Gemini Software Development Kit

Specification for Svs5Sequencer Library Interface DLL

(‘Svs5SeqLib.dll’)

Notes:

Applies to Gemini Sonar VDSL/Ethernet/Serial connections.
The ‘Svs5SeqLib.dll’ and relevant header files will be provided with this documentation.

Revision History

Rev.	Date	Author	Changes
01	08/07/2021	AS	Initial version of interface specification.
02	28/02/2022	AS	Added Micron Gemini support
03	10/03/2022	AS	Added support for firmware upgrade and Noise Reduction configuration for Micron Gemini
04	18/08/2023	MS	Added External TTL Trigger and H264 Compression
05	02/10/2023	MS	Added AHRS Data Handling Added COM_PORT_STATUS message type. Added ListOfFileNames structure

Index

Overview

This document describes the DLL (Svs5SeqLib.dll) which provides a wrapper interface to the GeminiComms (DLL). This library hides the complexity of the GeminiComms and provides a very simple interface. This library is designed specifically for the SDK users for the ease of use. It will provide details of all the DLL functions allowing users to gain programmatic access to the Gemini sonar in their own software. This document will describe all the functionality exposed by the DLL that is relevant for users to write their own control and display programs.

Only one application (and only one instance of that application) may use the DLL at any given time to avoid resource conflicts.

The DLL is written in C++ and is provided on cross platforms. For Windows, this library is compiled using Microsoft Visual Studio 2017.

This document describes version 2.0.38 of the DLL, which returns the version string `"V2.0.38 Copyright (C) Tritech International Ltd."`

This library has the capabilities to log ECD/GLF data, therefore this library is dependent on GenesisSerializer, ECDLogDataTypes and MathsLib. ECD logging library is also dependent on boost libraries version 1.66.0. ECD logger support can be removed from the test example during compile time by setting -DECD_LOGGER=no

Constants

The following messages are passed to the user defined data handling callback function along with the type of the data.

Callback Message Type	**ID**	**DataTypes**
GEMINI_STATUS	0	CGemStatusPacket (defined in GeminiStructuresPublic.h)
* ECD_LIVE_TARGET_IMAGE	1	CTgtImg (defined in ecdlogtargetimage.h)
GLF_LIVE_TARGET_IMAGE	2	GLF::GLogTargetImage (defined in GlfApi.h)
* SENSOR_RECORD	3	Not supported
LOGGER_REC_UPDATE	4	GLF::SOutputFileInfo (defined in GlfLoggerStatusStructure.h)
LOGGER_PLAYBACK_UPDATE	5	GLF::SInputFileListInfo (defined in GlfLoggerStatusStructure.h)
TGT_IMG_PLAYBACK	6	GLF::GenesisPlaybackTargetImage
* ECD_IMG_PLAYBACK	7	CTgtImg (defined in ecdlogtargetimage.h)
LOGGER_FILE_INDEX	8	Integer value (index value of the playback file)
LOGGER_STATUS_INFO	9	GLF::GnsLoggerStatusInfo (defined in GlfLoggerStatusStructure.h)
FRAME_RATE	10	Integer value (Frames per second)
GPS_RECORD	11	GLF::GLogV4ReplyMessage (defined in GlfLoggerV4Structure.h) m_header.m_ciHeader.m_dataType == 98 (Raw ascii data) m_header.m_ciHeader.m_dataType == 99 (GLF::GpsDataRecord structure defined in GpsRecord.h)
COMPASS_RECORD	12	GLF::GLogV4ReplyMessage (defined in GlfLoggerV4Structure.h) m_header.m_ciHeader.m_dataType == 98 (Raw ascii data) m_header.m_ciHeader.m_dataType==99(GLF::CompassDataRecord structure defined in CompassRecord.h)
** AUXPORT1_DATA	13	Binary data received from the Gemini Aux port 1
** AUXPORT2_DATA	14	Binary data received from the Gemini Aux port 2
UPGRADE_AVAILABLE	15	Reports back to the application, if upgrade available. See GeminiSDKConsole.cpp for an example
FIRMWARE_UPGRADE_INFO	16	Reports back to the application for the firmware upgrade status. See FirmwareUpgradeStatus (defined in FirmwareUpgradeDef.h)
COM_PORT_STATUS	17	Status of Open Ports
AHRS_RAW_DATA	18	RAW AHRS Data from Compass (GLF::AHRSRawDataRecord structure defined in CompassRecord.h)
AHRS_HPR_DATA	19	HPR Calculated Data from the Library (GLF::CompassDataRecord structure defined in CompassRecord.h)

Table 1.0 Callback Message Types

```
* Deprecated.
** Data received from sonar AUX ports.
```

The following configuration messages are passed to the Svs5Sequencer library to modify the default configuration

ESvs5ConfigType	**Description**
SVS5_CONFIG_ONLINE	Enable/Disable streaming (bool)
SVS5_CONFIG_RANGE	Range (1 - 120) in meters (double)
SVS5_CONFIG_GAIN	Gain (1 - 100) in percentage (int)
SVS5_CONFIG_SIMULATE_ADC	Enable/Disable simulation (bool)
SVS5_CONFIG_PING_MODE	Ping configuration parameters in SequencerPingMode struct
SVS5_CONFIG_SOUND_VELOCITY	Configure sound velocity in SequencerSosConfig (Using sonar SOS or user configured)
SVS5_CONFIG_SONAR_ORIENTATION	Sonar orientation (See ESvs5SonarOrientation structures)

SVS5_CONFIG_RANGE_RESOLUTION	Configure sonar range resolution (Only applies to 1200ik product)
SVS5_CONFIG_HIGH_RESOLUTION	Configure sonar improved range resolution (Only applies to 1200ik product)
SVS5_CONFIG_CHIRP_MODE	Configure sonar chirp mode (0: Disabled, 1: Enabled, 2: Auto)
SVS5_CONFIG_LOG_RAW_GPS	Log RAW GPS data
SVS5_CONFIG_LOG_RAW_COMPASS	Log RAW Compass data
SVS5_CONFIG_CPU_PERFORMANCE	Configure SDK based on the CPU performance (SonarImageQualityLevel struct)
SVS5_CONFIG_APERTURE	Configure sonar Aperture (Can switch between 120 / 65 degrees)
SVS5_CONFIG_REBOOT_SONAR	Reboot Gemini sonar
SVS5_CONFIG_LOG_GPS	Log formatted GPS data
SVS5_CONFIG_LOG_COMPASS	Log formatted Compass data
SVS5_CONFIG_OPEN_720IM_COM_PORT	Open 720im / Micron Gemini on regular com port (See ComPortConfig struct in GeminiStructuresPublic.h)
SVS5_CONFIG_AUX_PORT	Configure Aux port (See AuxPortConfig structure)
SVS5_CONFIG_NOISE_REDUCTION	Configure Noise Reduction Filter (bool)
SVS5_CONFIG_FIRMWARE_LOCATION	Configure Firmware file location
SVS5_CONFIG_UPGRADE_FIRMWARE	Upgrade firmware on the target device
SVS5_CONFIG_ABORT_UPGRADE	Cancel firmware Upgrade on the target device
SVS5_CONFIG_LOGGER	Configure logger (default : GLF)
SVS5_CONFIG_FILE_LOCATION	Configure default location (const char*)
SVS5_CONFIG_REC	Start/Stop logger (bool)
SVS5_CONFIG_PLAY_START	List of files (struct ListOfFileNames)
SVS5_CONFIG_PLAY_FILE_INDEX	File Index
SVS5_CONFIG_PLAY_PAUSE	Playback in pause state
SVS5_CONFIG_PLAY_REPEAT	Playback in the loopback state
SVS5_CONFIG_PLAY_STOP	Stop playback
SVS5_CONFIG_PLAY_SPEED	0 for free running, 1: RealTime
SVS5_CONFIG_PLAY_FRAME	Explicitly request frame
SVS5_CONFIG_RLE_COMPRESSION	RLE Data Compression Level (0..255), 0=Compression Off
SVS5_CONFIG_H264_COMPRESSION	H264 Compression Enabled/Disabled, true = ON, false = OFF

Table 2.0 Configuration Message Types

The following table specifies the error code returned by the library

SVS5_SEQUENCER_STATUS_OK	0
SVS5_SEQUENCER_ALREADY_STARTED	1
SVS5_SEQUENCER_NOT_RUNNING	2
SVS5_SEQUENCER_ANOTHER_INSTANCE_RUNNING	3
SVS5_SEQUENCER_INVALID_CONFIG	4
SVS5_SEQUENCER_INVALID_PARAMETERS_SIZE	5
SVS5_SEQUENCER_INVALID_PARAMETERS_VALUE	6
SVS5_SEQUENCER_INVALID_PROPERTY_ID	7
SVS5_SEQUENCER_INVALID_PLAYBACK_FILE	8
SVS5_SEQUENCER_UPGRADE_IN_PROGRESS	9

Table 3.0 Error codes

Structures

The following structures are used in the interface of the DLL. The C++ class definitions for these are contained in the file 'Svs5SequencerApi.h'.

SequencerPingMode

The SequencerPingMode structure used to configure the ping mode i.e. Free running mode, External TTL Trigger or ping at a fixed interval.

The fields within the SequencerPingMode structure are defined below

Type	**Field**	**Definition**
bool	m_bFreeRun	True: as fast as possible, False: ping based on interval in milliseconds
Unsigned short	m_msInterval	Millisecond interval (0-999)
bool	m_extTTLTrigger	True: External TTL Triggers a Ping; False: External TTL Trigger does not Trigger a Ping

SequencerSosConfig

The SequencerSosConfig structure used to configure the speed of sound mode. User can either select manual speed of sound or sonar speed of sound calculated (Not supported for 720im and Micron Gemini)

The fields within the SequencerSosConfig structure are defined below

Type	**Field**	**Definition**
bool	m_bUsedUserSos	True: User defined, False: Sonar SOS
float	m_manualSos	Default: 1500, Range(1400 – 1584)

AuxPortConfig

The AuxPortConfig structure used to configure the Aux port configuration on sonar. Only 720is, 720im and Micron Gemini platform supports to switch between RS232/RS485 mode. Rest of the platforms only support RS232 mode.

The fields within the AuxPortConfig structure are defined below

Type	**Field**	**Definition**
unsigned char	m_portNum	0 for Aux port 1, Only 720is support 2 AUX ports
unsigned int	m_baudRate	9600 – 115200
bool	m_rs232	True: RS232, False: RS485

SonarImageQualityLevel

The SonarImageQualityLevel structure used to configure the bandwidth for sonar data rate. The CPU usage will be higher if the data received from sonar is the highest quality. For embedded devices where the screen resolution is not even HD quality then user can configure these setting to reduce the CPU usage.

The fields within the SonarImageQualityLevel structure are defined below

Type	Field	Definition
unsigned char	m_performance	See below ESdkPerformanceControl
int	m_screenPixels	To use the highest quality user screen pixels to 2048, 1024, 512, 256 respectively

```
// Performance configuration parameters
// User can configure these settings based on the CPU usage
enum ESdkPerformanceControl {
    LOW_CPU,           // Low image quality       ( 256 beams )
    MEDIUM_CPU,        // Medium image quality ( 256 beams )
    HIGH_CPU,          // High image quality      ( 512 beams )
    UL_HIGH_CPU        // Ultra high quality
                       // 512 beams for 720is and 720ik,
                      // 1024 beams for 1200ik in higher resolution
};
```

ListOfFileNames

The ListOfFileNames structure used to store a list of files for recorded playback.

The fields within the ListOfFileNames structure are defined below

Type	Field	Definition
size_t	m_numberOfFiles	Number of files in the list
char **	m_fileNames	Pointer to a Pointer containing the file names within the list

Callback Functions

The DLL requires a callback function to be defined for it, so that it can inform the calling process when data has been received from the sonar head.

Main Callback Function

The main callback function has to have the following definition

```
void Svs5Callback (int msgType, int size, const char* const value)
```

`msgType` is the type of data being passed from the DLL to the calling program. See Table 1.0.

`len` is the length of the value being passed to the calling program, See Table 1.0 for size of message structure.

`value` is the actual data being passed to the calling program, and is a pointer to one of the structures defined in Table 1.0.

The callback function is set up by calling the DLL function StartSvs5 as follows

```
StartSvs5(CallBackFn);
```

The design of the DLL requires that the callback function copies the data from `value` into local storage under its own control before returning, as the buffer holding the data in the DLL may be overwritten by other data coming from the sonar head if the block is used by the parent process after the callback function has returned.

Functions

The following functions are defined as the interface of the DLL. The API's definitions for these are contained in the file 'Svs5SequencerApi.h'. The DLL itself is known as Svs5SeqLib.dll.

For each of the functions exposed by the DLL, the following paragraphs give the name of the function, the C definition of the function, a description of the function, its parameters and return value (if any), an example of calling the function in C, and a summary of the default values and limits for the parameters (where applicable).

GetLibraryVersionInfo

```
const char* GetLibraryVersionInfo();
```

This function returns the version string which identifies the DLL.

StartSvs5

```
Svs5ErrorCode StartSvs5 (Svs5Callback fnSvs5);
```

This function takes the callback function as in input argument and returns the messages as defined in table 1.0. The library starts pumping data out as it receives from the sonar over network/serial interface.

Example usage:

```
SequencerApi::StartSvs5( std::bind(
                                &onGeminiMessageReceived,
                                std::placeholders::_1,
                                std::placeholders::_2,
                                std::placeholders::_3
                                 ) );
```

If the function is successful then returns SVS5_SEQUENCER_STATUS_OK else error code defined in Table 3.0.

Svs5SetConfiguration

```
Svs5ErrorCode Svs5SetConfiguration(
                    ESvs5ConfigType configType,
                    size_t          size,
                    const void*     value,
                    unsigned int    deviceID
                    );
```

This function configures the library with messages types defined in table 2.0. e.g. To start pinging to sonar, application calls “Svs5SetConfiguration” with message type “SequencerApi::SVS5_CONFIG_ONLINE” , size of datatype and pointer to the value. Device ID = 0 specifies to send it to all devices otherwise specific to device ID.

Example usage:

```
bool fOnline = true;
// configure sonar to go online
SequencerApi::Svs5SetConfiguration(
            SequencerApi::SVS5_CONFIG_ONLINE,
            sizeof(bool),
            &fOnline,
            0            // 0: all devices otherwise specific to device
            );
```

If the function is successful then SVS5_SEQUENCER_STATUS_OK is returned otherwise one of the error code is returned as defined in Table 3.0.

Svs5GetConfiguration

```
Svs5ErrorCode Svs5GetConfiguration(
                    ESvs5ConfigType configType,
                    size_t          size,
                    const void*     value,
                    unsigned int    deviceID
                    );
```

This function retrieves the library configuration status with messages types defined in table 2.0. e.g. To enquire sonar pinging status, application calls “Svs5GetConfiguration” with message type “SequencerApi::SVS5_CONFIG_ONLINE” , size of datatype and pointer to the value. If more than 1 devices are attached and “deviceID = 0” is specified then the library will returns the last device configuration.

Example usage:

```
bool fOnline = true;
// configure sonar to online
SequencerApi::Svs5GetConfiguration(
            SequencerApi::SVS5_CONFIG_ONLINE,
            sizeof(bool),
            &fOnline,
            );
```

If the function is successful then SVS5_SEQUENCER_STATUS_OK is returned otherwise one of the error code is returned as defined in Table 3.0.

StopSvs5

```
Svs5ErrorCode StopSvs5();
```

This function will shutdown the library i.e. stops listening to messages from COM interface and stops internal thread. This API should be called when application is exiting.

Example usage:

```
StopSvs5();
```

If the function is successful then SVS5_SEQUENCER_STATUS_OK is returned otherwise one of the error code is returned as defined in Table 3.0.

Example of Callback function

```
void onSvs5MessageReceived(
                    unsigned int      msgType,
                    size_t            size,
                    const char* const  value
                    )
{
   switch( msgType)
   {
      case SequencerApi::GEMINI_STATUS:
      {
         const CGemStatusPacket* const pStatus = (const CGemStatusPacket* const)value;
         // If status message is coming with 0 Alternate IP address then don't print anything
         if( !pStatus->m_sonarAltIp )
         {
            return;
         }
         std::cout<<  "Status  message received from : ";
         unsigned int from = pStatus->m_sonarAltIp;
         std::cout << std::dec << std::setfill( '0' ) << std::setw( 2 )
            << (int)( (from>>24) & 0xFF ) << "." << (int)( (from>>16) & 0xFF ) << "."
            << (int)( (from>>8) & 0xFF ) << "." << (int)( (from>>0) & 0xFF ) << "\n";

         std::string s;

         if ((pStatus->m_BOOTSTSRegister & 0x000001ff) == 0x00000001)
         {
            s = "Bootloader booted";
         }
         else
         {
            if (pStatus->m_shutdownStatus & 0x0001)
            {
               s = " (Over temp)";
            }
            else if (pStatus->m_shutdownStatus & 0x0006)
            {
               s = " (Out of water)";
            }
         }
         if( s.size() )
         {
            std::cout<< "Sonar status " << s << std::endl;
         }
      }
      break;
      case SequencerApi::GLF_LIVE_TARGET_IMAGE:
      {
         GLF::GLogTargetImage* logTgtImage = (GLF::GLogTargetImage*)value;

         std::cout
         << ( ( logTgtImage->m_mainImage.m_usPingFlags & 0x8000 ) ? " User Selected
SOS " : "Sonar Speed of Sound " )
         << logTgtImage->m_mainImage.m_fSosAtXd
         << ", Range Compression used " << logTgtImage-
>m_mainImage.m_usRangeCompUsed
         << ", Width " << logTgtImage->m_mainImage.m_uiEndBearing
         << ", Height "<< logTgtImage->m_mainImage.m_uiEndRange
         << ", Frequency " << ( ( logTgtImage->m_mainImage.m_usPingFlags & 0x0100 ) ?
"High" : " Low" )
         << ", Chirp " << ( ( logTgtImage->m_mainImage.m_fChirp ) ? "On" : "off" )
         << std::endl;

         // This information is provided by the application that sonar is
         // mounted inverted or not
         if( ( logTgtImage->m_mainImage.m_uiStateFlags & 0xE000 ) )
         {
            std::cout
            << "Sonar orientation is inverted "
            << std::endl;
         }
      }
      break;
      case SequencerApi::LOGGER_REC_UPDATE:
      {
         const GLF::SOutputFileInfo* sLoggerInfo = (const GLF::SOutputFileInfo*)value;
         std::cout<< "Record Info :\n"
            << "\tFileName:\t" << sLoggerInfo->m_strFileName << std::endl
            << "\tNo Of Records:\t" << sLoggerInfo->m_uiNumberOfRecords<< std::endl
            << "\tFile Size(bytes):\t" << sLoggerInfo->m_fileSizeBytes<< std::endl
            << "\tFree Disk Space:\t" << sLoggerInfo->m_diskSpaceFreeBytes<< std::endl
            << "\tPercentage Disk Space Free:\t" << sLoggerInfo-
>m_percentDiskSpaceFree<< std::endl
            << "\tRecording Time Left :\t" << sLoggerInfo->m_recordingTimeLeftSecs<<
std::endl;
      }
      break;
      case SequencerApi::LOGGER_PLAYBACK_UPDATE:     // handle log file playback
information
      {
         const GLF::SInputFileListInfo* info = (const GLF::SInputFileListInfo*)value;

         if (info && info->m_uiPercentProcessed && ( ( info->m_uiPercentProcessed % 100 )
== 0 ) )
         {
            // Print total records (frames) 100 frames means ( 0 -99 )
            std::cout<< "Number of Records " << info->m_uiNumberOfRecords << std::endl;
            // Print filenames
            for ( unsigned int i = 0; i < info->m_uiNumberOfFiles; ++i )
            {
               std::cout<< "Playingback filename " << info->m_filenames[ i ] << std::endl;
            }
            }
        }
        break;
        case SequencerApi::TGT_IMG_PLAYBACK:
        {
            GLF::GenesisPlaybackTargetImage*                logTgtImage                    =
(GLF::GenesisPlaybackTargetImage*)value;
            std::cout
            << "Playing back..."
            << " Width " << logTgtImage->m_pLogTgtImage->m_mainImage.m_uiEndBearing
            << " Height "<< logTgtImage->m_pLogTgtImage->m_mainImage.m_uiEndRange
            << " frame "<< logTgtImage->m_frame
            << std::endl;
        }
        break;
        case SequencerApi::LOGGER_FILE_INDEX:
        {
            std::cout<< "Playing File Index..." << *(int*)value<< std::endl;
        }
        break;
        case SequencerApi::LOGGER_STATUS_INFO:
        {
            GLF::GnsLoggerStatusInfo* statusInfo = (GLF::GnsLoggerStatusInfo*)value;
            std::cout<<  "Logger  status  info...\r\n"  <<  statusInfo->m_errType  <<  statusInfo-
>m_loggerStatusInfo<< std::endl;
        }
        break;
        case SequencerApi::FRAME_RATE:
        {
            unsigned int fps = *(unsigned int*)value;
            std::cout<< "Frames per seconds " << fps << std::endl;
        }
        break;
        case SequencerApi::GPS_RECORD:
        {
            GLF::GLogV4ReplyMessage*                gnsV4ReplyMsg                    =
(GLF::GLogV4ReplyMessage*)value;
            std::cout<< "GPS data received at " << std::setprecision (15)
                << gnsV4ReplyMsg->m_header.m_ciHeader.m_timestamp << std::endl;
            std::vector<unsigned        char>&        vecGps        =        *gnsV4ReplyMsg-
>m_v4GenericRec.m_vecData;

            // Raw ascii
            if( gnsV4ReplyMsg->m_header.m_ciHeader.m_dataType == 98 )
            {
                for( size_t i = 0; i < vecGps.size(); ++i )
                {
                    std::cout<< vecGps[ i ];
                }
            } //V4 recorded message
            else if( gnsV4ReplyMsg->m_header.m_ciHeader.m_dataType == 99 )
        {
          //GLF::Gps
          GLF::GpsDataRecord* pNewFixRec = (GLF::GpsDataRecord*)&vecGps[0];
          printf("Received GPS Fix rec (Lat = %.4f, Lon = %.4f, E = %.3f, N = %.3f)\n",
              pNewFixRec->m_llRec.m_latDegrees,                          pNewFixRec-
>m_llRec.m_longDegrees,
              pNewFixRec->m_enRec.m_easting, pNewFixRec->m_enRec.m_northing);
        }
        std::cout<< "\t Frame Number " << gnsV4ReplyMsg->m_uiFrame <<std::endl;
      }
      break;
      case SequencerApi::COMPASS_RECORD:
      {
        GLF::GLogV4ReplyMessage*                    gnsV4ReplyMsg                    =
(GLF::GLogV4ReplyMessage*)value;
        std::cout<<  "Compass  data  received  at  "  <<  std::setprecision  (15)  <<
gnsV4ReplyMsg->m_header.m_ciHeader.m_timestamp <<std::endl;
        std::vector<unsigned      char>&      vecCompass      =      *gnsV4ReplyMsg-
>m_v4GenericRec.m_vecData;

        if( gnsV4ReplyMsg->m_header.m_ciHeader.m_dataType == 98 ) // Raw ascii
        {
          for( size_t i = 0; i < vecCompass.size(); ++i )
          {
            std::cout<< vecCompass[ i ];
          }
        }
        else if( gnsV4ReplyMsg->m_header.m_ciHeader.m_dataType == 99 ) //V4 Recorded
message
        {
          GLF::CompassDataRecord*                    pNewRec                    =
(GLF::CompassDataRecord*)&vecCompass[0];
          // output Heading, Pitch and Roll to test...
          printf("Received Compass Rec (Hdg = %.2f, Pitch = %.2f, Roll = %.2f)\n",
              pNewRec->m_heading, pNewRec->m_pitch, pNewRec->m_roll);
        }
        std::cout<< "\t Frame Number " << gnsV4ReplyMsg->m_uiFrame <<std::endl;
      }
      break;
      case SequencerApi::AUXPORT1_DATA:
      {
        std::cout<< "Gemini AUX Port 1 data received " << std::endl;
        for ( unsigned int i = 0; i < size; ++i )
        {
          std::cout<< std::hex << (value[ i ] & 0xFF) << " ";
        }
        std::cout<< std::endl;
      }
      break;
      case SequencerApi::AUXPORT2_DATA:
      {
            std::cout<< "Gemini AUX Port 2 data received " << std::endl;
            for ( unsigned int i = 0; i < size; ++i )
            {
                std::cout<< std::hex << (value[ i ] & 0xFF) << " ";
            }
            std::cout<< std::endl;
        }
        break;
    }
}
```

Example of live configuration message

These functions relate to a specific `deviceID` or `0`. If 0 is specified then the configuration will apply to all sonar devices attached to the system otherwise it will apply to specific sonar ID. If user does not specify the device ID then device ID 0 will be used.
Note: Each Gemini sonar has a unique ID and will only accept commands which have a matching ID to its own. The unique ID is printed on the rear of the sonar, and is broadcast in status messages from that sonar.

Configure Online / Offline

```
bool fOnline = true; // True for online, False: Offline
// configure sonar to online
SequencerApi::Svs5SetConfiguration(
                    SequencerApi::SVS5_CONFIG_ONLINE,
                    sizeof(bool),
                    &fOnline,
                    );
```

Set Range

```
double rangeLinesInMeter = 20.0;
// Set range value
SequencerApi::Svs5SetConfiguration(
                   SequencerApi::SVS5_CONFIG_RANGE,
                   sizeof(double),
                   &rangeLinesInMeter
                   );
```

Set Gain

```
int gain = 70;
// Configure Gain value
SequencerApi::Svs5SetConfiguration(
                 SequencerApi::SVS5_CONFIG_GAIN,
                 sizeof(int),
                 &gain
                 );
```

Configure Simulation Mode

```
bool enableSim = true; // true for enable simulation, False : disable simulation
// configure simulation mode
SequencerApi::Svs5SetConfiguration(
                 SequencerApi::SVS5_CONFIG_SIMULATE_ADC,
                 sizeof(bool),
                 &enableSim
                 );
```

Free Run Ping Mode

```
SequencerApi::SequencerPingMode sPingMode;
sPingMode.m_bFreeRun = true;

// Configure ping mode
SequencerApi::Svs5SetConfiguration(
                    SequencerApi::SVS5_CONFIG_PING_MODE,
                    sizeof(sPingMode),
                    &sPingMode
                    );
```

Fixed interval Ping Mode

```
SequencerApi::SequencerPingMode sPingMode;
sPingMode.m_bFreeRun = false;
sPingMode.m_msInterval = 20;// Sonar will automatically ping after every 20 milliseconds,
                                   // limited by range settings

// Configure ping mode
SequencerApi::Svs5SetConfiguration(
                    SequencerApi::SVS5_CONFIG_PING_MODE,
                    sizeof(sPingMode),
                    &sPingMode
                    );
```

External TTL Trigger Ping Mode

```
SequencerApi::SequencerPingMode sPingMode;
sPingMode.m_bFreeRun = false;
sPingMode.m_extTTLTrigger = true;

// Configure ping mode
SequencerApi::Svs5SetConfiguration(
                    SequencerApi::SVS5_CONFIG_PING_MODE,
                    sizeof(sPingMode),
                    &sPingMode
                    );
```

Configure Manual speed of sound

```
SequencerApi::SequencerSosConfig sSosConfig;
sSosConfig.m_bUsedUserSos = true;
sSosConfig.m_manualSos = 1500;// Manually specified SOS

// Configure Speed of Sound
SequencerApi::Svs5SetConfiguration(
                            SequencerApi::SVS5_CONFIG_SOUND_VELOCITY,
                            sizeof(sSosConfig),
                            &sSosConfig
                            );
```

Configure Sonar speed of sound

```
SequencerApi::SequencerSosConfig sSosConfig;
sSosConfig.m_bUsedUserSos = false;

// Configure Speed of Sound
SequencerApi::Svs5SetConfiguration(
                              SequencerApi::SVS5_CONFIG_SOUND_VELOCITY,
                              sizeof(sSosConfig),
                              &sSosConfig
                              );
```

Configure Sonar Orientation

Note: This is only used for sonar imagery display purposes to identify the sonar orientation when mounted.

```
SequencerApi::ESvs5SonarOrientation orientation;
orientation = SequencerApi::SONAR_ORIENTATION_UP;

// Record the sonar orientation
SequencerApi::Svs5SetConfiguration(
                              SequencerApi::SVS5_CONFIG_SONAR_ORIENTATION,
                              sizeof(SequencerApi::ESvs5SonarOrientation),
                              &orientation
                              );
```

Configure Sonar Range Resolution

Note: This is only applicable for 1200ik platform.

Following example switch to the high resolution when crossing 20m range.

```
RangeFrequencyConfig rangeResolution;
rangeResolution.m_frequency      = FREQUENCY_AUTO;
rangeResolution.m_rangeThreshold  = 20;
SequencerApi::Svs5SetConfiguration(
                    SequencerApi::SVS5_CONFIG_RANGE_RESOLUTION,
                    sizeof(RangeFrequencyConfig),
                    &rangeResolution
                    );
```

Enable Sonar High Range Resolution

Following example enables the high resolution.

```
bool highRangeResolution = true;
// configure high range resolution
SequencerApi::Svs5SetConfiguration(
                    SequencerApi::SVS5_CONFIG_HIGH_RESOLUTION,
                    sizeof(bool),
                    &highRangeResolution
                    );
```

Auto Chirp Mode

Following example configure the chirp in AUTO mode. Manually disable chirp mode use ‘chMode = CHIRP_DISABLED’ and to enable chirp mode use ‘chMode =CHIRP_ENABLED’

```
CHIRP_MODE chMode = CHIRP_AUTO;
// configure high range resolution
SequencerApi::Svs5SetConfiguration(
                    SequencerApi::SVS5_CONFIG_CHIRP_MODE,
                    sizeof(CHIRP_MODE),
                    &chMode
                    );
```

Log Raw GPS Data

Calling this API will log GPS Raw serial data in the GLF log file. This data can be any custom ascii string and the user application will be responsible for decoding.

Note: This function with message type can only be used during logging and this data cannot be played back in Genesis.

```
const char GPS[] =
"$GPGGA,143305,4722.68807,N,00813.95741,E,2,07,1.4,0.0,M,0.0,M,2.2,0362*5C";

// GPS data string
SequencerApi::Svs5SetConfiguration(
                    SequencerApi::SVS5_CONFIG_LOG_RAW_GPS,
                    strlen(GPS), (const char* const)GPS
                    );
```

Log Raw Compass Data

Calling this API will log Compass Raw data in the GLF log file. This data can be any custom ascii string and the user application will be responsible for decoding.

Note: This function with message type can only be used during logging and this data cannot be played back in Genesis.

```
const char COMPASS[] = "H170.5P-1.2R-1.1T12.3D501.26B12.3A59W59LN47F0";
// Compass data string
SequencerApi::Svs5SetConfiguration(
                    SequencerApi::SVS5_CONFIG_LOG_RAW_COMPASS,
                    strlen(COMPASS), (const char* const)COMPASS
```

Log Compass Data

Calling this API will log a GLF::CompassDataRecord in the GLF log file. The user application will be responsible for decoding.

Note: This function with message type can only be used during logging and this data cannot be played back in Genesis.

```
GLF::CompassDataRecord hpr;     //Defaults to 0 for every axis and calculated value
//Populate as appropriate

SequencerApi::Svs5SetConfiguration(
            SequencerApi::SVS5_CONFIG_LOG_COMPASS,
            sizeof(GLF::CompassDataRecord), (const char* const)&hpr
);
```

Open 720im / Micron Gemini device on Regular COM port

The following message type is only specific to 720im / Micron Gemini platform.

```
//For Windows: use COM1, COM2.....
//For Linux: use /dev/ttyUSB0, /dev/ttyUSB1.....

ComPortConfig comPort1(false, false, 921600, "COM1");
SequencerApi::Svs5SetConfiguration(
                    SequencerApi::SVS5_CONFIG_OPEN_720IM_COM_PORT,
                    sizeof(ComPortConfig),
                    &comPort1,
                    0xFFFF
                    );
```

Configure Gemini AUX port

Configure Gemini AUX port, see SequencerApi::AuxPortConfig structure for further details

```
SequencerApi::AuxPortConfig auxPortConfig;
SequencerApi::Svs5SetConfiguration(
                    SequencerApi::SVS5_CONFIG_AUX_PORT,
                    sizeof(SequencerApi::AuxPortConfig), (const char* const)&auxPortConfig
                    );
```

Configure Noise Reduction filter

Configure Noise Reduction filter for Micron Gemini platform, following example turn on the Noise Reduction filter

```
bool enable = true;
SequencerApi::Svs5SetConfiguration(
                    SequencerApi::SVS5_CONFIG_NOISE_REDUCTION,
                    sizeof(bool), (const char* const)&enable
                    );
```

Enable / Disable H264 Compression

When connected via Ethernet, this will Enable/Disable H264 Compression (Enabled by default for Serial Comms)

```
bool enable = true; //True = Turn ON. False = Turn OFF
SequencerApi::Svs5SetConfiguration(
                 SequencerApi::SVS5_CONFIG_H264_COMPRESSION,
                 sizeof(bool), (const char* const)&enable
                 );
```

Example of firmware upgrade message

Configure firmware file location

This is the firmware file location (typically the data directory where Genesis is installed). The SDK will find the updated firmware version of the selected platform. If new firmware file is available then "UPGRADE_AVAILABLE" message will be sent through call back function. Please see the GeminiSDKConsole.cpp for an example

```
const char* firmwareLocationPath = "C:\\Program Files (x86)\\Tritech\\Genesis\\Application Files\\data";
SequencerApi::Svs5SetConfiguration(
                              SequencerApi::SVS5_CONFIG_FIRMWARE_LOCATION,
                              strlen(firmwareLocationPath),
                              firmwareLocationPath,
                              SonarID
                              );
```

Upgrade firmware

User can upgrade firmware on the target device by calling the following API or user can force to upgrade with the same version by setting forceUpdate to true.

```
bool forceUpdate = false;
SequencerApi::Svs5SetConfiguration(
                              SequencerApi::SVS5_CONFIG_UPGRADE_FIRMWARE,
                              sizeof(forceUpdate),
                              &forceUpdate,
                              SonarID
                              );
```

Abort firmware upgrade

Once the firmware upgrade is started then user can abort the firmware upgrade by calling the following API.

```
SequencerApi::Svs5SetConfiguration(
                              SequencerApi:: SVS5_CONFIG_ABORT_UPGRADE,
                              0
                              NULL,
                              SonarID
                              );
```

Example of logger configuration message

Switch logger between GLF/ECD

The Svs5Sequencer library has the capability to support both GLF and ECD log file formats. By default GLF file format is used, however user can switch logger to use legacy ECD log file format. To log data in GLF or in ECD format, user can switch the logger between GLF/ECD by using the following message.

Note: Logger can only be switched before start logging the data. ECD is deprecated and is dependent on boost libraries. ECD logger can be removed by during compiling time by setting -DECD_LOGGER=no.

```
bool glfLogger = false; // true for GLF, false for ECD. Default is GLF
SequencerApi::Svs5SetConfiguration(
                    SequencerApi::SVS5_CONFIG_LOGGER,
                    sizeof(bool),
                    &glfLogger
                    );
```

Configure default log location

Default log location applies to both ECD and GLF logger.

```
std::string recFileLocation(“C:\\LogData”);
SequencerApi::Svs5SetConfiguration(
                      SequencerApi::SVS5_CONFIG_FILE_LOCATION,
                      recFileLocation.size(),
                      recFileLocation.c_str()
                      );
```

Configure Recording

User can configure logger to start/stop recording. Recording can be started using the following message.

```
bool recStart = true; // true: start recording, false: stop recording
// Start recording
SequencerApi::Svs5SetConfiguration(
                    SequencerApi::SVS5_CONFIG_REC,
                    sizeof(bool),
                    &recStart
                    );
```

Once the recoding is started, the logger library will start sending message “SequencerApi::LOGGER_REC_UPDATE” in the callback function with data structure value ‘GLF::SOutputFileInfo’.See structure parameters for further details.

Example of playback configuration message

Configure logger to start playback log files

The Svs5Sequencer library also supports playing back log files. User has to select log file to playback and the logger will automatically switched to ECD/GLF based on the playback file selection.

```
std::vector<std::string> filesList;
filesList.push_back("C:\\Users\\ashehzad\\Documents\\log_2021-05-05-075044.glf");

SequencerApi::ListOfFileNames listOfFileNames( filesList );

// Start playback
SequencerApi::Svs5SetConfiguration(
                SequencerApi::SVS5_CONFIG_PLAY_START,
                sizeof(listOfFileNames),
                &listOfFileNames
                );
```

Based on the log file selected by the user, the logger will start sending playback log messages in the call back function. The library will first send the file index in the callback function with message “SequencerApi::LOGGER_FILE_INDEX”. For a selection of a single log file, it will always be 0.

For GLF log file selected, the logger will return SequencerApi::TGT_IMG_PLAYBACK message with structure “GLF::GenesisPlaybackTargetImage”.

For ECD log file selected the logger will return SequencerApi::ECD_IMG_PLAYBACK message with structure CTgtImg.

Configure logger to play a specific log file from the file index

These messages will be only applicable when logging the playback file. The following configuration message will only be useful when more than 1 log files are selected. This configuration will configure the logger to playback specific log file based on index provided in the SequencerApi::ListOfFileNames.

```
unsigned int index = 0;
// first pause the video then request for frame
SequencerApi::Svs5SetConfiguration(
                  SequencerApi::SVS5_CONFIG_FILE_INDEX,
                  sizeof(unsigned int),
                  &index
                  );
```

Configure logger to pause/resume playback log file

These messages will be only applicable when logging the playback file. The following configuration message will pause / resume playing back the log file.

```
// Pause/Resume playback
bool pause = true; // True: pause, False: resume playback
SequencerApi::Svs5SetConfiguration(
                SequencerApi::SVS5_CONFIG_PLAY_PAUSE,
                sizeof(bool),
                &pause
                );
```

Configure logger to play back log files in loop

The following configuration message will start playing back log files when reaching at the end of the record.

```
// Repeat playback
bool repeat = true;// True for repeat, false for stop at the last record of log file
SequencerApi::Svs5SetConfiguration(
                SequencerApi::SVS5_CONFIG_PLAY_REPEAT,
                sizeof(bool),
                &repeat
                );
```

Configure logger to stop playback log files

The following configuration message will stop playing back the log files.

```
// Stop playback
SequencerApi::Svs5SetConfiguration(
                SequencerApi::SVS5_CONFIG_PLAY_STOP,
                0,
                NULL
                );
```

Configure logger to set speed of playback log files

The following configuration message will set the logging speed when playing back the log files.

```
// Speed playback
Double speed = 1.0;// 1.0 (Time synchronized), 0: Free run
SequencerApi::Svs5SetConfiguration(
                SequencerApi::SVS5_CONFIG_PLAY_STOP,
                sizeof(double),
                &speed
                );
```

Configure logger to jump to a specific frame in log file

These messages will be only applicable when logging the playback file. The following configuration message will jump to a specific frame in the log file. The requested frame should be

```
unsigned int frame = 0;

// first pause the video then request for frame
SequencerApi::Svs5SetConfiguration(
                    SequencerApi::SVS5_CONFIG_PLAY_FRAME,
                    sizeof(unsigned int),
                    &frame
                    );
```

Example of live sonar image

```
#include "Svs5Seq/Svs5SequencerApi.h"
#include "Gemini/GeminiStructuresPublic.h"
#include "GenesisSerializer/GlfApi.h"

#include <iostream>

#include <iomanip>
#include <limits>

static void onGeminiMessageReceived(
                        unsigned int       msgType,
                        size_t             size,
                        const char* const   value
                        )
{
    switch( msgType)
    {
        case SequencerApi::GEMINI_STATUS:
        {
            const CGemStatusPacket* const pStatus = (const CGemStatusPacket* const)value;
            // If status message is coming with 0 Alternate IP adrress then don't print anything
            if( !pStatus->m_sonarAltIp )
            {
                return;
            }
            std::cout<<  "Status  message received from : ";
            unsigned int from = pStatus->m_sonarAltIp;
            std::cout << std::dec << std::setfill( '0' ) << std::setw( 2 )
                << (int)( (from>>24) & 0xFF ) << "." << (int)( (from>>16) & 0xFF ) << "."
                << (int)( (from>>8) & 0xFF ) << "." << (int)( (from>>0) & 0xFF ) << "\n";

            std::string s;

            if ((pStatus->m_BOOTSTSRegister & 0x000001ff) == 0x00000001)
            {
                s = "Bootloader booted";
            }
            else
            {
                if (pStatus->m_shutdownStatus & 0x0001)
                {
                    s = " (Over temp)";
                }
                else if (pStatus->m_shutdownStatus & 0x0006)
                {
                    s = " (Out of water)";
                }
            }
            if( s.size() )
            {
                std::cout<< "Sonar status " << s << std::endl;
            }
        }
        break;
        case SequencerApi::GLF_LIVE_TARGET_IMAGE:
        {
            GLF::GLogTargetImage* logTgtImage = (GLF::GLogTargetImage*)value;

            std::cout
            << ( ( logTgtImage->m_mainImage.m_usPingFlags & 0x8000 ) ? " User Selected
SOS " : "Sonar Speed of Sound " )
            << logTgtImage->m_mainImage.m_fSosAtXd
            << ", Range Compression used " << logTgtImage-
>m_mainImage.m_usRangeCompUsed
            << ", Width " << logTgtImage->m_mainImage.m_uiEndBearing
            << ", Height "<< logTgtImage->m_mainImage.m_uiEndRange
            << ", Frequency " << ( ( logTgtImage->m_mainImage.m_usPingFlags & 0x0100 ) ?
"High" : " Low" )
            << ", Chirp " << ( ( logTgtImage->m_mainImage.m_fChirp ) ? "On" : "off" )
            << std::endl;

            // This information is provided by the application that sonar is
            // mounted inverted or not
            if( ( logTgtImage->m_mainImage.m_uiStateFlags & 0xE000 ) )
            {
                std::cout
                << "Sonar orientation is inverted "
                << std::endl;
            }
        }
        break;
        case SequencerApi::FRAME_RATE:
        {
            unsigned int fps = *(unsigned int*)value;
            std::cout<< "Frames per seconds " << fps << std::endl;
        }
        break;
        case SequencerApi::AHRS_HPR_DATA:
        {
            GLF::CompassDataRecord *record =
            const_cast<GLF::CompassDataRecord*>(reinterpret_cast<const
            GLF::CompassDataRecord *>(value));
            //Output as desired
            printf("Console Test App:: Callback, received Compass Rec (Hdg = %.2f, Pitch =
            %.2f, Roll = %.2f)\n", record->m_heading, record->m_pitch, record->m_roll);
        }
        break;
    }
```

```
}

int main( int argc, char* argv[] )
{

   std::cout<< "SVS5 sequencer library version :  "<< SequencerApi::GetLibraryVersionInfo()
<< std::endl;

   SequencerApi::StartSvs5( std::bind(
                            &onGeminiMessageReceived,
                            std::placeholders::_1,
                            std::placeholders::_2,
                            std::placeholders::_3 ) );


   // default to go online
   bool fOnline = true;
   // configure sonar to online / offline
   SequencerApi::Svs5SetConfiguration(
                        SequencerApi::SVS5_CONFIG_ONLINE,
                        sizeof(bool),
                        &fOnline
                        );
   fOnline =! fOnline;

   char a = 'a';

   do
   {
      std::cin >> a;
      switch ( a )
      {
         case 'a': //online / offline
         {
            // Toggle online / offline
            SequencerApi::Svs5SetConfiguration(
                                 SequencerApi::SVS5_CONFIG_ONLINE,
                                 sizeof(bool),
                                 &fOnline
                                 );
            fOnline =! fOnline;
         }
         break;
      }
   }while ( a != 'z');

   return 0;
}
```

Example of playback log files

```
#include "Svs5Seq/Svs5SequencerApi.h"
#include "Gemini/GeminiStructuresPublic.h"
#include "GenesisSerializer/GlfApi.h"
#include "GenesisSerializer/GenericDataTypes.h"
#include "GenesisSerializer/GpsRecord.h"
#include "GenesisSerializer/CompassRecord.h"

#include <iostream>

#include <iomanip>
#include <limits>

static void onGeminiMessageReceived(
                    unsigned int      msgType,
                    size_t            size,
                    const char* const   value
                    )
{
   switch( msgType)
   {
      case SequencerApi::LOGGER_PLAYBACK_UPDATE:     // handle log file playback
information
      {
         const GLF::SInputFileListInfo* info = (const GLF::SInputFileListInfo*)value;

         if (info && info->m_uiPercentProcessed && ( ( info->m_uiPercentProcessed % 100 )
== 0 ) )
         {
            // Print total records (frames) 100 frames means ( 0 -99 )
            std::cout<< "Number of Records " << info->m_uiNumberOfRecords << std::endl;
            // Print filenames
            for ( unsigned int i = 0; i < info->m_uiNumberOfFiles; ++i )
            {
               std::cout<< "Playingback filename " << info->m_filenames[ i ] << std::endl;
            }
         }
      }
      break;
      case SequencerApi::TGT_IMG_PLAYBACK:
      {
         GLF::GenesisPlaybackTargetImage*                   logTgtImage                  =
(GLF::GenesisPlaybackTargetImage*)value;
         std::cout
         << "Playing back..."
         << " Width " << logTgtImage->m_pLogTgtImage->m_mainImage.m_uiEndBearing
         << " Height "<< logTgtImage->m_pLogTgtImage->m_mainImage.m_uiEndRange
         << " frame "<< logTgtImage->m_frame
         << std::endl;
```

```
        }
        break;
        case SequencerApi::LOGGER_FILE_INDEX:
        {
           std::cout<< "Playing File Index..." << *(int*)value<< std::endl;
        }
        break;
        case SequencerApi::LOGGER_STATUS_INFO:
        {
           GLF::GnsLoggerStatusInfo* statusInfo = (GLF::GnsLoggerStatusInfo*)value;
           std::cout<< "Logger status info...\r\n" << statusInfo->m_errType << statusInfo-
>m_loggerStatusInfo<< std::endl;
        }
        break;
        case SequencerApi::GPS_RECORD:
        {
           GLF::GLogV4ReplyMessage*                gnsV4ReplyMsg                =
(GLF::GLogV4ReplyMessage*)value;
           std::cout<< "GPS data received at " << std::setprecision (15)
              << gnsV4ReplyMsg->m_header.m_ciHeader.m_timestamp << std::endl;
           std::vector<unsigned        char>&        vecGps        =        *gnsV4ReplyMsg-
>m_v4GenericRec.m_vecData;

           // Raw ascii
           if( gnsV4ReplyMsg->m_header.m_ciHeader.m_dataType == 98 )
           {
              for( size_t i = 0; i < vecGps.size(); ++i )
              {
                 std::cout<< vecGps[ i ];
              }
           } //V4 recorded message
           else if( gnsV4ReplyMsg->m_header.m_ciHeader.m_dataType == 99 )
           {
              //GLF::Gps
              GLF::GpsDataRecord* pNewFixRec = (GLF::GpsDataRecord*)&vecGps[0];
              printf("Received GPS Fix rec (Lat = %.4f, Lon = %.4f, E = %.3f, N = %.3f)\n",
                   pNewFixRec->m_llRec.m_latDegrees,                          pNewFixRec-
>m_llRec.m_longDegrees,
                   pNewFixRec->m_enRec.m_easting, pNewFixRec->m_enRec.m_northing);
           }
           std::cout<< "\t Frame Number " << gnsV4ReplyMsg->m_uiFrame <<std::endl;
        }
        break;
        case SequencerApi::COMPASS_RECORD:
        {
           GLF::GLogV4ReplyMessage*                gnsV4ReplyMsg                =
(GLF::GLogV4ReplyMessage*)value;
           std::cout<<  "Compass  data  received  at  "  <<  std::setprecision  (15)  <<
gnsV4ReplyMsg->m_header.m_ciHeader.m_timestamp <<std::endl;
           std::vector<unsigned        char>&        vecCompass        =        *gnsV4ReplyMsg-
>m_v4GenericRec.m_vecData;
```

```
         if( gnsV4ReplyMsg->m_header.m_ciHeader.m_dataType == 98 ) // Raw ascii
         {
            for( size_t i = 0; i < vecCompass.size(); ++i )
            {
               std::cout<< vecCompass[ i ];
            }
         }
         else if( gnsV4ReplyMsg->m_header.m_ciHeader.m_dataType == 99 ) //V4 Recorded
message
         {
            GLF::CompassDataRecord*                          pNewRec                          =
(GLF::CompassDataRecord*)&vecCompass[0];
            // output Heading, Pitch and Roll to test...
            printf("Received Compass Rec (Hdg = %.2f, Pitch = %.2f, Roll = %.2f)\n",
                 pNewRec->m_heading, pNewRec->m_pitch, pNewRec->m_roll);
         }
         std::cout<< "\t Frame Number " << gnsV4ReplyMsg->m_uiFrame <<std::endl;
      }
      break;
   }
}

int main( int argc, char* argv[] )
{
   // List of playback file names
   std::vector<std::string> filesList;
   filesList.push_back( "C:\\GeminiData\\LD20190122\\data_2019-01-22-103513.ecd");

   std::cout<< "SVS5 sequencer library version :  "<< SequencerApi::GetLibraryVersionInfo()
<< std::endl;

   // check if file has selected
   if( filesList.size() == 0 )
   {
      std::cout<< "No file selected to playback !!" << std::endl;
      return -1;
   }

   bool playbackSTart = true;
   bool pause = true;
   SequencerApi::ListOfFileNames listOfFileNames( filesList );


   // initialize the Svs5Sequencer library
   SequencerApi::StartSvs5( std::bind(
                        &onGeminiMessageReceived,
                        std::placeholders::_1,
                        std::placeholders::_2,
                        std::placeholders::_3 ) );
```

```
// Configure logger to playback log files
if( SequencerApi::Svs5SetConfiguration(
               SequencerApi::SVS5_CONFIG_PLAY_START,
               sizeof(listOfFileNames),
               &listOfFileNames
               ) != SVS5_SEQUENCER_STATUS_OK )
{
   // failed to open log files
   return -1;
}

char cmd = 'a';
bool repeat = true;
double speed = 0;

do
{
   std::cin >> cmd;
   switch( cmd )
   {
      case 'p':
      {
      playbackSTart = false;
      // Stop playback
      SequencerApi::Svs5SetConfiguration(
                  SequencerApi::SVS5_CONFIG_PLAY_STOP,
                  0,
                  NULL
                  );
      }
      break;
      case 's':
      {
         // Pause / resume playback
         SequencerApi::Svs5SetConfiguration(
                  SequencerApi::SVS5_CONFIG_PLAY_PAUSE,
                  sizeof(bool),
                  &pause
                  );
         pause = !pause;
      }
      break;
      case 'f': //play brame by frame
      {
         UInt32 frame = 0;
         std::cin >> frame;
         // first pause the video then request for frame
         SequencerApi::Svs5SetConfiguration(
                  SequencerApi::SVS5_CONFIG_PLAY_FRAME,
                  sizeof(UInt32),
                  &frame
```

```
                              );
          }
          break;
          case '0': // Set speed ( run as fast as possible )
          {
             speed = 0.0;
             SequencerApi::Svs5SetConfiguration(
                              SequencerApi::SVS5_CONFIG_PLAY_SPEED,
                              sizeof(double),
                              &speed
                              );
          }
          break;

          case '1': // Set speed ( 1x real time )
          {
             speed = 1.0;
             SequencerApi::Svs5SetConfiguration(
                              SequencerApi::SVS5_CONFIG_PLAY_SPEED,
                              sizeof(double),
                              &speed
                              );
          }
          break;
          case 'l': //Loop Playback
          {
             // configure playback repeat mode
             SequencerApi::Svs5SetConfiguration(
                              SequencerApi::SVS5_CONFIG_PLAY_REPEAT,
                              sizeof(bool),
                              &repeat
                              );
             repeat = !repeat;
          }
          break;
       }
    }while ( cmd != 'z');

    // Stop sequencer library
    SequencerApi::StopSvs5();

    return 0;
}
```