/A | Rat N/A

The solution…

The bearing lines require to be radially spaced further apart near the edges of the image and to achieve this an Inverse Sin correction needs to be applied. For this correction, a Correction Look-Up table with the 256 bearing positions within the 120 degree scan sector needs to be created.

Bearing Correction Look-Up Table

The following loop and formula is applied:

```
For j := 1 to 256 do begin
  Rad := ArcSin(((2 * j - 256) / 256) * 0.86602540);
  GemBrgTable[j] := Rad * 180 / pi;
end;
```

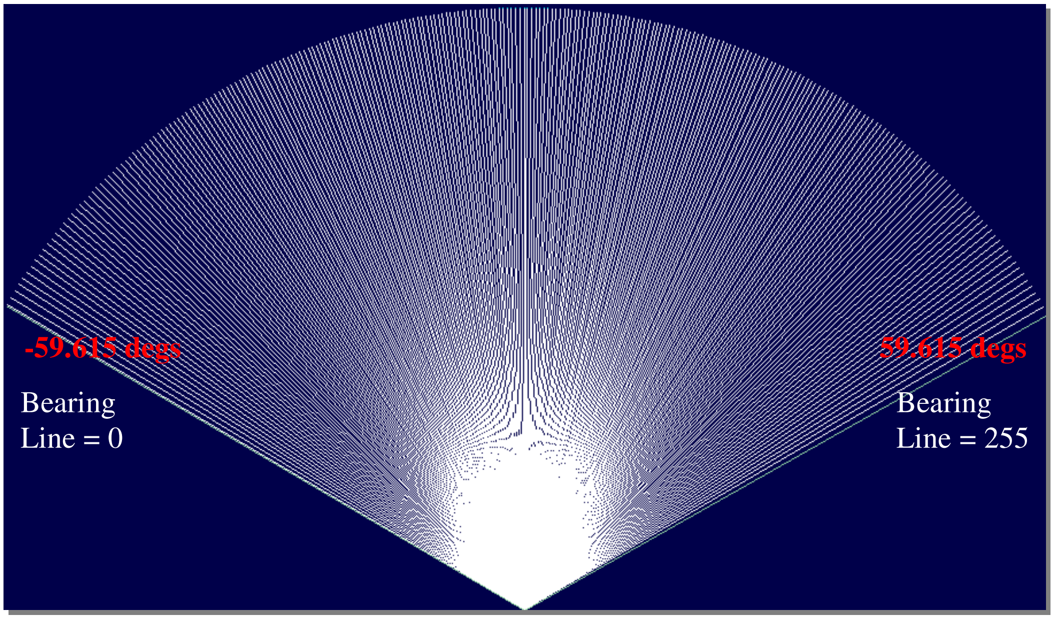
where;

Rad = *Bearing position in radians.*
ArcSin = *Inverse Sin function.*
GemBrgTable[] = *Array of 256 elements with corrected bearing positions for each of the 256 beams.*

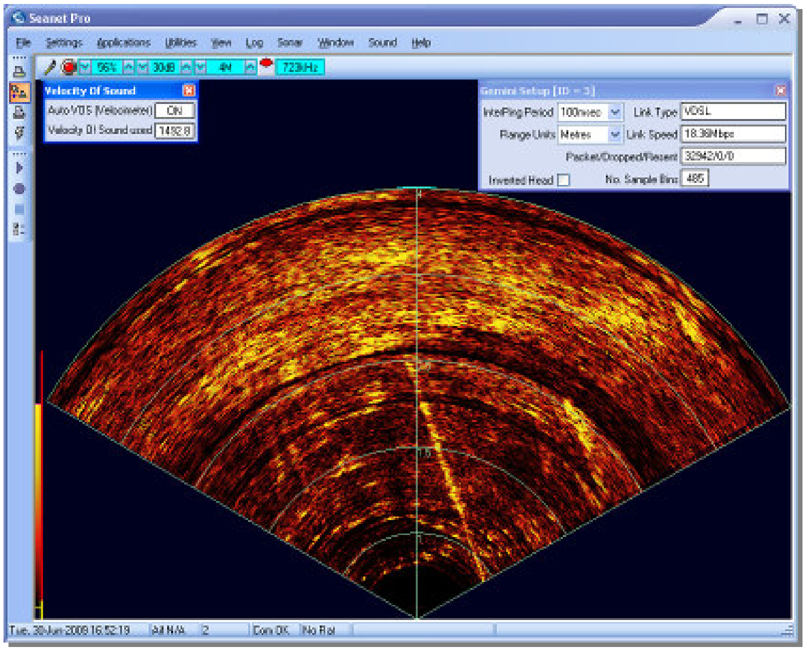
The following 256 element Look-Up table is produced…
(e.g. values have been rounded to 3d.p.)

```
[-59.615, -58.857, -58.115, -57.389, -56.676, -55.977, -55.290, -54.615, -53.951,
 -53.297, -52.654, -52.019, -51.394, -50.777, -50.168, -49.566, -48.972, -48.385,
 -47.805, -47.231, -46.663, -46.101, -45.545, -44.994, -44.448, -43.908, -43.372,
 -42.841, -42.315, -41.792, -41.275, -40.761, -40.251, -39.745, -39.243, -38.744,
 -38.249, -37.757, -37.268, -36.782, -36.299, -35.820, -35.344, -34.869, -34.399,
 -33.931, -33.464, -33.001, -32.540, -32.081, -31.625, -31.170, -30.718, -30.269,
 -29.821, -29.375, -28.931, -28.489, -28.049, -27.611, -27.174, -26.739, -26.306,
 -25.874, -25.444, -25.016, -24.588, -24.163, -23.739, -23.316, -22.894, -22.474,
 -22.055, -21.638, -21.221, -20.806, -20.392, -19.979, -19.567, -19.156, -18.746,
 -18.337, -17.929, -17.522, -17.116, -16.711, -16.307, -15.903, -15.501, -15.099,
 -14.698, -14.297, -13.898, -13.499, -13.100, -12.703, -12.305, -11.909, -11.513,
 -11.118, -10.723, -10.329, -9.935, -9.542, -9.149, -8.756, -8.364, -7.973, -7.581,
 -7.190, -6.799, -6.409, -6.019, -5.630, -5.241, -4.851, -4.463, -4.074, -3.685,
 -3.297, -2.909, -2.521, -2.133, -1.745, -1.357, -0.969, -0.581, -0.194, 0.194,
  0.581, 0.969, 1.357, 1.745, 2.133, 2.521, 2.909, 3.297, 3.685, 4.074, 4.463,
  4.851, 5.241, 5.630, 6.019, 6.409, 6.799, 7.190, 7.581, 7.973, 8.364, 8.756,
  9.149, 9.542, 9.935, 10.329, 10.723, 11.118, 11.513, 11.909, 12.305, 12.703,
  13.100, 13.499, 13.898, 14.297, 14.698, 15.099, 15.501, 15.903, 16.307, 16.711,
  17.116, 17.522, 17.929, 18.337, 18.746, 19.156, 19.567, 19.979, 20.392, 20.806,
  21.221, 21.638, 22.055, 22.474, 22.894, 23.316, 23.739, 24.163, 24.588, 25.016,
  25.444, 25.874, 26.306, 26.739, 27.174, 27.611, 28.049, 28.489, 28.931, 29.375,
  29.821, 30.269, 30.718, 31.170, 31.625, 32.081, 32.539, 33.001, 33.464, 33.930,
  34.399, 34.869, 35.344, 35.820, 36.299, 36.782, 37.268, 37.757, 38.248, 38.744,
  39.243, 39.745, 40.251, 40.761, 41.275, 41.792, 42.314, 42.841, 43.372, 43.908,
  44.448, 44.994, 45.544, 46.101, 46.663, 47.231, 47.805, 48.385, 48.972, 49.566,
  50.168, 50.777, 51.394, 52.019, 52.654, 53.298, 53.951, 54.615, 55.290, 55.977,
  56.676, 57.389, 58.115, 58.857, 59.615]
```

These are plotted over a 120 degree scan sector as follows…



Applying these corrected bearings to the test tank example shown earlier, with the applied bearing corrections the tank wall is now graphically straightened, as shown below…



Auto VOS (Velocimeter) ON

Velocity Of Sound used 1492.8

InterPing Period 100msec

Link Type VDSL

Range Units Metres

Link Speed 18.36Mbps

Packet/Dropped/Resent 32942/0/0

Inverted Head □

No. Sample Bins 485

Tue, 30-Jun-2009 16:52:19 | Alt N/A | 2 | Com OK | No Rot