StarFish

StarFish

Software Development Kit
Programmers Guide

Contents

I. Introduction

Thank you for downloading the Software Development Kit (SDK) for the StarFish range of side-scanning sonar's. This document and the accompanying support files have been put together to help you write your own applications that interface with the StarFish hardware and collect the data back from it.

In summary, this SDK contains…

- This document, describing the API functions available to the programmer in the software library.
- A 'DLL' library file that provides function to create, configure and control instances of a StarFish sonar object.
- Example header files for various programming languages.
- An installer and documentation for the USB drivers required by the software library.
- Various other useful support tools and scripts.

Once you've installed the SDK, you will need to develop the following features in your application …

- Include the USB driver files and DLL in your applications installer.
- Create an instance of a StarFish sonar – this will them provide you with a means to configure basic parameters such as the number of samples to collect, the velocity-of-sound and the sampling range.
- Create a timer or background thread object – this will be used to poll the sonar object at regular intervals and read the collected data out of its buffer.
- Log the data if required – the SDK does not provide any means or support for data logging to a storage medium. There are several documented industry standard formats available on the internet, but perhaps the most widely supported if the XTF (eXtended Triton Format).
- Display the data – the SDK does not include any software components or controls for data visualisation. The sonar objects will return an array of signal data for each sonar channel. The data is presented in a logarithmic format ranging from 0 to 127.5 Decibels (dB's) in 0.5dB steps. Subsequent sections of this document discuss ways of applying gain, contrast and colour mapping algorithms to this.

2. Getting Started

2.1. Requirements

Operating System
This SDK has been developed for use on Microsoft Windows operating systems (from Windows 2000 onwards), and is compiled for 32-bit applications, although it has been successfully tested on 64-bit systems. Currently no support is available for Linux or Mac systems.

Supported StarFish Products
This document, and the accompanying DLL, covers interfacing to the following products…

- StarFish 450 Series Top Box - capable of controlling the StarFish450 and StarFish452 sonar heads.
- StarFish 990 Series Top Box - capable of controlling the StarFisho990 sonar head.

Subsequent sections of this document discuss the basic differences between the various StarFish models.

Background Knowledge
It is assumed that the reader has a reasonable level of programming expertise and is familiar with the basic concepts of the Microsoft Windows operating systems, device drivers, numerical data types (bytes, integers, floating-point numbers, Boolean logic etc), and the principles of the USB bus.
It is also assumed that the reader has a good and basic grasp of the principles of acoustics theory and Sonar operation – further details of this can be found on the StarFish website and in the product user manuals (www.starfishsonar.com).

Technical Support
The contents of this document and SDK pack are provided on an “as is” basis and **only minimal support** can be offered by “StarFish Technical Support” when integrating with your application.

In no circumstances, whether in tort, contract, or otherwise, shall “Tritech International Ltd” or StarFish partner “Blueprint Design Engineering Ltd” accept responsibility whatsoever for any general, special, indirect, consequential, incidental or other damages arising out of the use of information contained in the documentation by other parties, and makes no guarantees, expressed or implied, about its quality, reliability, or any other characteristic.

Please make sure you are using the latest version of the DLL available from the StarFish website.

2.2. Product Range Overview

StarFish towed systems are extremely portable, with each sonar less than 15" long. This enables them to be easily shared between groups of users or vessels when required, and excel in remote shallow water locations where other side scan systems struggle to perform.

All our systems have been specifically designed for shallow water survey in water depths up to 30m/100ft. This makes them ideal for port & harbour survey & security work, inland water survey of rivers/canals/lakes, wreck location and Search and Recovery (SAR) missions.

From the software development aspect, the key features and differences you should be aware of are…

StarFish 450F Our entry level sonar that provides the lowest along-track resolution of the range…

- Uses the '450 series' electronics top box
- **Black** polyurethane moulded rubber
- 450kHz, 400µs CHIRP
- **1.7°** horizontal beam width, 60° vertical beam width
- **100m** typical maximum acoustic range per channel

StarFish 452F Our entry level sonar that provides the lowest along-track resolution of the range…

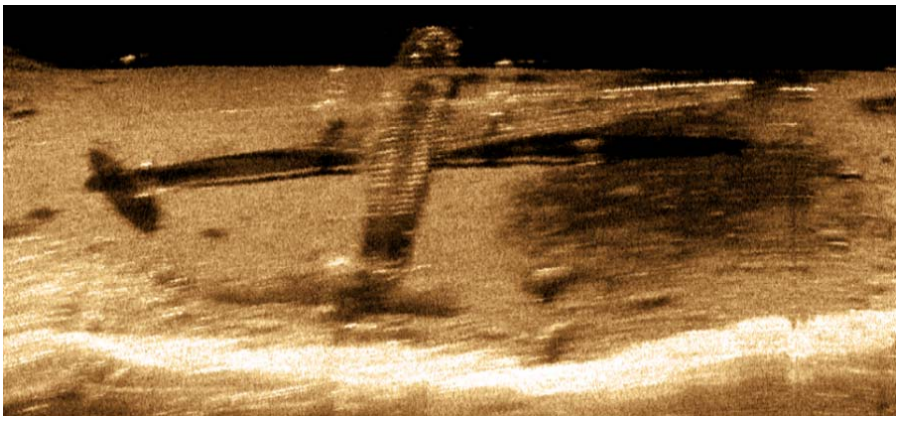
- Uses the '450 series' electronics top box
- **Yellow** polyurethane moulded rubber
- 450kHz, 400µs CHIRP
- **0.8°** horizontal beam width, 60° vertical beam width
- **100m** typical maximum acoustic range per channel

StarFish 990F Designed for search-and-rescue and inshore/port survey applications, the 990 is our highest definition sonar…

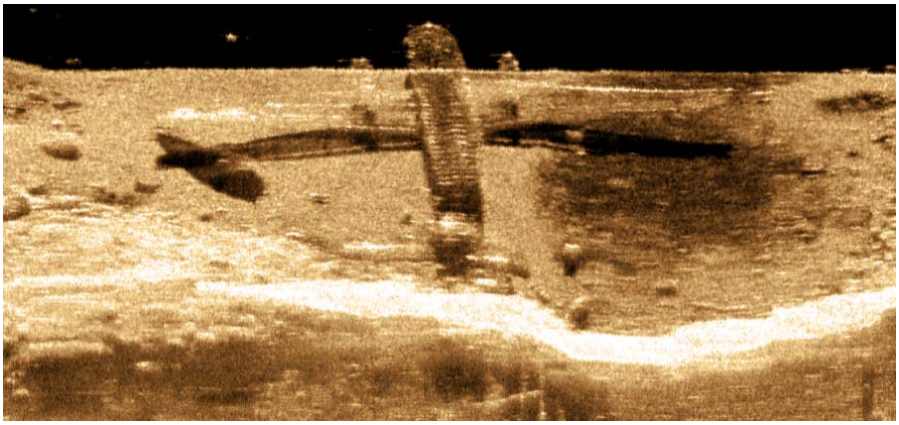
- Uses the '990 series' electronics top box
- **Red** polyurethane moulded rubber
- 1MHz, 100µs CHIRP
- **0.3°** horizontal beam width, 60° vertical beam width
- **35m** typical maximum acoustic range per channel

The images below show a comparison of sonar performance when scanning a sunken aeroplane (in approx 15m of water)…

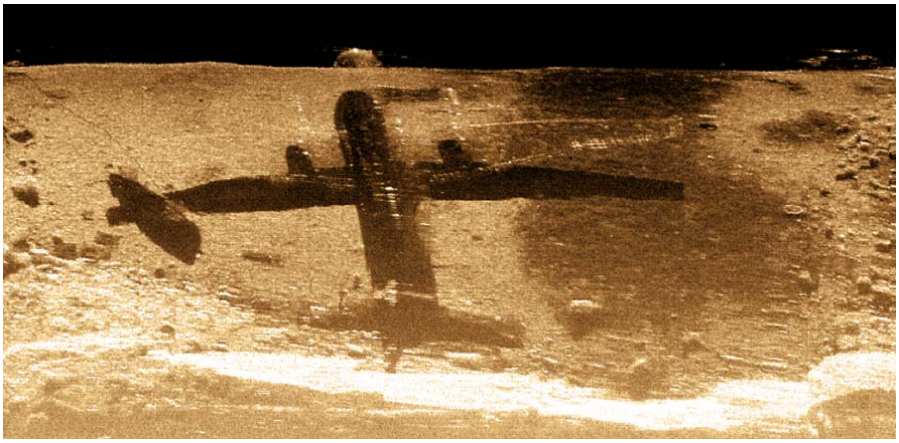
StarFish 450F



StarFish 452F



StarFish 990F



The images also show the wreck of a small boat at the tip of the aircrafts port wing, and the internal structure of the fuselage can be clearly seen.

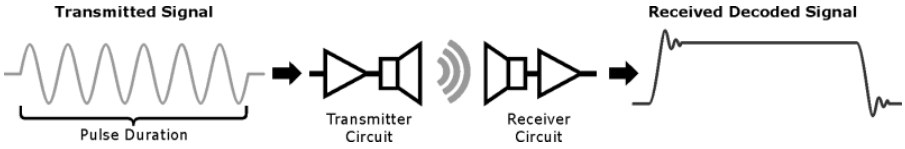
2.3. Sonar Operation

Utilising the latest in advanced CHIRP (**C**ompressed **H**igh **I**ntensity **R**adar **P**ulse) acoustic technology and digital-signal-processing (DSP) techniques at the core of its imagining capabilities, StarFish sonar systems have the ability to detect small closely spaced targets at far greater distances than conventional single frequency, monotonic systems.

By sweeping the acoustic transmission from one frequency to another, the bandwidth of this 'chirped' signal allows closely spaced targets to be imaged individually instead of typically becoming merged into one larger target. CHIRP techniques also help to remove random or out-of-band noise, therefore reducing the risk of acoustic interference.

Traditional Monotonic Operation

The figure below shows the relationship that exists between the transmitted single frequency signal and the output produced by the reflected target echo in the receiver circuitry of the sonar. It can be seen that the receiver does not decode each cycle of the transmitted pulse, but instead produces the envelope of its overall amplitude…



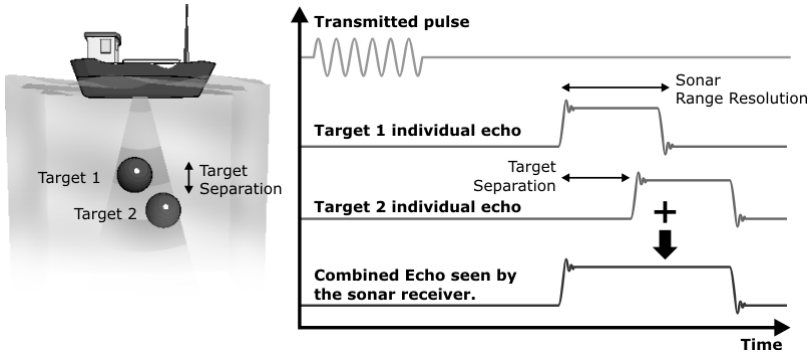
The ability of monotonic acoustic systems to resolve targets is better if the pulse duration is short (so these decoded pulses don't overlap); this, however, has its drawbacks.

Ideally, we need long transmit pulses to get enough acoustic energy into the water for -good target identification of the furthermost targets, but due to the Velocity Of Sound (VOS) through water (typically around 1500 metres/second), each pulse will occupy an equivalent distance related to its pulse duration – this is referred to as "range resolution", and can be found by the following equation…

$$\text{Range Resolution} = \frac{(\text{Pulse Length} \times \text{Velocity of Sound})}{2}$$

Example…
In a monotonic side scan sonar system, the pulse duration is typically 100 micro-seconds, and combining this with the VOS, a range resolution of approximately 75mm is obtained.

The "range resolution" effectively determines the ability of the sonar to identify separate targets; so, using the example above, if two targets are less than 75mm apart then they cannot be distinguished from each other. The net effect is that the system will display a single large combined target, rather than multiple smaller targets, and any fine sonar details are lost, as shown below…

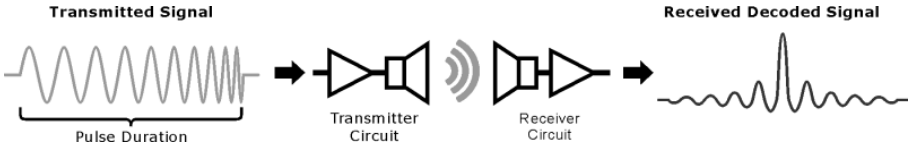


StarFish Chirp Operation

Instead of using a pulse of a single carrier frequency, the frequency within the burst is changed (swept) through the duration of the transmission, from one frequency to another. For the StarFish, at the start of the transmission the sonar operates at 430KHz, and at the end, it reaches 470KHz (giving it a 40KHz bandwidth).

By constantly changing its frequency over time, this "chirped" transmission can be thought of as having a unique acoustic signature, and so if two pulses now overlap (as the targets are closer than the range resolution), we can use the known frequency-versus-time information to tell them apart.

The StarFish sonar receiver contains a pattern-matching circuit that looks for its transmitted Chirp being echoed back from targets, and its receiver now produces a sharp spike when a good match is found (compare this to the monotonic sonar, described previously, that produces an output the same duration as its transmit pulse)…



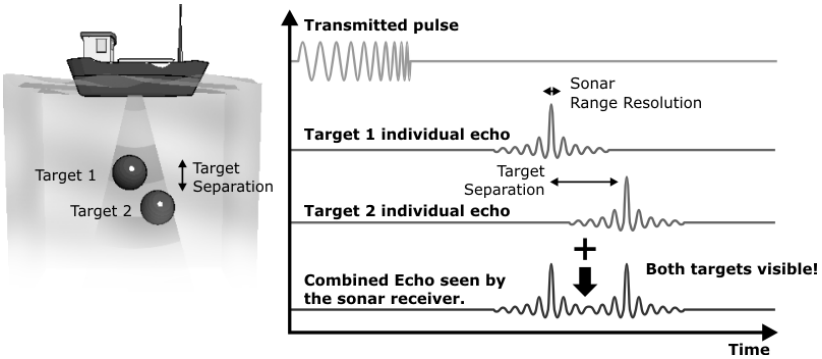
This means that the critical factor in determining range resolution is no longer the pulse length, but the bandwidth of the Chirp, so the range resolution can be found by…

$$\text{Range Resolution} = \frac{\text{Velocity of Sound}}{(\text{Bandwidth} \times 2)}$$

Example…
If bandwidth of the StarFish CHIRP system is 40kHz, and using the same VOS of 1500 metres/second, our new range resolution is approximately 18.75mm... a theoretical improvement by a factor of 4 over the monotonic example above!

This figure below shows that on a chirped sonar, when two acoustic echoes overlap, the Chirp pulses do not merge into a single acoustic return (as their frequency is different from each other at the overlapping points), and the sonar is able to resolve and display the two targets independently.

Therefore, we now can have longer transmissions (and see targets further away) without a loss in resolution; and additionally, Chirp signal processing techniques offer improvements in background noise rejection (as the side scan sonar is only looking for a swept frequency echoes, removing random noise or out-of-band noise).



2.4. USB "FTDI" Driver Installation

The StarFish API provides a set of functions that will initiate a communications link with the hardware, configure the device, issue control commands and decode the received data stream. However, in order to do this, it relies on third party drivers from "Future Technology Devices International" (FTDI) being installed on the target system.

Specifically, the StarFish uses their D2XX driver, now part of their Combined Driver Model (CDM), and available for download from www.ftdichip.com/Drivers/D2XX.htm. At the time of writing, their V2.06.00 driver is recommended as this is Microsoft WHQL certified for 32-bit and 64-bit versions of Windows 7, and available both as a standalone executable or individual files depending on the needs of your applications installer. A copy of their installer has been included in the SDK support files.

Please Note…

Other than the information available via their website, FTDI will provide **no technical support** for the use of their product in the StarFish family of sonar's.

If selecting/downloading future driver releases, you should ensure that they support the "FT2232" USB hardware device.

The StarFish hardware uses the standard FTDI USB "Vendor ID" (VID) and "Product ID" (PID) values of 0x0403 and 0x6010 respectively.

For further information, you may wish to read the following documents available for download from the FTDI website (www.ftdichip.com)...

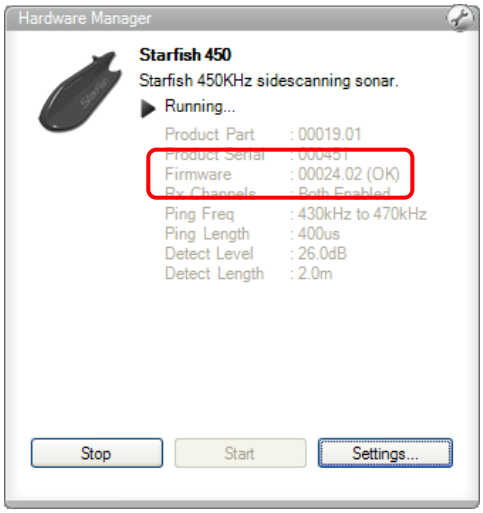
- FT2232D datasheet
- Appropriate "D2XX" drivers for your operating system (version 2.04.16 is recommended as a tested and stable set for Microsoft Windows operating systems).
- FTDI application note "AN232R-04 Windows Combined Driver Model", for details on the driver architecture and individual files.
- Appropriate driver installation guide & release notes for your operating system.
- See FTDI application note "AN_103" for Windows Vista
- See FTDI application note "AN_104" for Windows XP
- "D2XX Programmers Guide" documentation that details the specific function of each of the drivers API functions.

2.5. StarFish Hardware Support

Before continuing, you should check your StarFish module has the latest firmware installed. An easy way to check the firmware status is by running the StarFish Scanline software application, and using the “Hardware Manager” widget.

With the StarFish hardware connected and powered up, check the “Firmware” entry reads “**00024.02**”, where “02” indicates firmware version 2…

If the version number is less than 2, please contact StarFish technical support for the latest firmware files to program using Scanlines “Firmware Upgrade” wizard (available from the StarFish450 hardware settings window).



If the version number is larger than 2, this represents a future release of firmware not covered by this document. Please check the StarFish Technical Library for subsequent releases of this documentation.

2.6. Debugging Tools

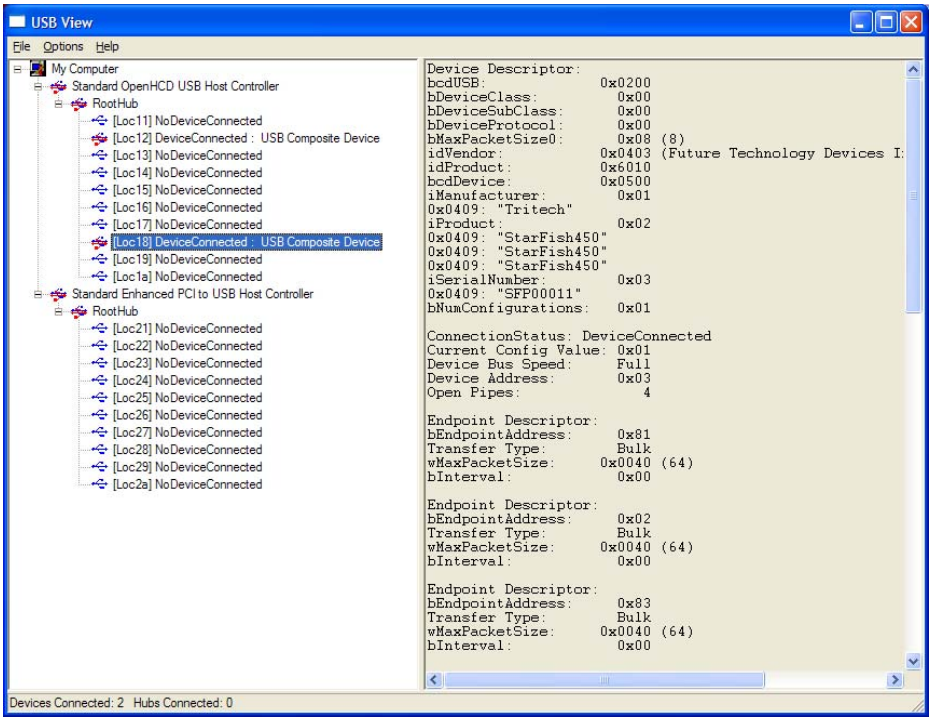
Once the above USB driver has been installed, you should familiarise yourself with the other support tools included with the SDK to help you develop and debug your application…

Show_Windows_Hidden_Devices.bat
This batch file will run the Windows Device Manager with a command line option that allows you to see and change the settings of all hardware devices known to the system, regardless of weather they are currently connected or not. To use the feature, select "Show Hidden Devices" fro the Device Managers "View" menu.

The simplified contents of the "bat" file reads…

```
@echo off
set devmgr_show_nonpresent_devices=1
start devmgmt.msc
pause
exit
```

USBView.exe
USBView is a free utility from Microsoft that displays the USB connection tree and shows the USB devices that are connected to it together with their configuration data. This is very useful for debugging USB enumeration errors.



FTClean.exe
Uninstalling Windows drivers should always be done by selecting to remove the drivers through the "Add/Remove Programs" utility.
It is possible however, to selectively remove devices with a specific VID and PID combination by running the FTDI uninstaller manually with edited INI files containing the desired VID and PID. The FTClean utility provides the user with an easy way of running the uninstaller without having to edit INI files. FTClean generates the required INI files and then runs the uninstaller automatically.

2.7. Other Resources

- A good reference on the subject of the principles and concepts of the USB plug-and-play system is the book “USB Complete: Everything You Need to Develop Custom USB Peripherals” by Jan Axelson (Paperback, 572 pages, 3rd Edition (Jun 2005), ISBN 1-931-44802-7)
- For StarFish Technical support, visit http://www.starfishsonar.com/support/contact.htm.
- For the latest StarFish downloads, visit http://www.starfishsonar.com/support/updates.htm.
- For further information on the D2XX USB driver from FTDI, visit http://www.ftdichip.com/Drivers/D2XX.htm.

3. Using the API Function Library

This section aims to provide the reader with a simple functional overview of the API functions and the sequence in which to call them to correctly configure and control a StarFish sonar.

3.1. Data Modes

When running, the StarFish library will capture and produce data in one of two modes, depending on the requirements of the user application…

- **Sonar Mode** – This is the default mode the sonar assumed, where data is presented in a structure optimised for a small memory footprint, with samples encoded in an array of individual bytes.

  This mode is designed survey and graphical applications that require data logging and graphical display on palette based components.

- **Scope Mode** – Data is formatted into a "waveform" type structure with time and range indices accompanying each floating-point data sample.

  This mode is designed for diagnostic and academic style applications that require an oscilloscope type signal display.

The "SfSetCapture" function is used to specify the current operational mode, and to allow ease of swapping between modes during development and debugging processes, both have their own independent control functions, usually identified within the function names.

For example "SfSetSonarSamples" will specify how many data samples the sonar should collect while in "Sonar" mode, while the "SfSetScope" function is used for "Scope" mode. Similarly "SfGetRxCountSonar" and "SfGetRxDataSonar" are used for reading "Sonar" data, while "SfGetRxCountScope" and "SfGetRxDataScope" are used for "Scope" data.

When in "Sonar" mode, **no** data will be decoded onto the "Scope" buffers, and vice versa with "Scope" mode and the "Sonar" buffer. By default, the library will use "Sonar" mode, and for most applications the user is advised to use this.

3.2. Function Overview

For ease of use, the functions in the library have been grouped into three categories…

- **Creation & Disposal** – discussed in section 7 (from page 32 onwards), these functions cover the basics of creating, managing and disposing of instances of the sonar control objects.
- **Control Functions** – discussed in section 8 (from page 36 onwards), these functions provide the means to open and close communication links, start and stop the sonar, read data, and update settings.
- **Standard Configuration** – discussed in section 9 (from page 44 onwards), these functions allow the user to get and set the status of basic operating parameters (for "Sonar" mode), and read back the collected data.
- **Advanced Configuration** – discussed in section 10 (from page 61 onwards), these function provide greater control over the sonar operation, and not needed for basic operating in "Sonar" mode.

With the exception of a couple of global functions, all sonar control functions return a "SfResult" value that describes if the function executed correctly, or what type of error prevented its successful completion.

When using most of the functions starting "SfSet…", you will need to call an "SfUpdate" function after their execution to apply the changes to the hardware. Only one call to "SfUpdate" is required for many "SfSet…" calls, as all the changes will be applied in one operation.

3.3. Controlling the Sonar & Collecting Data

The general procedure for connecting to and initialising the sonar is…

- First ensure the FTDI USB driver has been installed on the target system, and the "BpStarFish1.dll" file is in the same directory as your application, or the appropriate Windows System directory.
- Optionally, you could now read the library version using the "SfGetLibraryVersion" function. From this you can determine any specific storage or operational requirements of the library, especially if any changes or new requirements have been added since a previous release.
- Create an instance of a sonar object using a call to "SfCreate", specifying the type of sonar you with to create. This will return a "handle" that you will then use with all subsequent function calls to control and configure the sonar.
- Open the communications link with the sonar using a call to "SfOpen". If the function result code isn't "SF_RES_OK" then there is a problem with the USB driver or connection to the sonar hardware.
- Optionally you could retrieve the product information here with a call to "SfGetProductInfo" (to obtain the part numbers and serial number of the sonar hardware and firmware), or the sonar settings and configuration with a call to "SfGetConfig".
- At this point you may want to initialise the default values of the sonar using the following functions…
    - "SfSetSonarRange" – this adjusts in metres the range to collect data over for each channel.
    - "SfSetSonarSamples" – this adjusts the number of data samples to collect on each channel. By default 500 is used.
    - "SfSetSonarVos" – this adjusts the velocity of sound in water, in metres-per-second, and should be used to help accurately calibrate the range settings of the sonar. By default 1475 is used.
- After a call to any of the above "SfSet…" functions, you will need to issue an "SfUpdate" command to apply the changes to the hardware.
- At any point, you can restore the factory defaults for the sonar with a call to "SfDefaults". This is internally called by the library when the sonar instance was created.

Once the sonar has been opened and configured, you can start to collect and process data by…

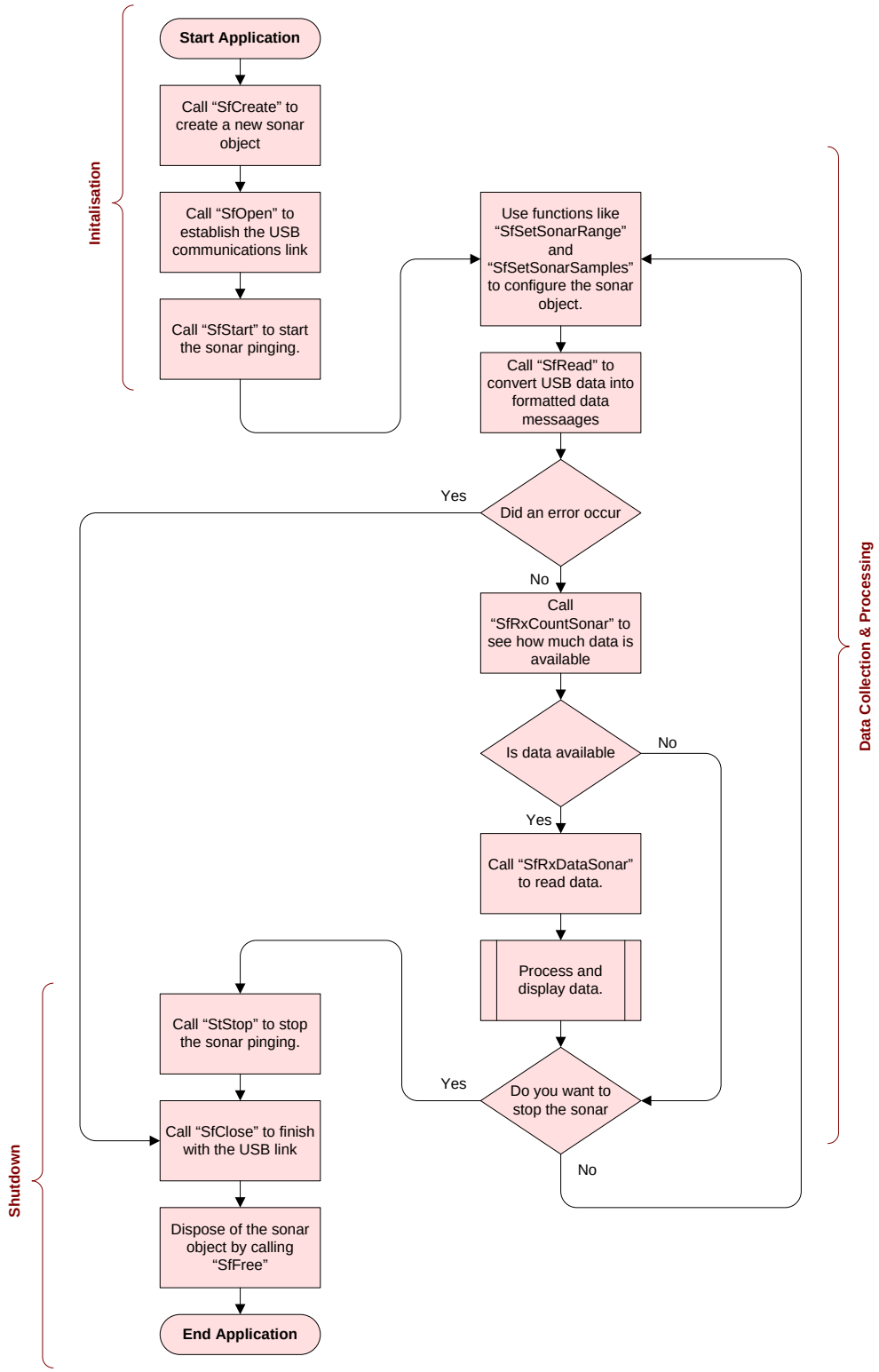
- Call the "SfStart" function to instruct the sonar to start its continuous cycle of transmission and reception. This will then produce a stream of data until the "SfStop" function is called. Alternately, for a single ping use the "SfStartOnce" function.
- Start a timer (or background thread) that will call the "SfRead" function at approximately 50ms to 200ms intervals. This will decode any raw data on the USB buffer into formatted messages and place them on the "Sonar" (or "Scope) FIFO buffer.
- If the "SfRead" function returns a code other than "SF_RES_OK", then you can determine if the sonar has been unplugged or an error has occurred. At this point you should follow the sequence below for closing the sonar.
- Within the timer event, after "SfRead" has been called, use the "SfGetRxCountSonar" (or "SfGetRxCountScope") function to determine how many messages are available for reading from the "Sonar" (or "Scope") buffer.
- If messages are available from the previous functions, call "SfGetRxDataSonar" (or "SfGetRxDataScope") function to read and remove the next message from the FIFO buffer. It is important that all messages are read to clear the buffer otherwise it will become full and no further messages will be added.
- Each data message from the sonar is described in the "SfDataSonar" (or "SfDataScope") structure, you will then need to process and display this data within your application. See the following section for a discussion of applying "Gain" and "Constrast" controls to the data, and palette mapping.
- At any point you can issue commands like "SfSetSonarRange" to adjust operating parameters of the sonar. After these commands, you should issue an "SfUpdate" command (just once) to re-compute internal parameters and apply any changes to the hardware.

Once you have finished collecting data (or an error has occurred), you can shut down the sonar by…

- Issue an "SfStop" command, to abort data collection (if an error has occurred this can be skipped as the sonar should have already stopped).
- Issue an "SfClose" command to shut down the USB communications link to the sonar hardware.

- Stop the applications polling timer (or background task), as no further data will be produced by the sonar.
- Call the "SfFree" function when you are ready to dispose of the resources used by the sonar object. This is usually only done when the application is shutting down.

The flowchart below summarises the sequence of function calls required for basic operation…



4. Displaying Data

Due to the many different programming languages and development environments available, the discussion of developing cascading 'Waterfall' type side-scan sonar displays lies outside the scope of this document. However, whatever type of display is used, at some point it is likely that the sonar data will need to be mapped to a display colour palette, and for the purposes of simplicity an 8-bit colour palette is often chosen (comprising of 256 discrete colours, that can be individually defined and mapped to the sonar signal levels).

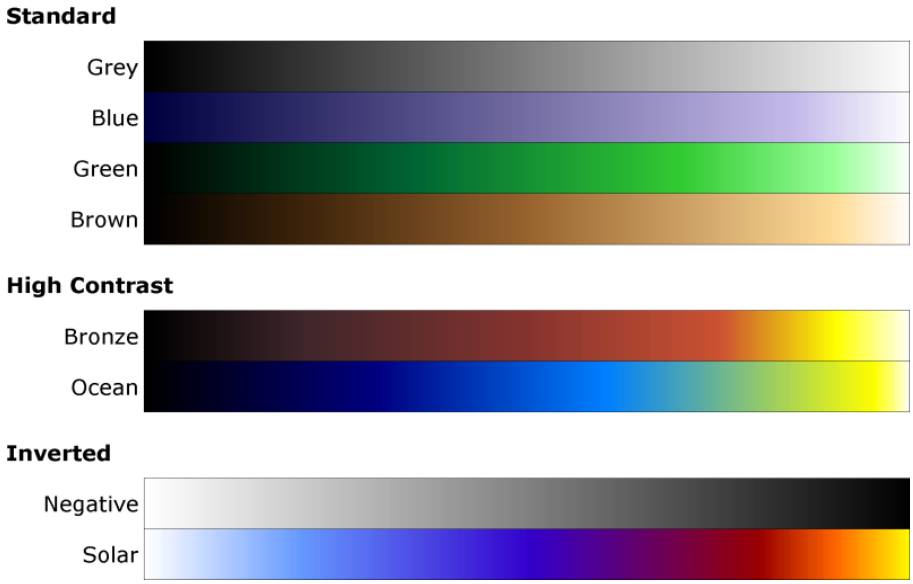
The following sections present the basic theory behind manipulating and mapping the sonar data onto an 8-bit display palette…

4.1. Choosing Palette Colours

The first step is to choose an appropriate swatch of colours. The simplest approach is to map each of the 256 grey shades to a palette index, creating a smooth transition from black to white (i.e. Index 0 = RGB(0,0,0), Index 1 = RGB(1,1,1) etc. to Index 254 = RGB(254,254,254) and Index 255 = RGB(255,255,255).

If needed, these colour mappings can be pre-defined in a look up table that contains the Red, Green and Blue colour components for each palette index. Specific details of how to achieve this can vary significantly between different programming platforms, and it is left up to the implementer to decide the best way to achieve this.

The Scanline software supplied with the StarFish uses several colour palettes, so the user can choose the best suited for their environment and personal preference. Some people prefer 'negative' palettes for daylight viewing, and find green or red ones easier to view at night. Other prefer non-linear palettes (such as the 'ocean' and 'bronze' in the image below), as the Gain and Contrast signal levels can be adjusted to significantly highlight more reflective targets against the background echoes.



4.2. Mapping Signal Levels – Gain & Contrast

Having chosen a palette, the simplest approach is to directly map the sonar data to the palette index, as both have 256 levels. However, this leads to a display that appears very washed-out or over-exposed, as the sonar data very rarely occupies the entire 127dB span available to it.

To make better use of the palette, two software implemented controls are recommended – "Gain" and "Contrast"…

- **Gain** – the purpose of the gain control is to apply an offset (also in Decibels) to the sonar signal, effectively allowing it to be shifted across the palette range. As the sonar signal is also in Decibels, the gain value can be added to each signal value.

- **Contrast** – the purpose of the contrast control is to define the span (or width) of the colour palette in Decibels. In other words, if the black colour index (0) of the palette represented an arbitrary 0dB level, then a contrast of 40dB would mean that the palette index 255 would be used to represent a signal of 40dB, with linear mapping of all in-between palette index's to signal levels. Changing the contrast effectively stretches or compresses the sonar signal against the palette colours.

When your application receives the sonar data, the most effective way to compute the palette index is to use a look up table that maps each 0.5dB discrete signal level from the sonar to a palette index.
It is recommended that you create an array of Bytes with 256 entries (representing signal levels), then for each entry, compute the palette index it maps to, using an algorithm similar to the pseudo code below…

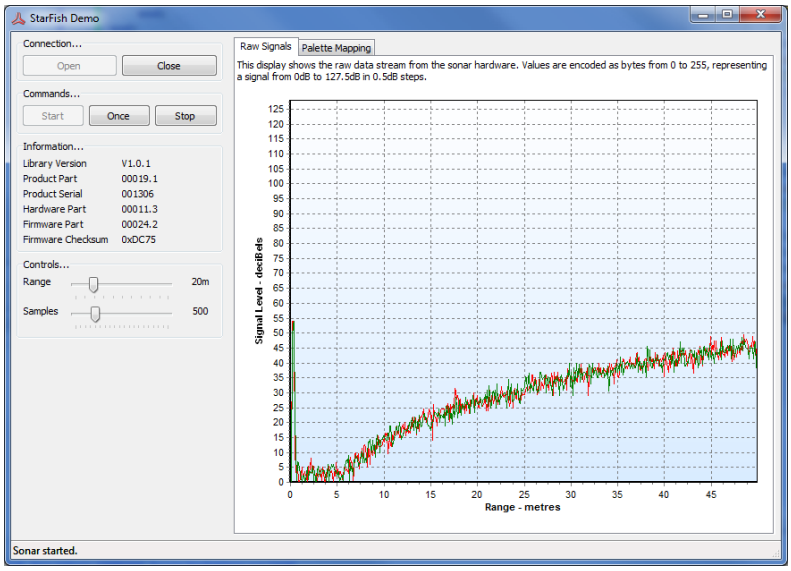
```
//Loop for all incoming signal levels (0 to 127.5dB)
for SignalLevel = 0 to 255
{
        //Compute the normalised signal level, in the interval 0 to 1
        //Gain & Contrast are in dB's, SignalLevel is divided by 2
        //to obtain dB values
        Normalised = ((SignalLevel / 2) + Gain) / Contrast;

        //Clip the signal to the 0 to 1 interval
        Normalised = Max(Min(Normalised, 1), 0);

        //Compute the palette index for each signal level, where 0 is mapped
        //to index 0 and 1 represents 255
        PaletteIndex = Round(Normalised * 255);

        //Store this in the look up table
        MappingTable[SignalLevel] = PaletteIndex;
}
```

When executed, "MappingTable" will contain the palette indices to use for each discrete incoming signal level.

5. Example Application

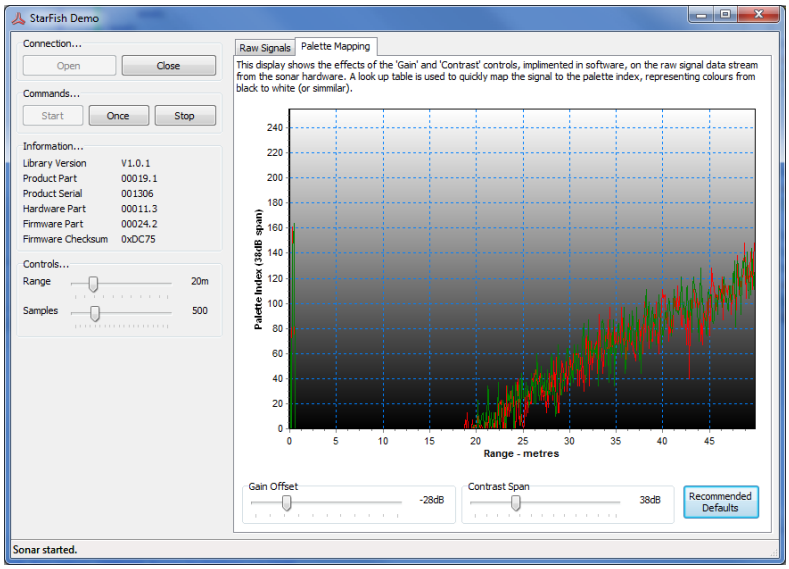
Included in the SDK is the “StarFishDemo.exe” application. The program, and the included source code, demonstrate not only how to programmatically use the SDK library, but also how to process the data using the above “Gain” and “Contrast” control theory.

When you run the program, connect a StarFish 450 series top box and click the “Open” button, then the “Start” button to set the sonar continuously pinging. The raw data will be shown on the “Raw Signals” tab as the figure below shows…



The sonar is being used in “Sonar” mode, and you can use the sliders to control the “Range” and “Samples” parameters.

If you switch to the “Palette Mapping” tab, you will be able to adjust the Gain and Contrast sliders, and see the effect these have on the signal being displayed. The vertical axis of the display chart represent the palette index each signal level is being mapped to, and the background colour of the chart is meant to give an impression of the display colour being used (assuming a greyscale palette). For efficiency, the application uses a mapping look up table to convert the signal level to a palette, and the algorithm behind this can be seen in the “CalculatePalette” function in the source code.



6. Types, Constants & Structures

The following section aims to provide a complete reference of all the types, constants, enumerations and structures (records) used by the API functions (described in subsequent sections). For ease of use, please see the accompanying "header" files included in this SDK for code that can be copied and pasted into your application.

Unless otherwise stated, assume that all multi-byte numerical values (integers, word, doubles etc.) are stored in a 'Little-Endian' format, where the least significant byte value of the entity is at the lowest address. The other bytes follow in increasing order of significance.

As the SDK is written for 32-bit systems, all pointers and other relevant memory access structures should be assumed to be this size.

To describe numerical values in this document, in addition to standard fractional and integer values, hexadecimal notation is also used. Values described this way are prefixed with '0x' to distinguish them from other numerical formats.

The core DLL is written in Delphi, but data types are compatible with unmanaged C++ style type declarations, and for further information please refer to the appropriate MSDN documentation.

6.1. Byte (Value Type)

Represents an unsigned 8-bit integer (**Byte**) value in the binary list, where each value occupies one byte.
Values can lie in the range 0 to 255.

6.2. Double (Value Type)

Represents a double-precision 64-bit number in the binary list, as 8 consecutive bytes. Each value complies with the IEC 60559:1989 (IEEE 754) standard for binary floating-point arithmetic, and is encoded in a 'little endian' format, with the least significant byte being stored first.
Values can lie in the range -1.79769313486232$\times 10^{308}$ to 1.79769313486232e308$\times 10^{308}$, as well as positive or negative zero, Positive-Infinity, Negative-Infinity, and Not-a-Number (NaN).

6.3. Int32 (Value Type)

Represents a 32-bit signed integer value in the binary list, as 4 consecutive bytes. Each value is encoded in a 'little endian' format, with the least significant byte being stored first.
Values can lie in the range -2,147,483,648 (0x80000000) to 2,147,483,647 (0x7FFFFFFF).

6.4. PByte (Pointer)

A pointer value (see "Pointer") referencing a Byte value.

6.5. PDouble (Pointer)

A pointer value (see "Pointer") referencing a Double value.

6.6. PInt32 (Pointer)

A pointer value (see "Pointer") referencing an Int32 value.

6.7. Pointer (Pointer)

A 32-bit value denoting the memory address of the reference object.

6.8. PSfCaptureMode (Pointer)

A pointer value (see "Pointer") referencing an SfCaptureMode value.

6.9. PSfConfig (Pointer)

A pointer value (see "Pointer") referencing an SfConfig structure.

6.10. PSfHandle (Pointer)

A pointer value (see "Pointer") referencing an SfHandle value.

6.11. PSfRxMode (Pointer)

A pointer value (see "Pointer") referencing an SfRxMode value.

6.12. PSfSource (Pointer)

A pointer value (see "Pointer") referencing an SfSource value.

6.13. PSfStatus (Pointer)

A pointer value (see "Pointer") referencing an SfStatus value.

6.14. PSfType (Pointer)

A pointer value (see "Pointer") referencing an SfType value.

6.15. PSfDataSonar (Pointer)

A pointer value (see "Pointer") referencing an SfDataSonar structure.

6.16. PSfDataScope (Pointer)

A pointer value (see "Pointer") referencing an SfDataScope structure.

6.17. PSfProductInfo (Pointer)

A pointer value (see "Pointer") referencing an SfProductInfo structure.

6.18. PUInt32 (Pointer)

A pointer value (see "Pointer") referencing a UInt32 value.

6.19. SfCaptureMode (Enumerated Type)

A Uint32 value that specifies the mode the data capture circuitry in the sonar is operating in…

- SF_CAPTURE_SONAR = 0 — Specifies that the sonar should operate as a sonar, and output data using the least amount of bandwidth/memory.
- SF_CAPTURE_SCOPE = 1 — Specifies that the sonar should operate in a diagnostic mode, allowing the output from different stages in the data processing pipeline to be retrieved with specific timing and range information. See the "SfSource" type for further details.

6.20. SfChannel (Enumerated Type)

A UInt32 value that specifies a channel number, for functions that operate on a single sonar channel…

- SF_CHANNEL_STBD = 0 — Specifies the Starboard (right, with respect to the bow of the towing/mounting vessel) sonar channel.
- SF_CHANNEL_PORT = 1 — Specifies the Port (left, with respect to the bow of the towing/mounting vessel) sonar channel.

6.21. SfConfig (Structure)

The SfConfig structure is used with the "SfGetConfig" function to retrieve all of the sonar and status settings as a single continuous block of data.

Please note that this structure is "packed", meaning that the fields run continuously from one to the next without any byte padding or alignment to specific memory/address boundaries.
The table below lists the individual fields of the structure in the order the library will fill them…

Field Name	*Field Type*	*Comment*
LibVersion	UInt32	Gets the StarFish library version – for further details, see "SfGetLibraryVersion".
SonarType	SfType	Get the type of sonar being controlled.
Status	SfStatus	Get the current status flags for the sonar.
RxMode	SfRxMode	Gets the mode the receiver is currently running in.
CaptureMode	SfCaptureMode	Get the mode that data is currently being captured and output in (i.e. Sonar or Scope)
TxFreqStart	Double	Gets the starting frequency of transmission, in Hertz.
TxFreqEnd	Double	Gets the ending frequency of transmission, in Hertz.
TxFreqCentre	Double	Gets the centre frequency of transmission, in Hertz.
TxFreqBW	Double	Gets the bandwidth of transmission, in Hertz.
TxFreqQ	Double	Gets the Q-Factor (centre ÷ bandwidth) of transmission.
TxLength	Double	Gets the length of the transmission pulse, in seconds.
TxPower	UInt32	Gets the power level of transmission (from 0 to 15).

• RxFreqStart	Double	Gets the starting frequency of reception, in Hertz.
• RxFreqEnd	Double	Gets the ending frequency of reception, in Hertz.
• RxFreqCentre	Double	Gets the centre frequency of reception, in Hertz.
• RxFreqBW	Double	Gets the bandwidth of reception, in Hertz.
• RxFreqQ	Double	Gets the Q-Factor (centre ÷ bandwidth) of reception.
• RxLength	Double	Gets the length of the receiver, in seconds.
• RxGainStbd	Int32	Gets the hardware gain adjustment of the Starboard channel, in Decibels.
• RxGainPort	Int32	Gets the hardware gain adjustment of the Port channel, in Decibels.
• RxOffsetAdcStbd	Int32	Gets the ADC numerical offset of the Starboard channel.
• RxOffsetAdcPort	Int32	Gets the ADC numerical offset of the Port channel.
• RxOffsetRsl	Int32	Gets the Receiver Signal Level offset, in Decibels.
• RxOversample	Double	Gets the receiver oversampling factor (at the centre of reception).
• RxOversampleMin	Double	Gets the receiver oversampling factor (minimum).
• RxOversampleMax	Double	Gets the receiver oversampling factor (maximum).
• TvgAlpha	Double	Gets the absorption loss coefficient of the TVG system.
• TvgAttnMax :	Double	Gets the maximum applied attenuation of the TVG system.
• TvgRangeMax	Double	Gets the range at which not attenuation is applied by the TVG system.
• TvgSamples	UInt32	Gets the number of samples the TVG characteristic is defined in.
• TvgSlope	Double	Gets the slope correction factor of the TVG system.
• AdcFreq	Double	Gets the ADC sampling frequency of the receiver front ends, in Hertz.
• AdcLatch	UInt32	Gets the ADC latch rate.
• FltTaps	UInt32	Gets the number of taps defined in the receiver filters.
• FltTapsPerBlock	UInt32	Gets the number of taps per processing block in the receiver filters.
• FltTapsPerBlockMax	UInt32	Gets the maximum number of taps per processing block available in the receiver filters.
• PingDuration	Double	Gets the duration of the acoustic ping, in seconds.
• PingIntervalScalar	Double	Gets the intervals scalar used to multiply the duration by and determine the gap between pings.
• PingIntervalMin	Double	Gets the minimum inter-ping interval allowed, in seconds.
• PingInterval	Double	Gets the computed and used current ping interval, in seconds.

• ScopeSource	SfSource	Gets the data source used by the capture hardware when the sonar is in “Scope” mode.
• ScopeRange	Double	Get the range to collect data over, in metres, when the sonar is in “Scope” mode.
• ScopeSamples	UInt32	Gets the number of data samples to collect, when the sonar is in “Scope” mode.
• ScopeDelay	Int32	Gets the delay between a transmission start and data collection, in micro-seconds, when the sonar is in “Scope” mode.
• SonarRange	Double	Get the range to collect data over, in metres, when the sonar is in “Sonar” mode.
• SonarSamples	Int32	Gets the number of data samples to collect, when the sonar is in “Sonar” mode.
• SonarVos	Double	Gets the velocity of sound to use when computing the capture timing, from the specified range, in metres-per-second.
• CaptureSource	SfSource	Gets the source of data currently being used, based on the CaptureMode setting.
• CaptureRange	Double	Gets the range the data is currently being collected over, in metres.
• CaptureSamples	UInt32	Gets the number of samples currently being collected.
• CaptureDelay	Int32	Gets the current delay between a transmission start and data collection, in micro-seconds.
• CaptureFreq	Double	Gets the sampling frequency data is being collected at, in Hertz.
• CaptureDuration	Double	Gets the duration it will take to collect all the required data, in seconds.

6.22. SfDataScope (Structure)

Unlike the "SfDataSonar" structure, this structure is designed to provide more useful and accurate acoustic information at the expense of memory usage. The structure is returned by a call to the "SfGetRxDataScope" function, only when the sonar is operating in "Scope" mode.
It is not designed for use with graphical Waterfall type display applications, but with diagnostic or academic applications in mind where oscilloscope-style data collection and analysis is required. Data is grouped by time index rather than channel, and range and timing resolutions are pre-computed for each sample.

Please note that this structure is "packed", meaning that the fields run continuously from one to the next without any byte padding or alignment to specific memory/address boundaries.
The table below lists the individual fields of the structure in the order the library will fill them…

Field Name	*Field Type*	*Comment*
• Source	Byte	Specifies a constant describing which part of the internal signal-path within the electronics module has been used as a source of the data within this structure. The numeric value written to this byte should be the same as defined in SfSource.
• Range	Double	Specifies the overall range (per channel) that data was collected at, in metres.
• Period	Double	Specifies the timing period (in seconds) that data was collected at (i.e. 1 ÷ sampling frequency).
• Offset	Double	Specifies any timing offset (delay), in seconds, from the start of the acoustic ping to when data collection commenced.
• Samples	UInt32	Specifies the number of samples that the hardware collected, and are stored in the Signal array.
• Signal[2048]	Array of SfDataScopeSample	Specifies an array of sample values, each described by the SfDataScopeSample type.

6.23. SfDataScopeSample (Structure)

This structure describes a single sample value comprising part of the SfDataScope structure.

Please note that this structure is "packed", meaning that the fields run continuously from one to the next without any byte padding or alignment to specific memory/address boundaries.
The table below lists the individual fields of the structure in the order the library will fill them…

Field Name	*Field Type*	*Comment*
• Time	Single	The time, in microseconds, the sample represents from the start of the ping (including the offset).
• Range	Single	The target range, in metres, the sample represents. This is not the acoustic range (there and back), but half of that – being the range from the sonar to the target.
• Channel[2]	Array of Single	Array of sample values, in Decibels, for each acoustic channel – see the "SfChannel" type for the mapping of channels to array index's… o Channel[0] = Starboard channel acoustic data. o Channel[1] = Port channel acoustic data.

6.24. SfDataSonar (Structure)

This data structure is used with the "SfGetRxDataSonar" function, to describe the results from a single sonar ping (in the most memory effective way). Unlike the SfDataScope function, data is organised by channel, and accompanied by the range it was collected over. The channel sample resolution can be computed by dividing this range by the number of samples in the channel.

Please note that this structure is "packed", meaning that the fields run continuously from one to the next without any byte padding or alignment to specific memory/address boundaries.
The table below lists the individual fields of the structure in the order the library will fill them…

Field Name	*Field Type*	*Comment*
• Range	Double	The range in metres (per channel) that the data was collected over.
• Channel[2]	Array of SfDataSonarEcho	An array of channels containing the sonar data – see the "SfChannel" type for the mapping of channels to array index's… o Channel[0] = Starboard channel acoustic data o Channel[1] = Port channel acoustic data

6.25. SfDataSonarEcho (Structure)

This structure is used within the SfDataSonar structure to encode data from a single acoustic channel collected during a sonar ping operation.

Please note that this structure is "packed", meaning that the fields run continuously from one to the next without any byte padding or alignment to specific memory/address boundaries.
The table below lists the individual fields of the structure in the order the library will fill them…

Field Name	*Field Type*	*Comment*
• Samples	UInt32	Contains the number of samples collected in the Data array. If the value is 0, then the channel was disabled, and Data will be empty.
• Data[2048]	Array of Byte	Array of the acoustic data collected on a single sonar channel. Only data from array index 0 to 'samples – 1' should be read, as further data may be undefined. To reduce memory usage, data is encoded in 0.5dB steps as a byte rather than single or double precision numbers.

6.26. SfHandle (Value Type)

An Int32 value that is used to indentify which sonar instance the API functions should work with. A handle to a sonar is obtained when an "SfCreate" function is used to initialise a new instance of the sonar.

6.27. SfPartInfo (Structure)

This structure is used to represent the information about a system part (i.e. physical or firmware), and allows the part number and part version to be grouped together.
Please note that this structure is "packed", meaning that the fields run continuously from one to the next without any byte padding or alignment to specific memory/address boundaries.
The table below lists the individual fields of the structure in the order the library will fill them…

Field Name	*Field Type*	*Comment*
• Number	UInt16	Specifies the part number of the item the structure is referring to.
• Version	Byte	Specifies the version (or revision) of the part number the structure is referring to.

Typically this structure will be displayed in string form as "<number>.<version>", with appropriate zero-padding preceding the number to make it up to 5 digits – i.e. "00123.4".

6.28. SfProductInfo (Structure)

The SfProductInfo structure is returned by the "SfGetProductInfo" function, and contains an overview of the hardware and firmware used in the sonar electronics module. This can be used to identify if firmware is outdated or any specific configuration considerations to apply.

Please note that this structure is "packed", meaning that the fields run continuously from one to the next without any byte padding or alignment to specific memory/address boundaries.
The table below lists the individual fields of the structure in the order the library will fill them…

Field Name	*Field Type*	*Comment*
• ProductPart	SfPartInfo	Specifies the overall part number of the electronics module used by the sonar. Examples would be "00011.3" for a 450 series top box, or "00205.3" for a 990 series top box.
• ProductSerial	UInt32	Specifies the serial number of the part – when displayed serial numbers are usually zero-padded to be 6 digits long.
• HardwarePart	SfPartInfo	Specifies the part number of the circuit board within the sonar electronics module.
• FirmwarePart	SfPartInfo	Specifies the part number and version of the firmware currently programmed onto the hardware.
• FirmwareChecksum	UInt16	Specifies the checksum of the firmware.

6.29. SfResult (Enumerated Type)

A UInt32 value that contains a result code from the API functions…

- SF_RES_OK = 0 — Returned if the function completes successfully.
- SF_RES_STATE_ERROR = 1 — Returned if the function cannot complete as the sonar is in an incorrect state to allow it to execute (i.e. not opened).
- SF_RES_COMMS_ERROR = 2 — Returned if the function cannot complete as a USB communications error has occurred. When this code is returned it is most likely the sonar will have also closed the port, and will need re-opening to initialise the communications link again.
- SF_RES_LIBRARY_ERROR = 3 — Returned if the function cannot complete as an error with the USB driver library has occurred.
- SF_RES_SYSTEM_ERROR = 4 — Returned if the function cannot complete as a general system error (not covered by any of the other errors) has occurred. When this code is returned it is most likely the sonar will have also closed the port, and will need re-opening to initialise the communications link again.
- SF_RES_INVALID_HANDLE = 16 — Returned when the specified sonar handle is invalid.
- SF_RES_INVALID_CHANNEL = 17 — Returned when the specified sonar channel is invalid.
- SF_RES_NO_DATA = 18 — Returned in read operations where do data is present on the buffer.

6.30. SfRxMode (Enumerated Type)

A Uint32 value that specifies how the sonar receiver is operating…

- SF_RX_MANUAL = 0 — The receiver is in manual mode – this mode allows the receiver parameters to be adjusted manually and independently of the transmitter. However, a detailed discussion of this mode lies outside the scope of this document.
- SF_RX_STANDARD = 1 — The receiver is operating in standard sampling mode.
- SF_RX_SUBSAMPLE = 2 — The receiver is sub-sampling – this mode is not intended for normal sonar use, and lies outside the scope of this document.

6.31. SfSource (Enumerated Type)

A Uint32 value that specifies which section in the internal data pipeline the sonar's capture module should collect data from…

- SF_SOURCE_WAVE = 0 — Data values represent the ADC waveform (with numerical offset applied), in signed 16-bit format, captured at the input sample rate.
- SF_SOURCE_FILTER_RAW = 1 — Data values represent raw filtered logarithmic data, in an unsigned 16-bit "8.8" integer/fractional binary format, captured at the input sample rate.
- SF_SOURCE_FILTER_GAIN = 2 — Data values represent gain compensated filtered logarithmic data, in an unsigned 16-bit "8.8" integer/fractional binary format, captured at the input sample rate.
- SF_SOURCE_GAIN = 3 — Data values represent the TVG and Fixed Gain characteristic. Values are in Decibels (dB's), encoded in an unsigned 16-bit "8.8" integer/fractional binary format, sampled at the output sample rate.
- SF_SOURCE_RSL_LONG = 4 — Data values represent the received signal level (long) data, in dB's, encoded as an unsigned 16-bit "8.8" integer/fractional binary format, sampled at the output sample rate.
- SF_SOURCE_RSL_SHORT = 5 — Data values represent the received signal level (short) data, in dB's, encoded as an unsigned 8-bit "7.1" integer/fractional binary format, sampled at the output sample rate (representing values from 0dB to 127.5dB).

NB: In the case of the SfDataScope structure, the 'Source' field uses the above values, but is only a single byte in length.

6.32. SfStatus (Enumerated Type)

A Uint32 value that contains flags indicating the status of the sonar. Bit values are defined as…

- Bit 0, SF_STATUS_OPEN — True when the communications port with the sonar is open.
- Bit 1, SF_STATUS_ACTIVE — True when the sonar is actively pinging (does not get set for single ping operations).
- Bit 8, SF_STATUS_TXEN — True if the sonar's transmitter is enabled.
- Bit 9, SF_STATUS_TXMUTE — True is the sonar is muting the receiver during transmission.
- Bit 10, SF_STATUS_TVGEN — True if the sonar's TVG system is enabled.
- Bit 11, SF_STATUS_RXEN_STBD — True if the sonar's starboard channel receiver is enabled.
- Bit 12, SF_STATUS_RXEN_PORT — True if the sonar's port channel receiver is enabled.

6.33. SfType (Enumerated Type)

A Uint32 value that specifies the type of sonar hardware and presets. Valid values are…

- SF_TYPE_450 = 1 — The sonar is a 450 series StarFish, covering the following models…
  - StarFish450F
  - StarFish450H
  - StarFish452F
- SF_TYPE_990 = 2 — The sonar is a 990 series StarFish, covering the following models…
  - StarFish990F

6.34. UInt32 (Value Type)

Represents a 32-bit unsigned integer value in the binary list, as 4 consecutive bytes. Each value is encoded in a 'little endian' format, with the least significant byte being stored first.
Values can lie in the range 0 (0x00000000) to 4,294,967,295 (0xFFFFFFFF).

7. Sonar Creation & Disposal Functions

These functions have been grouped together for documentation purposes as they all relate to the creation, management and disposal of the sonar interface objects…

7.1. SfCreate

Description…
This function creates a new instance of a sonar object in the API's sonar manage, and returns the handle to it.

The library maintains an internal list of sonar objects, and handles all the creation and destruction of them. Before any other library function can operate on a sonar, the application must first call this function to create an object and get its handle.
When the application is shutting down, it should then call the "SfFree" function to release the resources used by the sonar object.

Parameters & Result…

Name	*Type*	*Description*
stype	SfType	Specifies a value that determines the type of sonar object to create. Valid values are.. • SF_TYPE_450, to create a StarFish450 or StarFish452 sonar object. • SF_TYPE_990, to create an StarFish990 sonar object.
son	PSfHandle	Pointer to a value that will receive the handle of the sonar object created. Other functions will then need to use this value in order to access and control the sonar. The handle is the zero-based index of the created sonar object in the internal management list of the library. If an error occurs, the handle will return a value of -1.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_LIBRARY_ERROR, the starfish libraries are not loaded properly and cannot perform the specified function. • SF_SYSTEM_ERROR, an unknown exception occurred while trying to create the sonar.

7.2. SfCreated

Description…
This function can be called to find out how many sonar objects have been created, and are currently being managed, but the StarFish library.

Parameters & Result…

Name	*Type*	*Description*
result	UInt32	The function returns the number of sonar objects that have been created and its currently managing.

7.3. SfFree

Description…
This function should be called to dispose of a sonar object when it is no longer needed, and free any resources it may be using. If the USB communications link is open and the sonar is active when this function is called, then the sonar will be gracefully shut down before being disposed off.

When the StarFish library is unloaded from memory, it will automatically perform this function on any sonar objects that are still created.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

7.4. SfGetLibraryVersion

Description…
This function should be called to retrieve to version of the StarFish library. The application can use this value to determine function support (compare to previous library versions) and memory structure definitions.

Parameters & Result…

Name	*Type*	*Description*
result	UInt32	The functions return value, containing the library version encoded into the lower 24-bits of the number. The lowest byte contains the revision, followed by the minor version in the next 8, and the major version in the next after that… `0x00<major><minor><revision>` For example, a value 0x00010203 would translate to V1.2.3

8. Sonar Control Functions

These functions have been grouped together for documentation purposes as they are all used to issue commands that control instances of the sonar objects created with the “SfCreate” function.

8.1. SfClose

Description…

This function manually closes the USB communications port to the StarFish hardware, and can be used as part of a manual disconnection procedure.

If the sonar is active when the “SfClose” function is called, the hardware will first be stopped then the communications link disconnected.

Please note this function does not dispose of or free up any of the resources used by the sonar object, to do this a call should be made to the “SfFree” function.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the “SfCreate”). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

8.2. SfDefaults

Description…

Call this function to initialise the sonar object back to its factory default settings. When the object is first created it will be initialised with these defaults, but if you change them then wish to undo the changes, call this function.

If the sonar isn't currently connected, this function will still adjust the local (PC) settings, which will then be synchronised with the hardware when it next connects.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

8.3. SfOpen

Description…

This function opens the USB communications port to the StarFish hardware. Until this function is called any settings changes that are made through the other functions will only be applied locally to the sonar object in memory. Calling the function will synchronise the settings with the hardware and allow any further changes to be applied in real-time.
To close the USB communications link, use the "SfClose" function.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed. • SF_STATE_ERROR, the sonar is in an incorrect state to allow it to execute (i.e. not opened). • SF_COMMS_ERROR, an error occurred with the USB communications, and the sonar object has been put into a closed state. • SF_SYSTEM_ERROR

8.4. SfRead

Description…
This function instructs the sonar to process all the pending data on the USB communications buffers (in the FTDI USB driver), and then convert and format it into messages available for reading from the DLL. Normally an interval timer or background thread will be used to periodically call this function (at approx 50ms to 100ms intervals). The total number of messages decoded is returned in the "msgs" parameter.

As mentioned earlier, the sonar can be either instructed to operate in "Sonar" or "Scope" mode by the "CaptureMode" property (see the "SfGetCapture" and "SfSetCapture" functions). Depending on the capture mode, completed data messages are placed on only one of the two available thread-safe receiver FIFO buffers. After the "SfRead" function is executed, the application can then use either the "SfGetRxCountSonar" or "SfGetRxCountScope" functions to determine if there are pending messages available, and the "SfGetRxDataSonar" or "SfGetRxDataScope" function to retrieve the data.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
msgs	PUInt32	Value receives the number of messages that were decoded by the read command.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

8.5. SfStop

Description…
This function is used to stop the sonar from pinging if the "SfStart" command has been issued. The USB link is kept in the open state, but if the "SfClose" command is called, then it will internally call "SfStop" before closing the USB link.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed. • SF_STATE_ERROR, the sonar is in an incorrect state to allow it to execute (i.e. not opened). • SF_COMMS_ERROR, an error occurred with the USB communications, and the sonar object has been put into a closed state. • SF_SYSTEM_ERROR

8.6. SfStartOnce

Description…
This function instructs the sonar to perform a single ping (transmit and receive cycle), using the current settings. If the "SfStart" command has previously been issued then the sonar will issue a self "SfStop" before performing the ping.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed. • SF_STATE_ERROR, the sonar is in an incorrect state to allow it to execute (i.e. not opened). • SF_COMMS_ERROR, an error occurred with the USB communications, and the sonar object has been put into a closed state. • SF_SYSTEM_ERROR

8.7. SfStart

Description…

This function is used to start the sonar continuously pinging (transmitting and receiving data). The USB communications port should fist have been opened with the "SfOpen" command before issuing "SfStart". Any other settings, such a samples, velocity of sound etc. should also have been setup before starting pinging, although most parameter commands, such as Range, can adjusted while the sonar is pinging.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed. • SF_STATE_ERROR, the sonar is in an incorrect state to allow it to execute (i.e. not opened). • SF_COMMS_ERROR, an error occurred with the USB communications, and the sonar object has been put into a closed state. • SF_SYSTEM_ERROR

8.8. SfUpdate

Description…
The update function is used to synchronise any settings between the sonar object in the library with the StarFish hardware. For the command to be successful the hardware should be connected and an "SfOpen" command successfully issued.

In addition to synchronisation of settings, the "SfUpdate" function should also be used to apply any new settings (or batch of settings) that may have changed, such as range or velocity-of-sound.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed. • SF_STATE_ERROR, the sonar is in an incorrect state to allow it to execute (i.e. not opened). • SF_COMMS_ERROR, an error occurred with the USB communications, and the sonar object has been put into a closed state. • SF_SYSTEM_ERROR

9. Standard Configuration & Status Functions

These functions have been grouped together for documentation purposes as they are all used to adjust the most commonly used settings that control the sonar's function.

9.1. SfGetChannelEn

Description…
This function is used to retrieve the enabled state of a specified channel. When a channel is disabled, no data will be produced for it, although its transmitter will remain enabled.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
channel	SfChannel	A value specifying which channel should have its enabled status queried by this function, possible values are… • SF_CHAN_STBD, to select the Starboard channel. • SF_CHAN_PORT, to select the Port channel.
enable	PByte	Pointer to a Byte value that will receive a non-zero value if the specified channel is enabled.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed. • SF_INVALID_CHANNEL, the channel parameter is invalid.

9.2. SfGetConfig

Description…
This function is used to retrieve a memory structure that contains all the parameters for the current sonar configuration, including transmitter and receiver setup, TVG operating parameters and data collection settings.

Most of the information in the "SfConfig" structure can be read individually through the other functions described in this document. For further details on the "SfConfig" structure, see section **Error! Reference source not found.**.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
cfg	PSfConfig	Pointer to an "SfConfig" structure, that will receive the current configuration and status values.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

9.3. SfGetOpen

Description…
This function is used to query the current "open" state of the sonar – i.e. if the "SfOpen" command has been successfully used to establish a communications link to the hardware.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
value	PByte	Pointer to a Byte value that will receive a non-zero value if the sonar communications link is open.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

9.4. SfGetProductInfo

Description…
This function can be used to retrieve information about the products identity (part number, revision and serial number), as well as the firmware information.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
info	PSfProductInfo	Pointer to an "SfProductInfo" structure that will receive the details of the products hardware, serial number and firmware.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

9.5. SfGetRxCountScope

Description…
This function should be used after calls to the "SfRead" function (that polls and decodes any pending USB data) when the sonar has been set into "Scope" mode (see the "SfGetCapture" and "SfSetCapture" functions). Completed data messages will have been placed onto the Scope FIFO, and this function will return how many messages and pending reading.
Use the "SfGetRxDataScope" function to read (and remove) the next available message from the Scope FIFO buffer.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
count	PUInt32	Pointer to an unsigned integer value that receives the number of messages on the Scope FIFO that are available for reading.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

9.6. SfGetRxCountSonar

Description…
This function should be used after calls to the "SfRead" function (that polls and decodes any pending USB data) when the sonar has been set into "Sonar" mode (which it assumes by default, unless changed by the "SfGetCapture" and "SfSetCapture" functions). Completed data messages will have been placed onto the Sonar FIFO, and this function will return how many messages and pending reading.
Use the "SfGetRxDataSonar" function to read (and remove) the next available message from the Sonar FIFO buffer.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
count	PUInt32	Pointer to an unsigned integer value that receives the number of messages on the Sonar FIFO that are available for reading.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

9.7. SfGetRxDataScope

Description…
Once the sonar has been opened with "SfOpen", and the "SfStart" command issued (or "SfStartOnce"), the "SfRead" function should be called periodically to convert any accumulated raw data on the USB buffers into formatted data messages.

As mentioned in section 3.1, the sonar can be either instructed to operate in "Sonar" or "Scope" mode by the "CaptureMode" property (see the "SfGetCapture" and "SfSetCapture" functions). Depending on the capture mode, completed data messages are placed on only one of the two available thread-safe receiver FIFO buffers.
If the sonar is operating in "Scope" mode, then this function will remove and return the next data message from the Scope FIFO buffer. If no data is available on the FIFO, this function will not modify any memory pointed to by the "data" parameter and an "SF_RES_NO_DATA" value will be returned from the function – use the "SfGetRxCountScope" function prior to this function to determine is data is available to be read.

The FIFO has been designed with a memory locking mechanism that allows it to be thread safe, so the calling application can use a background thread to read the FIFO if required without having to worry about the DLL.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
data	PSfDataScope	Pointer to an "SfDataScope" structure that will be filled with the next Scope message from the receiver FIFO, if available.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed. • SF_RES_NO_DATA, there was no data in the FIFO available for reading.

9.8. SfGetRxDataSonar

Description…
Once the sonar has been opened with "SfOpen", and the "SfStart" command issued (or "SfStartOnce"), the "SfRead" function should be called periodically to convert any accumulated raw data on the USB buffers into formatted data messages.

As mentioned in section 3.1, the sonar can be either instructed to operate in "Sonar" or "Scope" mode by the "CaptureMode" property (see the "SfGetCapture" and "SfSetCapture" functions). Depending on the capture mode, completed data messages are placed on only one of the two available thread-safe receiver FIFO buffers.
If the sonar is operating in "Sonar" mode (the default setting), then this function will remove and return the next data message from the Sonar FIFO buffer. If no data is available on the FIFO, this function will not modify any memory pointed to by the "data" parameter and an "SF_RES_NO_DATA" value will be returned from the function – use the "SfGetRxCountSonar" function prior to this function to determine is data is available to be read.

The FIFO has been designed with a memory locking mechanism that allows it to be thread safe, so the calling application can use a background thread to read the FIFO if required without having to worry about the DLL.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
data	PSfDataSonar	Pointer to an "SfDataSonar" structure that will be filled with the next Sonar message from the receiver FIFO, if available.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed. • SF_RES_NO_DATA, there was no data in the FIFO available for reading.

9.9. SfGetSonarRange

Description…
This function is used to retrieve the current range settings of the sonar, when CaptureMode is set to "Sonar" (see "SfGetCapture" and "SfSetCapture", and section 3.1 for further details).

Please note that the range setting should be used in conjunction with the velocity-of-sound setting to achieve properly calibrated readings. Both values are used to derive the sampling rate of the sonar.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
range	PDouble	Pointer to a Double value that will receive the current sonar range, in metres.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

9.10. SfGetSonarSamples

Description…
Function that will get the number of data samples the sonar will collect on each channel over the specified range, when CaptureMode is set to "Sonar" (see "SfGetCapture" and "SfSetCapture", and section 3.1 for further details).

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
samples	PUInt32	Pointer to a UInt32 value that will receive the number of samples the sonar will collect on each channel.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

9.11. SfGetSonarType

Description…
Function that returns the type of sonar that the sonar object is currently configured to control.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
stype	PSfType	Pointer to a SfType value that will receive the sonar type, returned values are… • SF_TYPE_450 • SF_TYPE_990
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

9.12. SfGetSonarVos

Description…
This function is used to retrieve the current velocity of sound setting being used by the sonar, in conjunction with the range setting, to determine the data acquisition duration.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
vos	PDouble	Pointer to a Double value that will receive the velocity-of-sound, in metres-per-second.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

9.13. SfGetStatus

Description…
This function is used to return the current status flags of the sonar hardware, a bits within a single integer.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
flags	PSfStatus	Pointer to an "SfStatus" integer that will receive the status flags. For specific bit definitions, refer to the "SfStatus" entry in section **Error! Reference source not found.**.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

9.14. SfSetChannelEn

Description…
This function is used to enable reception and data output from a specific channel.

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
channel	SfChannel	A value specifying which channel should have its enabled status set by this function, possible values are… • SF_CHAN_STBD, to select the Starboard channel. • SF_CHAN_PORT, to select the Port channel.
enable	Byte	A value which should be non-zero to enable the specified channel.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed. • SF_INVALID_CHANNEL, the channel parameter is invalid.

9.15. SfSetSonarRange

Description…
This function is used to set the range the sonar will collect data over on each channel (port and starboard), when CaptureMode is set to "Sonar" (see "SfGetCapture" and "SfSetCapture", and section 3.1 for further details).

The range should be set to specify the maximum distance to a target that the sonar will receive for – not the acoustic round trip distance (i.e. there and back).
The reception duration and sampling rate is computed using both the range and velocity-of-sound settings from the formulas…

$$t_{rx} = \frac{(2 \times range)}{vos} \qquad t_{sample} = \frac{t_{rx}}{samples}$$

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
range	Double	Specifies the range the sonar should collect data over, in metres. Valid values lie in the range 1m to 200m.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

9.16. SfSetSonarSamples

Description…
This function is used to set the number of data samples that the sonar will collect on each channel for each ping, when CaptureMode is set to "Sonar" (see "SfGetCapture" and "SfSetCapture", and section 3.1 for further details).

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
samples	UInt32	Specifies the number of samples that the sonar should collect on each channel. Valid values lie in the range from 100 to 2047.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

9.17. SfSetSonarVos

Description…
This function is used to set up the velocity-of-sound value used by the sonar in conjunction with the range value to compute the sampling duration during of the receiver.

For further details of how timing and samples rates are calculated, see the "SfSetSonarRange" function.

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
vos	Double	This value specifies the velocity of sound used by the sonar, in metres-per-second. Valid values lie in the range 1300ms^{-1} to 2000ms^{-1} - typically this value is set to 1475 as default.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10. Advanced Configuration & Status Functions

10.1. SfGetAdcFreq

Description…
This function returns the front-end sampling frequency of the receiver Analogue-to-Digital converters.

This function is provided for debugging and diagnostic applications.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
freq	PDouble	Pointer to a Double value that will receive the ADC sampling frequency, in Hertz.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.2. SfGetAdcLatch

Description…
Function that returns an internal hardware setting specifying the clock divisor rate the ADC is latching data at.

This function is provided for debugging and diagnostic applications, and does not require adjustment for normal operation of the sonar.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
latch	PUInt32	Pointer to a UInt32 value that receives the ADC latch divisor.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.3. SfGetCapture

Description…
This function is used to retrieve the current settings that control how the sonar will acquire data. For details of changing the CaptureMode see "SfSetCapture", and section 3.1.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
mode	PSfCaptureMode	Pointer to an SfCapture value the receives the current capture mode of the hardware ("Sonar" or "Scope"). See functions such as "SfGetRxDataSonar", "SfGetRxDataScope" for further details.
source	PSfSource	Pointer to an SfSource value that receive the output from the internal hardware signal path that data will be sampled from.
samples	PUInt32	Pointer to a UInt32 value that will receive the number of samples that will be collected on each ping.
delay	PInt32	Pointer to an Int32 value that will receive the capture delay, in micro-seconds. The delay represents the time between the start of the ping and when data acquisition started.
range	PDouble	Pointer to a Double value that will receive the current capture range, in metres.
freq	PDouble	Pointer to a Double value that will receive the current capture frequency (not ADC frequency!), in Hertz. This is the rate at which data samples are collected from the decoded echo, and used in conjunction with range can determine the sampling resolution.
duration	PDouble	Pointer to a Double value that will receive the duration it takes to capture the specified number of samples over the range, in seconds.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.4. SfGetPingTiming

Description…
This function is used to retrieve information about the ping rate and timing parameters – i.e. the rate at which the sonar will send transmissions, and how long it will wait in between.

Typically, the sonar will compute the capture duration (based on range and velocity-of-sound settings), then multiply this by a scalar to obtain the period it should wait before the next ping. By adjusting the scalar, the effects of reverberation in confined spaces can be reduced if required. However, to prevent excessive power consumption in the transmitters, the 'min' parameter is used to specify a lower timing threshold that cannot be exceeded.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
min	PDouble	Pointer to a Double value that will receive the minimum inter-ping time the sonar will observe, in seconds.
scalar	PDouble	Pointer to a Double value that will receive the timing scalar used by the sonar when computing the inter-ping delay. This is a fractional value with ×1 representing the standard duration.
interval	PDouble	Pointer to a Double value that will receive the actual implemented inter-ping time, in seconds. The inter-ping time is the duration the sonar will wait from one capture cycle to the next start of transmission.
duration	PDouble	Pointer to a Double value that will receive the time it takes for the sonar to perform a transmission and capture the required amount of data (based on the range and velocity-of-sound settings) in seconds.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.5. SfGetRxMode

Description…
This function is used to retrieve the current receiver mode of operation. This should not be confused with the CaptureMode setting, controlling how data is output from the sonar.

This function is provided for debugging and diagnostic applications, and does not require adjustment for normal operation of the sonar. Normally this mode is set to "SF_RX_STANDARD", and the discussion of the other receiver modes lies beyond the scope of this document.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
mode	PSfRxMode	Pointer to an SfRxMode value that will receive the current reception mode.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.6. SfGetRxOffsetAdc

Description…
This function is used to retrieve any numerical offset that is being applied to an ADC channel.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
channel	SfChannel	A value specifying which channel to retrieve information for.
offset	PInt32	Pointer to an Int32 value that will receive the signed numerical offset being used by the ADC for the specified channel.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.7. SfGetRxOffsetRsl

Description…
This function is used to retrieve the Receiver Signal Level offset, a value applied to normalise the numerical output level of the internal digital filters.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
rsl	PInt32	Pointer to an Int32 value that will receive the signal offset, in Decibels.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.8. SfGetRxOversample

Description…
This function is used to retrieve information relating to the oversampling rates of the ADC's relative to required reception parameters.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
ovs	PDouble	Pointer to a Double value that receives the required set minimum oversampling factor the receiver is using when computing operational parameters.
ovsMin	PDouble	Pointer to a Double value that receives the actual minimum oversampling factor achieved by the receiver.
ovsMax	PDouble	Pointer to a Double value that receives the actual maximum oversampling factor achieved by the receiver.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.9. SfGetRxPulse

Description…
This function is used to retrieve information about the current receiver configuration being used by the sonar.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
fstart	PDouble	Pointer to a Double value that receives the starting frequency of reception, in Hertz.
fend	PDouble	Pointer to a Double value that receives the ending frequency of reception, in Hertz.
length	PDouble	Pointer to a Double value that receives the length of the reception pulse, in seconds.
q	PDouble	Pointer to a Double value that receives the Q-factor of the reception pulse, expressed as centre-frequency divided by bandwidth.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.10.SfGetScope

Description…
This function is used to retrieve the current settings that the hardware will use when the CaptureMode is set to "Scope" (see "SfGetCapture" and "SfSetCapture", and section 3.1 for further details).

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
source	PSfSource	Pointer to an SfSource value that receive the output from the internal hardware signal path that data will be sampled from.
samples	PUInt32	Pointer to a UInt32 value that will receive the number of samples that will be collected on each ping.
delay	PInt32	Pointer to an Int32 value that will receive the capture delay, in micro-seconds. The delay represents the time between the start of the ping and when data acquisition started.
range	PDouble	Pointer to a Double value that will receive the current capture range, in metres.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.11.SfGetTvgEnable

Description…
This function is used to retrieve the current enabled state of the sonar's Time-Varying-Gain system.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
value	PByte	Pointer to a Byte value that will receive a non-zero value if the TVG system is enabled.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.12.SfGetTvgFunc

Description…
This function is used to retrieve the control parameters that the sonar uses to calculate its Time-Varying-Gain compensation characteristic.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
alpha	PDouble	Pointer to a Double value that receives the acoustic absorption loss coefficient (commonly referred to by sonar text's as 'alpha'), expressed in Decibels-per-metre.
attnMax	PDouble	Pointer to a Double value that receives the maximum attenuation applied to the signal at the start of reception, in Decibels.
rangeMax	PDouble	Pointer to a Double value that receives the range at which no attenuation is applied to the received signal, in metres.
samples	PUInt32	Pointer to a UInt32 value that receives how many points will be computed in the TVG characteristic. This does not relate to reception samples.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.13.SfGetTvgSlope

Description…
Function that retrieves the current slope-multiplier being applied to the Timer-Varying-Gain characteristic.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
slope	PDouble	Pointer to a Double value that receives the slope-multiplier value (typically a value of 1 means the slope is unmodified).
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.14.SfGetTxEnable

Description…
This function is used to retrieve the current enabled state of the sonar transmitters.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
value	PByte	Pointer to a Byte value that will receive a non-zero value if the transmitter is enabled.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.15.SfGetTxMuteRx

Description…
This function is used to retrieve a flag indicating if the receiver input will be muted during transmission.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
value	PByte	Pointer to a Byte value that will receive a non-zero value if the receiver is muted during transmission.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.16.SfGetTxPower

Description…
This function is used to retrieve the current power setting used by the sonar transmitters.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
power	PByte	Pointer to a Byte value that will receive the current transmitter power level (from 0 to 15).
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.17.SfGetTxPulse

Description…
This function is used to retrieve information about the current transmitter configuration being used by the sonar.

If any of the pointer parameters that will receive values are specified as null (zero), then no value will be written into them and no exception raised.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
fstart	PDouble	Pointer to a Double value that receives the starting frequency of transmission, in Hertz.
fend	PDouble	Pointer to a Double value that receives the ending frequency of transmission, in Hertz.
length	PDouble	Pointer to a Double value that receives the length of the transmission pulse, in seconds.
q	PDouble	Pointer to a Double value that receives the Q-factor of the transmission pulse, expressed as centre-frequency divided by bandwidth.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.18.SfSetAdcLatch

Description…
Function that allows the application to specify the clock divisor rate the ADC is latching data at, when the receiver is operating in "Manual" mode (see "SfSetRxMode").

This function is primarily provided for debugging and diagnostic applications, and should not be adjusted under normal circumstances (and it will only be applied when the receiver is operating in "Manual" mode).

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
latch	UInt32	Value that specifies the ADC latch divisor. Valid values lie in the range 19 to 255.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.19.SfSetCapture

Description…
This function is used to set the current mode the sonar will acquire data in. Depending on the CaptureMode specified, operational values will be taken from those pecified by the "SfSetSonar…" or "SfSetScope…" functions.

For further details about the CaptureMode refer to section 3.1 and the "SfGetRxCountSonar", "SfGetRxCountScope", "SfGetRxDataSonar" and "SfGetRxDataScope" functions.

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
mode	SfCaptureMode	Value that specifies how the sonar will capture and output data ("Sonar" or "Scope").
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.20.SfSetPingTiming

Description…
Function that can be used to setup the timing parameters for the sonar ping rate – i.e. the rate at which the sonar will send transmissions, and how long it will wait in between.

Typically, the sonar will compute the capture duration (based on range and velocity-of-sound settings), then multiply this by a scalar to obtain the period it should wait before the next ping. By adjusting the scalar, the effects of reverberation in confined spaces can be reduced if required. However, to prevent excessive power consumption in the transmitters, the 'min' parameter is used to specify a lower timing threshold that cannot be exceeded.

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
min	Double	Value that will specify the minimum inter-ping time the sonar will observe, in seconds. Valid values lie in the range 0.01s to 10s – a default of 0.02s is used.
scalar	Double	Value that will specify the timing scalar used by the sonar when computing the inter-ping delay. This is a fractional value with ×1 representing the standard duration. Valid values lie in the range 1.0 to 10.0.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.21.SfSetRxMode

Description…
This function can be used to set the receiver mode and determine how (and if) values are auto-computed.

This function is provided for debugging and diagnostic applications, and does not require adjustment for normal operation of the sonar. Normally this mode is set to “SF_RX_STANDARD”, and the discussion of the other receiver modes lies beyond the scope of this document.

When you have executed this function, you will need to make a call the “SfUpdate” to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the “SfCreate”). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
mode	SfRxMode	Value that specifies the current mode of operation of the receiver.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.22.SfSetRxOffsetAdc

Description…
This function is used to specify a numerical offset to apply to data captured on an ADC channel, and is useful for removing any DC loading effects from the sonar transducer.

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
channel	SfChannel	Value specifying which channel will have the offset applied.
offset	Int32	Value specifying the signed numerical offset to apply to the ADC for the specified channel. Valid values lie from -4096 to +4095.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.23.SfSetOffsetRsl

Description…
This function is used to retrieve the Receiver Signal Level offset, a value applied to normalise the numerical output level of the internal digital filters. Under normal circumstances, the offset of either channel will not need any adjustment.

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
rsl	Int32	Value that specifies the signal offset, in Decibels. Valid values lie from 0dB to 120dB.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.24.SfSetRxOversample

Description…
This function is used to specify the minimum acceptable oversampling rate the receiver should use when computing parameters for the ADC sampling rate, when the receiver is operating in "Standard" mode (see "SfSetRxMode").

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
ovs	Double	Value that specifies the required minimum oversampling factor the receiver should use when computing operational parameters. Valid values lie in the range 2.2 to 100 – typically values around 3 or 4 should be used.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.25.SfSetRxPulse

Description…
This function is used to specify how the receiver should decode signals, when operating in "Manual" mode (see "SfSetRxMode").

This function is primarily provided for debugging and diagnostic applications, and should not be adjusted under normal circumstances (and it will only be applied when the receiver is operating in "Manual" mode).

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
fstart	Double	Value that specifies the starting frequency of reception, in Hertz. Valid values line in the range 10000Hz to 2000000Hz.
fend	Double	Value that specifies the ending frequency of reception, in Hertz. Valid values line in the range 10000Hz to 2000000Hz.
length	Double	Value that specifies the length of the reception pulse, in seconds. Valid values lie in the range 0.00001s to 0.05s.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.26.SfSetScope

Description…
This function is used to specify the settings that the hardware will use when the CaptureMode is set to "Scope" (see "SfGetCapture" and "SfSetCapture", and section 3.1 for further details).

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
source	SfSource	Value that specifies the output from the internal hardware signal path that data will be sampled from.
samples	UInt32	Value that specifies the number of samples that will be collected on each ping. Valid values lie in the range from 1 to 1023 for all capture sources other than "SF_SOURCE_RSSI_SHORT" where the maximum value is 2047.
delay	Int32	Value that specifies the capture delay, in micro-seconds. The delay represents the time between the start of the ping and when data acquisition started. Valid values lie in the range from 0μs to 50000μs.
range	Double	Value that specifies the current capture range, in metres. Valid values lie in the range 1m to 200m.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.27.SfSetTvgEnable

Description…
This function is used to set the enabled state of the sonar's Time-Varying-Gain system. When TVG is applied, the sonar will compensate for the acoustic signal loss as the sound travels through the water. Without TVG applied target echoes will appear diminished the further the targets are away from the sonar.

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
value	Byte	Value that should contain a non-zero value if the TVG system is to be enabled, or zero to disable.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.28.SfSetTvgFunc

Description…
This function is used to specify the control parameters that the sonar uses to calculate its Time-Varying-Gain compensation characteristic.

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
alpha	Double	Value that specifies the acoustic absorption loss coefficient (commonly referred to by sonar text's as 'alpha'), expressed in Decibels-per-metre. Valid values lie in the range 0 to 1.
attnMax	Double	Value that specifies the maximum attenuation applied to the signal at the start of reception, in Decibels. Valid values lie in the range 0 to 60.
rangeMax	Double	Value that specifies the range at which no attenuation is applied to the received signal, in metres. Valid values lie in the range 1 to 200.
samples	UInt32	Value that specifies how many points will be computed in the TVG characteristic. This does not relate to reception samples. Valid values lie in the range 1 to 1000.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.29.SfSetTvgSlope

Description…
This function specifies the current slope-multiplier being applied to the Timer-Varying-Gain characteristic. In advanced applications, this could be used as an additional control to Range, Gain and Contrast – allowing the far-field signal level to be varied relative to the near field signal.

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
slope	Double	Value that receives the slope-multiplier value. Valid values lie in the range 0.01 to 10 – typically a value of 1 is used as a default.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.30.SfSetTxEnable

Description…
This function is used to set the enabled state of the sonar's transmitter (for all channels).

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
value	Byte	Value that should contain a non-zero value if the transmitter is to be enabled, or zero to disable.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.31.SfSetTxMuteRx

Description…
This function is used to specify if the receiver is muted during the sonar's transmission cycle. As the transmission an reception occur on the same transducer element, muting the receiver prevents a large high-intensity echo being present at the start of the received and decoded data.

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
value	Byte	Value that should contain a non-zero value if the receiver is muted during transmission, or zero to un-mute. By default the receiver is normally muted.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.32.SfSetTxPower

Description…
This function is used to set the transmission power by manipulating the duty cycle of the transmitter drive circuitry on the sonar hardware.
Primarily this function should be used to turn down the transmit power in situations where reverberation is occurring cause by the sonar operating in a confined acoustic environment.

Increasing the transmitter power to maximum levels does not necessarily give a increase in range. Due to transducer efficiency, above a certain level some components may start to develop more heat and prolonged use at higher powers may reduce the operational life of the sonar components, or cause a thermal shut-down to occur.
For each type of sonar, the default transmit power level has been chosen to give the best balance of range against power consumption. If in doubt, use a power level of 9 as a standard value.

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
power	Byte	Value that will specifies the current transmitter power level. Valid values are in the range 0 to 15, where 15 represents full power. Typically a value of 8 to 10 is best suited.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

10.33.SfSetTxPulse

Description…
This function is used to specify parameters controlling the transmitter configuration used by the sonar.

When you have executed this function, you will need to make a call the "SfUpdate" to compute internal coefficients and apply the changes to the hardware.

Parameters & Result…

Name	*Type*	*Description*
hSon	SfHandle	Should contain a valid handle of the sonar object to control (obtained from a call the "SfCreate"). If an incorrect handle is specified, the function will return an SF_INVALID_HANDLE code.
fstart	Double	Value that specifies the starting frequency of transmission, in Hertz. Valid values line in the range 10000Hz to 2000000Hz.
fend	Double	Value that specifies the ending frequency of transmission, in Hertz. Valid values line in the range 10000Hz to 2000000Hz.
length	Double	Value that specifies the length of the transmission pulse, in seconds. Valid values lie in the range 0.00001s to 0.05s.
result	SfResult	The functions return value, containing the status code of the operation, possible values include… • SF_OK, the operation completed successfully. • SF_INVALID_HANDLE, the hSon parameter contained an invalid handle, and the operation could not be performed.

II. History

Version	Date	Comments
V1.0.1	11th August 2010	Initial library release
V1.0.2	19th August 2010	Fixed “Debug” message appearing when library is first loaded.
V1.0.3	16th September 2010	Fixed various syntax errors in C++ header file.
V1.0.4	31st May 2012	Text corrections in the ‘requirements’ section.
V1.0.5	21st June 2012	Corrected the length of ‘Source’ field in the SfDataScope structure to be one byte in length, not a UInt32 as implied from the definition of a SfSource type. The numeric values written to this byte still remain the same as SfSource, but without the upper padding.
V1.0.6	27th June 2012	Corrected the ‘Time’ field units description of the SfDataScopeSample structure to be in microseconds and not seconds as previously stated. The values output by the DLL have not changed in magnitude, only the documentation text was incorrect – in future the ‘Time’ field type may be amended to an Int32 type to be in line with other fields that hold microsecond values.