BlueView®

TECHNOLOGIES

ProViewer Sonar Development Kit

User's Guide

Contents

Listings

Chapter 1

Welcome

This manual will guide you through the installation and use of this of the BlueView Sonar Development Kit (SDK). Our goal is to get you started on the image acquisition portion of your sonar imaging project, and to help you develop a successful application.

1.1 General Overview

This SDK consists of a set of function calls that allow you to communicate with a BlueView sonar. Functions are provided to acquire pings, produce images, as well as read and write the BlueView .son file format.

1.2 System Requirements

While the final system requirements depend greatly on the target application, meeting the following minimum requirements will ensure that the sonar and the SDK perform acceptably.

- Windows 2000 or XP operating system
- 750MHz or faster processor
- 512MB or more of RAM
- 200MB or more of free disk space
- Ethernet port (100 mbs or faster recommended)

1.3 Installation

To install the SDK, simply run **setup.exe** from the SDK directory on the distribution CD. BlueView recommends choosing a 'shallow' install path such as `c:/bvt` since it makes entering these paths into various development environments easier. Please see the Getting Started section for more information.

1.4 Technical Support

Although we have attempted to make this manual as complete as possible, we realize that there are always additional unanswered questions, as well as unique situations not covered by this document. BlueView is committed to providing industry leading customer service and technical support for all of our products. For technical assistance with this SDK or your BlueView sonar please email your questions to *support@blueviewtech.com*, or contact our customer service department at 206-545-7260 between the hours of 8am and 5pm Pacific Time.

For the latest contact information, data sheets and other support material please visit our web site at `http://www.blueviewtech.com`.

1.5 Typographic Conventions

This document uses the following typographic conventions:

- Commands that you type
  Ex: **setup.exe**
- Names of files and paths
  Ex: `include/bvt_sdk.h`
- Code listings
  Ex: **int** `i`=42;
- E-Mail addresses
  Ex: *support@blueviewtech.com*
- Uniform Resource Locators (URLs)
  Ex: `http://www.blueviewtech.com`

1.6 License Agreement

The accompanying Software and Documentation hereinafter referred to as the "SDK" are proprietary products owned by BlueView Technologies, Inc., and protected under U.S. and international copyright law. Except as authorized under this License Agreement, the Software may be used only on computers owned, leased, or otherwise controlled by you. You may not reverse assemble, reverse compile, or otherwise translate the SDK.

You may make copies of the SDK only for backup purposes. Except as authorized under this License Agreement, no copies of the SDK may be made by you or any person under your authority or control.

The use of the SDK for creating derivative works is covered under a separate SDK Derivative Works Distribution License Agreement available from BlueView Technologies, Inc..

Chapter 2

Getting Started

This section is designed as an introduction to building custom application using the BlueView SDK. Example code contained in this section isn't designed to be complete, but is to be used as a starting point for the reader's own exploration. As such, it often doesn't do important things such as error checking.

2.1 Overview

Although C isn't considered an object oriented language, it is still possible to use many common object oriented programming techniques. Classes can be implemented as opaque typedefs with conveniently name functions to act as member functions. The BlueView SDK follows this design pattern.

The SDK consists of eight such objects. They are as follows: Sonar, Head, Ping, MagImage, ColorMapper, ColorImage, Logger, and Error. To prevent name space pollution, all the objects carry the BVT prefix. The following sections offer basic descriptions of each of these classes.

Sonar

The Sonar object is the top level object in the SDK. It embodies communication with a single physical sonar unit, or file. Sonar also provides a function to create new data files using BlueView's .son format.

Head

Most user interaction occurs with the Head object. A head consists of a group of co-planar transducers which are operated simultaneously to produce (ultimately) a single 2d image. The Head object provides functions to change the range window as well as produce pings. While most BlueView sonar only have a single head, the SDK allows for sonar with multiple heads.

Ping

As its name implies, the Ping object represents the return from a single ping on a particular head. GetImage is the most important function in Ping as it does whatever processing is necessary to convert the ping to an image. In the future the Ping object will expose additional information about the ping, such as the orientation of the head when it was generated.

MagImage

MagImage is short for MagnitudeImage. It provides access to a 2d image where each pixel is intensity of the return from a particular point on a plane emanating from the head. I can be thought of as a 16bit gray-scale image.

ColorMapper

Unfortunately, a MagImage isn't overly useful for display. The ColorMapper object simplifies the process of mapping each pixel of a MagImage to a RGB value according to a colormap and several parameters.

ColorImage

The ColorImage object is identical to MagImage with the exception that each pixel is now a 32bit integer representing the Red,Green, and Blue values assigned by the colormap.

Logger

The SDK is capable of producing a significant amount of debugging output. The Logger object exists to allow the user to control (or disable) the output. Users can also use Logger to add their own custom log messages.

Error

The Error object provides access to the SDKs error reporting system. This allows the user to map from an error number to a human readable description of the error.

2.2 Directory Layout

The installation process creates several directories containing various parts of the SDK. This section briefly describes their contents.

InstallRoot/		
Readme.txt		Release Information
License.txt		The SDK license agreement
colormaps/		Colormap files
	bone.cmap	
	cool.cmap	
	copper.cmap	
	green.cmap	
	hot.cmap	
	jet.cmap	
data/		Example data files
	swimmer.son	Person swimming
doc/		SDK Documentation
include/		Include files
	bvt_sdk.h	Main SDK include file
	bvt_c	C language includes
	bvt_cpp	C++ language includes
lib/		Libraries
	bvtsdk.lib	Main SDK import library
	bvtsdk.dll	Main SDK DLL
	bvtutils.dll	Support DLL
examples/		SDK Examples
	file_sonar/	Demonstrate file reading
	net_sonar/	Demonstrate networked sonar connection
	multi_head_sonar/	Demonstrate using a sonar with multiple heads
	opencv/	Demonstrate interfacing with OpenCV

2.3 Reading a .son File

Most applications using the BlueView SDK generally follow the same basic pattern. Listing 2.1 shows the process of connecting to a sonar (in this case, a .son file), accessing one of it's heads, and doing pings. The SDK is designed so that reading files is nearly identical to working with real hardware. The differences between the two are highlighted in section 2.4.

```
#include <bvt_sdk.h>

BVTSonar son = BVTSonar_Create();
BVTSonar_Open(sonar, "FILE", "swimmer.son");

BVTHead head = NULL;
BVTSonar_GetHead(son, 0, &head);

BVTHead_SetRange(head, 10, 40);

BVTPing ping = NULL;
BVTHead_GetPing(head, 0, &ping);

/* Do something useful with the ping here */

BVTPing_Destroy(ping);
```

```
17 BVTSonar_Destroy(son);
```

Listing 2.1: Reading a File

Let's dig into Listing 2.1. Line 1 includes the main SDK header, `bvt_sdk.h`. This is the only header you need in order to access the SDK's functionality. It automatically includes the needed files from `include/bvt_c`. In order for the compiler/preprocessor to find it, you must make sure that the path to `include/` is in the compiler's include path.

Line 3 creates a new Sonar object for use. To prevent memory leaks, this object is then destroyed when we are done with it (line 17). The next step is to actually open a 'connection' to the sonar, which is on line 4. In this case, we specify that we will be opening a file named `swimmer.son`. The SDK is designed so that reading files is nearly identical to working with real hardware.

Now that we have an open Sonar object, the next thing we usually want to do is retrieve a head object. In this case, we get the first head on line 7. There is no need to destroy the Head as it's 'owned' by the Sonar object. It will be cleaned up when the Sonar is destroyed. Note that a file collected from sonar with multiple heads will also have multiple Heads. In this case, we know that `swimmer.son` only has one head.

Each sonar has a default range window. For files, this is the range window at which it was recorded. As show in line 9, the SetRange function can be used to set the start and stop of the window. You can change the range window at any time except when you are saving data (See below).

Finally, we ask the Head for it's first ping on line 12. A file will typically have multiple pings available (see GetPingCount) where as an Ethernet device will only have a single ping. Either way, using a ping number of -1 gets the *next* ping available. We destroy the ping to prevent memory leaks (line 16). Note that GetPing only does the ping and acquisition phases. See the next section for information about actually building an image.

2.4 Connecting to a ProViewer

Many applications using this SDK will eventually connect to a ProViewer to collect live data. Luckily, the process of using a Ethernet-connected ProViewer is nearly identical to using a file. See Section 2.3 for more information on using the SDK with a file.

```
#include <bvt_sdk.h>

BVTSonar son = BVTSonar_Create();
BVTSonar_Open(sonar, "NET", "");

BVTHead head = NULL;
BVTSonar_GetHead(son, 0, &head);

BVTHead_SetRange(head, 10, 40);

BVTPing ping = NULL;
BVTHead_GetPing(head, -1, &ping);

/* Do something useful with the ping here */

BVTPing_Destroy(ping);
BVTSonar_Destroy(son);
```

Listing 2.2: Connecting to a ProViewer sonar

There are only two differences between Listing 2.1 and Listing 2.2. First, the call to `BVTSonar_Open()` on line 4 is slightly different. When opening a file, the type parameter is **"FILE"** whereas Ethernet sonar devices use **"NET"**, and adds an IP address as a parameter rather than a file name. This instructs the SDK to connect the ProViewer sonar with that IP address.

Second, `BVTHead_GetPing()` is called with a ping number of -1. This means that the sonar will return the next available ping, which is all it really has anyway. This function causes the sonar to immediately generate a ping in the water and return the data. It does not actually process the return to build an image. See Section 2.5 for more information on how to do that.

2.5 Image Building

As mentioned above, `BVTHead_GetPing()` does not actually take the time to process an image. Image processing can be time consuming, so splitting the two operations gives the user a lot of needed flexibility. The following code snippet illustrates the process of building an image from a ping.

```
1  /* Assume that we're connected to a Sonar and have a Head... */
2  BVTPing ping = NULL;
3  BVTHead_GetPing(head, -1, &ping);

5  BVTMagImage img;
6  BVTPing_GetImage(ping, &img);

8  int height = BVTMagImage_GetHeight(img);
9  int width = BVTMagImage_GetWidth(img);

11 BVTMagImage_SavePGM(img, "img.pgm");

13 BVTMagImage_Destroy(img);
14 BVTPing_Destroy(ping);
```

Listing 2.3: Building an Image

In Listing 2.5, we first acquire a ping on line 3 exactly as we've done before. Next, we ask the SDK to process the ping into an image with the call to `BVTPing_GetImage` on line 6. This operation could potentially be time consuming depending on the resolution and range requested (See the BVTHead documentation for more information). Lines 8, 9, and 11 illustrate some common image operations. Finally, to prevent leaks, we destroy the image on line 12.

2.6 Color Mapping

The images produced by `BVTPing_GetImage` are in gray scale with a depth of 16bits. Therefore they aren't really suitable for viewing by humans. A very common operation on such images is to map them into RGB color space using a colormap. To make this process easier, the SDK provides the BVTColorMapper object along with several example colormap files. Listing 2.4 illustrates the use of the BVTColorMapper object.

```
1  /* Assume that we've got a ping... */
2  BVTMagImage img;
3  BVTPing_GetImage(ping, &img);
```

```
5  BVTColorMapper mapper = BVTColorMapper_Create();

7  BVTColorMapper_Load(mapper, "colormaps/bone.cmap");

9  BVTColorImage cimg;
10 BVTColorMapper_MapImage(mapper, img, &cimg);

12 BVTMagImage_Destroy(img);
13 BVTColorMapper_Destroy(mapper);

15 BVTColorImage_SavePPM(cimg, "cimg.ppm");

17 BVTColorImage_Destroy(cimg);
18 BVTPing_Destroy(ping);
```

Listing 2.4: Using the Color Mapper

As with several other objects, we need to create a BVTColorMapper (line 5). It gets destroyed on line 13 to prevent leaks. For this example, we're going to use the color map stored in `"bone.cmap"` which is a common 'nearly gray scale' color map. Although they aren't shown in Listing 2.4, BVTColorMapper has two functions available to control exactly how this colormap is applied to the image, `BVTColorMapper_SetGamma` and `BVTColorMapper_SetThresholds`. The latter is probably the most important of the two. See its documentation for more information.

Once the color map is loaded and configured, it is time to call `BVTColorMapper_MapImage` to convert the MagImage to a ColorImage. Note that this allocates a BVTColorImage, we destroy it on line 17. Line 15 saves the image out to `cimg.ppm`.

2.7 Saving a .son File

The previous sections have dealt with operations designed to produce an image suitable for display or other processing. The ProViewer SDK is also able to save sonar data in BlueView's .son file format. A .son file stores the sonar data in a highly compressed, yet lossless, format. This allows the user to load the file back into ProViewer (or use the SDK) either for review or for higher-resolution processing. The following example shows how to save a series of pings to a file.

```
/* Assume that we're connected to a Sonar (son)... */
BVTSonar file = BVTSonar_Create();

BVTSonar_CreateFile(file, "out.son", son, "");

BVTHead in_head = NULL;
BVTSonar_GetHead(son, 0, &in_head);

BVTHead out_head = NULL;
BVTSonar_GetHead(file, 0, &out_head);

for(int i=0;i<10;i++)
{
    BVTPing ping = NULL;
    BVTHead_GetPing(in_head, -1, &ping);
16     BVTHead_PutPing(out_head, ping);
17     BVTPing_Destroy(ping);
18 }

20 BVTSonar_Destroy(file);
```

Listing 2.5: Saving Files

New files are created by ’cloning‘ an existing sonar. This ensures that the various required parameters propagate to the file. This ’cloning‘ is a accomplished on line 4. In this case, we’re creating a file named `out.son` by cloning the Sonar object, `son`. At this point in previous examples, we acquire a head object to operate on. However, this time we acquire *two* heads (Lines 7 and 7). The first, `in_head` becomes out ’source‘. The second, `out_head` becomes our target.

The actual meat of the example is the body of the for loop at line 12. First, we call the familiar `BVTHead_GetPing` function which acquires a ping. Next, we save the ping to the file using `BVTHead_PutPing`. Finally, we destroy the ping to keep leaks away. Once the loop completes, `out.son` will contain 10 pings.

2.8 Interfacing with OpenCV

OpenCV [1] is an open source library that contains many common image processing and computer vision algorithms. It is also capable of using Intel’s Integrated Performance Primitives for accelerated processing on Intel processors. `IplImage` is the core image structure used by OpenCV. Listings 2.6 and 2.7 illustrate how to create `IplImage` structures from `BVTMagImage` and `BVTColorImage` objects.

```
int height = BVTMagImage_GetHeight(img);
int width = BVTMagImage_GetWidth(img);

IplImage* gray_img = cvCreateImageHeader(cvSize(width,height), IPL_DEPTH_16U, 1);
cvSetImageData(gray_img, BVTMagImage_GetBits(img), width*2);

/* Use gray_img for something */

cvReleaseImageHeader(&gray_img);
BVTMagImage_Destroy(img);
```

Listing 2.6: Creating an `IplImage` from a `BVTMagImage`

```
int height = BVTColorImage_GetHeight(cimg);
int width = BVTColorImage_GetWidth(cimg);

IplImage* color_img = cvCreateImageHeader(cvSize(width,height), IPL_DEPTH_8U, 4);
cvSetImageData(color_img, BVTColorImage_GetBits(cimg), width*4);

/* Use color_img for something */

cvReleaseImageHeader(&color_img);
BVTColorImage_Destroy(cimg);
```

Listing 2.7: Creating an `IplImage` from a `BVTColorImage`

[1] Website: `http://sf.net/projects/opencvlibrary/`

This technique ‘wraps’ an `IplImage` around the image buffer ‘owned’ by the SDK by using `cvCreateImageHeader`. This means that you should call `cvReleaseImageHeader` to have OpenCV deallocate just it’s information without touching the image data buffer. That buffer will be deallocated in `BVTColorImage_Destroy`.

2.9 General Notes

2.9.1 Memory Management

Memory management in the BlueView SDK is designed to follow a couple simple rules. They are as follows:

1. If you Create it, Destroy it.
2. If there is a Destroy for an object, call it when you’re done with the object.

As a user you will be explicitly calling Create to construct Sonar and ColorMapper objects. You must remember to Destroy them when finished to prevent memory leaks. The second rule applies to Ping, MagImage, and ColorImage objects. The SDK handles the creation side of things, but the user needs to Destroy them to prevent leaks. The Head object is a special case. It is ’owned‘ by a Sonar object so Destroying a Sonar also frees up it’s heads. Logger and Error aren’t really allocated objects to start with, so no special memory management techniques are needed for them.

2.9.2 Threading

The SDK is designed to work in a threaded environment with a few restrictions. In general, you must ensure that only a single thread accesses an object at a time. This can either be accomplished through the use of mutexes or through careful programming techniques. Also, you shouldn’t access multiple heads on a single sonar simultaneously. The following example illustrates proper threading.

Main Thread	**Pinger Thread**	**Image Proc. Thread**
Start		
Open Sonar		
Start Image Proc. Thread		
Start Pinger Thread		
	Get Head 0	
	Get Head 1	
	Ping Head 0	
	Add to queue	
		GetImage 0
	Ping Head 1	⋮
	⋮	Process Image 0
	Add to queue	
		GetImage 1
	Ping Head 0	⋮
	⋮	Process Image 1
	Add to queue	
		GetImage 2
	Ping Head 1	⋮
	⋮	Process Image 2

Chapter 3

Module Documentation

3.1 BVTColorImage Object

Store a color image.

Typedefs

- typedef void ∗ BVTColorImage

Functions

- void BVTColorImage_Destroy (BVTColorImage obj)
- int BVTColorImage_GetHeight (BVTColorImage obj)
- int BVTColorImage_GetWidth (BVTColorImage obj)
- double BVTColorImage_GetRangeResolution (BVTColorImage obj)
- int BVTColorImage_GetOriginRow (BVTColorImage obj)
- int BVTColorImage_GetOriginCol (BVTColorImage obj)
- double BVTColorImage_GetPixelRange (BVTColorImage obj, int row, int col)
- double BVTColorImage_GetPixelRelativeBearing (BVTColorImage obj, int row, int col)
- double BVTColorImage_GetFOVMinAngle (BVTColorImage obj)
- double BVTColorImage_GetFOVMaxAngle (BVTColorImage obj)
- unsigned int BVTColorImage_GetPixel (BVTColorImage obj, int row, int col)
- unsigned int ∗ BVTColorImage_GetRow (BVTColorImage obj, int row)
- unsigned int ∗ BVTColorImage_GetBits (BVTColorImage obj)
- RetVal BVTColorImage_CopyBits (BVTColorImage obj, unsigned int ∗data, unsigned int len)
- RetVal BVTColorImage_SavePPM (BVTColorImage obj, const char ∗file_name)

3.1.1 Detailed Description

Store a color image.

The API is nearly identical to MagImage. The main difference is the pixel datatype. In ColorImage, each pixel is a single unsigned int.

- Byte 0: Red Value
- Byte 1: Green Value
- Byte 2: Blue Value
- Byte 3: Alpha Value

3.1.2 Typedef Documentation

3.1.2.1 typedef void∗ BVTColorImage

Opaque type for the BVTColorImage object.

3.1.3 Function Documentation

3.1.3.1 void BVTColorImage_Destroy (BVTColorImage *obj*)

Destroy a BVTColorImage object.

Parameters:
obj Object pointer

Referenced by BVTSDK::ColorImage::∼ColorImage().

3.1.3.2 int BVTColorImage_GetHeight (BVTColorImage *obj*)

Return the height (in pixels) of this image.

Parameters:
obj Object pointer

Referenced by BVTSDK::ColorImage::GetHeight().

3.1.3.3 int BVTColorImage_GetWidth (BVTColorImage *obj*)

Return the width (in pixels) of this image.

Parameters:
obj Object pointer

Referenced by BVTSDK::ColorImage::GetWidth().

3.1.3.4 double BVTColorImage_GetRangeResolution (BVTColorImage *obj*)

Return the range resolution of this image.

The resolution is returned in meters per pixel

Parameters:

obj Object pointer

Referenced by BVTSDK::ColorImage::GetRangeResolution().

3.1.3.5 int BVTColorImage_GetOriginRow (BVTColorImage *obj*)

Retrieve the image row of the origin.

In most cases the origin will be outside of the image boundaries. The origin is the ’location’ (in pixels) of the sonar head in image plane

Parameters:

obj Object pointer

Referenced by BVTSDK::ColorImage::GetOriginRow().

3.1.3.6 int BVTColorImage_GetOriginCol (BVTColorImage *obj*)

Retrieve the image column of the origin.

In most cases the origin will be outside of the image boundaries. The origin is the ’location’ (in pixels) of the sonar head in image plane

Parameters:

obj Object pointer

Referenced by BVTSDK::ColorImage::GetOriginCol().

3.1.3.7 double BVTColorImage_GetPixelRange (BVTColorImage *obj*, int *row*, int *col*)

Retrieve the range (from the sonar head) of the specified pixel.

Parameters:

obj Object pointer

row Origin row

col Origin col

Referenced by BVTSDK::ColorImage::GetPixelRange().

3.1.3.8 double BVTColorImage_GetPixelRelativeBearing (BVTColorImage *obj*, int *row*, int *col*)

Retrieve the bearing relative to the sonar head of the specified pixel.

Parameters:

obj Object pointer

row Origin row

col Origin col

Referenced by BVTSDK::ColorImage::GetPixelRelativeBearing().

3.1.3.9 double BVTColorImage_GetFOVMinAngle (BVTColorImage *obj*)

Return the minimum angle for the sonar's imaging field of view.

The angle is returned in degrees.

Parameters:

obj Object pointer

Referenced by BVTSDK::ColorImage::GetFOVMinAngle().

3.1.3.10 double BVTColorImage_GetFOVMaxAngle (BVTColorImage *obj*)

Return the maximum angle for the sonar's imaging field of view.

The angle is returned in degrees.

Parameters:

obj Object pointer

Referenced by BVTSDK::ColorImage::GetFOVMaxAngle().

3.1.3.11 unsigned int BVTColorImage_GetPixel (BVTColorImage *obj*, int *row*, int *col*)

Return the value of the pixel at (row, col).

Parameters:

obj Object pointer

row Requested row

col Requested col

Referenced by BVTSDK::ColorImage::GetPixel().

3.1.3.12 unsigned int∗ BVTColorImage_GetRow (BVTColorImage *obj*, int *row*)

Return a pointer to a row of pixels.

Parameters:

- ***obj*** Object pointer
- ***row*** Requested row

Referenced by BVTSDK::ColorImage::GetRow().

3.1.3.13 unsigned int∗ BVTColorImage_GetBits (BVTColorImage *obj*)

Return a pointer to the entire image.

The image or organized in Row-Major order (just like C/C++).

Parameters:

- ***obj*** Object pointer

Referenced by BVTSDK::ColorImage::GetBits().

3.1.3.14 RetVal BVTColorImage_CopyBits (BVTColorImage *obj*, unsigned int ∗ *data*, unsigned int *len*)

Copy the raw image data to the user specified buffer.

See GetBits for more info.

Parameters:

- ***obj*** Object pointer
- ***data*** Pointer to a valid buffer
- ***len*** The size of the buffer pointed to by data in pixels NOT bytes.

Referenced by BVTSDK::ColorImage::CopyBits().

3.1.3.15 RetVal BVTColorImage_SavePPM (BVTColorImage *obj*, const char ∗ *file_name*)

Save the image in PPM (PortablePixMap) format.

Parameters:

- ***obj*** Object pointer
- ***file_name*** File name to save to

Referenced by BVTSDK::ColorImage::SavePPM().

3.2 BVTColorMapper Object

Provide support for applying a colormap to a MagImage, thus generating a ColorImage.

Typedefs

- typedef void ∗ BVTColorMapper

Functions

- BVTColorMapper BVTColorMapper_Create ()
- void BVTColorMapper_Destroy (BVTColorMapper obj)
- RetVal BVTColorMapper_Load (BVTColorMapper obj, const char ∗file)
- RetVal BVTColorMapper_SetGamma (BVTColorMapper obj, float gamma)
- float BVTColorMapper_GetGamma (BVTColorMapper obj)
- RetVal BVTColorMapper_SetThresholds (BVTColorMapper obj, int top, int bottom)
- int BVTColorMapper_GetTopThreshold (BVTColorMapper obj)
- int BVTColorMapper_GetBottomThreshold (BVTColorMapper obj)
- int BVTColorMapper_GetAutoMode (BVTColorMapper obj)
- RetVal BVTColorMapper_SetAutoMode (BVTColorMapper obj, int mode)
- RetVal BVTColorMapper_MapImage (BVTColorMapper obj, const BVTMagImage input, BVTColorImage ∗output)

3.2.1 Detailed Description

Provide support for applying a colormap to a MagImage, thus generating a ColorImage.

3.2.2 Typedef Documentation

3.2.2.1 typedef void∗ BVTColorMapper

Opaque type for the BVTColorMapper object.

3.2.3 Function Documentation

3.2.3.1 BVTColorMapper BVTColorMapper_Create ()

Create a BVTColorMapper object.

Referenced by BVTSDK::ColorMapper::ColorMapper().

3.2.3.2 void BVTColorMapper_Destroy (BVTColorMapper *obj*)

Destroy a BVTColorMapper object.

Parameters:
- ***obj*** Object pointer

Referenced by BVTSDK::ColorMapper::∼ColorMapper().

3.2.3.3 RetVal BVTColorMapper_Load (BVTColorMapper *obj*, const char ∗ *file*)

Load a color map file.

Parameters:
- ***obj*** Object pointer
- ***file*** Colormap file name

Referenced by BVTSDK::ColorMapper::Load().

3.2.3.4 RetVal BVTColorMapper_SetGamma (BVTColorMapper *obj*, float *gamma*)

Set the gamma used when colormapping.

Parameters:
- ***obj*** Object pointer
- ***gamma*** Gamma value

Referenced by BVTSDK::ColorMapper::SetGamma().

3.2.3.5 float BVTColorMapper_GetGamma (BVTColorMapper *obj*)

Return the current gamma.

Parameters:
- ***obj*** Object pointer

Referenced by BVTSDK::ColorMapper::GetGamma().

3.2.3.6 RetVal BVTColorMapper_SetThresholds (BVTColorMapper *obj*, int *top*, int *bottom*)

Set the intensity values to be mapped to the top and bottom of the colormap.

If auto intensity is enabled, this function returns an error.

Parameters:
- ***obj*** Object pointer
- ***top*** Top colormap threshold (aka intensity)
- ***bottom*** Bottom colormap threshold

Referenced by BVTSDK::ColorMapper::SetThresholds().

3.2.3.7 int BVTColorMapper_GetTopThreshold (BVTColorMapper *obj*)

Return the upper threshold for the colormap.

The top threshold is also known as ’intensity’. Lowering the top threshold will make a brighter image.

Parameters:

obj Object pointer

Referenced by BVTSDK::ColorMapper::GetTopThreshold().

3.2.3.8 int BVTColorMapper_GetBottomThreshold (BVTColorMapper *obj*)

Return the lower threshold for the colormap.

Parameters:

obj Object pointer

Referenced by BVTSDK::ColorMapper::GetBottomThreshold().

3.2.3.9 int BVTColorMapper_GetAutoMode (BVTColorMapper *obj*)

Return a number greater than 0 if auto-threshold is enabled, 0 if it’s not.

Parameters:

obj Object pointer

Referenced by BVTSDK::ColorMapper::GetAutoMode().

3.2.3.10 RetVal BVTColorMapper_SetAutoMode (BVTColorMapper *obj*, int *mode*)

Enable or disable an internal auto-threshold algorithm.

Parameters:

obj Object pointer

mode $>$ 0 if auto-threshold should be enabled. 0 otherwise.

Referenced by BVTSDK::ColorMapper::SetAutoMode().

3.2.3.11 RetVal BVTColorMapper_MapImage (BVTColorMapper *obj*, const BVTMagImage *input*, BVTColorImage $*$ *output*)

Colormap an image.

Parameters:

obj Object pointer

input Input magnitude image

output Output color image

Referenced by BVTSDK::ColorMapper::MapImage().

3.3 BVTError Object

The Error object provides access to the SDKs error reporting system.

Typedefs

- typedef int RetVal

Functions

- char ∗ BVTError_GetString (RetVal code)
- char ∗ BVTError_GetName (RetVal code)

3.3.1 Detailed Description

The Error object provides access to the SDKs error reporting system.

This allows the user to map from an error number to a human readable description of the error.

3.3.2 Typedef Documentation

3.3.2.1 typedef int RetVal

Define our error code return type.

3.3.3 Function Documentation

3.3.3.1 char∗ BVTError_GetString (RetVal *code*)

Return a description of the error.

Parameters:

code Error code

Referenced by BVTSDK::Error::GetString().

3.3.3.2 char∗ BVTError_GetName (RetVal *code*)

Return a string version of the name of the error constant.

Parameters:

code Error code

Referenced by BVTSDK::Error::GetName().

3.4 BVTHead Object

A head consists of a group of co-planar transducers which are operated simultaneously to produce (ultimately) a single 2d image.

Defines

- #define BVTHEAD_FLUID_SALTWATER (int)(0)
- #define BVTHEAD_FLUID_FRESHWATER (int)(1)
- #define BVTHEAD_FLUID_OTHER (int)(2)
- #define BVTHEAD_RES_OFF (int)(0)
- #define BVTHEAD_RES_LOW (int)(1)
- #define BVTHEAD_RES_MED (int)(2)
- #define BVTHEAD_RES_HIGH (int)(3)
- #define BVTHEAD_RES_AUTO (int)(4)
- #define BVTHEAD_IMAGE_XY (int)(0)
- #define BVTHEAD_IMAGE_RTHETA (int)(1)

Typedefs

- typedef void ∗ BVTHead

Functions

- RetVal BVTHead_GetHeadName (BVTHead obj, char ∗buffer, int buffer_size)
- RetVal BVTHead_SetRange (BVTHead obj, float start, float stop)
- float BVTHead_GetStartRange (BVTHead obj)
- float BVTHead_GetStopRange (BVTHead obj)
- float BVTHead_GetMinimumRange (BVTHead obj)
- float BVTHead_GetMaximumRange (BVTHead obj)
- int BVTHead_GetFluidType (BVTHead obj)
- RetVal BVTHead_SetFluidType (BVTHead obj, int fluid)
- int BVTHead_GetSoundSpeed (BVTHead obj)
- RetVal BVTHead_SetSoundSpeed (BVTHead obj, int speed)
- float BVTHead_GetGainAdjustment (BVTHead obj)
- RetVal BVTHead_SetGainAdjustment (BVTHead obj, float gain)
- float BVTHead_GetTVGSlope (BVTHead obj)
- RetVal BVTHead_SetTVGSlope (BVTHead obj, float tvg)
- int BVTHead_GetCenterFreq (BVTHead obj)
- int BVTHead_GetPingCount (BVTHead obj)
- RetVal BVTHead_GetPing (BVTHead obj, int ping_num, BVTPing ∗ping)
- RetVal BVTHead_PutPing (BVTHead obj, const BVTPing ping)
- RetVal BVTHead_SetImageRes (BVTHead obj, int res)
- RetVal BVTHead_SetRangeResolution (BVTHead obj, float resolution_in_meters)
- RetVal BVTHead_SetImageReqSize (BVTHead obj, int height, int width)
- RetVal BVTHead_SetRemoteBeamForming (BVTHead obj, int enable)

- RetVal BVTHead_SetRawDataSending (BVTHead obj, int enable)
- RetVal BVTHead_SetRemoteImageForming (BVTHead obj, int enable)
- RetVal BVTHead_SetImageType (BVTHead obj, int type)
- int BVTHead_GetImageFilterFlags (BVTHead obj)
- RetVal BVTHead_SetImageFilterFlags (BVTHead obj, int flags)
- RetVal BVTHead_SetRangeDataThreshold (BVTHead obj, unsigned short noise_threshold)
- RetVal BVTHead_SetTxEnable (BVTHead obj, int enableTx)
- RetVal BVTHead_GetMountingOrientation (BVTHead obj, double ∗X_axis_degrees, double ∗Y_axis_degrees, double ∗Z_axis_degrees)
- RetVal BVTHead_SetMountingOrientation (BVTHead obj, double X_axis_degrees, double Y_axis_degrees, double Z_axis_degrees)

3.4.1 Detailed Description

A head consists of a group of co-planar transducers which are operated simultaneously to produce (ultimately) a single 2d image.

The Head object provides functions to change the range window as well as produce pings.

3.4.2 Define Documentation

3.4.2.1 #define BVTHEAD_FLUID_SALTWATER (int)(0)

3.4.2.2 #define BVTHEAD_FLUID_FRESHWATER (int)(1)

3.4.2.3 #define BVTHEAD_FLUID_OTHER (int)(2)

3.4.2.4 #define BVTHEAD_RES_OFF (int)(0)

Turn off image processing.

3.4.2.5 #define BVTHEAD_RES_LOW (int)(1)

Process at low resolution.

3.4.2.6 #define BVTHEAD_RES_MED (int)(2)

Process at med resolution.

3.4.2.7 #define BVTHEAD_RES_HIGH (int)(3)

Process at high resolution.

3.4.2.8 #define BVTHEAD_RES_AUTO (int)(4)

Select a good res for the current range automatically.

3.4.2.9 #define BVTHEAD_IMAGE_XY (int)(0)

Output a cartesian image.

3.4.2.10 #define BVTHEAD_IMAGE_RTHETA (int)(1)

Output a Range/Theta image.

3.4.3 Typedef Documentation

3.4.3.1 typedef void∗ BVTHead

Opaque type for the BVTHead object.

3.4.4 Function Documentation

3.4.4.1 RetVal BVTHead_GetHeadName (BVTHead *obj*, char ∗ *buffer*, int *buffer_size*)

Retrieves a copy of a the name of the head.

The head name is currently set only at the factory, and is simply "Head" on many sonars. Only special order sonars with multiple heads are likely to have a different name.

The length of the name has no actual limit, though 80 characters would seem to be more than enough.

Parameters:

obj Object pointer

buffer buffer to hold the null-terminated string to be passed back

buffer_size total number of characters the passed buffer can hold

Referenced by BVTSDK::Head::GetHeadName().

3.4.4.2 RetVal BVTHead_SetRange (BVTHead *obj*, float *start*, float *stop*)

Set the range to be acquired.

Parameters:

obj Object pointer

start Start range in meters

stop Stop range in meters

Referenced by BVTSDK::Head::SetRange().

3.4.4.3 float BVTHead_GetStartRange (BVTHead *obj*)

Retrieve the current starting range in meters.

Parameters:
obj Object pointer

Referenced by BVTSDK::Head::GetStartRange().

3.4.4.4 float BVTHead_GetStopRange (BVTHead *obj*)

Retrieve the current stopping range in meters.

Parameters:
obj Object pointer

Referenced by BVTSDK::Head::GetStopRange().

3.4.4.5 float BVTHead_GetMinimumRange (BVTHead *obj*)

Return the minimum allowable range for this sonar.

Parameters:
obj Object pointer

Referenced by BVTSDK::Head::GetMinimumRange().

3.4.4.6 float BVTHead_GetMaximumRange (BVTHead *obj*)

Return the maximum allowable range for this sonar.

Parameters:
obj Object pointer

Referenced by BVTSDK::Head::GetMaximumRange().

3.4.4.7 int BVTHead_GetFluidType (BVTHead *obj*)

Return the type of water the head is in.

The returned value will correspond to one of the FLUID_∗ constants.

Parameters:
obj Object pointer

Referenced by BVTSDK::Head::GetFluidType().

3.4.4.8 RetVal BVTHead_SetFluidType (BVTHead *obj*, int *fluid*)

Set the type of water the head is in.

Parameters:

obj Object pointer

fluid The fluid type (one of the FLUID_∗ constants)

Referenced by BVTSDK::Head::SetFluidType().

3.4.4.9 int BVTHead_GetSoundSpeed (BVTHead *obj*)

Return the speed of sound in water.

Parameters:

obj Object pointer

Referenced by BVTSDK::Head::GetSoundSpeed().

3.4.4.10 RetVal BVTHead_SetSoundSpeed (BVTHead *obj*, int *speed*)

Set the speed of sound in water.

Parameters:

obj Object pointer

speed Sound speed in water

Referenced by BVTSDK::Head::SetSoundSpeed().

3.4.4.11 float BVTHead_GetGainAdjustment (BVTHead *obj*)

Return the additional analog gain in dB.

Parameters:

obj Object pointer

Referenced by BVTSDK::Head::GetGainAdjustment().

3.4.4.12 RetVal BVTHead_SetGainAdjustment (BVTHead *obj*, float *gain*)

Set the additional analog gain.

Note: Some systems don't support gain adjustment.

Parameters:

obj Object pointer

gain Additional analog gain in dB

Referenced by BVTSDK::Head::SetGainAdjustment().

3.4.4.13 float BVTHead_GetTVGSlope (BVTHead *obj*)

Return the time variable gain in dB/meter.

Parameters:

obj Object pointer

Referenced by BVTSDK::Head::GetTVGSlope().

3.4.4.14 RetVal BVTHead_SetTVGSlope (BVTHead *obj*, float *tvg*)

Set the time variable analog gain.

Note: Some systems don't support TVG

Parameters:

obj Object pointer

tvg Time variable gain in dB/meter

Referenced by BVTSDK::Head::SetTVGSlope().

3.4.4.15 int BVTHead_GetCenterFreq (BVTHead *obj*)

Return the center frequency(in Hz) of this head.

Parameters:

obj Object pointer

Referenced by BVTSDK::Head::GetCenterFreq().

3.4.4.16 int BVTHead_GetPingCount (BVTHead *obj*)

Return the number of pings 'in' this head A head attached to a file might have more than one ping recorded.

However, a networked sonar will only have a single ping.

Parameters:

obj Object pointer

Referenced by BVTSDK::Head::GetPingCount().

3.4.4.17 RetVal BVTHead_GetPing (BVTHead *obj*, int *ping_num*, BVTPing ∗ *ping*)

Retrieve a Ping from the Head If ping_num is less than 0, return the next ping in the file.

Otherwise, load the specified ping. If the Head is attached to a 'live' sonar (network), then GetPing always acquires a new ping.

Parameters:

obj Object pointer

ping_num The ping number to return

ping The returned Ping object

Referenced by BVTSDK::Head::GetPing().

3.4.4.18 RetVal BVTHead_PutPing (BVTHead *obj*, const BVTPing *ping*)

Write a ping to a file.

Parameters:

obj Object pointer

ping The ping to write out

Referenced by BVTSDK::Head::PutPing().

3.4.4.19 RetVal BVTHead_SetImageRes (BVTHead *obj*, int *res*)

Set the image processing resolution.

The RES_AUTO setting is highly recommended, as it adapts via a formula according to the stop range, whereas the other ranges are fixed values, and should only be used in specialized cases, such as requesting high resolution for longer distances (which will increase the processing time required to create the image). R-Theta images may use either this funtion or SetRangeResolution(), depending on the degree of control required.

Parameters:

obj Object pointer

res Resolution constant (RES_∗)

Referenced by BVTSDK::Head::SetImageRes().

3.4.4.20 RetVal BVTHead_SetRangeResolution (BVTHead *obj*, float *resolution_in_meters*)

Requests a range resolution for R-Theta images.

Also affects the range resolution for RangeData. Note that the exact range resolution may not be available, and the closest resolution will be set. The actual resolution can be obtained by querying the returned image or RangeData object.

Parameters:

obj Object pointer

resolution_in_meters Range resolution, in meters

Referenced by BVTSDK::Head::SetRangeResolution().

3.4.4.21 RetVal BVTHead_SetImageReqSize (BVTHead *obj*, int *height*, int *width*)

Set the requested out image size The processing code will attempt to process images at the specified size.

However, it doesn't guarantee that the final output will match this size. NOTE: For R-Theta images, only the width is used, and the image will be created with that exact width. Height will depend on the range, and the resolution set. (See SetImageRes() and SetRangeResolution())

Parameters:

obj Object pointer

height The requested height

width The requested width

Referenced by BVTSDK::Head::SetImageReqSize().

3.4.4.22 RetVal BVTHead_SetRemoteBeamForming (BVTHead *obj*, int *enable*)

NOTE: this option is only valid for some sonars, in specific circumstances, and should only be used on advice from the factory.

By default, beamforming is done on the local system. If you call this function with enable=true, the SDK will request that the remote sonar handle the majority of the beamforming. This operation applies the next time GetPing is called.

Parameters:

obj Object pointer

enable Enable/Disable remote beamformer. (using 1 or 0 to enable or disable)

Referenced by BVTSDK::Head::SetRemoteBeamForming().

3.4.4.23 RetVal BVTHead_SetRawDataSending (BVTHead *obj*, int *enable*)

NOTE: this option is only valid for some sonars, in specific circumstances, and should only be used on advice from the factory.

By default, the sonar sends data suitable for saving to a .son file. If you are not saving files, AND are recieving processed data thru setting one of the other options, you can call this function with enable=false to reduce the amount of network bandwidth needed. . This operation applies the next time GetPing is called.

Parameters:

obj Object pointer

enable Enable/Disable raw ping data. (using 1 or 0 to enable or disable)

Referenced by BVTSDK::Head::SetRawDataSending().

3.4.4.24 RetVal BVTHead_SetRemoteImageForming (BVTHead *obj*, int *enable*)

NOTE: this option is only valid for some sonars, in specific circumstances, and should only be used on advice from the factory.

By default, image forming is done on the local system. If you call this function with en=true, the SDK will request that the remote sonar handle the image forming. This option is slightly different than remote beam-forming, with all processing done on the sonar, and only the complete image sent over the network connection. This operation applies the next time GetPing is called.

Parameters:

obj Object pointer

enable Enable/Disable remote image forming. (using 1 or 0 to enable or disable)

Referenced by BVTSDK::Head::SetRemoteImageForming().

3.4.4.25 RetVal BVTHead_SetImageType (BVTHead *obj*, int *type*)

Set the type of image output.

NOTE: See SetImageReqSize() for important issues regarding image size.

Parameters:

obj Object pointer

type Image type constant (IMAGE_∗)

Referenced by BVTSDK::Head::SetImageType().

3.4.4.26 int BVTHead_GetImageFilterFlags (BVTHead *obj*)

Return the filter flags.

Parameters:

obj Object pointer

Referenced by BVTSDK::Head::GetImageFilterFlags().

3.4.4.27 RetVal BVTHead_SetImageFilterFlags (BVTHead *obj*, int *flags*)

Set the filter flags.

Parameters:

obj Object pointer

flags Image filter flags (bit field)

Referenced by BVTSDK::Head::SetImageFilterFlags().

3.4.4.28 RetVal BVTHead_SetRangeDataThreshold (BVTHead *obj*, unsigned short *noise_threshold*)

∗∗ EXPERIMENTAL ∗∗ Sets the intensity value below which data is considered to be noise.

Values above this threshold are included in the algorithm which attempts to determine the target edge. This is NOT a simple threshold above which the first value encountered is considered the target edge. This is the same intensity value returned in a MagImage, with a range of an unsigned 16-bit integer. If not set, the default is currently set to 1000.

NOTE: This only applies to specialized BlueView sonars.

Parameters:

obj Object pointer

noise_threshold Threshold below which is considered noise

Referenced by BVTSDK::Head::SetRangeDataThreshold().

3.4.4.29 RetVal BVTHead_SetTxEnable (BVTHead *obj*, int *enableTx*)

By default, the sonar transmits pings.

This function allows the user to disable transmit. This can be useful to get background noise measurements. Note that this is not implemented on all sonars.

Parameters:

obj Object pointer

enableTx If 0, disable the sonar transmission of pings.

Referenced by BVTSDK::Head::SetTxEnable().

3.4.4.30 RetVal BVTHead_GetMountingOrientation (BVTHead *obj*, double ∗ *X_axis_degrees*, double ∗ *Y_axis_degrees*, double ∗ *Z_axis_degrees*)

∗∗ Preliminary support - may change in later SDK versions ∗∗

Position of the sonar positioner relative to the boat.

Parameters:

obj Object pointer

X_axis_degrees rotation about X axis

Y_axis_degrees rotation about Y axis

Z_axis_degrees rotation about Z axis

Referenced by BVTSDK::Head::GetMountingOrientation().

3.4.4.31 RetVal BVTHead_SetMountingOrientation (BVTHead *obj*, double *X_axis_degrees*, double *Y_axis_degrees*, double *Z_axis_degrees*)

∗∗ Preliminary support - may change in later SDK versions ∗∗

Position of the sonar positioner relative to the boat.

Parameters:

obj Object pointer

X_axis_degrees rotation about X axis

Y_axis_degrees rotation about Y axis

Z_axis_degrees rotation about Z axis

Referenced by BVTSDK::Head::SetMountingOrientation().

3.5 BVTLogger Object

The SDK is capable of producing a significant amount of debugging output.

Defines

- #define BVTLOGGER_NONE (int)(-1)
- #define BVTLOGGER_CRITICAL (int)(0)
- #define BVTLOGGER_WARNING (int)(1)
- #define BVTLOGGER_STATUS (int)(2)

Functions

- void BVTLogger_SetLevel (int level)
- RetVal BVTLogger_SetTarget (const char ∗target)

3.5.1 Detailed Description

The SDK is capable of producing a significant amount of debugging output.

The Logger object exists to allow the user to control (or disable) the output. Users can also use Logger to add their own custom log messages.

3.5.2 Define Documentation

3.5.2.1 #define BVTLOGGER_NONE (int)(-1)

Don't log anything.

3.5.2.2 #define BVTLOGGER_CRITICAL (int)(0)

Log critical events.

3.5.2.3 #define BVTLOGGER_WARNING (int)(1)

3.5.2.4 #define BVTLOGGER_STATUS (int)(2)

3.5.3 Function Documentation

3.5.3.1 void BVTLogger_SetLevel (int *level*)

Set the log threshold level.

Events above level will be logged to the target.

Parameters:

level Log level

Referenced by BVTSDK::Logger::SetLevel().

3.5.3.2 RetVal BVTLogger_SetTarget (const char ∗ *target*)

The log target can be a filename, "stdout", "stderr", or "null".

If null is specified, log output is disabled.

Parameters:

target File/device to log output to

Referenced by BVTSDK::Logger::SetTarget().

3.6 BVTMagImage Object

MagImage is short for MagnitudeImage.

Typedefs

- typedef void ∗ BVTMagImage

Functions

- void BVTMagImage_Destroy (BVTMagImage obj)
- int BVTMagImage_GetHeight (BVTMagImage obj)
- int BVTMagImage_GetWidth (BVTMagImage obj)
- double BVTMagImage_GetRangeResolution (BVTMagImage obj)
- double BVTMagImage_GetBearingResolution (BVTMagImage obj)
- int BVTMagImage_GetOriginRow (BVTMagImage obj)
- int BVTMagImage_GetOriginCol (BVTMagImage obj)
- double BVTMagImage_GetPixelRange (BVTMagImage obj, int row, int col)
- double BVTMagImage_GetPixelRelativeBearing (BVTMagImage obj, int row, int col)
- double BVTMagImage_GetFOVMinAngle (BVTMagImage obj)
- double BVTMagImage_GetFOVMaxAngle (BVTMagImage obj)
- unsigned short BVTMagImage_GetPixel (BVTMagImage obj, int row, int col)
- unsigned short ∗ BVTMagImage_GetRow (BVTMagImage obj, int row)
- unsigned short ∗ BVTMagImage_GetBits (BVTMagImage obj)
- RetVal BVTMagImage_CopyBits (BVTMagImage obj, unsigned short ∗data, unsigned int len)
- RetVal BVTMagImage_SavePGM (BVTMagImage obj, const char ∗file_name)

3.6.1 Detailed Description

MagImage is short for MagnitudeImage.

It provides access to a 2d image where each pixel is intensity of the return from a particular point on a plane emanating from the head. It can be thought of as a 16bit grey-scale image.

3.6.2 Typedef Documentation

3.6.2.1 typedef void∗ BVTMagImage

Opaque type for the BVTMagImage object.

3.6.3 Function Documentation

3.6.3.1 void BVTMagImage_Destroy (BVTMagImage *obj*)

Destroy a BVTMagImage object.

Parameters:
obj Object pointer

Referenced by BVTSDK::MagImage::∼MagImage().

3.6.3.2 int BVTMagImage_GetHeight (BVTMagImage *obj*)

Return the height (in pixels) of this image.

Parameters:
obj Object pointer

Referenced by BVTSDK::MagImage::GetHeight().

3.6.3.3 int BVTMagImage_GetWidth (BVTMagImage *obj*)

Return the width (in pixels) of this image.

Parameters:
obj Object pointer

Referenced by BVTSDK::MagImage::GetWidth().

3.6.3.4 double BVTMagImage_GetRangeResolution (BVTMagImage *obj*)

Return the range resolution of this image.

The resolution is returned in meters per pixel.

Parameters:
obj Object pointer

Referenced by BVTSDK::MagImage::GetRangeResolution().

3.6.3.5 double BVTMagImage_GetBearingResolution (BVTMagImage *obj*)

Only valid for R-Theta images.

Returns the bearing resolution, in degrees per column.

Parameters:
obj Object pointer

Referenced by BVTSDK::MagImage::GetBearingResolution().

3.6.3.6 int BVTMagImage_GetOriginRow (BVTMagImage *obj*)

Retrieve the image row of the origin.

In most cases the origin will be outside of the image boundaries. The origin is the 'location' (in pixels) of the sonar head in image plane

Parameters:

obj Object pointer

Referenced by BVTSDK::MagImage::GetOriginRow().

3.6.3.7 int BVTMagImage_GetOriginCol (BVTMagImage *obj*)

Retrieve the image column of the origin.

In most cases the origin will be outside of the image boundaries. The origin is the 'location' (in pixels) of the sonar head in image plane

Parameters:

obj Object pointer

Referenced by BVTSDK::MagImage::GetOriginCol().

3.6.3.8 double BVTMagImage_GetPixelRange (BVTMagImage *obj*, int *row*, int *col*)

Retrieve the range (from the sonar head) of the specified pixel.

Parameters:

obj Object pointer

row Origin row

col Origin col

Referenced by BVTSDK::MagImage::GetPixelRange().

3.6.3.9 double BVTMagImage_GetPixelRelativeBearing (BVTMagImage *obj*, int *row*, int *col*)

Retrieve the bearing relative to the sonar head of the specified pixel.

Parameters:

obj Object pointer

row Origin row

col Origin col

Referenced by BVTSDK::MagImage::GetPixelRelativeBearing().

3.6.3.10 double BVTMagImage_GetFOVMinAngle (BVTMagImage *obj*)

Return the minimum angle for the sonar's imaging field of view.

The angle is returned in degrees.

Parameters:

obj Object pointer

Referenced by BVTSDK::MagImage::GetFOVMinAngle().

3.6.3.11 double BVTMagImage_GetFOVMaxAngle (BVTMagImage *obj*)

Return the maximum angle for the sonar's imaging field of view.

The angle is returned in degrees.

Parameters:

obj Object pointer

Referenced by BVTSDK::MagImage::GetFOVMaxAngle().

3.6.3.12 unsigned short BVTMagImage_GetPixel (BVTMagImage *obj*, int *row*, int *col*)

Return the value of the pixel at (row, col).

Parameters:

obj Object pointer

row Requested row

col Requested col

Referenced by BVTSDK::MagImage::GetPixel().

3.6.3.13 unsigned short* BVTMagImage_GetRow (BVTMagImage *obj*, int *row*)

Return a pointer to a row of pixels.

Parameters:

obj Object pointer

row Requested row

Referenced by BVTSDK::MagImage::GetRow().

3.6.3.14 unsigned short* BVTMagImage_GetBits (BVTMagImage *obj*)

Return a pointer to the entire image.

The image or organized in Row-Major order (just like C/C++).

Parameters:

obj Object pointer

Referenced by BVTSDK::MagImage::GetBits().

3.6.3.15 RetVal BVTMagImage_CopyBits (BVTMagImage *obj*, unsigned short ∗ *data*, unsigned int *len*)

Copy the raw image data to the user specified buffer.

See GetBits for more info.

Parameters:

obj Object pointer

data Pointer to a valid buffer

len The size of the buffer pointed to by data in pixels NOT bytes.

Referenced by BVTSDK::MagImage::CopyBits().

3.6.3.16 RetVal BVTMagImage_SavePGM (BVTMagImage *obj*, const char ∗ *file_name*)

Save the image in PGM (PortableGreyMap) format.

Note that few programs actually support loading a 16bit PGM.

Parameters:

obj Object pointer

file_name File name to save to

Referenced by BVTSDK::MagImage::SavePGM().

3.7 BVTNavData Object

NavData contains various types of user-accessible navigation parameter, which can be saved to and retrieved from a sonar file on a per ping basis.

Typedefs

- typedef void ∗ BVTNavData

Functions

- BVTNavData BVTNavData_Create ()
- void BVTNavData_Destroy (BVTNavData obj)
- RetVal BVTNavData_Clone (BVTNavData obj, const BVTNavData navdata_to_clone)
- RetVal BVTNavData_GetLatitude (BVTNavData obj, double ∗degrees)
- RetVal BVTNavData_SetLatitude (BVTNavData obj, double degrees)
- RetVal BVTNavData_GetLongitude (BVTNavData obj, double ∗degrees)
- RetVal BVTNavData_SetLongitude (BVTNavData obj, double degrees)
- RetVal BVTNavData_GetHorizontalPrecisionError (BVTNavData obj, float ∗error_meters)
- RetVal BVTNavData_SetHorizontalPrecisionError (BVTNavData obj, float error_meters)
- RetVal BVTNavData_GetHeight (BVTNavData obj, float ∗meters_above_geoid)
- RetVal BVTNavData_SetHeight (BVTNavData obj, float meters_above_geoid)
- RetVal BVTNavData_GetVerticalPrecisionError (BVTNavData obj, float ∗error_meters)
- RetVal BVTNavData_SetVerticalPrecisionError (BVTNavData obj, float error_meters)
- RetVal BVTNavData_GetDepth (BVTNavData obj, float ∗meters_below_surface)
- RetVal BVTNavData_SetDepth (BVTNavData obj, float meters_below_surface)
- RetVal BVTNavData_GetAltitude (BVTNavData obj, float ∗meters_above_bottom)
- RetVal BVTNavData_SetAltitude (BVTNavData obj, float meters_above_bottom)
- RetVal BVTNavData_GetHeading (BVTNavData obj, float ∗degrees_true)
- RetVal BVTNavData_SetHeading (BVTNavData obj, float degrees_true)
- RetVal BVTNavData_GetHeadingVelocity (BVTNavData obj, float ∗meters_per_second)
- RetVal BVTNavData_SetHeadingVelocity (BVTNavData obj, float meters_per_second)
- RetVal BVTNavData_GetCourseOverGround (BVTNavData obj, float ∗degrees_true)
- RetVal BVTNavData_SetCourseOverGround (BVTNavData obj, float degrees_true)
- RetVal BVTNavData_GetSpeedOverGround (BVTNavData obj, float ∗meters_per_second)
- RetVal BVTNavData_SetSpeedOverGround (BVTNavData obj, float meters_per_second)
- RetVal BVTNavData_GetPitchAngle (BVTNavData obj, float ∗degrees_bow_up)
- RetVal BVTNavData_SetPitchAngle (BVTNavData obj, float degrees_bow_up)
- RetVal BVTNavData_GetRollAngle (BVTNavData obj, float ∗degrees_port_up)
- RetVal BVTNavData_SetRollAngle (BVTNavData obj, float degrees_port_up)
- RetVal BVTNavData_GetYawAngle (BVTNavData obj, float ∗degrees_bow_to_starboard)
- RetVal BVTNavData_SetYawAngle (BVTNavData obj, float degrees_bow_to_starboard)
- RetVal BVTNavData_GetPitchRate (BVTNavData obj, float ∗degrees_per_second)
- RetVal BVTNavData_SetPitchRate (BVTNavData obj, float degrees_per_second)
- RetVal BVTNavData_GetRollRate (BVTNavData obj, float ∗degrees_per_second)
- RetVal BVTNavData_SetRollRate (BVTNavData obj, float degrees_per_second)
- RetVal BVTNavData_GetYawRate (BVTNavData obj, float ∗degrees_per_second)

- RetVal BVTNavData_SetYawRate (BVTNavData obj, float degrees_per_second)
- RetVal BVTNavData_GetAccelerationX (BVTNavData obj, float ∗accel_mg)
- RetVal BVTNavData_SetAccelerationX (BVTNavData obj, float accel_mg)
- RetVal BVTNavData_GetAccelerationY (BVTNavData obj, float ∗accel_mg)
- RetVal BVTNavData_SetAccelerationY (BVTNavData obj, float accel_mg)
- RetVal BVTNavData_GetAccelerationZ (BVTNavData obj, float ∗accel_mg)
- RetVal BVTNavData_SetAccelerationZ (BVTNavData obj, float accel_mg)
- RetVal BVTNavData_GetOffsetNorth (BVTNavData obj, double ∗meters)
- RetVal BVTNavData_SetOffsetNorth (BVTNavData obj, double meters)
- RetVal BVTNavData_GetOffsetEast (BVTNavData obj, double ∗meters)
- RetVal BVTNavData_SetOffsetEast (BVTNavData obj, double meters)
- RetVal BVTNavData_GetOffsetIsFromLatLongFlag (BVTNavData obj, int ∗is_true)
- RetVal BVTNavData_SetOffsetIsFromLatLongFlag (BVTNavData obj, int is_true)
- char ∗ BVTNavData_GetUserNavString (BVTNavData obj)
- RetVal BVTNavData_SetUserNavString (BVTNavData obj, const char ∗string_in)
- double BVTNavData_GetTimestamp (BVTNavData obj)
- RetVal BVTNavData_SetTimestamp (BVTNavData obj, double sec)

3.7.1 Detailed Description

NavData contains various types of user-accessible navigation parameter, which can be saved to and retrieved from a sonar file on a per ping basis.

The NavData objects can be created and destroyed as needed. When the ping functions are called to get or put the data, the data is copied. This allows NavData objects to be pre-allocated and filled from various instrument sources. It also allows the data to be copied from one NavData object to the other. NOTE: NavData changes will only be saved to a sonar of type FILE.

3.7.2 Typedef Documentation

3.7.2.1 typedef void∗ BVTNavData

Opaque type for the BVTNavData object.

3.7.3 Function Documentation

3.7.3.1 BVTNavData BVTNavData_Create ()

Create a BVTNavData object.

Referenced by BVTSDK::NavData::NavData().

3.7.3.2 void BVTNavData_Destroy (BVTNavData *obj*)

Destroy a BVTNavData object.

Parameters:

obj Object pointer

Referenced by BVTSDK::NavData::∼NavData().

3.7.3.3 RetVal BVTNavData_Clone (BVTNavData *obj*, const BVTNavData *navdata_to_clone*)

Clones the data from the passed NavData object to this object.

Both objects must have already been created.

Parameters:

obj Object pointer

navdata_to_clone existing NavData object to copy from

Referenced by BVTSDK::NavData::Clone().

3.7.3.4 RetVal BVTNavData_GetLatitude (BVTNavData *obj*, double ∗ *degrees*)

Returns the latitude.

If no latitude was stored, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

degrees latitude in degrees

Referenced by BVTSDK::NavData::GetLatitude().

3.7.3.5 RetVal BVTNavData_SetLatitude (BVTNavData *obj*, double *degrees*)

Stores the latitude, as a signed floating point number of degrees.

Latitudes in the Western hemisphere are expressed as negative numbers.

Parameters:

obj Object pointer

degrees latitude in degrees

Referenced by BVTSDK::NavData::SetLatitude().

3.7.3.6 RetVal BVTNavData_GetLongitude (BVTNavData *obj*, double ∗ *degrees*)

Returns the longitude.

If no longitude was stored, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

degrees longitude in degrees

Referenced by BVTSDK::NavData::GetLongitude().

3.7.3.7 RetVal BVTNavData_SetLongitude (BVTNavData *obj*, double *degrees*)

Stores the longitude, as a signed floating point number of degrees.

Longitudes in the Southern hemisphere are expressed as negative numbers.

Parameters:

obj Object pointer

degrees longitude in degrees

Referenced by BVTSDK::NavData::SetLongitude().

3.7.3.8 RetVal BVTNavData_GetHorizontalPrecisionError (BVTNavData *obj*, float ∗ *error_meters*)

Returns the estimated horizontal error (see next function for details).

If none was stored, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

error_meters potential error distance, in meters

Referenced by BVTSDK::NavData::GetHorizontalPrecisionError().

3.7.3.9 RetVal BVTNavData_SetHorizontalPrecisionError (BVTNavData *obj*, float *error_meters*)

Stores the estimated possible horizontal error.

This is primarily (but not exclusively) intended for GPS systems, where there may be some doubt as to the quality of the position fix. HDOP is common, but not used here, as that is a unitless measure and varies between different manufacturers and models.

The idea is to use whatever calculations are appropriate for the local navigation system, and store a possible error value in meters. Some GPS units will attempt to give this directly. (for example, the HPE field in the PGRME sentence, supplied by some Garmin units.) In the case of large errors, or old data, it may be best to either not store a position, or not store new data. (also consider using the NavData time parameter to store the time of last fix, which can then be compared to the ping time when the data is read back to determine the age of the GPS reading.)

Parameters:

obj Object pointer

error_meters potential error distance, in meters

Referenced by BVTSDK::NavData::SetHorizontalPrecisionError().

3.7.3.10 RetVal BVTNavData_GetHeight (BVTNavData *obj*, float ∗ *meters_above_geoid*)

Returns the height above mean sea level.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

meters_above_geoid height in floating point meters

Referenced by BVTSDK::NavData::GetHeight().

3.7.3.11 RetVal BVTNavData_SetHeight (BVTNavData *obj*, float *meters_above_geoid*)

Store the height above Mean Sea Level (usually the EGM96 geoid)).

Parameters:

obj Object pointer

meters_above_geoid altitude in floating point meters

Referenced by BVTSDK::NavData::SetHeight().

3.7.3.12 RetVal BVTNavData_GetVerticalPrecisionError (BVTNavData *obj*, float ∗ *error_meters*)

Returns the estimated vertical error (see next function for details).

If none was stored, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

error_meters potential error distance, in meters

Referenced by BVTSDK::NavData::GetVerticalPrecisionError().

3.7.3.13 RetVal BVTNavData_SetVerticalPrecisionError (BVTNavData *obj*, float *error_meters*)

Stores the estimated possible vertical error (height) above Mean Sea Level (EGM96 geoid).

For other notes, see the functions or Horizontal Precision Error, above.

Parameters:

obj Object pointer

error_meters potential error distance, in meters

Referenced by BVTSDK::NavData::SetVerticalPrecisionError().

3.7.3.14 RetVal BVTNavData_GetDepth (BVTNavData *obj*, float ∗ *meters_below_surface*)

Returns the depth.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

meters_below_surface depth in floating point meters

Referenced by BVTSDK::NavData::GetDepth().

3.7.3.15 RetVal BVTNavData_SetDepth (BVTNavData *obj*, float *meters_below_surface*)

Store the depth.

Parameters:

obj Object pointer

meters_below_surface depth in floating point meters

Referenced by BVTSDK::NavData::SetDepth().

3.7.3.16 RetVal BVTNavData_GetAltitude (BVTNavData *obj*, float ∗ *meters_above_bottom*)

Returns the altitude.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

meters_above_bottom altitude in floating point meters

Referenced by BVTSDK::NavData::GetAltitude().

3.7.3.17 RetVal BVTNavData_SetAltitude (BVTNavData *obj*, float *meters_above_bottom*)

Store the altitude.

Parameters:

obj Object pointer

meters_above_bottom altitude in floating point meters

Referenced by BVTSDK::NavData::SetAltitude().

3.7.3.18 RetVal BVTNavData_GetHeading (BVTNavData *obj*, float ∗ *degrees_true*)

Returns the heading relative to True North.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

degrees_true True heading, in degrees

Referenced by BVTSDK::NavData::GetHeading().

3.7.3.19 RetVal BVTNavData_SetHeading (BVTNavData *obj*, float *degrees_true*)

Store the heading relative to True North.

Parameters:

obj Object pointer

degrees_true True heading, in degrees

Referenced by BVTSDK::NavData::SetHeading().

3.7.3.20 RetVal BVTNavData_GetHeadingVelocity (BVTNavData *obj*, float ∗ *meters_per_second*)

Returns the velocity along the heading.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

meters_per_second velocity, in meters per second

Referenced by BVTSDK::NavData::GetHeadingVelocity().

3.7.3.21 RetVal BVTNavData_SetHeadingVelocity (BVTNavData *obj*, float *meters_per_second*)

Store the velocity along the heading.

Parameters:

obj Object pointer

meters_per_second velocity, in meters per second

Referenced by BVTSDK::NavData::SetHeadingVelocity().

3.7.3.22 RetVal BVTNavData_GetCourseOverGround (BVTNavData *obj*, float ∗ *degrees_true*)

Returns the course over ground (true).

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

degrees_true course over ground, true, in degrees

Referenced by BVTSDK::NavData::GetCourseOverGround().

3.7.3.23 RetVal BVTNavData_SetCourseOverGround (BVTNavData *obj*, float *degrees_true*)

Store the course over ground, true.

Parameters:

obj Object pointer

degrees_true course over ground, true, in degrees

Referenced by BVTSDK::NavData::SetCourseOverGround().

3.7.3.24 RetVal BVTNavData_GetSpeedOverGround (BVTNavData *obj*, float ∗ *meters_per_second*)

Returns the speed over ground.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

meters_per_second speed over ground, in meters per second

Referenced by BVTSDK::NavData::GetSpeedOverGround().

3.7.3.25 RetVal BVTNavData_SetSpeedOverGround (BVTNavData *obj*, float *meters_per_second*)

Store the speed over ground.

Parameters:

obj Object pointer

meters_per_second speed over ground, in meters per second

Referenced by BVTSDK::NavData::SetSpeedOverGround().

3.7.3.26 RetVal BVTNavData_GetPitchAngle (BVTNavData *obj*, float ∗ *degrees_bow_up*)

Get the pitch angle.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

degrees_bow_up pitch angle, in floating point degrees, bow up is positive

Referenced by BVTSDK::NavData::GetPitchAngle().

3.7.3.27 RetVal BVTNavData_SetPitchAngle (BVTNavData *obj*, float *degrees_bow_up*)

Store the pitch angle.

Parameters:

obj Object pointer

degrees_bow_up pitch angle, in floating point degrees, bow up is positive

Referenced by BVTSDK::NavData::SetPitchAngle().

3.7.3.28 RetVal BVTNavData_GetRollAngle (BVTNavData *obj*, float ∗ *degrees_port_up*)

Get the roll angle.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

degrees_port_up roll angle, in floating point degrees, port side up is positive

Referenced by BVTSDK::NavData::GetRollAngle().

3.7.3.29 RetVal BVTNavData_SetRollAngle (BVTNavData *obj*, float *degrees_port_up*)

Store the roll angle.

Parameters:

obj Object pointer

degrees_port_up roll angle, in floating point degrees, port side up is positive

Referenced by BVTSDK::NavData::SetRollAngle().

3.7.3.30 RetVal BVTNavData_GetYawAngle (BVTNavData *obj*, float ∗ *degrees_bow_to_starboard*)

Get the roll angle (but see notes with SetYawAngle()).

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

Referenced by BVTSDK::NavData::GetYawAngle().

3.7.3.31 RetVal BVTNavData_SetYawAngle (BVTNavData *obj*, float *degrees_bow_to_starboard*)

Store the yaw angle.

NOTE: This is NOT the same as the Heading field. Heading is for the normal navigation use of Heading, often from a compass. This field is intended to store raw data from other research instruments, in case you need another storage spot. To keep everyone using the fields the same way so that files can be interchanged, please use SetHeading for the normal heading, and SetYawAngle() only for special uses.

Parameters:

obj Object pointer

Referenced by BVTSDK::NavData::SetYawAngle().

3.7.3.32 RetVal BVTNavData_GetPitchRate (BVTNavData *obj*, float ∗ *degrees_per_second*)

Returns the rate of pitch change.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

degrees_per_second rate of pitch change in degrees per second

Referenced by BVTSDK::NavData::GetPitchRate().

3.7.3.33 RetVal BVTNavData_SetPitchRate (BVTNavData *obj*, float *degrees_per_second*)

Store the the rate of pitch change.

Parameters:

obj Object pointer

degrees_per_second rate of pitch change in degrees per second

Referenced by BVTSDK::NavData::SetPitchRate().

3.7.3.34 RetVal BVTNavData_GetRollRate (BVTNavData *obj*, float ∗ *degrees_per_second*)

Returns the rate of roll change.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

degrees_per_second rate of roll change in degrees per second

Referenced by BVTSDK::NavData::GetRollRate().

3.7.3.35 RetVal BVTNavData_SetRollRate (BVTNavData *obj*, float *degrees_per_second*)

Store the the rate of change in roll.

Parameters:

obj Object pointer

degrees_per_second rate of roll change in floating point degrees per second

Referenced by BVTSDK::NavData::SetRollRate().

3.7.3.36 RetVal BVTNavData_GetYawRate (BVTNavData *obj*, float ∗ *degrees_per_second*)

Returns the rate of change in yaw (heading).

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

degrees_per_second rate of yaw change in floating point degrees per second

Referenced by BVTSDK::NavData::GetYawRate().

3.7.3.37 RetVal BVTNavData_SetYawRate (BVTNavData *obj*, float *degrees_per_second*)

Store the rate of change in yaw (heading).

Parameters:

obj Object pointer

degrees_per_second rate of yaw change in floating point degrees per second

Referenced by BVTSDK::NavData::SetYawRate().

3.7.3.38 RetVal BVTNavData_GetAccelerationX (BVTNavData *obj*, float ∗ *accel_mg*)

Gets the stored acceleration along the X axis.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

accel_mg acceleration along X axis, in milli-g

Referenced by BVTSDK::NavData::GetAccelerationX().

3.7.3.39 RetVal BVTNavData_SetAccelerationX (BVTNavData *obj*, float *accel_mg*)

Sets the stored acceleration along the X axis.

Intended to store raw values of accelerometers.

Parameters:

obj Object pointer

accel_mg acceleration along X axis, in milli-g

Referenced by BVTSDK::NavData::SetAccelerationX().

3.7.3.40 RetVal BVTNavData_GetAccelerationY (BVTNavData *obj*, float ∗ *accel_mg*)

Gets the stored acceleration along the Y axis.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

accel_mg acceleration along Y axis, in milli-g

Referenced by BVTSDK::NavData::GetAccelerationY().

3.7.3.41 RetVal BVTNavData_SetAccelerationY (BVTNavData *obj*, float *accel_mg*)

Sets the stored acceleration along the Y axis.

Intended to store raw values of accelerometers.

Parameters:

obj Object pointer

accel_mg acceleration along Y axis, in milli-g

Referenced by BVTSDK::NavData::SetAccelerationY().

3.7.3.42 RetVal BVTNavData_GetAccelerationZ (BVTNavData *obj*, float ∗ *accel_mg*)

Gets the stored acceleration along the Z axis.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

accel_mg acceleration along Z axis, in milli-g

Referenced by BVTSDK::NavData::GetAccelerationZ().

3.7.3.43 RetVal BVTNavData_SetAccelerationZ (BVTNavData *obj*, float *accel_mg*)

Sets the stored acceleration along the Z axis.

Intended to store raw values of accelerometers.

Parameters:

obj Object pointer

accel_mg acceleration along Z axis, in milli-g

Referenced by BVTSDK::NavData::SetAccelerationZ().

3.7.3.44 RetVal BVTNavData_GetOffsetNorth (BVTNavData *obj*, double ∗ *meters*)

Returns the offset, to the north, from a user-defined fixed point.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

meters offset to the north, in floating point meters

Referenced by BVTSDK::NavData::GetOffsetNorth().

3.7.3.45 RetVal BVTNavData_SetOffsetNorth (BVTNavData *obj*, double *meters*)

Store the offset from a user-defined fixed point.

Parameters:

obj Object pointer

meters offset to the north, in floating point meters

Referenced by BVTSDK::NavData::SetOffsetNorth().

3.7.3.46 RetVal BVTNavData_GetOffsetEast (BVTNavData *obj*, double ∗ *meters*)

Returns the offset, to the east, from a user-defined fixed point.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

meters offset to the east, in floating point meters

Referenced by BVTSDK::NavData::GetOffsetEast().

3.7.3.47 RetVal BVTNavData_SetOffsetEast (BVTNavData *obj*, double *meters*)

Store the offset from a user-defined fixed point.

Parameters:

obj Object pointer

meters offset to the east, in floating point meters

Referenced by BVTSDK::NavData::SetOffsetEast().

3.7.3.48 RetVal BVTNavData_GetOffsetIsFromLatLongFlag (BVTNavData *obj*, int ∗ *is_true*)

Gets a flag value which indicates if the East and North offset values are from the stored Latitude and Longitude.

(see more below...) If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

obj Object pointer

is_true either 1 or 0, to indicate true or false, respectively

Referenced by BVTSDK::NavData::GetOffsetIsFromLatLongFlag().

3.7.3.49 RetVal BVTNavData_SetOffsetIsFromLatLongFlag (BVTNavData *obj*, int *is_true*)

Sets a flag to indicate if the East and North offset values are from the stored Latitude and Longitude.

If so, then software reading the file will know to adjust appropriately. If from some special location, perhaps the UserNavString could be used to indicate the reference point? If nothing is stored, then the value is assumed to be false.

Parameters:

obj Object pointer

is_true either 1 or 0, to indicate true or false, respectively

Referenced by BVTSDK::NavData::SetOffsetIsFromLatLongFlag().

3.7.3.50 char∗ BVTNavData_GetUserNavString (BVTNavData *obj*)

Returns the stored user string, in null-terminated form.

If no string was stored, returns a null string.

Parameters:

obj Object pointer

Referenced by BVTSDK::NavData::GetUserNavString().

3.7.3.51 RetVal BVTNavData_SetUserNavString (BVTNavData *obj*, const char ∗ *string_in*)

Stores a user-defined string related to navigation.

It is highly recommended to store some identifier such that the data is recognizable as you own.

Examples of possible uses might be to store locations based on coordinate systems other than GPS, indicating information about the use of the north and east offset parameters, additional fields from a GPS receiver, or any other information which might change dynamically.

Parameters:

obj Object pointer

string_in string to be stored, null terminated, max length 80 chars

Referenced by BVTSDK::NavData::SetUserNavString().

3.7.3.52 double BVTNavData_GetTimestamp (BVTNavData *obj*)

Return the ping's timestamp in seconds since 00:00:00 UTC, January 1, 1970 Pings are timestamped using a standard UNIX time stamp.

This is a similar value to that returned by the time() C standard library function. In fact, the only difference is the addition of fractional seconds.

Parameters:

obj Object pointer

Referenced by BVTSDK::NavData::GetTimestamp().

3.7.3.53 RetVal BVTNavData_SetTimestamp (BVTNavData *obj*, double *sec*)

Set the NavData's internal time stamp.

See GetTimestamp() for more information.

Parameters:

obj Object pointer

sec Timestamp in seconds since 00:00:00 UTC, January 1, 1970

Referenced by BVTSDK::NavData::SetTimestamp().

3.8 BVTPing Object

As its name implies, the Ping object represents the return from a single ping on a particular head.

Defines

- #define BVTPING_VIDEO_RGB (int)(0)
- #define BVTPING_VIDEO_JPEG (int)(1)

Typedefs

- typedef void ∗ BVTPing

Functions

- void BVTPing_Destroy (BVTPing obj)
- int BVTPing_GetPingNumber (BVTPing obj)
- double BVTPing_GetTimestamp (BVTPing obj)
- int BVTPing_GetTimeZoneOffset (BVTPing obj)
- RetVal BVTPing_SetTimestamp (BVTPing obj, double sec)
- RetVal BVTPing_GetImage (BVTPing obj, BVTMagImage ∗img)
- RetVal BVTPing_GetRangeData (BVTPing obj, BVTRangeData ∗data)
- float BVTPing_GetSonarPitchAngle (BVTPing obj)
- float BVTPing_GetSonarRollAngle (BVTPing obj)
- RetVal BVTPing_GetNavDataCopy (BVTPing obj, BVTNavData ∗nav_data)
- RetVal BVTPing_PutNavData (BVTPing obj, const BVTNavData nav_data)
- RetVal BVTPing_GetVideoFrame (BVTPing obj, unsigned char ∗∗frame, int ∗height, int ∗width, int ∗length, int ∗type)
- RetVal BVTPing_PutVideoFrameJPEG (BVTPing obj, const unsigned char ∗frame, int height, int width, int length)
- RetVal BVTPing_GetPositionerOrientation (BVTPing obj, double ∗X_axis_degrees, double ∗Y_axis_degrees, double ∗Z_axis_degrees)
- RetVal BVTPing_SetPositionerOrientation (BVTPing obj, double X_axis_degrees, double Y_axis_degrees, double Z_axis_degrees)

3.8.1 Detailed Description

As its name implies, the Ping object represents the return from a single ping on a particular head.

GetImage is the most important function in Ping as it does whatever processing is necessary to convert the ping to an image.

Each ping may have a video frame associated with it, and saved in the same file. These images are typically from a video camera mounted near the sonar, such as on a ROV.

Each ping may also store navigation data to indicate the position and orientation of the vehicle at the time of the ping.

A ping is essentially a container for data. As such, after you get a ping from the head and extract the data (or save it to a file), it is necessary to destroy the ping object to free up memory. In the future the Ping object will expose additional information about the ping, such as the orientation of the head when it was generated.

3.8.2 Define Documentation

3.8.2.1 #define BVTPING_VIDEO_RGB (int)(0)

Video frame is raw RGB (RGBRGB.

..)

3.8.2.2 #define BVTPING_VIDEO_JPEG (int)(1)

Video frame is a JPEG image.

3.8.3 Typedef Documentation

3.8.3.1 typedef void∗ BVTPing

Opaque type for the BVTPing object.

3.8.4 Function Documentation

3.8.4.1 void BVTPing_Destroy (BVTPing *obj*)

Destroy a BVTPing object.

Parameters:

obj Object pointer

Referenced by BVTSDK::Ping::∼Ping().

3.8.4.2 int BVTPing_GetPingNumber (BVTPing *obj*)

Return the ping number.

Ping numbers only have meaning if the ping came from a file.

Parameters:

obj Object pointer

Referenced by BVTSDK::Ping::GetPingNumber().

3.8.4.3 double BVTPing_GetTimestamp (BVTPing *obj*)

Return the ping's timestamp in seconds since 00:00:00, January 1, 1970 This is local time.

Pings are timestamped using a standard UNIX time stamp. This is a similar value to that returned by the time() C standard library function. In fact, the only difference is the addition of fractional seconds.

Parameters:

obj Object pointer

Referenced by BVTSDK::Ping::GetTimestamp().

3.8.4.4 int BVTPing_GetTimeZoneOffset (BVTPing *obj*)

Return the ping's timestamp's offset in seconds from UTC time.

Add this value to that returned by GetTimestamp() to obtain UTC time.

Parameters:

obj Object pointer

Referenced by BVTSDK::Ping::GetTimeZoneOffset().

3.8.4.5 RetVal BVTPing_SetTimestamp (BVTPing *obj*, double *sec*)

Set the ping's internal time stamp.

See GetTimestamp() for more information. Note: BlueView strongly recommends that users NOT directly set the time stamp as it is set internally when the ping is actually initiated. If you are trying to synchronize two systems, it is far better to simply make sure that the system clocks are synchronized, as the ping timestamp is created from the PC's internal clock. Network Time Protocol and GPS sources provide highly accurate ways to accomplish this.

Parameters:

obj Object pointer

sec Timestamp in seconds since 00:00:00 UTC, January 1, 1970

Referenced by BVTSDK::Ping::SetTimestamp().

3.8.4.6 RetVal BVTPing_GetImage (BVTPing *obj*, BVTMagImage ∗ *img*)

Retrieve an image of this ping, according to the parameters set in the head used to get this ping.

See Head and MagImage documentation for more details.

Parameters:

obj Object pointer

img Output image

Referenced by BVTSDK::Ping::GetImage().

3.8.4.7 RetVal BVTPing_GetRangeData (BVTPing *obj*, BVTRangeData ∗ *data*)

∗∗ EXPERIMENTAL ∗∗ See RangeData class for more details, and the Head's SetRangeDataThreshold function.

NOTE: This only applies to specialized BlueView sonars.

Parameters:

obj Object pointer

data set of ranges at angles for this ping

Referenced by BVTSDK::Ping::GetRangeData().

3.8.4.8 float BVTPing_GetSonarPitchAngle (BVTPing *obj*)

Get the pitch angle, in floating point degrees.

Bow up is positive. Some BlueView sonar have an internal tilt sensor that is capable of reporting the pitch angle. If the sonar doesn't have the sensor, this function returns 0

Parameters:

obj Object pointer

Referenced by BVTSDK::Ping::GetSonarPitchAngle().

3.8.4.9 float BVTPing_GetSonarRollAngle (BVTPing *obj*)

Get the roll angle, in floating point degrees.

Port side up is positive. Some BlueView sonar have an internal tilt sensor that is capable of reporting the roll angle. If the sonar doesn't have the sensor, this function returns 0.

Parameters:

obj Object pointer

Referenced by BVTSDK::Ping::GetSonarRollAngle().

3.8.4.10 RetVal BVTPing_GetNavDataCopy (BVTPing *obj*, BVTNavData ∗ *nav_data*)

Retrieves a copy of the navigation data stored with this ping.

Note that the data is copied out of the ping into the local NavData object, a pointer to internal data is not returned. Thus, the NavData object may be used after the Ping is destroyed.

Parameters:

obj Object pointer

Referenced by BVTSDK::Ping::GetNavDataCopy().

3.8.4.11 RetVal BVTPing_PutNavData (BVTPing *obj*, const BVTNavData *nav_data*)

Stores a copy of the navigation data with the other ping data, so the data will be saved if the ping is saved to a file.

Parameters:

obj Object pointer

Referenced by BVTSDK::Ping::PutNavData().

3.8.4.12 RetVal BVTPing_GetVideoFrame (BVTPing *obj*, unsigned char ∗∗ *frame*, int ∗ *height*, int ∗ *width*, int ∗ *length*, int ∗ *type*)

Returns the video frame associated with this ping.

The video frame may be in any of the supported image formats. Some image formats may already contain parameters such as height and width (and more), but valid pointers must be passed in anyway. The same pointer can be passed in for multiple parameters, if those parameters will not be used. However, they are provided both for formats which do not have embedded size information, and so that the display window may be created and/or sized without parsing the image data.

NOTE: This function will return BVT_NO_VIDEO_FRAME if there is no video frame stored for the ping.

WARNING: The data buffer must NOT be accessed after the ping object is destroyed, as the pointer will no longer point to valid data and will likely crash your application! So copy off the data before destroying the Ping object.

The single value pointers must be pointers to allocated data, not just pointer types. For example:

int height, width, length, type, retval;

int ∗ frame_ptr;

retval = GetVideoFrame(frame_ptr, &height, &width, &length, &type);

Parameters:

obj Object pointer

frame Pointer to a pointer to the image data to be returned

height Pointer to return the uncompressed height of the image, in pixels

width Pointer to return the uncompressed width of the image, in pixels

length Pointer to return the actual size of the data buffer returned, in bytes, which may include additional metadata for some image types

type pointer to return the type of image returned: FRAME_RGB or FRAME_JPEG

Referenced by BVTSDK::Ping::GetVideoFrame().

3.8.4.13 RetVal BVTPing_PutVideoFrameJPEG (BVTPing *obj*, const unsigned char ∗ *frame*, int *height*, int *width*, int *length*)

Store a JPEG image to save with this ping.

Note that the height and width values will simply be stored and available to read when the frame is retrieved. These have no effect on the actual image size (the image will not be resized). The length however is very important, as it determines how far from the passed image pointer data will be read. An incorrect length could result in an application crash.

Parameters:

obj Object pointer

frame Pointer to a single video frame

height Uncompressed height of the image, in pixels

width Uncompressed width of the image, in pixels

length Actual number of bytes being passed in

Referenced by BVTSDK::Ping::PutVideoFrameJPEG().

3.8.4.14 RetVal BVTPing_GetPositionerOrientation (BVTPing *obj*, double ∗ *X_axis_degrees*, double ∗ *Y_axis_degrees*, double ∗ *Z_axis_degrees*)

∗∗ Preliminary support - may change in later SDK versions ∗∗

Get orientation of the sonar head relative to the positioner.

Effectively the raw position data from a ROS pan/tilt unit.

Parameters:

obj Object pointer

X_axis_degrees rotation about X axis

Y_axis_degrees rotation about Y axis

Z_axis_degrees rotation about Z axis

Referenced by BVTSDK::Ping::GetPositionerOrientation().

3.8.4.15 RetVal BVTPing_SetPositionerOrientation (BVTPing *obj*, double *X_axis_degrees*, double *Y_axis_degrees*, double *Z_axis_degrees*)

∗∗ Preliminary support - may change in later SDK versions ∗∗

Set orientation of the sonar head relative to the positioner.

Effectively the raw position data from a ROS pan/tilt unit.

Parameters:

obj Object pointer

X_axis_degrees rotation about X axis

Y_axis_degrees rotation about Y axis

Z_axis_degrees rotation about Z axis

Referenced by BVTSDK::Ping::SetPositionerOrientation().

3.9 BVTRangeData Object

∗∗ EXPERIMENTAL ∗∗ This functionality is still under development! ∗∗∗ RangeData is a set of ranges from the sonar head, at various angles from the sonar head.

Defines

- #define BVTRANGEDATA_MAX_RANGE (int)(999)

Typedefs

- typedef void ∗ BVTRangeData

Functions

- int BVTRangeData_GetCount (BVTRangeData obj)
- double BVTRangeData_GetRangeResolution (BVTRangeData obj)
- double BVTRangeData_GetBearingResolution (BVTRangeData obj)
- float BVTRangeData_GetFOVMinAngle (BVTRangeData obj)
- float BVTRangeData_GetFOVMaxAngle (BVTRangeData obj)
- RetVal BVTRangeData_CopyRangeValues (BVTRangeData obj, float ∗ranges, int number_of_ranges)
- float BVTRangeData_GetRangeValue (BVTRangeData obj, int index)
- float BVTRangeData_GetBearingValue (BVTRangeData obj, int index)
- int BVTRangeData_GetColorImagePixelX (BVTRangeData obj, int rangeDataIndex, const BVTColorImage image)
- int BVTRangeData_GetColorImagePixelY (BVTRangeData obj, int rangeDataIndex, const BVTColorImage image)

3.9.1 Detailed Description

∗∗ EXPERIMENTAL ∗∗ This functionality is still under development! ∗∗∗ RangeData is a set of ranges from the sonar head, at various angles from the sonar head.

For each angle, the range, bearing and intensity of the return beam at that range is stored. NOTE: RangeData only applies to specialized BlueView sonars, and has no use for our standard imaging sonars.

3.9.2 Define Documentation

3.9.2.1 #define BVTRANGEDATA_MAX_RANGE (int)(999)

Values greater than this indicate no range could be measured.

3.9.3 Typedef Documentation

3.9.3.1 typedef void∗ BVTRangeData

Opaque type for the BVTRangeData object.

3.9.4 Function Documentation

3.9.4.1 int BVTRangeData_GetCount (BVTRangeData *obj*)

Returns the number of range values stored for this ping.

Parameters:

obj Object pointer

Referenced by BVTSDK::RangeData::GetCount().

3.9.4.2 double BVTRangeData_GetRangeResolution (BVTRangeData *obj*)

Returns the resolution of the range values, in meters.

Parameters:

obj Object pointer

Referenced by BVTSDK::RangeData::GetRangeResolution().

3.9.4.3 double BVTRangeData_GetBearingResolution (BVTRangeData *obj*)

Returns the resolution of the bearing stored with each range value.

This is the difference in bearing between each range value in the array.

Parameters:

obj Object pointer

Referenced by BVTSDK::RangeData::GetBearingResolution().

3.9.4.4 float BVTRangeData_GetFOVMinAngle (BVTRangeData *obj*)

Return the minimum angle for the sonar's imaging field of view.

In other words, this is the angle of the first range value, as all angles are "left referenced."The angle is returned in degrees. Note that this may not represent the actual physical field of view of a particular sonar, but does represent the field of view of the data being returned. Some outside values may have range values indicating they are out of range.

Parameters:

obj Object pointer

Referenced by BVTSDK::RangeData::GetFOVMinAngle().

3.9.4.5 float BVTRangeData_GetFOVMaxAngle (BVTRangeData *obj*)

Return the maximum angle for the sonar's imaging field of view.

In other words, this is the angle of the last range value, as all angles are "left referenced."The angle is returned in degrees. Note that this may not represent the actual physical field of view of a particular sonar, but does represent the field of view of the data being returned. Some outside values may have range values indicating they are out of range.

Parameters:

obj Object pointer

Referenced by BVTSDK::RangeData::GetFOVMaxAngle().

3.9.4.6 RetVal BVTRangeData_CopyRangeValues (BVTRangeData *obj*, float ∗ *ranges*, int *number_of_ranges*)

Copies the range values into the user specified buffer.

The buffer must hold the entire number of ranges (See GetCount() above), or an error is returned.

Parameters:

obj Object pointer

ranges Pointer to a valid buffer of type float.

number_of_ranges Number of values the buffer can hold.

Referenced by BVTSDK::RangeData::CopyRangeValues().

3.9.4.7 float BVTRangeData_GetRangeValue (BVTRangeData *obj*, int *index*)

Returns the range from the sonar head, in meters, at a particular index into the array.

NOTE: Check all returned values for validity. If range > BVTRANGEDATA_MAX_RANGE then the range could not be determined within the capabilities of the sonar. Meaning that the closest object at that bearing was either out of view of the sonar, or the threshold was set too high to be detected.

Parameters:

obj Object pointer

index index into the array of RangeData values

Referenced by BVTSDK::RangeData::GetRangeValue().

3.9.4.8 float BVTRangeData_GetBearingValue (BVTRangeData *obj*, int *index*)

Returns the bearing from the center of the sonar head, in degrees (+/-), at a particular index into the array.

Parameters:

obj Object pointer

index index into the array of RangeData values

Referenced by BVTSDK::RangeData::GetBearingValue().

3.9.4.9 int BVTRangeData_GetColorImagePixelX (BVTRangeData *obj*, int *rangeDataIndex*, const BVTColorImage *image*)

Returns the X coordinate for the pixel in the passed ColorImage, which maps to the range and bearing at the index passed.

This allows placing of the range data on a colorimage, easing analysis of the algorithm used for thresholding.

Parameters:

obj Object pointer

image ColorImage object where the pixel coordinate is needed

Referenced by BVTSDK::RangeData::GetColorImagePixelX().

3.9.4.10 int BVTRangeData_GetColorImagePixelY (BVTRangeData *obj*, int *rangeDataIndex*, const BVTColorImage *image*)

Returns the Y coordinate for the pixel in the passed ColorImage which maps to the range and bearing at the index passed.

(see similar function, above, for more details)

Parameters:

obj Object pointer

image ColorImage object where the pixel coordinate is needed

Referenced by BVTSDK::RangeData::GetColorImagePixelY().

3.10 BVTSonar Object

The Sonar object is the top level object in the SDK.

Typedefs

- typedef void ∗ BVTSonar

Functions

- BVTSonar BVTSonar_Create ()
- void BVTSonar_Destroy (BVTSonar obj)
- RetVal BVTSonar_Open (BVTSonar obj, const char ∗type, const char ∗type_params)
- RetVal BVTSonar_CreateFile (BVTSonar obj, const char ∗file_name, const BVTSonar src, const char ∗create_params)
- int BVTSonar_GetFileSize (BVTSonar obj)
- RetVal BVTSonar_GetHead (BVTSonar obj, int head_num, BVTHead ∗head)
- int BVTSonar_GetHeadCount (BVTSonar obj)
- RetVal BVTSonar_GetSonarTypeAsString (BVTSonar obj, char ∗buffer, int buffer_size)
- RetVal BVTSonar_GetSonarName (BVTSonar obj, char ∗buffer, int buffer_size)
- float BVTSonar_GetTemperature (BVTSonar obj)

3.10.1 Detailed Description

The Sonar object is the top level object in the SDK.

A sonar object embodies communication with a single physical sonar unit, or file. Each sonar contains several heads, which is where most of the functionality is implemented. Sonar also provides a function to create new data files using BlueView's .son format.

3.10.2 Typedef Documentation

3.10.2.1 typedef void∗ BVTSonar

Opaque type for the BVTSonar object.

3.10.3 Function Documentation

3.10.3.1 BVTSonar BVTSonar_Create ()

Create a BVTSonar object.

Referenced by BVTSDK::Sonar::Sonar().

3.10.3.2 void BVTSonar_Destroy (BVTSonar *obj*)

Destroy a BVTSonar object.

Parameters:

obj Object pointer

Referenced by BVTSDK::Sonar::∼Sonar().

3.10.3.3 RetVal BVTSonar_Open (BVTSonar *obj*, const char ∗ *type*, const char ∗ *type_params*)

Open the sonar type 'type' using the specified parameters.

Allowed types (and parameters):

- FILE

  [filename] - Required

- NET

  [host] - Connect to the specified host.

  Parameters:

  obj Object pointer

  type The type of sonar to open

  type_params Various type-specific parameters

Referenced by BVTSDK::Sonar::Open().

3.10.3.4 RetVal BVTSonar_CreateFile (BVTSonar *obj*, const char ∗ *file_name*, const BVTSonar *src*, const char ∗ *create_params*)

Create a new data file.

Files are always created by 'cloning' another Sonar object. This ensures that the file receives all the needed setup/configuration data needed to process images.

Parameters:

obj Object pointer

file_name The filename of the file to be created

src The Sonar object to clone when creating the file

create_params Parameters for (reserved for future use)

Referenced by BVTSDK::Sonar::CreateFile().

3.10.3.5 int BVTSonar_GetFileSize (BVTSonar *obj*)

Gets the size of a file created with CreateFile().

Only works with file type sonars. A networked sonar will return 0, as will a file type sonar if there is no open file associated with it. The return value must be multiplied by 1000 to get the actual file size in bytes.

Parameters:

obj Object pointer

Referenced by BVTSDK::Sonar::GetFileSize().

3.10.3.6 RetVal BVTSonar_GetHead (BVTSonar *obj*, int *head_num*, BVTHead ∗ *head*)

Retrieve a Head object from the sonar.

Parameters:

obj Object pointer

head_num The head number to return

head The returned Head object

Referenced by BVTSDK::Sonar::GetHead().

3.10.3.7 int BVTSonar_GetHeadCount (BVTSonar *obj*)

Return the number of heads on this sonar.

Parameters:

obj Object pointer

Referenced by BVTSDK::Sonar::GetHeadCount().

3.10.3.8 RetVal BVTSonar_GetSonarTypeAsString (BVTSonar *obj*, char ∗ *buffer*, int *buffer_size*)

Retrieves a copy of a short string with the model of the sonar.

At the time of this writing, 20 characters would easily hold all of the sonar model names.

Parameters:

obj Object pointer

buffer buffer to hold the null-terminated string to be passed back

buffer_size total number of characters the passed buffer can hold

Referenced by BVTSDK::Sonar::GetSonarTypeAsString().

3.10.3.9 RetVal BVTSonar_GetSonarName (BVTSonar *obj*, char ∗ *buffer*, int *buffer_size*)

Retrieves a copy of the name of the sonar.

The name is set only via the ProViewer application (at least at this time), or at the factory, and is separate from any BlueView model designations.

The length of the name could be considerably longer than the sonar type, and there is no actual limit, though 80 characters would seem to be more than enough.

Parameters:

obj Object pointer

buffer buffer to hold the null-terminated string to be passed back

buffer_size total number of characters the passed buffer can hold

Referenced by BVTSDK::Sonar::GetSonarName().

3.10.3.10 float BVTSonar_GetTemperature (BVTSonar *obj*)

Return the sonar's internal temperature in degrees Celsius If the sonar doesn't have a temp sensor this function returns absolute zero (-273.15).

Parameters:

obj Object pointer

Referenced by BVTSDK::Sonar::GetTemperature().

Chapter 4

Data Structure Documentation

4.1 BVTSDK::ColorImage Class Reference

Store a color image.

Public Member Functions

- ColorImage ()
- ∼ColorImage ()
- int GetHeight ()
- int GetWidth ()
- double GetRangeResolution ()
- int GetOriginRow ()
- int GetOriginCol ()
- double GetPixelRange (int row, int col)
- double GetPixelRelativeBearing (int row, int col)
- double GetFOVMinAngle ()
- double GetFOVMaxAngle ()
- unsigned int GetPixel (int row, int col)
- unsigned int ∗ GetRow (int row)
- unsigned int ∗ GetBits ()
- RetVal CopyBits (unsigned int ∗data, unsigned int len)
- RetVal SavePPM (std::string file_name)

4.1.1 Detailed Description

Store a color image.

The API is nearly identical to MagImage. The main difference is the pixel datatype. In ColorImage, each pixel is a single unsigned int.

- Byte 0: Red Value

- Byte 1: Green Value
- Byte 2: Blue Value
- Byte 3: Alpha Value

4.1.2 Constructor & Destructor Documentation

4.1.2.1 BVTSDK::ColorImage::ColorImage () `[inline]`

Create the object.

4.1.2.2 BVTSDK::ColorImage::∼ColorImage () `[inline]`

Destroy the object.

References BVTColorImage_Destroy().

4.1.3 Member Function Documentation

4.1.3.1 int BVTSDK::ColorImage::GetHeight () `[inline]`

Return the height (in pixels) of this image.

References BVTColorImage_GetHeight().

4.1.3.2 int BVTSDK::ColorImage::GetWidth () `[inline]`

Return the width (in pixels) of this image.

References BVTColorImage_GetWidth().

4.1.3.3 double BVTSDK::ColorImage::GetRangeResolution () `[inline]`

Return the range resolution of this image.

The resolution is returned in meters per pixel

References BVTColorImage_GetRangeResolution().

4.1.3.4 int BVTSDK::ColorImage::GetOriginRow () `[inline]`

Retrieve the image row of the origin.

In most cases the origin will be outside of the image boundaries. The origin is the ’location’ (in pixels) of the sonar head in image plane

References BVTColorImage_GetOriginRow().

4.1.3.5 int BVTSDK::ColorImage::GetOriginCol () `[inline]`

Retrieve the image column of the origin.

In most cases the origin will be outside of the image boundaries. The origin is the 'location' (in pixels) of the sonar head in image plane

References BVTColorImage_GetOriginCol().

4.1.3.6 double BVTSDK::ColorImage::GetPixelRange (int *row*, int *col*) `[inline]`

Retrieve the range (from the sonar head) of the specified pixel.

Parameters:

row Origin row

col Origin col

References BVTColorImage_GetPixelRange().

4.1.3.7 double BVTSDK::ColorImage::GetPixelRelativeBearing (int *row*, int *col*) `[inline]`

Retrieve the bearing relative to the sonar head of the specified pixel.

Parameters:

row Origin row

col Origin col

References BVTColorImage_GetPixelRelativeBearing().

4.1.3.8 double BVTSDK::ColorImage::GetFOVMinAngle () `[inline]`

Return the minimum angle for the sonar's imaging field of view.

The angle is returned in degrees.

References BVTColorImage_GetFOVMinAngle().

4.1.3.9 double BVTSDK::ColorImage::GetFOVMaxAngle () `[inline]`

Return the maximum angle for the sonar's imaging field of view.

The angle is returned in degrees.

References BVTColorImage_GetFOVMaxAngle().

4.1.3.10 unsigned int BVTSDK::ColorImage::GetPixel (int *row*, int *col*) `[inline]`

Return the value of the pixel at (row, col).

Parameters:

row Requested row

col Requested col

References BVTColorImage_GetPixel().

4.1.3.11 unsigned int∗ BVTSDK::ColorImage::GetRow (int *row*) `[inline]`

Return a pointer to a row of pixels.

Parameters:

row Requested row

References BVTColorImage_GetRow().

4.1.3.12 unsigned int∗ BVTSDK::ColorImage::GetBits () `[inline]`

Return a pointer to the entire image.

The image or organized in Row-Major order (just like C/C++).

References BVTColorImage_GetBits().

4.1.3.13 RetVal BVTSDK::ColorImage::CopyBits (unsigned int ∗ *data*, unsigned int *len*) `[inline]`

Copy the raw image data to the user specified buffer.

See GetBits for more info.

Parameters:

data Pointer to a valid buffer

len The size of the buffer pointed to by data in pixels NOT bytes.

References BVTColorImage_CopyBits().

4.1.3.14 RetVal BVTSDK::ColorImage::SavePPM (std::string *file_name*) `[inline]`

Save the image in PPM (PortablePixMap) format.

Parameters:

file_name File name to save to

References BVTColorImage_SavePPM().

4.2 BVTSDK::ColorMapper Class Reference

Provide support for applying a colormap to a MagImage, thus generating a ColorImage.

Public Member Functions

- ColorMapper ()
- ∼ColorMapper ()
- RetVal Load (std::string file)
- RetVal SetGamma (float gamma)
- float GetGamma ()
- RetVal SetThresholds (int top, int bottom)
- int GetTopThreshold ()
- int GetBottomThreshold ()
- int GetAutoMode ()
- RetVal SetAutoMode (int mode)
- RetVal MapImage (const MagImage &input, ColorImage ∗output)

4.2.1 Detailed Description

Provide support for applying a colormap to a MagImage, thus generating a ColorImage.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 BVTSDK::ColorMapper::ColorMapper () `[inline]`

Create the object.

References BVTColorMapper_Create().

4.2.2.2 BVTSDK::ColorMapper::∼ColorMapper () `[inline]`

Destroy the object.

References BVTColorMapper_Destroy().

4.2.3 Member Function Documentation

4.2.3.1 RetVal BVTSDK::ColorMapper::Load (std::string *file*) `[inline]`

Load a color map file.

Parameters:

file Colormap file name

References BVTColorMapper_Load().

4.2.3.2 RetVal BVTSDK::ColorMapper::SetGamma (float *gamma*) [inline]

Set the gamma used when colormapping.

Parameters:

gamma Gamma value

References BVTColorMapper_SetGamma().

4.2.3.3 float BVTSDK::ColorMapper::GetGamma () [inline]

Return the current gamma.

References BVTColorMapper_GetGamma().

4.2.3.4 RetVal BVTSDK::ColorMapper::SetThresholds (int *top*, int *bottom*) [inline]

Set the intensity values to be mapped to the top and bottom of the colormap.

If auto intensity is enabled, this function returns an error.

Parameters:

top Top colormap threshold (aka intensity)

bottom Bottom colormap threshold

References BVTColorMapper_SetThresholds().

4.2.3.5 int BVTSDK::ColorMapper::GetTopThreshold () [inline]

Return the upper threshold for the colormap.

The top threshold is also known as 'intensity'. Lowering the top threshold will make a brighter image.

References BVTColorMapper_GetTopThreshold().

4.2.3.6 int BVTSDK::ColorMapper::GetBottomThreshold () [inline]

Return the lower threshold for the colormap.

References BVTColorMapper_GetBottomThreshold().

4.2.3.7 int BVTSDK::ColorMapper::GetAutoMode () [inline]

Return a number greater than 0 if auto-threshold is enabled, 0 if it's not.

References BVTColorMapper_GetAutoMode().

4.2.3.8 RetVal BVTSDK::ColorMapper::SetAutoMode (int *mode*) `[inline]`

Enable or disable an internal auto-threshold algorithm.

Parameters:

mode > 0 if auto-threshold should be enabled. 0 otherwise.

References BVTColorMapper_SetAutoMode().

4.2.3.9 RetVal BVTSDK::ColorMapper::MapImage (const MagImage & *input*, ColorImage $*$ *output*) `[inline]`

Colormap an image.

Parameters:

input Input magnitude image

output Output color image

References BVTColorMapper_MapImage().

4.3 BVTSDK::Error Class Reference

The Error object provides access to the SDKs error reporting system.

Static Public Member Functions

- static std::string GetString (RetVal code)
- static std::string GetName (RetVal code)

4.3.1 Detailed Description

The Error object provides access to the SDKs error reporting system.

This allows the user to map from an error number to a human readable description of the error.

4.3.2 Member Function Documentation

4.3.2.1 static std::string BVTSDK::Error::GetString (RetVal *code*) `[inline, static]`

Return a description of the error.

Parameters:

code Error code

References BVTError_GetString().

4.3.2.2 static std::string BVTSDK::Error::GetName (RetVal *code*) `[inline, static]`

Return a string version of the name of the error constant.

Parameters:

code Error code

References BVTError_GetName().

4.4 BVTSDK::Head Class Reference

A head consists of a group of co-planar transducers which are operated simultaneously to produce (ultimately) a single 2d image.

Public Member Functions

- Head ()
- RetVal GetHeadName (char ∗buffer, int buffer_size)
- RetVal SetRange (float start, float stop)
- float GetStartRange ()
- float GetStopRange ()
- float GetMinimumRange ()
- float GetMaximumRange ()
- int GetFluidType ()
- RetVal SetFluidType (int fluid)
- int GetSoundSpeed ()
- RetVal SetSoundSpeed (int speed)
- float GetGainAdjustment ()
- RetVal SetGainAdjustment (float gain)
- float GetTVGSlope ()
- RetVal SetTVGSlope (float tvg)
- int GetCenterFreq ()
- int GetPingCount ()
- RetVal GetPing (int ping_num, Ping ∗ping)
- RetVal PutPing (const Ping &ping)
- RetVal SetImageRes (int res)
- RetVal SetRangeResolution (float resolution_in_meters)
- RetVal SetImageReqSize (int height, int width)
- RetVal SetRemoteBeamForming (int enable)
- RetVal SetRawDataSending (int enable)
- RetVal SetRemoteImageForming (int enable)
- RetVal SetImageType (int type)
- int GetImageFilterFlags ()
- RetVal SetImageFilterFlags (int flags)
- RetVal SetRangeDataThreshold (unsigned short noise_threshold)
- RetVal SetTxEnable (int enableTx)
- RetVal GetMountingOrientation (double ∗X_axis_degrees, double ∗Y_axis_degrees, double ∗Z_axis_degrees)
- RetVal SetMountingOrientation (double X_axis_degrees, double Y_axis_degrees, double Z_axis_degrees)

Static Public Attributes

- static const int FLUID_SALTWATER = 0
- static const int FLUID_FRESHWATER = 1
- static const int FLUID_OTHER = 2
- static const int RES_OFF = 0
- static const int RES_LOW = 1
- static const int RES_MED = 2
- static const int RES_HIGH = 3
- static const int RES_AUTO = 4
- static const int IMAGE_XY = 0
- static const int IMAGE_RTHETA = 1

4.4.1 Detailed Description

A head consists of a group of co-planar transducers which are operated simultaneously to produce (ultimately) a single 2d image.

The Head object provides functions to change the range window as well as produce pings.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 BVTSDK::Head::Head () `[inline]`

Create the object.

4.4.3 Member Function Documentation

4.4.3.1 RetVal BVTSDK::Head::GetHeadName (char $*$ *buffer*, int *buffer_size*) `[inline]`

Retrieves a copy of a the name of the head.

The head name is currently set only at the factory, and is simply "Head" on many sonars. Only special order sonars with multiple heads are likely to have a different name.

The length of the name has no actual limit, though 80 characters would seem to be more than enough.

Parameters:

buffer buffer to hold the null-terminated string to be passed back

buffer_size total number of characters the passed buffer can hold

References BVTHead_GetHeadName().

4.4.3.2 RetVal BVTSDK::Head::SetRange (float *start*, float *stop*) `[inline]`

Set the range to be acquired.

Parameters:

start Start range in meters

stop Stop range in meters

References BVTHead_SetRange().

4.4.3.3 float BVTSDK::Head::GetStartRange () `[inline]`

Retrieve the current starting range in meters.

References BVTHead_GetStartRange().

4.4.3.4 float BVTSDK::Head::GetStopRange () `[inline]`

Retrieve the current stopping range in meters.

References BVTHead_GetStopRange().

4.4.3.5 float BVTSDK::Head::GetMinimumRange () `[inline]`

Return the minimum allowable range for this sonar.

References BVTHead_GetMinimumRange().

4.4.3.6 float BVTSDK::Head::GetMaximumRange () `[inline]`

Return the maximum allowable range for this sonar.

References BVTHead_GetMaximumRange().

4.4.3.7 int BVTSDK::Head::GetFluidType () `[inline]`

Return the type of water the head is in.

The returned value will correspond to one of the FLUID_∗ constants.

References BVTHead_GetFluidType().

4.4.3.8 RetVal BVTSDK::Head::SetFluidType (int *fluid*) `[inline]`

Set the type of water the head is in.

Parameters:

fluid The fluid type (one of the FLUID_∗ constants)

References BVTHead_SetFluidType().

4.4.3.9 int BVTSDK::Head::GetSoundSpeed () `[inline]`

Return the speed of sound in water.

References BVTHead_GetSoundSpeed().

4.4.3.10 RetVal BVTSDK::Head::SetSoundSpeed (int *speed*) `[inline]`

Set the speed of sound in water.

Parameters:

speed Sound speed in water

References BVTHead_SetSoundSpeed().

4.4.3.11 float BVTSDK::Head::GetGainAdjustment () `[inline]`

Return the additional analog gain in dB.

References BVTHead_GetGainAdjustment().

4.4.3.12 RetVal BVTSDK::Head::SetGainAdjustment (float *gain*) `[inline]`

Set the additional analog gain.

Note: Some systems don't support gain adjustment.

Parameters:

gain Additional analog gain in dB

References BVTHead_SetGainAdjustment().

4.4.3.13 float BVTSDK::Head::GetTVGSlope () `[inline]`

Return the time variable gain in dB/meter.

References BVTHead_GetTVGSlope().

4.4.3.14 RetVal BVTSDK::Head::SetTVGSlope (float *tvg*) `[inline]`

Set the time variable analog gain.

Note: Some systems don't support TVG

Parameters:

tvg Time variable gain in dB/meter

References BVTHead_SetTVGSlope().

4.4.3.15 int BVTSDK::Head::GetCenterFreq () `[inline]`

Return the center frequency(in Hz) of this head.

References BVTHead_GetCenterFreq().

4.4.3.16 int BVTSDK::Head::GetPingCount () [inline]

Return the number of pings 'in' this head A head attached to a file might have more than one ping recorded.

However, a networked sonar will only have a single ping.

References BVTHead_GetPingCount().

4.4.3.17 RetVal BVTSDK::Head::GetPing (int *ping_num*, Ping ∗ *ping*) [inline]

Retrieve a Ping from the Head If ping_num is less than 0, return the next ping in the file.

Otherwise, load the specified ping. If the Head is attached to a 'live' sonar (network), then GetPing always acquires a new ping.

Parameters:

ping_num The ping number to return

ping The returned Ping object

References BVTHead_GetPing().

4.4.3.18 RetVal BVTSDK::Head::PutPing (const Ping & *ping*) [inline]

Write a ping to a file.

Parameters:

ping The ping to write out

References BVTHead_PutPing().

4.4.3.19 RetVal BVTSDK::Head::SetImageRes (int *res*) [inline]

Set the image processing resolution.

The RES_AUTO setting is highly recommended, as it adapts via a formula according to the stop range, whereas the other ranges are fixed values, and should only be used in specialized cases, such as requesting high resolution for longer distances (which will increase the processing time required to create the image). R-Theta images may use either this funtion or SetRangeResolution(), depending on the degree of control required.

Parameters:

res Resolution constant (RES_∗)

References BVTHead_SetImageRes().

4.4.3.20 RetVal BVTSDK::Head::SetRangeResolution (float *resolution_in_meters*) [inline]

Requests a range resolution for R-Theta images.

Also affects the range resolution for RangeData. Note that the exact range resolution may not be available, and the closest resolution will be set. The actual resolution can be obtained by querying the returned image or RangeData object.

Parameters:

resolution_in_meters Range resolution, in meters

References BVTHead_SetRangeResolution().

4.4.3.21 RetVal BVTSDK::Head::SetImageReqSize (int *height*, int *width*) `[inline]`

Set the requested out image size The processing code will attempt to process images at the specified size.

However, it doesn't guarantee that the final output will match this size. NOTE: For R-Theta images, only the width is used, and the image will be created with that exact width. Height will depend on the range, and the resolution set. (See SetImageRes() and SetRangeResolution())

Parameters:

height The requested height

width The requested width

References BVTHead_SetImageReqSize().

4.4.3.22 RetVal BVTSDK::Head::SetRemoteBeamForming (int *enable*) `[inline]`

NOTE: this option is only valid for some sonars, in specific circumstances, and should only be used on advice from the factory.

By default, beamforming is done on the local system. If you call this function with enable=true, the SDK will request that the remote sonar handle the majority of the beamforming. This operation applies the next time GetPing is called.

Parameters:

enable Enable/Disable remote beamformer. (using 1 or 0 to enable or disable)

References BVTHead_SetRemoteBeamForming().

4.4.3.23 RetVal BVTSDK::Head::SetRawDataSending (int *enable*) `[inline]`

NOTE: this option is only valid for some sonars, in specific circumstances, and should only be used on advice from the factory.

By default, the sonar sends data suitable for saving to a .son file. If you are not saving files, AND are recieving processed data thru setting one of the other options, you can call this function with enable=false to reduce the amount of network bandwidth needed. . This operation applies the next time GetPing is called.

Parameters:

enable Enable/Disable raw ping data. (using 1 or 0 to enable or disable)

References BVTHead_SetRawDataSending().

4.4.3.24 RetVal BVTSDK::Head::SetRemoteImageForming (int *enable*) [inline]

NOTE: this option is only valid for some sonars, in specific circumstances, and should only be used on advice from the factory.

By default, image forming is done on the local system. If you call this function with en=true, the SDK will request that the remote sonar handle the image forming. This option is slightly different than remote beam-forming, with all processing done on the sonar, and only the complete image sent over the network connection. This operation applies the next time GetPing is called.

Parameters:

enable Enable/Disable remote image forming. (using 1 or 0 to enable or disable)

References BVTHead_SetRemoteImageForming().

4.4.3.25 RetVal BVTSDK::Head::SetImageType (int *type*) [inline]

Set the type of image output.

NOTE: See SetImageReqSize() for important issues regarding image size.

Parameters:

type Image type constant (IMAGE_∗)

References BVTHead_SetImageType().

4.4.3.26 int BVTSDK::Head::GetImageFilterFlags () [inline]

Return the filter flags.

References BVTHead_GetImageFilterFlags().

4.4.3.27 RetVal BVTSDK::Head::SetImageFilterFlags (int *flags*) [inline]

Set the filter flags.

Parameters:

flags Image filter flags (bit field)

References BVTHead_SetImageFilterFlags().

4.4.3.28 RetVal BVTSDK::Head::SetRangeDataThreshold (unsigned short *noise_threshold*) [inline]

∗∗ EXPERIMENTAL ∗∗ Sets the intensity value below which data is considered to be noise.

Values above this threshold are included in the algorithm which attempts to determine the target edge. This is NOT a simple threshold above which the first value encountered is considered the target edge. This is the same intensity value returned in a MagImage, with a range of an unsigned 16-bit integer. If not set, the default is currently set to 1000.

NOTE: This only applies to specialized BlueView sonars.

Parameters:

noise_threshold Threshold below which is considered noise

References BVTHead_SetRangeDataThreshold().

4.4.3.29 RetVal BVTSDK::Head::SetTxEnable (int *enableTx*) `[inline]`

By default, the sonar transmits pings.

This function allows the user to disable transmit. This can be useful to get background noise measurements. Note that this is not implemented on all sonars.

Parameters:

enableTx If 0, disable the sonar transmission of pings.

References BVTHead_SetTxEnable().

4.4.3.30 RetVal BVTSDK::Head::GetMountingOrientation (double ∗ *X_axis_degrees*, double ∗ *Y_axis_degrees*, double ∗ *Z_axis_degrees*) `[inline]`

∗∗ Preliminary support - may change in later SDK versions ∗∗

Position of the sonar positioner relative to the boat.

Parameters:

X_axis_degrees rotation about X axis

Y_axis_degrees rotation about Y axis

Z_axis_degrees rotation about Z axis

References BVTHead_GetMountingOrientation().

4.4.3.31 RetVal BVTSDK::Head::SetMountingOrientation (double *X_axis_degrees*, double *Y_axis_degrees*, double *Z_axis_degrees*) `[inline]`

∗∗ Preliminary support - may change in later SDK versions ∗∗

Position of the sonar positioner relative to the boat.

Parameters:

X_axis_degrees rotation about X axis

Y_axis_degrees rotation about Y axis

Z_axis_degrees rotation about Z axis

References BVTHead_SetMountingOrientation().

4.4.4 Field Documentation

4.4.4.1 const int BVTSDK::Head::FLUID_SALTWATER = 0 `[static]`

4.4.4.2 const int BVTSDK::Head::FLUID_FRESHWATER = 1 `[static]`

4.4.4.3 const int BVTSDK::Head::FLUID_OTHER = 2 `[static]`

4.4.4.4 const int BVTSDK::Head::RES_OFF = 0 `[static]`

Turn off image processing.

4.4.4.5 const int BVTSDK::Head::RES_LOW = 1 `[static]`

Process at low resolution.

4.4.4.6 const int BVTSDK::Head::RES_MED = 2 `[static]`

Process at med resolution.

4.4.4.7 const int BVTSDK::Head::RES_HIGH = 3 `[static]`

Process at high resolution.

4.4.4.8 const int BVTSDK::Head::RES_AUTO = 4 `[static]`

Select a good res for the current range automatically.

4.4.4.9 const int BVTSDK::Head::IMAGE_XY = 0 `[static]`

Output a cartesian image.

4.4.4.10 const int BVTSDK::Head::IMAGE_RTHETA = 1 `[static]`

Output a Range/Theta image.

4.5 BVTSDK::Logger Class Reference

The SDK is capable of producing a significant amount of debugging output.

Static Public Member Functions

- static void SetLevel (int level)
- static RetVal SetTarget (std::string target)

Static Public Attributes

- static const int NONE = -1
- static const int CRITICAL = 0
- static const int WARNING = 1
- static const int STATUS = 2

4.5.1 Detailed Description

The SDK is capable of producing a significant amount of debugging output.

The Logger object exists to allow the user to control (or disable) the output. Users can also use Logger to add their own custom log messages.

4.5.2 Member Function Documentation

4.5.2.1 static void BVTSDK::Logger::SetLevel (int *level*) `[inline, static]`

Set the log threshold level.

Events above level will be logged to the target.

Parameters:

level Log level

References BVTLogger_SetLevel().

4.5.2.2 static RetVal BVTSDK::Logger::SetTarget (std::string *target*) `[inline, static]`

The log target can be a filename, "stdout", "stderr", or "null".

If null is specified, log output is disabled.

Parameters:

target File/device to log output to

References BVTLogger_SetTarget().

4.5.3 Field Documentation

4.5.3.1 const int BVTSDK::Logger::NONE = -1 `[static]`

Don't log anything.

4.5.3.2 const int BVTSDK::Logger::CRITICAL = 0 `[static]`

Log critical events.

4.5.3.3 const int BVTSDK::Logger::WARNING = 1 `[static]`

4.5.3.4 const int BVTSDK::Logger::STATUS = 2 `[static]`

4.6 BVTSDK::MagImage Class Reference

MagImage is short for MagnitudeImage.

Public Member Functions

- MagImage ()
- ∼MagImage ()
- int GetHeight ()
- int GetWidth ()
- double GetRangeResolution ()
- double GetBearingResolution ()
- int GetOriginRow ()
- int GetOriginCol ()
- double GetPixelRange (int row, int col)
- double GetPixelRelativeBearing (int row, int col)
- double GetFOVMinAngle ()
- double GetFOVMaxAngle ()
- unsigned short GetPixel (int row, int col)
- unsigned short ∗ GetRow (int row)
- unsigned short ∗ GetBits ()
- RetVal CopyBits (unsigned short ∗data, unsigned int len)
- RetVal SavePGM (std::string file_name)

4.6.1 Detailed Description

MagImage is short for MagnitudeImage.

It provides access to a 2d image where each pixel is intensity of the return from a particular point on a plane emanating from the head. It can be thought of as a 16bit grey-scale image.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 BVTSDK::MagImage::MagImage () `[inline]`

Create the object.

4.6.2.2 BVTSDK::MagImage::∼MagImage () `[inline]`

Destroy the object.

References BVTMagImage_Destroy().

4.6.3 Member Function Documentation

4.6.3.1 int BVTSDK::MagImage::GetHeight () `[inline]`

Return the height (in pixels) of this image.

References BVTMagImage_GetHeight().

4.6.3.2 int BVTSDK::MagImage::GetWidth () `[inline]`

Return the width (in pixels) of this image.

References BVTMagImage_GetWidth().

4.6.3.3 double BVTSDK::MagImage::GetRangeResolution () `[inline]`

Return the range resolution of this image.

The resolution is returned in meters per pixel.

References BVTMagImage_GetRangeResolution().

4.6.3.4 double BVTSDK::MagImage::GetBearingResolution () `[inline]`

Only valid for R-Theta images.

Returns the bearing resolution, in degrees per column.

References BVTMagImage_GetBearingResolution().

4.6.3.5 int BVTSDK::MagImage::GetOriginRow () `[inline]`

Retrieve the image row of the origin.

In most cases the origin will be outside of the image boundaries. The origin is the 'location' (in pixels) of the sonar head in image plane

References BVTMagImage_GetOriginRow().

4.6.3.6 int BVTSDK::MagImage::GetOriginCol () `[inline]`

Retrieve the image column of the origin.

In most cases the origin will be outside of the image boundaries. The origin is the 'location' (in pixels) of the sonar head in image plane

References BVTMagImage_GetOriginCol().

4.6.3.7 double BVTSDK::MagImage::GetPixelRange (int *row*, int *col*) `[inline]`

Retrieve the range (from the sonar head) of the specified pixel.

Parameters:

- ***row*** Origin row
- ***col*** Origin col

References BVTMagImage_GetPixelRange().

4.6.3.8 double BVTSDK::MagImage::GetPixelRelativeBearing (int *row*, int *col*) `[inline]`

Retrieve the bearing relative to the sonar head of the specified pixel.

Parameters:

- ***row*** Origin row
- ***col*** Origin col

References BVTMagImage_GetPixelRelativeBearing().

4.6.3.9 double BVTSDK::MagImage::GetFOVMinAngle () `[inline]`

Return the minimum angle for the sonar's imaging field of view.

The angle is returned in degrees.

References BVTMagImage_GetFOVMinAngle().

4.6.3.10 double BVTSDK::MagImage::GetFOVMaxAngle () `[inline]`

Return the maximum angle for the sonar's imaging field of view.

The angle is returned in degrees.

References BVTMagImage_GetFOVMaxAngle().

4.6.3.11 unsigned short BVTSDK::MagImage::GetPixel (int *row*, int *col*) `[inline]`

Return the value of the pixel at (row, col).

Parameters:

- ***row*** Requested row
- ***col*** Requested col

References BVTMagImage_GetPixel().

4.6.3.12 unsigned short* BVTSDK::MagImage::GetRow (int *row*) `[inline]`

Return a pointer to a row of pixels.

Parameters:

- ***row*** Requested row

References BVTMagImage_GetRow().

4.6.3.13 unsigned short* BVTSDK::MagImage::GetBits () `[inline]`

Return a pointer to the entire image.

The image or organized in Row-Major order (just like C/C++).

References BVTMagImage_GetBits().

4.6.3.14 RetVal BVTSDK::MagImage::CopyBits (unsigned short * *data*, unsigned int *len*) `[inline]`

Copy the raw image data to the user specified buffer.

See GetBits for more info.

Parameters:

data Pointer to a valid buffer

len The size of the buffer pointed to by data in pixels NOT bytes.

References BVTMagImage_CopyBits().

4.6.3.15 RetVal BVTSDK::MagImage::SavePGM (std::string *file_name*) `[inline]`

Save the image in PGM (PortableGreyMap) format.

Note that few programs actually support loading a 16bit PGM.

Parameters:

file_name File name to save to

References BVTMagImage_SavePGM().

4.7 BVTSDK::NavData Class Reference

NavData contains various types of user-accessible navigation parameter, which can be saved to and retrieved from a sonar file on a per ping basis.

Public Member Functions

- NavData ()
- ∼NavData ()
- RetVal Clone (const NavData &navdata_to_clone)
- RetVal GetLatitude (double ∗degrees)
- RetVal SetLatitude (double degrees)
- RetVal GetLongitude (double ∗degrees)
- RetVal SetLongitude (double degrees)
- RetVal GetHorizontalPrecisionError (float ∗error_meters)
- RetVal SetHorizontalPrecisionError (float error_meters)
- RetVal GetHeight (float ∗meters_above_geoid)
- RetVal SetHeight (float meters_above_geoid)
- RetVal GetVerticalPrecisionError (float ∗error_meters)
- RetVal SetVerticalPrecisionError (float error_meters)
- RetVal GetDepth (float ∗meters_below_surface)
- RetVal SetDepth (float meters_below_surface)
- RetVal GetAltitude (float ∗meters_above_bottom)
- RetVal SetAltitude (float meters_above_bottom)
- RetVal GetHeading (float ∗degrees_true)
- RetVal SetHeading (float degrees_true)
- RetVal GetHeadingVelocity (float ∗meters_per_second)
- RetVal SetHeadingVelocity (float meters_per_second)
- RetVal GetCourseOverGround (float ∗degrees_true)
- RetVal SetCourseOverGround (float degrees_true)
- RetVal GetSpeedOverGround (float ∗meters_per_second)
- RetVal SetSpeedOverGround (float meters_per_second)
- RetVal GetPitchAngle (float ∗degrees_bow_up)
- RetVal SetPitchAngle (float degrees_bow_up)
- RetVal GetRollAngle (float ∗degrees_port_up)
- RetVal SetRollAngle (float degrees_port_up)
- RetVal GetYawAngle (float ∗degrees_bow_to_starboard)
- RetVal SetYawAngle (float degrees_bow_to_starboard)
- RetVal GetPitchRate (float ∗degrees_per_second)
- RetVal SetPitchRate (float degrees_per_second)
- RetVal GetRollRate (float ∗degrees_per_second)
- RetVal SetRollRate (float degrees_per_second)
- RetVal GetYawRate (float ∗degrees_per_second)
- RetVal SetYawRate (float degrees_per_second)
- RetVal GetAccelerationX (float ∗accel_mg)
- RetVal SetAccelerationX (float accel_mg)
- RetVal GetAccelerationY (float ∗accel_mg)

- RetVal SetAccelerationY (float accel_mg)
- RetVal GetAccelerationZ (float ∗accel_mg)
- RetVal SetAccelerationZ (float accel_mg)
- RetVal GetOffsetNorth (double ∗meters)
- RetVal SetOffsetNorth (double meters)
- RetVal GetOffsetEast (double ∗meters)
- RetVal SetOffsetEast (double meters)
- RetVal GetOffsetIsFromLatLongFlag (int ∗is_true)
- RetVal SetOffsetIsFromLatLongFlag (int is_true)
- std::string GetUserNavString ()
- RetVal SetUserNavString (std::string string_in)
- double GetTimestamp ()
- RetVal SetTimestamp (double sec)

4.7.1 Detailed Description

NavData contains various types of user-accessible navigation parameter, which can be saved to and retrieved from a sonar file on a per ping basis.

The NavData objects can be created and destroyed as needed. When the ping functions are called to get or put the data, the data is copied. This allows NavData objects to be pre-allocated and filled from various instrument sources. It also allows the data to be copied from one NavData object to the other. NOTE: NavData changes will only be saved to a sonar of type FILE.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 BVTSDK::NavData::NavData () `[inline]`

Create the object.

References BVTNavData_Create().

4.7.2.2 BVTSDK::NavData::∼NavData () `[inline]`

Destroy the object.

References BVTNavData_Destroy().

4.7.3 Member Function Documentation

4.7.3.1 RetVal BVTSDK::NavData::Clone (const NavData & *navdata_to_clone*) `[inline]`

Clones the data from the passed NavData object to this object.

Both objects must have already been created.

Parameters:

navdata_to_clone existing NavData object to copy from

References BVTNavData_Clone().

4.7.3.2 RetVal BVTSDK::NavData::GetLatitude (double ∗ *degrees*) [inline]

Returns the latitude.

If no latitude was stored, returns BVT_NAV_NO_DATA.

Parameters:

degrees latitude in degrees

References BVTNavData_GetLatitude().

4.7.3.3 RetVal BVTSDK::NavData::SetLatitude (double *degrees*) [inline]

Stores the latitude, as a signed floating point number of degrees.

Latitudes in the Western hemisphere are expressed as negative numbers.

Parameters:

degrees latitude in degrees

References BVTNavData_SetLatitude().

4.7.3.4 RetVal BVTSDK::NavData::GetLongitude (double ∗ *degrees*) [inline]

Returns the longitude.

If no longitude was stored, returns BVT_NAV_NO_DATA.

Parameters:

degrees longitude in degrees

References BVTNavData_GetLongitude().

4.7.3.5 RetVal BVTSDK::NavData::SetLongitude (double *degrees*) [inline]

Stores the longitude, as a signed floating point number of degrees.

Longitudes in the Southern hemisphere are expressed as negative numbers.

Parameters:

degrees longitude in degrees

References BVTNavData_SetLongitude().

4.7.3.6 RetVal BVTSDK::NavData::GetHorizontalPrecisionError (float ∗ *error_meters*) [inline]

Returns the estimated horizontal error (see next function for details).

If none was stored, returns BVT_NAV_NO_DATA.

Parameters:

error_meters potential error distance, in meters

References BVTNavData_GetHorizontalPrecisionError().

4.7.3.7 RetVal BVTSDK::NavData::SetHorizontalPrecisionError (float *error_meters*) `[inline]`

Stores the estimated possible horizontal error.

This is primarily (but not exclusively) intended for GPS systems, where there may be some doubt as to the quality of the position fix. HDOP is common, but not used here, as that is a unitless measure and varies between different manufacturers and models.

The idea is to use whatever calculations are appropriate for the local navigation system, and store a possible error value in meters. Some GPS units will attempt to give this directly. (for example, the HPE field in the PGRME sentence, supplied by some Garmin units.) In the case of large errors, or old data, it may be best to either not store a position, or not store new data. (also consider using the NavData time parameter to store the time of last fix, which can then be compared to the ping time when the data is read back to determine the age of the GPS reading.)

Parameters:

error_meters potential error distance, in meters

References BVTNavData_SetHorizontalPrecisionError().

4.7.3.8 RetVal BVTSDK::NavData::GetHeight (float ∗ *meters_above_geoid*) `[inline]`

Returns the height above mean sea level.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

meters_above_geoid height in floating point meters

References BVTNavData_GetHeight().

4.7.3.9 RetVal BVTSDK::NavData::SetHeight (float *meters_above_geoid*) `[inline]`

Store the height above Mean Sea Level (usually the EGM96 geoid)).

Parameters:

meters_above_geoid altitude in floating point meters

References BVTNavData_SetHeight().

4.7.3.10 RetVal BVTSDK::NavData::GetVerticalPrecisionError (float ∗ *error_meters*) `[inline]`

Returns the estimated vertical error (see next function for details).

If none was stored, returns BVT_NAV_NO_DATA.

Parameters:

error_meters potential error distance, in meters

References BVTNavData_GetVerticalPrecisionError().

4.7.3.11 RetVal BVTSDK::NavData::SetVerticalPrecisionError (float *error_meters*) [inline]

Stores the estimated possible vertical error (height) above Mean Sea Level (EGM96 geoid).

For other notes, see the functions or Horizontal Precision Error, above.

Parameters:

error_meters potential error distance, in meters

References BVTNavData_SetVerticalPrecisionError().

4.7.3.12 RetVal BVTSDK::NavData::GetDepth (float ∗ *meters_below_surface*) [inline]

Returns the depth.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

meters_below_surface depth in floating point meters

References BVTNavData_GetDepth().

4.7.3.13 RetVal BVTSDK::NavData::SetDepth (float *meters_below_surface*) [inline]

Store the depth.

Parameters:

meters_below_surface depth in floating point meters

References BVTNavData_SetDepth().

4.7.3.14 RetVal BVTSDK::NavData::GetAltitude (float ∗ *meters_above_bottom*) [inline]

Returns the altitude.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

meters_above_bottom altitude in floating point meters

References BVTNavData_GetAltitude().

4.7.3.15 RetVal BVTSDK::NavData::SetAltitude (float *meters_above_bottom*) [inline]

Store the altitude.

Parameters:

meters_above_bottom altitude in floating point meters

References BVTNavData_SetAltitude().

4.7.3.16 RetVal BVTSDK::NavData::GetHeading (float ∗ *degrees_true*) [inline]

Returns the heading relative to True North.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

degrees_true True heading, in degrees

References BVTNavData_GetHeading().

4.7.3.17 RetVal BVTSDK::NavData::SetHeading (float *degrees_true*) [inline]

Store the heading relative to True North.

Parameters:

degrees_true True heading, in degrees

References BVTNavData_SetHeading().

4.7.3.18 RetVal BVTSDK::NavData::GetHeadingVelocity (float ∗ *meters_per_second*) [inline]

Returns the velocity along the heading.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

meters_per_second velocity, in meters per second

References BVTNavData_GetHeadingVelocity().

4.7.3.19 RetVal BVTSDK::NavData::SetHeadingVelocity (float *meters_per_second*) [inline]

Store the velocity along the heading.

Parameters:

meters_per_second velocity, in meters per second

References BVTNavData_SetHeadingVelocity().

4.7.3.20 RetVal BVTSDK::NavData::GetCourseOverGround (float ∗ *degrees_true*) [inline]

Returns the course over ground (true).

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

degrees_true course over ground, true, in degrees

References BVTNavData_GetCourseOverGround().

4.7.3.21 RetVal BVTSDK::NavData::SetCourseOverGround (float *degrees_true*) [inline]

Store the course over ground, true.

Parameters:

degrees_true course over ground, true, in degrees

References BVTNavData_SetCourseOverGround().

4.7.3.22 RetVal BVTSDK::NavData::GetSpeedOverGround (float ∗ *meters_per_second*) [inline]

Returns the speed over ground.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

meters_per_second speed over ground, in meters per second

References BVTNavData_GetSpeedOverGround().

4.7.3.23 RetVal BVTSDK::NavData::SetSpeedOverGround (float *meters_per_second*) [inline]

Store the speed over ground.

Parameters:

meters_per_second speed over ground, in meters per second

References BVTNavData_SetSpeedOverGround().

4.7.3.24 RetVal BVTSDK::NavData::GetPitchAngle (float ∗ *degrees_bow_up*) [inline]

Get the pitch angle.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

degrees_bow_up pitch angle, in floating point degrees, bow up is positive

References BVTNavData_GetPitchAngle().

4.7.3.25 RetVal BVTSDK::NavData::SetPitchAngle (float *degrees_bow_up*) [inline]

Store the pitch angle.

Parameters:

degrees_bow_up pitch angle, in floating point degrees, bow up is positive

References BVTNavData_SetPitchAngle().

4.7.3.26 RetVal BVTSDK::NavData::GetRollAngle (float ∗ *degrees_port_up*) `[inline]`

Get the roll angle.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

degrees_port_up roll angle, in floating point degrees, port side up is positive

References BVTNavData_GetRollAngle().

4.7.3.27 RetVal BVTSDK::NavData::SetRollAngle (float *degrees_port_up*) `[inline]`

Store the roll angle.

Parameters:

degrees_port_up roll angle, in floating point degrees, port side up is positive

References BVTNavData_SetRollAngle().

4.7.3.28 RetVal BVTSDK::NavData::GetYawAngle (float ∗ *degrees_bow_to_starboard*) `[inline]`

Get the roll angle (but see notes with SetYawAngle()).

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

References BVTNavData_GetYawAngle().

4.7.3.29 RetVal BVTSDK::NavData::SetYawAngle (float *degrees_bow_to_starboard*) `[inline]`

Store the yaw angle.

NOTE: This is NOT the same as the Heading field. Heading is for the normal navigation use of Heading, often from a compass. This field is intended to store raw data from other research instruments, in case you need another storage spot. To keep everyone using the fields the same way so that files can be interchanged, please use SetHeading for the normal heading, and SetYawAngle() only for special uses.

References BVTNavData_SetYawAngle().

4.7.3.30 RetVal BVTSDK::NavData::GetPitchRate (float ∗ *degrees_per_second*) `[inline]`

Returns the rate of pitch change.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

degrees_per_second rate of pitch change in degrees per second

References BVTNavData_GetPitchRate().

4.7.3.31 RetVal BVTSDK::NavData::SetPitchRate (float *degrees_per_second*) [inline]

Store the the rate of pitch change.

Parameters:

degrees_per_second rate of pitch change in degrees per second

References BVTNavData_SetPitchRate().

4.7.3.32 RetVal BVTSDK::NavData::GetRollRate (float ∗ *degrees_per_second*) [inline]

Returns the rate of roll change.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

degrees_per_second rate of roll change in degrees per second

References BVTNavData_GetRollRate().

4.7.3.33 RetVal BVTSDK::NavData::SetRollRate (float *degrees_per_second*) [inline]

Store the the rate of change in roll.

Parameters:

degrees_per_second rate of roll change in floating point degrees per second

References BVTNavData_SetRollRate().

4.7.3.34 RetVal BVTSDK::NavData::GetYawRate (float ∗ *degrees_per_second*) [inline]

Returns the rate of change in yaw (heading).

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

degrees_per_second rate of yaw change in floating point degrees per second

References BVTNavData_GetYawRate().

4.7.3.35 RetVal BVTSDK::NavData::SetYawRate (float *degrees_per_second*) [inline]

Store the rate of change in yaw (heading).

Parameters:

degrees_per_second rate of yaw change in floating point degrees per second

References BVTNavData_SetYawRate().

4.7.3.36 RetVal BVTSDK::NavData::GetAccelerationX (float ∗ *accel_mg*) [inline]

Gets the stored acceleration along the X axis.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

accel_mg acceleration along X axis, in milli-g

References BVTNavData_GetAccelerationX().

4.7.3.37 RetVal BVTSDK::NavData::SetAccelerationX (float *accel_mg*) [inline]

Sets the stored acceleration along the X axis.

Intended to store raw values of accelerometers.

Parameters:

accel_mg acceleration along X axis, in milli-g

References BVTNavData_SetAccelerationX().

4.7.3.38 RetVal BVTSDK::NavData::GetAccelerationY (float ∗ *accel_mg*) [inline]

Gets the stored acceleration along the Y axis.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

accel_mg acceleration along Y axis, in milli-g

References BVTNavData_GetAccelerationY().

4.7.3.39 RetVal BVTSDK::NavData::SetAccelerationY (float *accel_mg*) [inline]

Sets the stored acceleration along the Y axis.

Intended to store raw values of accelerometers.

Parameters:

accel_mg acceleration along Y axis, in milli-g

References BVTNavData_SetAccelerationY().

4.7.3.40 RetVal BVTSDK::NavData::GetAccelerationZ (float ∗ *accel_mg*) [inline]

Gets the stored acceleration along the Z axis.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

accel_mg acceleration along Z axis, in milli-g

References BVTNavData_GetAccelerationZ().

4.7.3.41 RetVal BVTSDK::NavData::SetAccelerationZ (float *accel_mg*) [inline]

Sets the stored acceleration along the Z axis.

Intended to store raw values of accelerometers.

Parameters:
accel_mg acceleration along Z axis, in milli-g

References BVTNavData_SetAccelerationZ().

4.7.3.42 RetVal BVTSDK::NavData::GetOffsetNorth (double ∗ *meters*) [inline]

Returns the offset, to the north, from a user-defined fixed point.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:
meters offset to the north, in floating point meters

References BVTNavData_GetOffsetNorth().

4.7.3.43 RetVal BVTSDK::NavData::SetOffsetNorth (double *meters*) [inline]

Store the offset from a user-defined fixed point.

Parameters:
meters offset to the north, in floating point meters

References BVTNavData_SetOffsetNorth().

4.7.3.44 RetVal BVTSDK::NavData::GetOffsetEast (double ∗ *meters*) [inline]

Returns the offset, to the east, from a user-defined fixed point.

If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:
meters offset to the east, in floating point meters

References BVTNavData_GetOffsetEast().

4.7.3.45 RetVal BVTSDK::NavData::SetOffsetEast (double *meters*) [inline]

Store the offset from a user-defined fixed point.

Parameters:
meters offset to the east, in floating point meters

References BVTNavData_SetOffsetEast().

4.7.3.46 RetVal BVTSDK::NavData::GetOffsetIsFromLatLongFlag (int ∗ *is_true*) [inline]

Gets a flag value which indicates if the East and North offset values are from the stored Latitude and Longitude.

(see more below...) If no value was stored for this ping, returns BVT_NAV_NO_DATA.

Parameters:

is_true either 1 or 0, to indicate true or false, respectively

References BVTNavData_GetOffsetIsFromLatLongFlag().

4.7.3.47 RetVal BVTSDK::NavData::SetOffsetIsFromLatLongFlag (int *is_true*) [inline]

Sets a flag to indicate if the East and North offset values are from the stored Latitude and Longitude.

If so, then software reading the file will know to adjust appropriately. If from some special location, perhaps the UserNavString could be used to indicate the reference point? If nothing is stored, then the value is assumed to be false.

Parameters:

is_true either 1 or 0, to indicate true or false, respectively

References BVTNavData_SetOffsetIsFromLatLongFlag().

4.7.3.48 std::string BVTSDK::NavData::GetUserNavString () [inline]

Returns the stored user string, in null-terminated form.

If no string was stored, returns a null string.

References BVTNavData_GetUserNavString().

4.7.3.49 RetVal BVTSDK::NavData::SetUserNavString (std::string *string_in*) [inline]

Stores a user-defined string related to navigation.

It is highly recommended to store some identifier such that the data is recognizable as you own.

Examples of possible uses might be to store locations based on coordinate systems other than GPS, indicating information about the use of the north and east offset parameters, additional fields from a GPS receiver, or any other information which might change dynamically.

Parameters:

string_in string to be stored, null terminated, max length 80 chars

References BVTNavData_SetUserNavString().

4.7.3.50 double BVTSDK::NavData::GetTimestamp () [inline]

Return the ping's timestamp in seconds since 00:00:00 UTC, January 1, 1970 Pings are timestamped using a standard UNIX time stamp.

This is a similar value to that returned by the time() C standard library function. In fact, the only difference is the addition of fractional seconds.

References BVTNavData_GetTimestamp().

4.7.3.51 RetVal BVTSDK::NavData::SetTimestamp (double *sec*) [inline]

Set the NavData's internal time stamp.

See GetTimestamp() for more information.

Parameters:

sec Timestamp in seconds since 00:00:00 UTC, January 1, 1970

References BVTNavData_SetTimestamp().

4.8 BVTSDK::Ping Class Reference

As its name implies, the Ping object represents the return from a single ping on a particular head.

Public Member Functions

- Ping ()
- ∼Ping ()
- int GetPingNumber ()
- double GetTimestamp ()
- int GetTimeZoneOffset ()
- RetVal SetTimestamp (double sec)
- RetVal GetImage (MagImage ∗img)
- RetVal GetRangeData (RangeData ∗data)
- float GetSonarPitchAngle ()
- float GetSonarRollAngle ()
- RetVal GetNavDataCopy (NavData ∗nav_data)
- RetVal PutNavData (const NavData &nav_data)
- RetVal GetVideoFrame (unsigned char ∗∗frame, int ∗height, int ∗width, int ∗length, int ∗type)
- RetVal PutVideoFrameJPEG (const unsigned char ∗frame, int height, int width, int length)
- RetVal GetPositionerOrientation (double ∗X_axis_degrees, double ∗Y_axis_degrees, double ∗Z_axis_degrees)
- RetVal SetPositionerOrientation (double X_axis_degrees, double Y_axis_degrees, double Z_axis_degrees)

Static Public Attributes

- static const int VIDEO_RGB = 0
- static const int VIDEO_JPEG = 1

4.8.1 Detailed Description

As its name implies, the Ping object represents the return from a single ping on a particular head.

GetImage is the most important function in Ping as it does whatever processing is necessary to convert the ping to an image.

Each ping may have a video frame associated with it, and saved in the same file. These images are typically from a video camera mounted near the sonar, such as on a ROV.

Each ping may also store navigation data to indicate the position and orientation of the vehicle at the time of the ping.

A ping is essentially a container for data. As such, after you get a ping from the head and extract the data (or save it to a file), it is necessary to destroy the ping object to free up memory. In the future the Ping object will expose additional information about the ping, such as the orientation of the head when it was generated.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 BVTSDK::Ping::Ping () `[inline]`

Create the object.

4.8.2.2 BVTSDK::Ping::∼Ping () `[inline]`

Destroy the object.

References BVTPing_Destroy().

4.8.3 Member Function Documentation

4.8.3.1 int BVTSDK::Ping::GetPingNumber () `[inline]`

Return the ping number.

Ping numbers only have meaning if the ping came from a file.

References BVTPing_GetPingNumber().

4.8.3.2 double BVTSDK::Ping::GetTimestamp () `[inline]`

Return the ping's timestamp in seconds since 00:00:00, January 1, 1970 This is local time.

Pings are timestamped using a standard UNIX time stamp. This is a similar value to that returned by the time() C standard library function. In fact, the only difference is the addition of fractional seconds.

References BVTPing_GetTimestamp().

4.8.3.3 int BVTSDK::Ping::GetTimeZoneOffset () `[inline]`

Return the ping's timestamp's offset in seconds from UTC time.

Add this value to that returned by GetTimestamp() to obtain UTC time.

References BVTPing_GetTimeZoneOffset().

4.8.3.4 RetVal BVTSDK::Ping::SetTimestamp (double *sec*) `[inline]`

Set the ping's internal time stamp.

See GetTimestamp() for more information. Note: BlueView strongly recommends that users NOT directly set the time stamp as it is set internally when the ping is actually initiated. If you are trying to synchronize two systems, it is far better to simply make sure that the system clocks are synchronized, as the ping timestamp is created from the PC's internal clock. Network Time Protocol and GPS sources provide highly accurate ways to accomplish this.

Parameters:

sec Timestamp in seconds since 00:00:00 UTC, January 1, 1970

References BVTPing_SetTimestamp().

4.8.3.5 RetVal BVTSDK::Ping::GetImage (MagImage ∗ *img*) `[inline]`

Retrieve an image of this ping, according to the parameters set in the head used to get this ping.

See Head and MagImage documentation for more details.

Parameters:

img Output image

References BVTPing_GetImage().

4.8.3.6 RetVal BVTSDK::Ping::GetRangeData (RangeData ∗ *data*) `[inline]`

∗∗ EXPERIMENTAL ∗∗ See RangeData class for more details, and the Head's SetRangeDataThreshold function.

NOTE: This only applies to specialized BlueView sonars.

Parameters:

data set of ranges at angles for this ping

References BVTPing_GetRangeData().

4.8.3.7 float BVTSDK::Ping::GetSonarPitchAngle () `[inline]`

Get the pitch angle, in floating point degrees.

Bow up is positive. Some BlueView sonar have an internal tilt sensor that is capable of reporting the pitch angle. If the sonar doesn't have the sensor, this function returns 0

References BVTPing_GetSonarPitchAngle().

4.8.3.8 float BVTSDK::Ping::GetSonarRollAngle () `[inline]`

Get the roll angle, in floating point degrees.

Port side up is positive. Some BlueView sonar have an internal tilt sensor that is capable of reporting the roll angle. If the sonar doesn't have the sensor, this function returns 0.

References BVTPing_GetSonarRollAngle().

4.8.3.9 RetVal BVTSDK::Ping::GetNavDataCopy (NavData ∗ *nav_data*) `[inline]`

Retrieves a copy of the navigation data stored with this ping.

Note that the data is copied out of the ping into the local NavData object, a pointer to internal data is not returned. Thus, the NavData object may be used after the Ping is destroyed.

References BVTPing_GetNavDataCopy().

4.8.3.10 RetVal BVTSDK::Ping::PutNavData (const NavData & *nav_data*) [inline]

Stores a copy of the navigation data with the other ping data, so the data will be saved if the ping is saved to a file.

References BVTPing_PutNavData().

4.8.3.11 RetVal BVTSDK::Ping::GetVideoFrame (unsigned char ∗∗ *frame*, int ∗ *height*, int ∗ *width*, int ∗ *length*, int ∗ *type*) [inline]

Returns the video frame associated with this ping.

The video frame may be in any of the supported image formats. Some image formats may already contain parameters such as height and width (and more), but valid pointers must be passed in anyway. The same pointer can be passed in for multiple parameters, if those parameters will not be used. However, they are provided both for formats which do not have embedded size information, and so that the display window may be created and/or sized without parsing the image data.

NOTE: This function will return BVT_NO_VIDEO_FRAME if there is no video frame stored for the ping.

WARNING: The data buffer must NOT be accessed after the ping object is destroyed, as the pointer will no longer point to valid data and will likely crash your application! So copy off the data before destroying the Ping object.

The single value pointers must be pointers to allocated data, not just pointer types. For example:

int height, width, length, type, retval;

int ∗ frame_ptr;

retval = GetVideoFrame(frame_ptr, &height, &width, &length, &type);

Parameters:

frame Pointer to a pointer to the image data to be returned

height Pointer to return the uncompressed height of the image, in pixels

width Pointer to return the uncompressed width of the image, in pixels

length Pointer to return the actual size of the data buffer returned, in bytes, which may include additional metadata for some image types

type pointer to return the type of image returned: FRAME_RGB or FRAME_JPEG

References BVTPing_GetVideoFrame().

4.8.3.12 RetVal BVTSDK::Ping::PutVideoFrameJPEG (const unsigned char ∗ *frame*, int *height*, int *width*, int *length*) [inline]

Store a JPEG image to save with this ping.

Note that the height and width values will simply be stored and available to read when the frame is retrieved. These have no effect on the actual image size (the image will not be resized). The length however is very important, as it determines how far from the passed image pointer data will be read. An incorrect length could result in an application crash.

Parameters:

frame Pointer to a single video frame

height Uncompressed height of the image, in pixels

width Uncompressed width of the image, in pixels

length Actual number of bytes being passed in

References BVTPing_PutVideoFrameJPEG().

4.8.3.13 RetVal BVTSDK::Ping::GetPositionerOrientation (double ∗ *X_axis_degrees*, double ∗ *Y_axis_degrees*, double ∗ *Z_axis_degrees*) [inline]

∗∗ Preliminary support - may change in later SDK versions ∗∗

Get orientation of the sonar head relative to the positioner.

Effectively the raw position data from a ROS pan/tilt unit.

Parameters:

X_axis_degrees rotation about X axis

Y_axis_degrees rotation about Y axis

Z_axis_degrees rotation about Z axis

References BVTPing_GetPositionerOrientation().

4.8.3.14 RetVal BVTSDK::Ping::SetPositionerOrientation (double *X_axis_degrees*, double *Y_axis_degrees*, double *Z_axis_degrees*) [inline]

∗∗ Preliminary support - may change in later SDK versions ∗∗

Set orientation of the sonar head relative to the positioner.

Effectively the raw position data from a ROS pan/tilt unit.

Parameters:

X_axis_degrees rotation about X axis

Y_axis_degrees rotation about Y axis

Z_axis_degrees rotation about Z axis

References BVTPing_SetPositionerOrientation().

4.8.4 Field Documentation

4.8.4.1 const int BVTSDK::Ping::VIDEO_RGB = 0 [static]

Video frame is raw RGB (RGBRGB.

..)

4.8.4.2 const int BVTSDK::Ping::VIDEO_JPEG = 1 [static]

Video frame is a JPEG image.

4.9 BVTSDK::RangeData Class Reference

∗∗ EXPERIMENTAL ∗∗ This functionality is still under development! ∗∗∗ RangeData is a set of ranges from the sonar head, at various angles from the sonar head.

Public Member Functions

- RangeData ()
- int GetCount ()
- double GetRangeResolution ()
- double GetBearingResolution ()
- float GetFOVMinAngle ()
- float GetFOVMaxAngle ()
- RetVal CopyRangeValues (float ∗ranges, int number_of_ranges)
- float GetRangeValue (int index)
- float GetBearingValue (int index)
- int GetColorImagePixelX (int rangeDataIndex, const ColorImage &image)
- int GetColorImagePixelY (int rangeDataIndex, const ColorImage &image)

Static Public Attributes

- static const int MAX_RANGE = 999

4.9.1 Detailed Description

∗∗ EXPERIMENTAL ∗∗ This functionality is still under development! ∗∗∗ RangeData is a set of ranges from the sonar head, at various angles from the sonar head.

For each angle, the range, bearing and intensity of the return beam at that range is stored. NOTE: RangeData only applies to specialized BlueView sonars, and has no use for our standard imaging sonars.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 BVTSDK::RangeData::RangeData () `[inline]`

Create the object.

4.9.3 Member Function Documentation

4.9.3.1 int BVTSDK::RangeData::GetCount () `[inline]`

Returns the number of range values stored for this ping.

References BVTRangeData_GetCount().

4.9.3.2 double BVTSDK::RangeData::GetRangeResolution () `[inline]`

Returns the resolution of the range values, in meters.

References BVTRangeData_GetRangeResolution().

4.9.3.3 double BVTSDK::RangeData::GetBearingResolution () `[inline]`

Returns the resolution of the bearing stored with each range value.

This is the difference in bearing between each range value in the array.

References BVTRangeData_GetBearingResolution().

4.9.3.4 float BVTSDK::RangeData::GetFOVMinAngle () `[inline]`

Return the minimum angle for the sonar's imaging field of view.

In other words, this is the angle of the first range value, as all angles are "left referenced."The angle is returned in degrees. Note that this may not represent the actual physical field of view of a particular sonar, but does represent the field of view of the data being returned. Some outside values may have range values indicating they are out of range.

References BVTRangeData_GetFOVMinAngle().

4.9.3.5 float BVTSDK::RangeData::GetFOVMaxAngle () `[inline]`

Return the maximum angle for the sonar's imaging field of view.

In other words, this is the angle of the last range value, as all angles are "left referenced."The angle is returned in degrees. Note that this may not represent the actual physical field of view of a particular sonar, but does represent the field of view of the data being returned. Some outside values may have range values indicating they are out of range.

References BVTRangeData_GetFOVMaxAngle().

4.9.3.6 RetVal BVTSDK::RangeData::CopyRangeValues (float ∗ *ranges*, int *number_of_ranges*) `[inline]`

Copies the range values into the user specified buffer.

The buffer must hold the entire number of ranges (See GetCount() above), or an error is returned.

Parameters:

ranges Pointer to a valid buffer of type float.

number_of_ranges Number of values the buffer can hold.

References BVTRangeData_CopyRangeValues().

4.9.3.7 float BVTSDK::RangeData::GetRangeValue (int *index*) `[inline]`

Returns the range from the sonar head, in meters, at a particular index into the array.

NOTE: Check all returned values for validity. If range > BVTRANGEDATA_MAX_RANGE then the range could not be determined within the capabilities of the sonar. Meaning that the closest object at that bearing was either out of view of the sonar, or the threshold was set too high to be detected.

Parameters:

index index into the array of RangeData values

References BVTRangeData_GetRangeValue().

4.9.3.8 float BVTSDK::RangeData::GetBearingValue (int *index*) [inline]

Returns the bearing from the center of the sonar head, in degrees (+/-), at a particular index into the array.

Parameters:

index index into the array of RangeData values

References BVTRangeData_GetBearingValue().

4.9.3.9 int BVTSDK::RangeData::GetColorImagePixelX (int *rangeDataIndex*, const ColorImage & *image*) [inline]

Returns the X coordinate for the pixel in the passed ColorImage, which maps to the range and bearing at the index passed.

This allows placing of the range data on a colorimage, easing analysis of the algorithm used for thresholding.

Parameters:

image ColorImage object where the pixel coordinate is needed

References BVTRangeData_GetColorImagePixelX().

4.9.3.10 int BVTSDK::RangeData::GetColorImagePixelY (int *rangeDataIndex*, const ColorImage & *image*) [inline]

Returns the Y coordinate for the pixel in the passed ColorImage which maps to the range and bearing at the index passed.

(see similar function, above, for more details)

Parameters:

image ColorImage object where the pixel coordinate is needed

References BVTRangeData_GetColorImagePixelY().

4.9.4 Field Documentation

4.9.4.1 const int BVTSDK::RangeData::MAX_RANGE = 999 [static]

Values greater than this indicate no range could be measured.

4.10 BVTSDK::SDK Class Reference

Static Public Member Functions

- static int MajorVersion ()
- static int MinorVersion ()
- static int BuildNumber ()

4.10.1 Member Function Documentation

4.10.1.1 static int BVTSDK::SDK::MajorVersion () `[inline, static]`

Return the major version number of the SDK.

References BVTSDK_MajorVersion().

4.10.1.2 static int BVTSDK::SDK::MinorVersion () `[inline, static]`

Return the minor version of the SDK.

References BVTSDK_MinorVersion().

4.10.1.3 static int BVTSDK::SDK::BuildNumber () `[inline, static]`

Return the build number of the SDK.

References BVTSDK_BuildNumber().

4.11 BVTSDK::Sonar Class Reference

The Sonar object is the top level object in the SDK.

Public Member Functions

- Sonar ()
- ∼Sonar ()
- RetVal Open (std::string type, std::string type_params)
- RetVal CreateFile (std::string file_name, const Sonar &src, std::string create_params)
- int GetFileSize ()
- RetVal GetHead (int head_num, Head ∗head)
- int GetHeadCount ()
- RetVal GetSonarTypeAsString (char ∗buffer, int buffer_size)
- RetVal GetSonarName (char ∗buffer, int buffer_size)
- float GetTemperature ()

4.11.1 Detailed Description

The Sonar object is the top level object in the SDK.

A sonar object embodies communication with a single physical sonar unit, or file. Each sonar contains several heads, which is where most of the functionality is implemented. Sonar also provides a function to create new data files using BlueView's .son format.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 BVTSDK::Sonar::Sonar () `[inline]`

Create the object.

References BVTSonar_Create().

4.11.2.2 BVTSDK::Sonar::∼Sonar () `[inline]`

Destroy the object.

References BVTSonar_Destroy().

4.11.3 Member Function Documentation

4.11.3.1 RetVal BVTSDK::Sonar::Open (std::string *type*, std::string *type_params*) `[inline]`

Open the sonar type 'type' using the specified parameters.

Allowed types (and parameters):

- FILE

  [filename] - Required

- NET

  [host] - Connect to the specified host.

  Parameters:

  type The type of sonar to open

  type_params Various type-specific parameters

References BVTSonar_Open().

4.11.3.2 RetVal BVTSDK::Sonar::CreateFile (std::string *file_name*, const Sonar & *src*, std::string *create_params*) `[inline]`

Create a new data file.

Files are always created by 'cloning' another Sonar object. This ensures that the file receives all the needed setup/configuration data needed to process images.

Parameters:

file_name The filename of the file to be created

src The Sonar object to clone when creating the file

create_params Parameters for (reserved for future use)

References BVTSonar_CreateFile().

4.11.3.3 int BVTSDK::Sonar::GetFileSize () `[inline]`

Gets the size of a file created with CreateFile().

Only works with file type sonars. A networked sonar will return 0, as will a file type sonar if there is no open file associated with it. The return value must be multiplied by 1000 to get the actual file size in bytes.

References BVTSonar_GetFileSize().

4.11.3.4 RetVal BVTSDK::Sonar::GetHead (int *head_num*, Head $*$ *head*) `[inline]`

Retrieve a Head object from the sonar.

Parameters:

head_num The head number to return

head The returned Head object

References BVTSonar_GetHead().

4.11.3.5 int BVTSDK::Sonar::GetHeadCount () `[inline]`

Return the number of heads on this sonar.

References BVTSonar_GetHeadCount().

4.11.3.6 RetVal BVTSDK::Sonar::GetSonarTypeAsString (char * *buffer*, int *buffer_size*) [inline]

Retrieves a copy of a short string with the model of the sonar.

At the time of this writing, 20 characters would easily hold all of the sonar model names.

Parameters:

buffer buffer to hold the null-terminated string to be passed back

buffer_size total number of characters the passed buffer can hold

References BVTSonar_GetSonarTypeAsString().

4.11.3.7 RetVal BVTSDK::Sonar::GetSonarName (char * *buffer*, int *buffer_size*) [inline]

Retrieves a copy of the name of the sonar.

The name is set only via the ProViewer application (at least at this time), or at the factory, and is separate from any BlueView model designations.

The length of the name could be considerably longer than the sonar type, and there is no actual limit, though 80 characters would seem to be more than enough.

Parameters:

buffer buffer to hold the null-terminated string to be passed back

buffer_size total number of characters the passed buffer can hold

References BVTSonar_GetSonarName().

4.11.3.8 float BVTSDK::Sonar::GetTemperature () [inline]

Return the sonar's internal temperature in degrees Celsius If the sonar doesn't have a temp sensor this function returns absolute zero (-273.15).

References BVTSonar_GetTemperature().

Appendix A

Example Source Code

A.1 File Sonar Example

```
/*
 * File Sonar Example
 * Demonstrate opening a file, accessing a head, and retriving a ping.
 * The ping is then processed into an image and saved to a file.
 * Finally, a colormap is loaded and the image is colormapped.
 */

#include <stdio.h>

#include <bvt_sdk.h>

char DataFile[] = "../../data/swimmer.son";

int main( int argc, char *argv[] )
{
    int ret;
    // Create a new BVTSonar Object
    BVTSonar son = BVTSonar_Create();
    if( son == NULL )
    {
        printf("BVTSonar_Create: failed\n");
        return 1;
    }

    // Open the sonar
    if ( argc == 2 )
        strcpy( DataFile, argv[1] );

    ret = BVTSonar_Open(son, "FILE", DataFile);
    if( ret != 0 )
    {
        printf("BVTSonar_Open: ret=%d\n", ret);
        return 1;
    }

    // Make sure we have the right number of heads
    int heads = -1;
    heads = BVTSonar_GetHeadCount(son);
    printf("BVTSonar_GetHeadCount: %d\n", heads);


    // Get the first head
    BVTHead head = NULL;
    ret = BVTSonar_GetHead(son, 0, &head);
    if( ret != 0 )
    {
        printf("BVTSonar_GetHead: ret=%d\n", ret);
        return 1;
    }

    // Check the ping count
    int pings = -1;
    pings = BVTHead_GetPingCount(head);
    printf("BVTHead_GetPingCount: %d\n", pings);

    // Check the min and max range in this file
    printf("BVTHead_GetMinimumRange: %0.2f\n", BVTHead_GetMinimumRange(head) );
    printf("BVTHead_GetMaximumRange: %0.2f\n", BVTHead_GetMaximumRange(head) );


    // Now, get a ping!
    BVTPing ping = NULL;
    ret = BVTHead_GetPing(head, 0, &ping);
    if( ret != 0 )
    {
        printf("BVTHead_GetPing: ret=%d\n", ret);
        return 1;
    }

    // Generate an image from the ping
    BVTMagImage img;
    ret = BVTPing_GetImage(ping, &img);
    if( ret != 0 )
    {
        printf("BVTPing_GetImage: ret=%d\n", ret);
        return 1;
    }

    printf("\n");

    ////////////////////////////////////////////////////////////

    // Check the image height and width out
    int height = BVTMagImage_GetHeight(img);
    printf("BVTMagImage_GetHeight: %d\n", height);
    int width = BVTMagImage_GetWidth(img);
    printf("BVTMagImage_GetWidth: %d\n", width);

    // Save it to a PGM (PortableGreyMap)
    ret = BVTMagImage_SavePGM(img, "img.pgm");
    if( ret != 0 )
    {
        printf("BVTMagImage_SavePGM: ret=%d\n", ret);
        return 1;
    }

    // ////////////////////////////////////////////////////////

    // Build a color mapper
    BVTColorMapper mapper;
    mapper = BVTColorMapper_Create();
    if( mapper == NULL )
    {
        printf("BVTColorMapper_Create: failed\n");
        return 1;
    }

    // Load the bone colormap
    ret = BVTColorMapper_Load(mapper, "../../colormaps/bone.cmap");
    if( ret != 0 )
    {
        printf("BVTColorMapper_Load: ret=%d\n", ret);
        return 1;
    }


    // Perform the colormapping
    BVTColorImage cimg;
    ret = BVTColorMapper_MapImage(mapper, img, &cimg);
    if( ret != 0 )
    {
        printf("BVTColorMapper_MapImage: ret=%d\n", ret);
        return 1;
    }
    printf("\n");

    // ////////////////////////////////////////////////////////
    // Check the image height and width out
    height = BVTColorImage_GetHeight(cimg);
    printf("BVTColorImage_GetHeight: %d\n", height);
    width = BVTColorImage_GetWidth(cimg);
    printf("BVTColorImage_GetWidth: %d\n", width);
    // Save it to a PPM (PortablePixMap)
    ret = BVTColorImage_SavePPM(cimg, "cimg.ppm");
    if( ret != 0 )
    {
        printf("BVTColorImage_SavePPM: ret=%d\n", ret);
        return 1;
    }

    // Clean up
    BVTColorImage_Destroy(cimg);
    BVTMagImage_Destroy(img);
    BVTColorMapper_Destroy(mapper);
    BVTPing_Destroy(ping);
    BVTSonar_Destroy(son);
    return 0;
}
```

Listing A.1: File Sonar Example

A.2 NET Sonar Example

```c
/*
 * NET Sonar Example
 * Connect to a networked device, do a ping, and save it to a .son file
 */

#include <stdio.h>

#include <bvt_sdk.h>

int main( void )
{
    int ret;

    // Create a new BVTSonar Object
    BVTSonar son = BVTSonar_Create();
    if( son == NULL )
    {
        printf("BVTSonar_Create: failed\n");
        return 1;
    }

    // Open the first sonar
    ret = BVTSonar_Open(son, "NET", "192.168.1.45"); // default ip address
    if( ret != 0 )
    {
        printf("BVTSonar_Open: ret=%d\n", ret);
        return 1;
    }

    // Make sure we have the right number of heads
    int heads = BVTSonar_GetHeadCount(son);
    printf("BVTSonar_GetHeadCount: %d\n", heads);

    // Get the first head
    BVTHead head = NULL;
    ret = BVTSonar_GetHead(son, 0, &head);
    if( ret != 0 )
    {
        // some sonars start at head 1 instead of zero...
        ret = BVTSonar_GetHead(son, 1, &head);
        if( ret != 0 )
        {
            printf("BVTSonar_GetHead: ret=%d\n", ret);
            return 1;

        }
    }

    // Set the range window to be 1m to 40m
    BVTHead_SetRange(head, 1, 40);

    // ////////////////////////////////////////////////
    // Now, Create a file to save some pings to
    BVTSonar file = BVTSonar_Create();
    if( file == NULL )
    {
        printf("BVTSonar_Create: failed\n");
        return 1;
    }

    ret = BVTSonar_CreateFile(file, "out.son", son, "");
    if( ret != 0 )
    {
        printf("BVTSonar_CreateFile: ret=%d\n", ret);
        return 1;
    }

    // Request the first head
    BVTHead out_head = NULL;
    ret = BVTSonar_GetHead(file, 0, &out_head);
    if( ret != 0 )
    {
        printf("BVTSonar_GetHead: ret=%d\n", ret);
        return 1;
    }


    // ////////////////////////////////////////////////
    // Now, let's go get some pings!
    int num_pings = 10;
    for (int i=0; i < num_pings; i++)
    {
        BVTPing ping = NULL;
        ret = BVTHead_GetPing(head, -1, &ping);
        if( ret != 0 )
        {
            printf("BVTHead_GetPing: ret=%d\n", ret);
            return 1;
        }

        ret = BVTHead_PutPing(out_head, ping);
        if( ret != 0 )
        {
            printf("BVTHead_PutPing: ret=%d\n", ret);
            return 1;
        }
        BVTPing_Destroy(ping);
    }

    printf("Saved %d pings to out.son file\n", num_pings);
    BVTSonar_Destroy(file);
    BVTSonar_Destroy(son);
    return 0;
}
```

Listing A.2: NET Sonar Example

A.3 MultiHead Sonar Example

```
/*
 * Multi-Head Sonar Example
 * Connect to an ethernet connected sonar, acquire and save pings on all heads
 */

#include <stdio.h>

#include <bvt_sdk.h>

#define MAX_SONAR_HEADS (2)

int main( void )
{
    int ret;

    // Create a new BVTSonar Object
    BVTSonar son = BVTSonar_Create();
    if( son == NULL )
    {
        printf("BVTSonar_Create: failed\n");
        return 1;
    }

    // Open the live sonar
    ret = BVTSonar_Open(son, "NET", "192.168.1.45");
    if( ret != 0 )
    {
        printf("BVTSonar_Open: ret=%s\n", BVTError_GetString(ret) );
        return 1;
    }

    BVTHead son_heads[MAX_SONAR_HEADS];

    int headcount = BVTSonar_GetHeadCount(son);
    printf("The sonar has %d heads.\n", headcount);


    // Get each head, and set the range
    for( int i = 0; i < headcount; i++ )
    {
        // Get the first head
        ret = BVTSonar_GetHead(son, i, &son_heads[i]);
        if( ret != 0 )
        {
            // Some sonars will return more heads than they actually have,
            // though shouldn't be the case with multiple head sonars.
            printf("BVTSonar_GetHead for head %d: ret=%s\n", i, BVTError_GetString(ret)
            headcount = i;
            break;
        }else
        {
            // Set the range window to be 1m to 40m
            BVTHead_SetRange(son_heads[i], 1, 40);
        }
    }


    //////////////////////////////////////////////////
    // Create a file "sonar" to save some pings to
    BVTSonar file = BVTSonar_Create();
    if( file == NULL )
    {
        printf("BVTSonar_Create: failed\n");
        return 1;
    }

    // Create the file, which "clones" the live sonar parameters to the
    // file "sonar."
    ret = BVTSonar_CreateFile(file, "out.son", son, "");
    if( ret != 0 )
    {
        printf("BVTSonar_CreateFile: ret=ret=%s\n", BVTError_GetString(ret) );
        return 1;
    }

    // Get each of the file heads
    BVTHead file_heads[MAX_SONAR_HEADS];
    for ( int i = 0; i < headcount; i++ )
    {
        ret = BVTSonar_GetHead(file, i, &file_heads[i]);
        if( ret != 0 )
        {
            printf("BVTSonar_GetHead: ret=%s\n", BVTError_GetString(ret) );
            return 1;
        }
    }


    //////////////////////////////////////////////////
    // Now, let's go get some pings!
    for ( int ping_num = 0; ping_num < 10; ping_num++ )
    {
        for (int head_num = 0; head_num < headcount; head_num++)
        {
            BVTPing ping = NULL;

            printf("Getting ping %d on head %d\n", ping_num, head_num);

            ret = BVTHead_GetPing (son_heads[head_num], -1, &ping );
            if( ret != 0 )
            {
                printf("BVTHead_GetPing: ret=%s\n", BVTError_GetString(ret) );
                return 1;
            }

            ret = BVTHead_PutPing( file_heads[head_num], ping );
            if( ret != 0 )
            {
                printf("BVTHead_PutPing: ret=%s\n", BVTError_GetString(ret) );
                return 1;
            }
            BVTPing_Destroy(ping);
        }
    }


    BVTSonar_Destroy( file );  // This also closes the file properly.
    BVTSonar_Destroy( son );

    return 0;
}
```

Listing A.3: MultiHead Sonar Example