# Building an Image Frame from ARIS File Data

## File Structure

The ARIS file format was derived from the earlier DIDSON DDF\_04 file format. It comprises a *Master Header* of 1024 bytes, followed by *N* frames of constant size, where the total frame length is the sum of the *Frame Header* (1024 bytes) plus the *Acoustic Data Size*.

Note: File offsets should be calculated with 64-bit values rather than 32-bit in order to read large files correctly.

Many of the parameters in both the *Master Header* and *Frame Header* are legacy parameters from DIDSON files, and are not used at all in ARIS files. They were preserved to allow backward compatibility. Following is the subset of parameters actively used in ARIS files and necessary or useful for image generation and post-processing. Refer to the *ARIS File Format DDF\_05.xlsx* document for a complete listing of all parameters.

### Necessary ARIS Frame Header Parameters

***PingMode***

***SamplePeriod***

***SoundSpeed***

***SamplesPerBeam***

***SampleStartDelay***

***LargeLens***

***ReorderedSamples***

The *Acoustic Data Size* is the product of *NumBeams* \* *SamplesPerBeam* in bytes. *NumBeams* may be calculated from the *PingMode* parameter in the *Frame Header* as:

switch (frameheader->PingMode)

{

case 1:

case 2:

return 48;

case 3:

case 4:

case 5:

return 96;

case 6:

case 7:

case 8:

return 64;

case 9:

case 10:

case 11:

case 12:

return 128;

}

It may also be found in the *Master Header* as *NumRawBeams*.

## Data Ordering

The acoustic data in a frame may or may not be ordered into a [beam][sample] array, depending on the state of the *ReorderedSamples* flag. See ***Appendix A*** for example code to reorder data when this flag is zero. For the rest of this document, it will be assumed that the data has been converted to [beam][sample] order.

## Sample Geometry

A single image frame is built from the array of N beams by M samples. The frame header information is used to calculate the start and end ranges, and the incremental down range sample distance. The cross range distance for any given sample is calculated from the beam spacing table (for angular limits) and linearly increases with sample range.

The *SoundSpeed* (meters/second) parameter in the frame headers has been calculated based on water temperature, depth and salinity [fresh=0, brackish=15, sea=35. Range dependent values may then be calculated from the frame header parameters as follows:

*WindowStart(m)* = *SampleStartDelay(us) \** 1e-6*(s/us)* \* *SoundSpeed* *(m/s)* / 2

*WindowLength(m) = SamplePeriod(us) \* SamplesPerBeam \** 1e-6*(s/us)* \* *SoundSpeed* *(m/s)* / 2

*RangeStart = WindowStart*

*RangeEnd = WindowStart + WindowLength*

*SampleRange****M*** *= WindowStart + SamplePeriod \* Sample****M*** *\** 1e-6 \* *SoundSpeed* / 2

*SampleLength = SamplePeriod \**  1e-6 \* *SoundSpeed* / 2

The geometric mapping from sample space to pixel space may be done in various ways, which is beyond the scope of this document. For each [beam][sample] value, a color is assigned based on acoustic intensity [0..255 corresponds to 0..80dB] and that color is assigned to all screen pixels that lie within the polar boundaries of *SampleRange****M*** +/- *SampleLength / 2* and *BeamAngle****N*** +/- *BeamWidth****N*** / 2.

A simple model assumes evenly spaced beams, but the acoustic lens introduces a non-linear beam spacing based on the lens type (ARIS 1200/1800, ARIS 3000 or Telephoto) and number of beams. See ***Table 1*** for beam spacing files which are based on a “composite” lens of each type, generated by averaging several measured beam patterns for each lens type.

**Table 1: Beam Spacing Table Files**

|  |  |  |
| --- | --- | --- |
| Sonar Lens | Number of Beams | Source File |
| 1200/1800 | 48 | BeamWidths\_ARIS1800\_1200\_48.h |
| 1800 | 96 | BeamWidths\_ARIS1800\_96.h |
| 3000 | 64 | BeamWidths\_ARIS3000\_64.h |
| 3000 | 128 | BeamWidths\_ARIS3000\_128.h |
| Telephoto | 48 | BeamWidths\_ARIS\_Telephoto\_48.h |
| Telephoto | 96 | BeamWidths\_ARIS\_Telephoto\_96.h |

## Appendix A: Reordering .aris Acoustic Data into [Beam][Sample] Array

// for example only...not meant to be compiled

// if the ReorderedSamples parameter in the .aris frame headers is zero, then the acoustic data is not in a tidy [beam][sample] array,

// but interleaved as the raw data was originally digitized. It is then highly desirable to convert the samples into a [beam][sample]

// array as shown below

// the result leaves the acoustic data as (for ARIS 3000 in PingModes 9-12, N=NumBeams-1 and M=SamplesPerBeam-1) :

// beam0:sample0, beam1:sample0... beamN:sample0

// beam0:sample1, beam1:sample1... beamN:sample1

// ...

// beamN:sampleM, beamN:sampleM... beamN:sampleM

int channelReverseMultipledMap [FileTraitsConsts::ARIS\_NUMBER\_A2D\_CHANNELS]; // ARIS\_NUMBER\_A2D\_CHANNELS = 16

static const int Bits16\_channelReverseMap[FileTraitsConsts::ARIS\_NUMBER\_A2D\_CHANNELS];

const int EngineCore::Bits16\_channelReverseMap[] = {

10, 2, 14, 6, 8, 0, 12, 4,

11, 3, 15, 7, 9, 1, 13, 5

};

void EngineCore::ReorderSamples(Frame \* frame, FrameInformation \* frameInfo)

{

(void) frameInfo;

const FrameHeader &header = frame->const\_header();

// mark frame as ordered if writing to .aris V1.x file format

frame->header().ReorderedSamples = 1;

// number of pings per frame (= NumBeams / 16)

uint PingsPerFrame = framecfg->numPings();

// beams per ping always 16 for V1.x .aris files

uint BeamsPerPing = framecfg->numBeams() / PingsPerFrame;

// samples per beam from frame header

uint SamplesPerBeam = framecfg->numSamples();

// number of beams from frame header

uint numBeams = framecfg->numBeams();

for (uint i=0; i<BeamsPerPing; ++i)

{

channelReverseMultipledMap[i] = Bits16\_channelReverseMap[i] \* PingsPerFrame;

}

// pointer to start of desired output data array

byte \* outbuf = frame->rawdataBegin();

// acoustic data size = number of beams \* samples per beam

byte \* temp = new byte[frame->acousticDataSize()];

// work 4 bytes (samples) at a time

uint \* inputW = (uint\*)(temp);

// copy 8-bit data array to 32-bit data buffer

memcpy\_s(inputW, frame->acousticDataSize(), outbuf, frame->acousticDataSize());

for (uint pingIndex=0; pingIndex<PingsPerFrame; ++pingIndex)

{

for (uint sampleIndex = 0; sampleIndex < SamplesPerBeam; ++sampleIndex)

{

uint composed = sampleIndex \* numBeams + pingIndex;

for (uint channel=0; channel < BeamsPerPing; channel += 4)

{

uint values1 = \*inputW++;

int outIndex1 = channelReverseMultipledMap[channel] + composed;

int outIndex2 = channelReverseMultipledMap[channel+1] + composed;

int outIndex3 = channelReverseMultipledMap[channel+2] + composed;

int outIndex4 = channelReverseMultipledMap[channel+3] + composed;

byte byte1 = byte(values1 & 0xFF);

byte byte2 = byte((values1 >> 8) & 0xFF);

byte byte3 = byte((values1 >> 16) & 0xFF);

byte byte4 = byte((values1 >> 24) & 0xFF);

outbuf[outIndex1] = byte1;

outbuf[outIndex2] = byte2;

outbuf[outIndex3] = byte3;

outbuf[outIndex4] = byte4;

}

}

}

delete [] temp;

}