Generic Data Format (GDF) file description

GDF files contain sonar data written in binary format. Each file can contain an arbitrary number of data blocks that each start with a standard block header. There are several types of data blocks, including: sensor configuration (type 1), kernel (type 2), and sonar (type 3). Note that data block type 5 can be ignored. In a single GDF file there will typically only be one each of data block types 1, 2, and 5, which are followed by multiple data blocks of type 3. These data blocks are described in the following sections. The last section provides example MATLAB code to read a GDF file.

Standard Block Header

The standard block header is always 12-bytes long and contains:

Byte	Format	Name	Description	Units
0	uint32	SYNC	Synchronization word, uint32(0x46444731)	
4	uint32	SIZE	Size of the following block	bytes
8	uint8	TYPE	Type of the following block	
9	uint8	VERSION	Version of the following block	
10	uint16	CHECKSUM	Checksum for validating the data integrity	

Type 1: Sensor Configuration Block

The sensor configuration is contained in data block type 1. These blocks define the nominal vehicle parameters and sensor parameters used for the duration of the track. This data block contains:

Byte	Format	Name	Description	Units
0	uint16	SYSTEM	Unique system identifier	
2	uint16	CHANNELS	Number of channels (hydrophones) in the array	
4	uint16	SPEED	Nominal forward vehicle speed	mm/us
6	uint16	ADVANCE	Nominal forward advance per ping	mm
8	uint32	INTERVAL	Nominal time interval per ping	usec
12	uint32	FS	Sampling rate of the receiver array	Hz
16	uint32	FC	Center frequency of the receive array	Hz
20	uint16	(spare)		
22	uint16	DATATYPE	Raw data format bitmask	
24	int32[3]	TXPOS	X, Y, and Z position of the transmitter	um
36	int32[3]	RXPOS	List of X, Y, and Z positions (one per channel)	um

The TXPOS and RXPOS are the positions of the elements relative to the vehicle origin, in forward (X), starboard (Y), down (Z) coordinates. RXPOS are in the same order as the time series data in the sonar data payload. For this simulated dataset the DATATYPE bitmask (at byte 22) is always 0x0401 and indicates that the data is complex (formatted as alternating real and imaginary components), 16-bit integers, and in little endian.

Type 2: Kernel Block

The kernel is contained in data block type 2. This data block contains:

Byte	Format	Name	Description	Units
0	uint16	KBID	Kernel block identifier	
2	uint16	(spare)		
4	uint32	SIZE	Number of complex samples to follow	
8	int32[2]	SAMPLES	List of complex samples	2^{30}

For this simulated data the KBID is 1 indicating that the kernel is the matched filter kernel for the sonar data, which can be used for pulse compression / replica correlation. The SAMPLES are a list of SIZE length complex samples. Each sample is normalized to the range of -1 to 1 and then serialized using 2^{30}, and a sample size of eight bytes per value (four bytes real, four bytes imaginary), explicitly in that order.

Type 3: Sonar Data Block

The sonar navigation and time series data is contained in data block type 3. For this simulated data, each sonar data block contains one navigation message and one ping of time series data. The header block contains:

Byte	Format	Name	Description	Units
0	uint32	NUMBER	Ping number	
4	uint32	TIME1	Time of acquisition – whole seconds since EPOCH	s
8	uint32	TIME2	Time of acquisition – fractional seconds	ns
12	uint32	SAMPLES	Number of samples per channel	
16	uint32	OFFSET1	Byte offset to sonar data (excluding this header)	bytes
20	uint32	SIZE1	Sonar data size	bytes
24	uint32	OFFSET2	Byte offset to navigation data (excluding this header)	bytes
28	uint32	SIZE2	Associated navigation data size	bytes

For this simulated data the ping NUMBER starts at 1 in the first file (*000.gdf) and monotonically increases. Note that for this dataset the byte offset to navigation data (OFFSET2) is smaller than the byte offset to sonar data (OFFSET1), indicating that the navigation data precedes the sonar data in the file. The order of the data contained in the sonar data block in this dataset is: header, navigation data header, navigation data message, and sonar data.

There is a navigation data header that immediately precedes each navigation message in the navigation data payload. This navigation data header contains:

Byte	Format	Name	Description	Units
0	uint32	TIME1	Time of acquisition – whole seconds since EPOCH	s
4	uint32	TIME2	Time of acquisition – fractional seconds	ns
8	uint16	SIZE	Navigation message size (excluding this header)	bytes
10	uint8	TYPE	Navigation message type	
11	uint8	VERSION	Navigation message version	

The message in the navigation data payload contains:

Byte	Format	Name	Description	Units
0	uint8	SYNC1	A syncword that should be 71	
1	uint8	SYNC2	A syncword that should be 78	
2	uint16	SIZE	Size of this navigation message, excluding first 8 bytes	bytes
4	uint16	(spare)		
6	uint16	CHECKSUM	Checksum for validating data integrity	
8	single	ALTITUDE	Distance to seafloor	m
12	single	DEPTH	Distance to sea surface	m
16	single	ROLL	Rotation around x-direction (positive is stbd down)	Radians
20	single	PITCH	Rotation around y-direction (positive is nose up)	Radians
24	single	HEADING	Rotation around z-direction (positive is nose stbd)	Radians
28	single	XVEL	Velocity in x-direction (stbd)	m/s
32	single	YVEL	Velocity in y-direction (forward)	m/s
36	single	ZVEL	Velocity in z-direction (down)	m/s
40	single	WATERSPEED	Sound velocity in the water	m/s

Note that CHECKSUM should equal the sum of the uint8 values from bytes 8 to the end of the message. For example, in MATLAB, CHECKSUM should be equal to "sum(msg(9:end))." Note that the tables in this document start indexing from 0 while MATLAB starts indexing from 1.

The message in the sonar data payload for this simulated dataset (with DATATYPE bitmask 0x0401) contains repeated entries of the following form:

Byte	Format	Name	Description	Units
0	int16[2]	SAMPLES	List of complex samples	2^{15}

The SAMPLES are a list of complex samples. Each sample is normalized to the range of -1 to 1 and then serialized using 2^{15}, and a sample size of four bytes per value (two bytes real, two bytes imaginary), explicitly in that order. The total number of data samples is SAMPLES (see byte 12 of the Sonar data block header) times number of channels of the sonar (see byte 2 of the Sensor configuration block). The data payload will contain all of the samples of hydrophone 1 data, then all of the samples of hydrophone 2 data, and so forth until all of the hydrophone data are read.

Example MATLAB Code

The code below demonstrates how a folder containing GDF files can be read into MATLAB. Each data block is processed and the information is saved into a structure variable, G1.

```
% Initialize data structure
G1 = struct;

% Indicate folder containing GDF files
folderNameGDF = 'path\to\folder';

% Determine file names
fileList = dir([folderNameGDF '\*.gdf']);

% Loop over all GDF files and load in the data
for fileNdex = 1:length(fileList)

    fileName = fileList(fileNdex).name;

    % Read in the GDF data
    fid = fopen(fullfile(folderNameGDF,fileName),'r');
    bin32 = fread(fid,'uint32=>uint32');
    bin32 = bin32(:).';
    fclose(fid);

    % Find the locations of the syncword indicating the start of a data block
    % Note: this simple method assumes the time series data doesn't contain the syncword
    syncWord = uint32(0x46444731);
    k = find(bin32 == syncWord);

    % Loop through each data block
    for kNdex = 1:length(k)

        % Read Standard Block Header information
        hdr = bin32(k(kNdex):k(kNdex)+2);
        size = hdr(2);
        temp = typecast(hdr(3),'uint8');
        type = temp(1);
        version = temp(2);
        checksum = typecast(temp(3:4),'uint16');
        data = typecast(bin32(k(kNdex)+2+1:k(kNdex)+2+size/4),'uint8');

        % Read the data according to the data block type
        switch type

            case 1 % Sensor Configuration Block
                G1.SensorConfig.system = typecast(data(1:2),'uint16');
                G1.SensorConfig.channels = typecast(data(3:4),'uint16');
                G1.SensorConfig.speed = single(typecast(data(5:6),'uint16'))/1000;
                G1.SensorConfig.advance = single(typecast(data(7:8),'uint16'))/1000;
                G1.SensorConfig.interval = single(typecast(data(9:12),'uint32'))/1e6;
                G1.SensorConfig.fs = single(typecast(data(13:16),'uint32'));
                G1.SensorConfig.fc = single(typecast(data(17:20),'uint32'));
                G1.SensorConfig.datatype = typecast(data(23:24),'uint16');
                 % Note: type 1025 [uint16(0x0401)] is data that is
                 %   complex (real, imag), 16-bit integer, & little endian
                G1.SensorConfig.txPos = single(typecast(data(25:36),'int32'))./1e6; % x,y,z
                G1.SensorConfig.rxPos = zeros(G1.SensorConfig.channels,3);
                for chNdex = 1:G1.SensorConfig.channels % Read each rx location
                    indStart = 37 + (chNdex-1)*3*4;
                    indEnd = indStart + 3*4 - 1;
                    G1.SensorConfig.rxPos(chNdex,:) = ...
                        single(typecast(data(indStart:indEnd),'int32'))./1e6;
                end

            case 2 % Kernel Block
                % Read header information
                G1.Kernel.kbid = typecast(data(1:2),'int16');
                % Note: KBID = 1 is matched filter kernel
                G1.Kernel.size = typecast(data(5:8),'uint32');
                G1.Kernel.waveform = zeros(G1.Kernel.size,1);
                % Read each kernel value and assemble them into a vector
                for sampNdex = 1:G1.Kernel.size
                    indStart = 9 + (sampNdex-1)*2*4;
```

```
                indEnd = indStart + 2*4 - 1;
                temp = single(typecast(data(indStart:indEnd),'int32'))./2^30;
                G1.Kernel.waveform(sampNdex,1) = temp(1) + 1i*temp(2);
            end

        case 3 % Sonar Data Block
            % Read header information
            pingNum = typecast(data(1:4),'uint32');
            pingTime = single(typecast(data(5:8),'uint32')) + ...
                       single(typecast(data(9:12),'uint32'))/1e9;
            nSamples = typecast(data(13:16),'uint32');
            offsetSonarData = typecast(data(17:20),'uint32');
            sizeSonarData = typecast(data(21:24),'uint32');
            offsetNavData = typecast(data(25:28),'uint32');
            sizeNavData = typecast(data(29:32),'uint32');
            % Read navigation data
            hdrNav = data(33+offsetNavData:33+offsetNavData+11);
            navTime = single(typecast(hdrNav(1:4),'uint32')) + ...
                      single(typecast(hdrNav(5:8),'uint32'))./1e9;
            navSizeNoHdr = typecast(hdrNav(9:10),'uint16');
            navType = typecast(hdrNav(11),'uint8');
            navTypeVer = typecast(hdrNav(12),'uint8');
            dataNav = data(33+offsetNavData+12:33+ ...
                offsetNavData+12+uint32(navSizeNoHdr)-1);
            G1.Ping{pingNum}.Nav.altitude(pingNum,1) = ...
                typecast(dataNav(9:12),'single');
            G1.Ping{pingNum}.Nav.depth = typecast(dataNav(13:16),'single');
            G1.Ping{pingNum}.Nav.roll = typecast(dataNav(17:20),'single');
            G1.Ping{pingNum}.Nav.pitch = typecast(dataNav(21:24),'single');
            G1.Ping{pingNum}.Nav.heading = typecast(dataNav(25:28),'single');
            G1.Ping{pingNum}.Nav.xVel = typecast(dataNav(29:32),'single');
            G1.Ping{pingNum}.Nav.yVel = typecast(dataNav(33:36),'single');
            G1.Ping{pingNum}.Nav.zVel = typecast(dataNav(37:40),'single');
            G1.Ping{pingNum}.Nav.waterspeed = typecast(dataNav(41:44),'single');
            % Read sonar data
            temp = typecast(data(33+offsetSonarData:33+ ...
                offsetSonarData+sizeSonarData-1),'int16');
            temp = single(temp(1:2:end)) + 1i*single(temp(2:2:end));
            temp = reshape(temp,nSamples,[])./2^15;
            G1.Ping{pingNum}.data = temp;
            % Use next lines to plot ping data
            % figure, plot(20.*log10(abs(temp(:,1)))),
            % xlabel('Sample'), ylabel('Amplitude [dB]')

        case 5 % Processing Mode Block
            % This information is not needed

    end % switch statement for type

  end % loop over kNdex

end % loop over fileNdex

%% Create plots of some of the data

% Plot sensor positions
figure, hold on
plot3(G1.SensorConfig.txPos(1,1),G1.SensorConfig.txPos(1,2),...
    G1.SensorConfig.txPos(1,3),'s','linewidth',2)
plot3(G1.SensorConfig.rxPos(:,1),G1.SensorConfig.rxPos(:,2),...
    G1.SensorConfig.rxPos(:,3),'o','linewidth',2)
xlabel('x'), ylabel('y'), zlabel('z')
view([60 40])
title('Sensor positions')
legend('Tx','Rx','location','northeast')

% create figure of a ping of data, averaged:
pingNum = 10;
figure
plot(20.*log10(mean(abs(G1.Ping{1,pingNum}.data),2)),'linewidth',2)
xlabel('Sample number')
ylabel('Amplitude, normalized [dB]')
title('Average data from a single ping')

% create figure of nav data
xVel = zeros(length(G1.Ping),1);
```

```
yVel = xVel;
zVel = xVel;
for pingNdex = 1:length(G1.Ping)
    xVel(pingNdex) = G1.Ping{1,pingNdex}.Nav.xVel;
    yVel(pingNdex) = G1.Ping{1,pingNdex}.Nav.yVel;
    zVel(pingNdex) = G1.Ping{1,pingNdex}.Nav.zVel;
    % Similarly, you can look at attitude, and altitude, etc.
end
figure, hold on
plot(xVel,'linewidth',2)
plot(yVel,'--','linewidth',2)
plot(zVel,':','linewidth',2)
xlabel('Ping number')
ylabel('Velocity [m/s]')
legend('x','y','z','location','northeast')
grid on
ylim([-.1 1])
title('Navigation (velocity) data')
```