ARIS Integration SDK 2.1

Contents

Purpose

The ARIS Integration SDK is a software development kit for integrating ARIS Explorer and Voyager sonars with an integrating partner's own software. This SDK provides the technical knowledge and definitions necessary to integrate external software (your Controller) with ARIS.

Exclusions and Dependencies

This SDK supports ARIS v2 and later; integration with ARIS 1.X is not supported by this SDK.

This SDK does not address the physical or electrical integration of an ARIS.

This SDK does not address the reading, visualization, or writing of `.aris` recording files. These topics are addressed by the ARIS File SDK.

This SDK does not support integrating an ARIS Defender; a Defender must be converted to an Explorer in order to successfully control it with your custom Controller.

Terminology

ARIS – refers to the ARIS sonar, regardless of model.

Controller – the integrator's software which controls the ARIS.

Command Stream – the TCP stream over which the Controller sends commands to the ARIS.

Frame – an assemblage of meta data and sample data that comprises a single, complete, static image. Each frame is assembled from multiple "frame parts."

Frame Stream – the stream of acoustic frame data received from the sonar; this data is sent via UDP packets.

Development Requirements

Communicating with ARIS requires the use of the TCP and UDP protocols over a TCP/IP network. The wire format of the message sent to and received from the ARIS is defined as Google Protocol Buffers. The messages are defined as `.proto` files (contained in this SDK) and can be used to generate message serialization/deserialization code in C, C++, Java, Python, and other programming languages.

> ***Note**: many of the field definitions in ARIS messages' Protocol Buffer definitions are marked* `optional`*; this is designed to allow easier obsolescence of fields in the future. For commands sent to the ARIS no fields should be considered optional unless otherwise clearly annotated. For messages received from the ARIS, it should be assumed that all fields are indeed optional and their presence should be checked before use.*

Certain critical implementations of code necessary to integrate with the ARIS are provided as C++ files. This code may depend on other libraries; see FrameStream Listener Reference Implementation and Reordering Frame Data.

SDK Contents

ARIS Integration SDK

This document provides an overview and technical detail of connecting to and controlling ARIS.

Reference Code

This document contains references to source files contained in the source repository at

https://github.com/SoundMetrics/aris-integration-sdk/

Protocol Buffer Definition Files

These files are contained in `common/protobuf/` which also contains `.h` header files for Microsoft Visual C++ (useful for warning suppression when compiling the generated protocol buffer code). These files are implicitly version 2 of the protocol buffer syntax.

FrameStream Implementation

The reference C++ implementation of a `FrameStream` listener is located in `common/code/FrameStream/`. This is discussed more in FrameStream Listener Reference Implementation below.

Diagnostics

Syslog

ARIS writes syslog entries to rotating syslog files. These files are located on the sonar in

```
/var/log/syslog/
```

The current syslog file may be followed with

```
tail -f /var/log/syslog/syslog
```

Login credentials for `ssh` access to syslog entries are available to qualified integration partners (please write support@soundmetrics.com). Also, see Syslog Relay.

Syslog Relay

Upon a Controller successfully connecting to an ARIS, the ARIS will also relay its syslog messages to the Controller's host on port 514 (syslog) via UDP. These syslog relay packets can also be observed with a network packet capture tool like Wireshark.

Arislog

In the tools folder there is a Windows console program named arislog which displays messages received from syslog relays, which may be useful for debugging client code.

> Note that a successful connection must be made after power on to set up forwarding of ARIS syslog messages to the client PC. Note also that this program will display **all** syslog messages forwarded to the PC, even from other applications.

Detecting an ARIS

Shortly after being powered on and booting up ARIS begins to broadcast "availability beacons" as UDP packets to port 56124; these beacons are sent at a 1-second interval and announce the presence of the sonar, along with various other information. The availability beacon is defined as the `Availability` message in file `availability.proto`.

Availability Fields of Interest

`serialNumber` – never changes.
`systemType` – never changes.
`connectionState` – connection will be possible only when the connection state is `AVAILABLE`. If the state is BUSY your or another client already have an open connection to the ARIS.
`cpuTemp` – current CPU temperature.
`powerDownReason` – when the sonar determines it needs to power itself down (e.g., it's over-temp) it populates this field and ensures another couple of beacons get out before powering down.
`overTempLimitDegreesC` – the maximum CPU operating temperature allowed
`overTempLimitSeconds` – the duration at which a continuing over-temp condition causes self-power down.

System Variants

The `SystemVariants` field in the availability beacon contains information on variants in the unit configuration. The following strings may appear in `SystemVariants`:

DF – the unit is a Defender variant of ARIS.
VG – the unit is a Voyager variant of ARIS.

Please note that your controller code cannot connect to an ARIS Defender unless the switch handle has been removed. Without the switch handle the DF variant will not be reported in the availability beacon—it is then an ARIS Explorer.

Connecting to an ARIS

ARIS accepts connections from only one Controller at a time. If the `Availability` message's connection state indicates BUSY a Controller will not be able to successfully connect.

To initiate a connection to an ARIS, open a Command Stream—a TCP stream to port 56888 on the ARIS. You may identify an ARIS by the serial number is in its availability beacon, and its IP address is that of the host from which the availability beacon's UDP packet is received.

> ***Note:*** *ARIS supports IPv4.*

The Controller sends commands to control the ARIS as described in Commanding an ARIS, and receives acoustic frame data as described in Receiving Acoustic Frames from an ARIS.

The Controller closes the TCP stream described above to terminate the connection to the ARIS. The ARIS will no longer receive commands from the Controller and will cease to send acoustic frame data to the Controller.

See connect-command for an example of how to connect to ARIS.

Commanding an ARIS

Having opened a TCP stream to port 56888 on the ARIS, the Controller may send commands to the ARIS to control its behavior. The Controller should send a common initial series of commands in order to put the ARIS in a known state.

Commands are defined in `common\protobuf\commands.proto`. The `Command` message is a tagged union, in which the field named `type` is the tag and the `CommandType` enumeration defines the known message types. Each tag has an associated field that must be set; e.g., a message tagged `SET_DATETIME` requires that the `dateTime` field be set. Unrelated fields are ignored, depending on the message type.

Currently there is no acknowledgement from the ARIS that a command has been received.

See connect-command for an example of how to send commands to ARIS.

> ***Suggestion****: The Protocol Buffer library you choose may implement a "to string"-like method or function on each message defined. This can be handy for logging during initial bring-up of your client code.*

Message Framing

Commands to the ARIS over TCP must be properly framed in order for the ARIS to find command boundaries. Each command must be preceded by a 32-bit integer, in network order (e.g., `htonl()`), containing the serialized length of the command. This size prefix must be followed immediately by the serialized command. The Controller must not allow the command length prefix and its associated command to be interrupted by another write to the socket.

> ***Suggestion****: size your "send buffer" to 'length of serialized command' + 4; populate the buffer with the command size followed by the serialized command; transmit the entire buffer in one write request. Your chosen protobuf library can calculate a message's size before you write it to a buffer.*

Verifying the Command Length Prefix

If commanding the ARIS does not appear to work, you can verify that your commands sent to the ARIS are correctly prefixed with the command length using Wireshark, a widely-used network protocol analyzer.

After capturing network traffic containing a connection attempt from your client software, examine the packets from your client computer. You should see evidence of several commands sent from your client computer to the ARIS; these are sent to port 56888 on the ARIS. In Wireshark, select the "Data" portion of the packet. You should observe that the first 4 bytes of the message contain the length of the serialized command, which is a relatively small number.

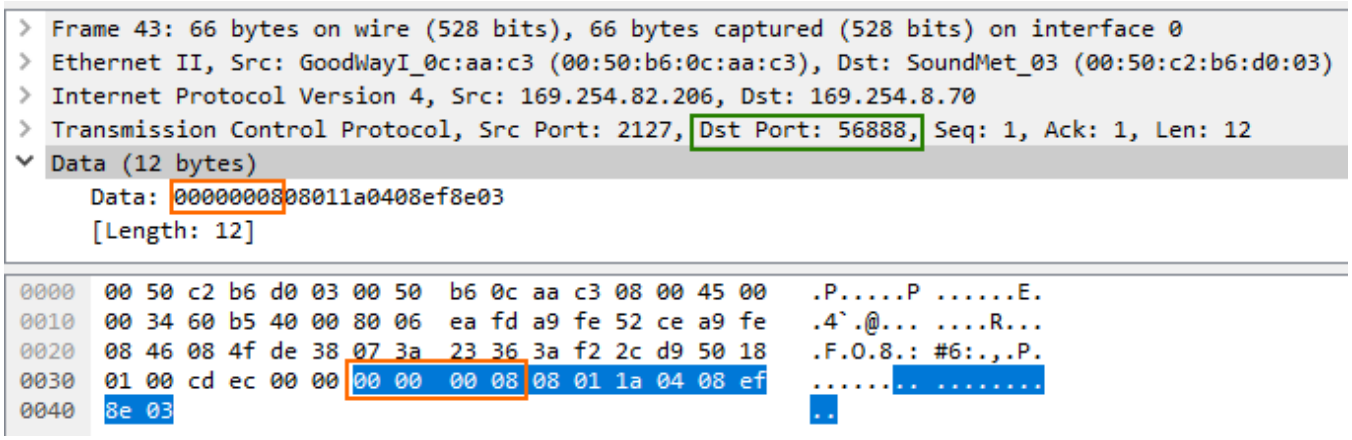

Figure 1

For example, as shown in Figure 1 above, the command length prefix is 8, and the Data portion of the TCP frame is 12 bytes in length, 4 bytes for the command length prefix and 8 for the serialized command.

Maintaining the Connection

In order to avoid zombie TCP connections, ARIS requires that the Controller periodically send a `Ping` message. The Controller should send a `Ping` once per second to maintain an active connection. ARIS will terminate an inactive connection.

Initial Commands

The following commands should be sent on initial connection or re-connection, in this order:

1. `SetFrameStreamReceiver` (optional)
2. `SetDateTime`
3. `SetAcousticSettings`
4. `SetSalinity`
5. `SetFocusPosition`

`SetFrameStreamReceiver` and `SetDateTime` should be sent only once. Other commands may be sent as needed.

The ARIS will not send any frame data until valid acoustic settings are received in a `SetAcousticSettings` command.

Commands

SetDateTime

Sets the ARIS' system clock. The date string must be in the format:

```
2014-Jan-31 23:58:15
```

The month name must be one of the following:

```
"Jan", "Feb", "Mar", "Apr", "May", "Jun",
"Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
```

SetFrameStreamReceiver

Indicates the UDP port on which the Controller will receive Frame Stream packets and the IP (IPv4) address to which the packets will be sent. The Controller will likely already have opened the port and will use its own IP address. Assembly of the Frame Stream packets is covered in Receiving Acoustic Frames from an ARIS.

SetFrameStreamSettings

Configures the frame stream. Not commonly used.

> ***Note**: inter-packet delay is seldom employed, but has been used to accommodate network configurations containing equipment that may not be able to keep up with the FrameStream UDP packets; e.g., a media converter with a small receive buffer.*

See Acoustic Settings: Constraints and Limits.

SetSalinity

Indicates the salinity of the water in which the ARIS rests; salinity affects the speed of sound in water and, therefore, distance measurements. The value passed for salinity must be one of the enumerated values: `FRESH`, `BRACKISH`, or `SALTWATER`. This value is remembered by the ARIS until it is powered down.

SetTelephotoLens

Informs the ARIS that it has the telephoto lens mounted. Not commonly used.

SetFocusPosition

Positions the focus lens. Focus is set by specifying the focus range, in meters, in field `focusRange`. Salinity affects the calculation for focus range, so be sure to use `SetSalinity` before `SetFocusPosition`.

> ***Note:** The focus distance was formerly specified in "focus units" which were specific to the ARIS model; that method is now obsolete, as is the field it used, named* `position`.

ForceFocus

Forces the lens focusing motor to move in a specific direction. Not commonly used.

HomeFocus

Commands the lens focusing motor to move to home position. Not commonly used; this is automatically performed on power-up.

SetRotatorMount

Indicates the type of mount employed with the attached optional AR2/AR3 rotator.

SetRotatorVelocity

Sets the velocity for an axis of the attached AR2/AR3 rotator.

SetRotatorAcceleration

Sets the acceleration for an axis of the attached AR2/AR3 rotator.

SetRotatorPosition

Sets the position for an axis of the attached AR2/AR3 rotator.

StopRotator

Stops rotator movement on the specified axis.

Ping

Informs the ARIS that the Controller is still connected.

SetAcousticSettings

The Controller requests specific acoustic settings for frame acquisition via `SetAcousticSettings`. ARIS will not return any acoustic data until valid acoustic settings are received. Valid values for `SetAcousticSettings`' fields are described here; further knowledge required to combine settings successfully is described in Acoustic Settings: Constraints and Limits.

Field Name	Units	Valid Range
`cookie`	n/a	1 – 0xFFFFFFFF; 0 is invalid
`frameRate`	frames/second	1.0 – 15.0
`pingMode`	n/a	1, 3, 6, 9; system dependent
`frequency`	n/a	Low/High
`samplesPerBeam`	n/a	128 – 4,096
`sampleStartDelay`	µs	930 – 60,000
`cyclePeriod`	µs	1,802 – 150,000
`samplePeriod`	µs	4 – 100
`pulseWidth`	µs	5 – 80; system dependent
`enableTransmit`	n/a	True/False
`enable150Volts`	n/a	True/False
`receiverGain`	dB	0 – 24

Choosing appropriate values for these fields is discussed in depth in ARIS Acoustic Images.

ARIS Acoustic Images

Theory of Operation

ARIS creates images by transmitting acoustic energy and then detecting, or sampling, the energy reflected back from objects. Because we know the "time in flight" for each sample taken, we also know the range of the object that reflected the sample; this calculation depends on the speed of sound in water.

The near and far boundaries of the image are defined by the settings used to collect the image. While we discuss these boundaries in terms of physical distance, the ARIS is commanded in units of time (microseconds). Because sample collection occurs in units of whole microseconds, controlling the ARIS in units of time allows us to precisely know the range from which a sample was reflected.

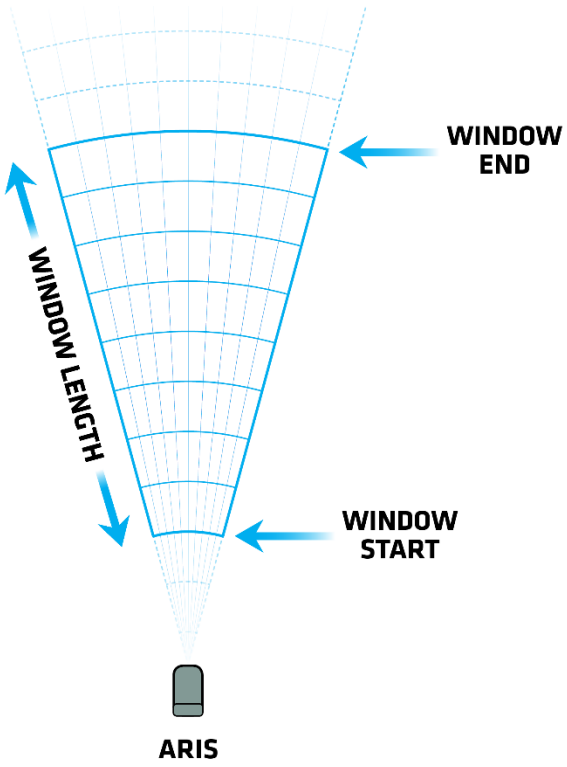

Figure 2

In order to calculate the range of a sample you need to know the speed of sound in water. The speed of sound, in turn, is approximated based on the water temperature, salinity, and depth. Temperature and salinity are the primary factors in determining the speed of sound in water. Water temperature and depth are provided by sensors onboard the ARIS. Salinity, however, is manually provided by the Controller by sending the `SetSalinity` command. Three acceptable salinity levels are pre-defined as fresh (0 ppt), brackish (15 ppt), and saltwater (35 ppt). (See protobuf definition file `common\protobuf\commands.proto`.)

Perspective View

Three terms concerning distance and direction relative to the acoustic image are Down Range, Cross Range, and Perspective View. Down Range refers to distance away from the ARIS along the length of the image. Cross Range speaks to distance directly across the width of the wedge. Perspective View refers to viewing the image from directly above the imaged area.

The width of the acoustic image is a fan of narrow slices. Each beam from an ARIS produces a thin slice of the image from near to far along the Down Range axis.

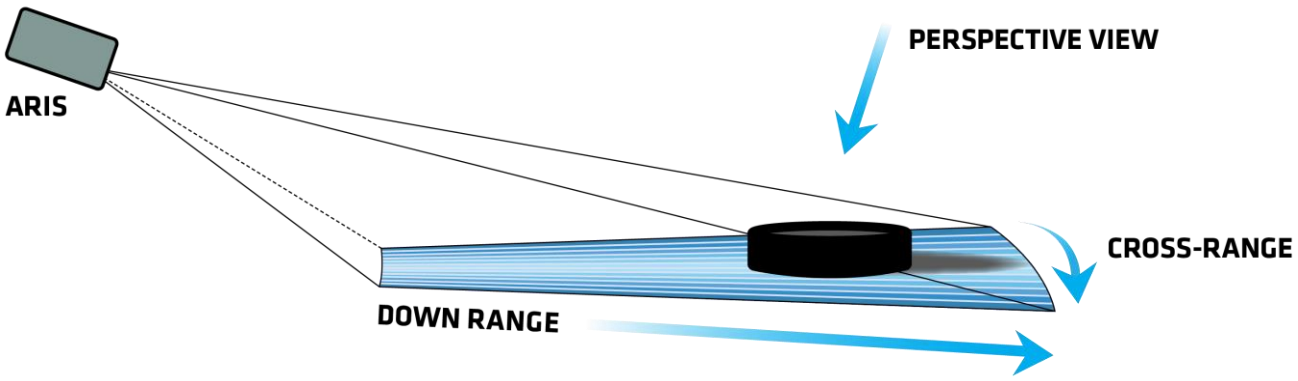

Figure 3

Practical Considerations

Water Temperature vs Range

Because water temperature changes over time, the calculated Window Start and Window Length (for constant acoustic settings) will also vary over time. For fixed window operations in which you don't vary the acoustic settings, you may wish to select acoustic settings that work as expected over the range of operating conditions.

Some applications prefer or require a consistent image size over time (always the same sample count). Such applications should also consider selecting acoustic settings that work as expected over the range of operating conditions.

Sample Storage Size

Sample storage size may be a consideration for applications. ARIS sample data is stored as one unsigned byte per sample with valid values of 0 – 255. (For more technical details, see Constructing Images from Acoustic Samples and the documents referenced there.) The size of each frame of data is a fixed header size plus the product of the beam count and sample count.

$$Frame\ Size = 1024 + (Number\ of\ Beams * Number\ of\ Samples)$$

For example, an ARIS 3000 operating with 128 beams and 1000 samples produces a frame of 129KB. At a maximum of 15 frames per second this would produce frame data at a rate of approximately 9 GB per hour.

The `.aris` file format is documented at https://github.com/SoundMetrics/aris-file-sdk.

Selecting Acoustic Settings

This section discusses how to select settings for the SetAcousticSettings message. Care must be taken to send valid acoustic settings to the ARIS. If your chosen settings exceed certain limits they may be constrained (adjusted) by the ARIS to fall within those limits. Invalid settings will be ignored.

In order to select appropriate acoustic settings, you'll want to know your desired Window Start, Window End, and Window Length, where

$$Window\ Length = Window\ End - Window\ Start$$

Figure 2 above illustrates Window Start, Window End, and Window Length.

SetAcousticSettings Fields

`cookie` – The cookie is a monotonically increasing integer provided by the controller; the controller must increment this value every time this message is sent. The cookie is used by the controller to track application of acoustic settings. The acoustic frame header sent by the ARIS contains several fields (`AppliedSettings`, `ConstrainedSettings`, `InvalidSettings`) that indicate whether and how ARIS applied the settings requested by the Controller.

If the settings are considered good the cookie will appear in the `AppliedSettings` field. If the settings were constrained to respect certain limits the cookie will appear in the `ConstrainedSettings` field. If the settings were considered invalid the cookie will appear in the `InvalidSettings` field.

`frequency` – selects the transmit frequency of the sonar. Set to zero (0) for low frequency, one (1) for high frequency. Low frequency allows imaging at longer ranges; high frequency gives better image resolution at closer ranges.

System Type	Low Frequency	High Frequency
ARIS 1200	0.7 MHz	1.2 MHz
ARIS 1800	1.1 MHz	1.8 MHz
ARIS 3000	1.8 MHz	3.0 MHz

It is recommended to select frequency by the Window End value. If Window End exceeds (is larger than) the crossover in the table below, set the frequency Low; otherwise, set the frequency High. In water colder than 15°C, the crossover distance may need to be reduced, and conversely may be increased if warmer than 15°C

System Type	Crossover Distance (Window End)
ARIS 1200	25 m
ARIS 1800	15 m
ARIS 3000	5 m

`pingMode` – selects the total number of beams in a frame. For the ARIS 1800 and ARIS 3000, changing the ping mode allows a trade-off between better resolution (more beams) and higher frame rate (fewer beams).

System Type	Ping Mode	Number of Beams
ARIS 1200	1	48
ARIS 1800	1	48
ARIS 1800	3	96
ARIS 3000	6	64
ARIS 3000	9	128

`sampleStartDelay` – determined by Window Start, where sampling begins. The minimum value is 930 µs (~0.7 m); the maximum value is 60,000 µs (~45 m).

$$Sample\ Start\ Delay = \frac{2 * Window\ Start}{Speed\ of\ Sound}$$

Note that `sampleStartDelay` field value must be an integer value in microseconds, so multiply by 10^6 and round to the nearest integer.

`samplePeriod` – Determined by the down-range resolution of samples. Because sample period is specified in an integral number of microseconds the down-range resolution is not infinitely adjustable (resolution is quantized).

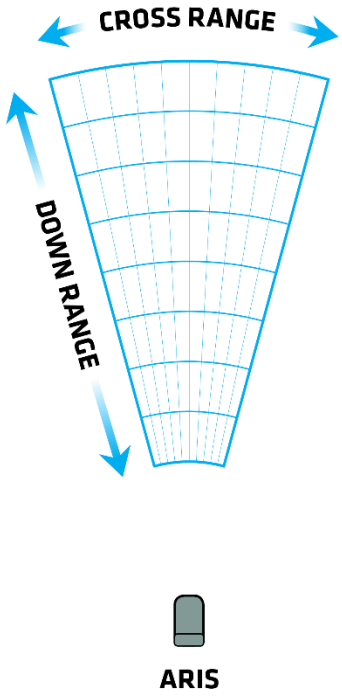

Figure 4

Sample period is typically selected to allow a cross-range to down-range aspect ratio of between 4:1 and 8:1, however you may wish to adjust the sample period based on storage needs: a smaller sample period gives a shorter down-range sample size, with more samples, and needs more storage space.

In order to select a sample period relative to the cross-range sample size, first calculate cross range resolution ($Beam\ Spacing$ is given below):

$$Cross\ Range\ Resolution = \left(Window\ Start + \frac{Window\ Length}{2}\right) * \sin(Beam\ Spacing)$$

Then, calculate down range resolution use the multiplier N (given below):

$$Down\ Range\ Resolution = \frac{Cross\ Range\ Resolution}{N}$$

The $Beam\ Spacing$ and N factors are determined based on your ARIS system type and which ping mode you are using:

System Type	Ping Mode	Beam Spacing	N
ARIS 1200	1	0.6°	8
ARIS 1800	1	0.6°	8
ARIS 1800	3	0.3°	4
ARIS 3000	6	0.5°	8
ARIS 3000	9	0.25°	4

Finally, calculate sample period:

$$Sample\ Period = \frac{2 * Down\ Range\ Resolution}{Speed\ of\ Sound}$$

Note that `samplePeriod` field value must be an integer value in microseconds, so multiply by 10^6 and round to the nearest integer.

`samplesPerBeam` – having selected a sample period above, it is easy to calculate the number of samples in the sample window. Here $Sample\ Period$ is in microseconds.

$$Samples\ per\ Beam\ = \frac{2 *\ Window\ Length}{Sample\ Period * Speed\ of\ Sound} *\ 10^6$$

`cyclePeriod` – the minimum time required to acquire the samples for a single ping.

$$Cycle\ Period = Sample\ Start\ Delay + (Sample\ Period * Samples\ per\ Beam) + 360$$

This is the minimum value of `cyclePeriod` that allows for the fastest `frameRate`. Additional time may be added (within the constraints below) to correct for aliasing in the acoustic image, which will reduce the maximum possible `frameRate`.

`frameRate` – the number of frames acquired per second. Faster frame rates allow smoother image motion when the sonar or targets are moving, at the cost of more data transfer. The fastest allowable frame rates is, approximately:

$$Frame\ Rate = \frac{1}{Frame\ Period} = \frac{1}{(Pings\ per\ Frame * Cycle\ Period)}$$

System Type	Ping Mode	Pings per Frame
ARIS 1200	1	3
ARIS 1800	1	3
ARIS 1800	3	6
ARIS 3000	6	4
ARIS 3000	9	8

Note that you can calculate the maximum frame rate based on given acoustic settings as shown in `common/code/FrameRate`. Currently ARIS is limited to a maximum of 15 frames per second.

`pulseWidth` – the duration of the transmitted energy pulse. Suggested values for transmit pulse width based on system type and frequency:

$$Width = Modifier * Window\ End$$

System Type	Frequency	Modifier
ARIS 1200	LF	1.0
ARIS 1200	HF	1.0
ARIS 1800	LF	1.0
ARIS 1800	HF	1.5
ARIS 3000	LF	1.5
ARIS 3000	HF	2.0

`receiverGain` – relative receiver gain in dBs. Suggested initial values based on system type:

System Type	Receiver Gain
ARIS 1200	20
ARIS 1800	18
ARIS 3000	12

`enable150Volts` – turns on 150V transmit circuit. You generally want this "on" (1).

`enableTransmit` – enables transmission of acoustic energy. You can turn this off to check for self-noise or look for external acoustic interference (other sonars, DVL, etc.).

This SDK contains two other resources for using correct acoustic settings:

- The appendix titled Example Acoustic Settings, which contains some baseline examples that are handy for making your first images during integration.
- The Excel spreadsheet "Integration SDK suggested settings.xlsx," which provides some guided calculation of valid settings.

Acoustic Settings: Validation and Constraints

Validation

The following pseudo code illustrates how validation is performed by the ARIS; settings that fail these checks are considered invalid and are not applied.

```
pingModeConfigurations = // look-up table, ping mode is the key
    { { pingMode = 01u; channelCount =  48u; pingsPerFrame = 3u }
      { pingMode = 03u; channelCount =  96u; pingsPerFrame = 6u }
      { pingMode = 06u; channelCount =  64u; pingsPerFrame = 4u }
      { pingMode = 09u; channelCount = 128u; pingsPerFrame = 8u } }

// Valid input ranges:
samplesPerBeamRange   =  128u  ↔    4096u
sampleStartDelayRange =  930u  ↔   60000u
cyclePeriodRange      = 1802u  ↔  150000u
samplePeriodRange     =    4u  ↔     100u
pulseWidthRange       =    5u  ↔      80u
receiverGainRange     =  0.0f  ↔    24.0f

// Validate:
pingsPerFrame = pingModeConfigurations[pingMode].pingsPerFrame
framePeriod = uint32 (ceil (1000000.0f / frameRate)) // microseconds
adjustedCyclePeriod =
     sampleStartDelay + (samplePeriod * samplesPerBeam) + 360u

isValid =   samplesPerBeam      is in range  samplesPerBeamRange
            && sampleStartDelay is in range  sampleStartDelayRange
            && cyclePeriod      is in range  cyclePeriodRange
            && samplePeriod     is in range  samplePeriodRange
            && pulseWidth       is in range  pulseWidthRange
            && receiverGain     is in range  receiverGainRange
            && framePeriod > cyclePeriod * pingsPerFrame
            && cyclePeriod >= adjustedCyclePeriod
```

Constraints

In order to protect certain components, ARIS limits the amount of energy allowed through them. The code below illustrates the steps taken by ARIS to limit energy by constraining the pulse with of the settings. If valid settings are supplied to the ARIS, this constraint is then supplied; if the settings are unchanged they are considered “valid,” otherwise “constrained.”

```
enum SystemType { ARIS1800 = 0, ARIS3000 = 1, ARIS1200 = 2 };

const float pulse_width_limits[3][2] = {
    { 40.0, 30.0 }, { 24.0, 16.0 }, { 80.0 , 60.0 }
    // ARIS1800,    ARIS3000,       ARIS1200
};

const float energy_limits[3] = {
    300.0, 240.0, 400.0 // ARIS1800, ARIS3000, ARIS1200
};
```

```
bool LimitPulseWidth(settings &settings)
{
    const uint origPulseWidth = settings.pulse_width; // cycle count
    const float actual = FreqSelToMHz(settings.acquisition.frequency_select);

    // First limit pulse width by system type and frequency to ensure
    // image quality
    const auto freq = settings.acquisition.frequency_select;
    const float maxPulseWidth = pulse_width_limits[_cfg.TheSystemType][freq];
      // usecs
    settings.pulse_width = min((uint)floor(maxPulseWidth * actual),
                                    origPulseWidth);

    // Then limit pulse with by total allowed energy for system type
    const float fps = 1000000.0 / (float)settings.frame_period;
    const float maxEnergy = energy_limits[_cfg.TheSystemType];
    settings.pulse_width = min((uint)floor((maxEnergy * actual) / fps),
                                    (uint)settings.pulse_width);

    return settings.pulse_width != origPulseWidth;
}

float FreqSelToMHz(uint frequency)
{
    float actual;
    if (_cfg.TheSystemType == 1) { // ARIS3000
        actual = (frequency == 0) ? 1.800f : 3.000f;
    } else if (_cfg.TheSystemType == 2) { // ARIS1200
        actual = (frequency == 0) ? 0.6923125f : 1.200f;
    } else { // ARIS1800
        actual = (frequency == 0) ? 1.1077f : 1.800f;
    }

    return actual;
}
```

Receiving Acoustic Frames from an ARIS

ARIS uses a “Frame Stream” protocol to pass acoustic frame data to the Controller; the port on the controller used to receive the data is specified by the Controller in the `SetFrameStreamReceiver` command.

Acoustic Frame Data

The final product of the Frame Stream protocol, a completely assembled frame of acoustic samples and associated metadata, is a 1024-byte Frame Header (described in the “015207 ARIS File SDK”; see “ArisFileSDK\ArisFileFormat\ARIS File Format DDF_05.xlsx”) followed by payload of acoustic samples whose length is ‘number of beams’ × ‘number of samples per beam.’

Frame Stream Protocol

The Frame Stream protocol utilizes UDP to send packets containing acoustic frame data to the Controller. The wire format of each frame is a Google Protocol Buffer as defined by `FramePart` in `protocols\ArisMessages\frame_stream.proto`.

Frame Parts

Each frame is sent as one or more `FramePart` messages. The first `FramePart` of a frame has a `data_offset` of `0`, a non-zero `total_data_size` field, and its `header` field is populated with the Frame Header; its `data` field contains the initial bytes of the acoustic samples. Subsequent `FramePart` messages for the same frame have the same `frame_index` as the first `FramePart` and a larger `data_offset` value; the final `FramePart` of a frame is determined by the condition

```
data_offset + length-of(data) == total_data_size
```

Only the first `FramePart` contains the frame header.

> ***Note:*** *the frame header contained in the first* `FramePart` *of each frame is truncated to eliminate transmission of the unused tail-end portion of the structure. The implementer should pad the frame header to 1024 bytes for further manipulation and storage; this will ensure the frame header is forward-compatible as well as compatible with existing ARIS software.*

Frame Part Acknowledgements

Each `FramePart` received by the Controller should be acknowledged by sending a `FramePartAck` message to the ARIS. Presently, the ARIS will not re-send lost packets, but acknowledging the frame parts will support basic retries when that retry policy is implemented.

Lost Frame Parts

Because the ARIS does not currently implement retries of lost packets the Controller may receive incomplete frames.

FrameStream Listener Reference Implementation

Our `FrameStream` listener implementation is the canonical C++ example of how to receive and assemble frame parts sent by the sonar. It currently relies on the Boost libraries for asynchronous I/O and is not generalized for easy substitution of other libraries. See `FrameStreamListener` in `common/code/FrameStream/`.

Multicasting ARIS Frames

It is not common, but some deployments wish to make ARIS frames available to more than one application.

The FrameStream Listener reference implementation supports point-to-point delivery of the frames as its default behavior. It can also be configured to receive frames via multicast (in `FrameStreamListener`'s constructor, via `receiveFrom`), but this is not a common configuration.

When configuring for multicast, you will need to choose a multicast group address, most likely in the range `239.0.0.0`–`239.255.255.255`; you will also need to specify a pre-determined port. See http://www.tcpipguide.com/free/t_IPMulticastAddressing.htm for more information on IP multicast addressing and how to select a multicast group address.

You may run across a number of issues may occur when multicasting frames:

- Not all network hubs and switches are compatible with multicasting. You may see no frames using multicast, while point-to-point works fine.

- Direct link-local setups (two network nodes, no switch) may not work for multicast (possibly useful for testing, but not for actual multicasting). Again, you may see no frames using multicast, while point-to-point works fine.
- Other network equipment may impede multicasting bandwidth. In this case, you may see many incomplete frames.

The sample program vc-using-framestream in `sample-code/vc-using-framestream` may be useful for testing your network configuration for multicast. In particular, the +/- output should indicate whether there is adequate bandwidth or if possible interference from other network equipment is occurring.

> ***Note:*** *multicast is supported by the onboard software in version is 2.6.8634 and later*.

No retries

ARIS' `FrameStream` sends frame parts over UDP, which is not a reliable protocol. ARIS does not presently resend missed frame parts, though the above `FrameStream` listener implementation does send acknowledgements to the ARIS in support of that future feature. It is the implementer's choice whether to discard an incomplete frame or save it with some accounting for which parts are missing.

Reordering Frame Data

Frame data delivered by the ARIS' FrameStream is generally not in an order that's appropriate for display or other usage—it is in the form received directly from the device rather than in a bitmap form. The frame header contains a field named `ReorderedSamples`, which is non-zero when the data has been properly re-ordered so it is useful for display.

When connecting your custom Controller directly to the ARIS you may see data that is not yet re-ordered, where `ReorderedSamples` is zero. In this case, you will need to re-order the data yourself.

The canonical C++ code to re-order the samples in a frame can be found in `common/code/Reordering/`; note that you should set frame header field `ReorderedSamples` to `1` after re-ordering the samples. There is also sample code in `sample-code/reorder-frame`.

Constructing Images from Acoustic Samples

Please see Constructing Images From Samples in the ARIS File SDK for guidance on correctly constructing an image from the acoustic samples.

Injecting ARIS Frame Header Data

A number of the frame header fields may be updated by your Controller or other software, including GPS coordinates. The file `common/code/UpdateFrameHeader/ArisHeaderUpdate.h` provides more information on what fields may be updated, how to create the update message, and sample functions for initializing and setting fields in an update message.

Appendices

Ports

Port Number	Protocol	Description
56124	UDP	ARIS broadcasts availability beacons to this port. See Detecting an ARIS.
56888	TCP	ARIS Explorer and ARIS Voyager accept a TCP connection from a Controller on this port; this initiates an ongoing dialog between the Controller and the ARIS. See Connecting to an ARIS.
700	UDP	ARIS accepts platform header updates as described in Injecting ARIS Frame Header Data. See Injecting ARIS Frame Header Data.

Example Acoustic Settings

The following example settings are valid settings that will work with your ARIS Explorer 1200, 1800, or 3000; these should be helpful during your initial integration with the ARIS.

Please note that the frame rates listed are the maximum available for the selected acoustic settings, as discussed in Selecting Acoustic Settings.

ARIS 1200

These acoustic settings are based on a sampling window from 4 – 24 meters in fresh water at 19°C. Recommended focus range is 14 meters.

Field Name	Units	Value
cookie	n/a	[you provide]
frameRate	frames/second	10.0
pingMode	n/a	1
frequency	n/a	1
samplesPerBeam	n/a	1082
sampleStartDelay	µs	5408
cyclePeriod	µs	32,818
samplePeriod	µs	25
pulseWidth	µs	24
enableTransmit	n/a	1
enable150Volts	n/a	1
receiverGain	dB	20

ARIS 1800

These acoustic settings are based on a sampling window from 1.5 – 7.5 meters in fresh water at 19°C. Recommended focus range is 4.5 meters.

Field Name	Units	Value
cookie	n/a	[you provide]
frameRate	frames/second	15.0
pingMode	n/a	3
frequency	n/a	1
samplesPerBeam	n/a	1014
sampleStartDelay	µs	2028
cyclePeriod	µs	10500
samplePeriod	µs	8
pulseWidth	µs	11
enableTransmit	n/a	1
enable150Volts	n/a	1
receiverGain	dB	18

ARIS 3000

These acoustic settings are based on a sampling window from 1.5 – 5.0 meters in fresh water at 19°C. Recommended focus range is 3.25 meters.

Field Name	Units	Value
cookie	n/a	[you provide]
frameRate	frames/second	15.0
pingMode	n/a	9
frequency	n/a	1
samplesPerBeam	n/a	946
sampleStartDelay	µs	2028
cyclePeriod	µs	7118
samplePeriod	µs	5
pulseWidth	µs	10
enableTransmit	n/a	1
enable150Volts	n/a	1
receiverGain	dB	12

Cycle Period & Sampling Terminology

Speed of Sound	SSPD	The velocity at which sound travels (in this case in water).
Sample Start Delay	SSD	The period of time that the sonar waits after transmit before sampling data.
Number of Samples	N	The number of samples to be taken for each beam.
Sample Period	SP	The time over which an individual sample is taken.
Window Start	WS	The distance at which the nearest sample is taken. This distance is $$WS = \frac{SSD * SSPD}{2}$$ (the ping has to round trip, hence the divide by two).
Window Length	WL	This is the distance over which samples are taken. This distance is $$WL = \frac{N * SP * SSPD}{2}$$
Window End	WE	Window end, the farthest that we sample. $$WE = WS + WL = \frac{SSD * SSPD}{2} + \frac{N * SP * SSPD}{2} = \frac{SSPD}{2} * \big(SSD + (N * SP)\big)$$
Cycle Period	CP	The minimum time it takes to complete one round of sampling. This can be derived from WE: (multiply WE by two to get time of round trip distance) $$CP = \frac{2 * WE}{SSPD} = 2 * \frac{\frac{SSPD}{2} * \big(SSD + (N * SP)\big)}{SSPD} = SSD + (N * SP)$$
Anti-Aliasing	AA	Time may be added to the cycle period in order to correct for aliasing in the acoustic image.
Adjusted Minimum Cycle Period	AMCP	The minimum cycle time necessary for data acquisition to go well. $$AMCP = CP + 360\mu s + AA$$ $$AMCP = SSD + (N * SP) + 360\mu s + AA$$