INTERFACE REQUIREMENT SPECIFICATION FOR PXGF STREAMING AND FILE FORMAT

This document specifies the requirements for implementing the PXGF streaming format.

TABLE OF CONTENTS

TABLE OF TABLES

TABLE OF FIGURES

1 SCOPE

1.1 IDENTIFICATION

This is the functional specification of the PXGF Streaming and File Format.

1.2 OVERVIEW

The PXGF streaming and file format provides a framework for the streaming and storage of sampled data along with the meta data required to process the sampled data. It is a streaming format in that synchronisation can be regained if lost.

A file using the PXGF format contains a PXGF stream with a prepended header. The header was designed to allow an application to identify a file without processing the file. The capability to identify files becomes more important as file sizes get bigger. The PXGF file format supports large file sizes.

1.3 DOCUMENT OVERVIEW

This functional specification comprises the following sections:

Section 1: Scope

This section identifies the functional specification.

Section 2: PXGF Description

This section introduces the PXGF streaming and file format. It gives an overview of the framework and its functions.

Section 3: Abbreviations

This section contains the list of the abbreviations that are used in this document.

2 PXGF DESCRIPTION

The PXGF format was designed to represent sampled data with additional information pertaining to the way in which the data was sampled. This document is written with all examples and values in big endian notation.

2.1 BACKGROUND

The PXGF format is loosely based on the Microsoft RIFF file format. The RIFF format is based on the concept of a chunk. Chunks are blocks that contain specific application defined data. In the RIFF format the complete file is a single RIFF chunk. RIFF chunks and LIST chunks are currently the only two types of chunks that may contain sub-chunks. All the remaining chunks in the file are children of the global RIFF chunk.

The RIFF format is unsuitable for our purposes for two primary reasons:

1. The global RIFF chunk is limited in size to 4GB, thereby effectively restricting the file size to 4GB.
2. The RIFF format is unsuitable for streaming applications as one needs to read the whole file sequentially to be able to parse it. There is no synchronisation mechanism available.

For these reasons a new file and streaming format was proposed and developed, namely the PXGF format.

2.2 THE PXGF CHUNK STRUCTURE

The PXGF format puts data into chunks. Different types of chunks are defined to store different information. The type of a chunk is specified by an int32 field in the chunk as shown in Table 2.1. An application that requires data from a particular chunk will register to receive data from that type of chunk. Chunks that are not recognised are simply skipped over. The size field in the chunk allows unrecognised chunks to be skipped over. Each chunk starts with the sync number 0xA1B2C3D4.

Table 2.1: Structure of PXGF chunks.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, e.g. "SOFH", "EOFH", "SSNC".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
data	byte[size]	The chunk data in a format specific to the type.

- The maximum amount of data in a chunk is limited to 69632 (0x11000) bytes. This limits the separation between sync patterns.
- The length of each chunk must contain an integral number of 32 bit words even though the size element in the chunk header is specified as a number of bytes.
- The PXGF format supports both little and big endian byte ordering, although it may be necessary to provide the stream reader with the endian used depending on its implementation. The endian format for a file or stream may be determined by reading the

sync pattern. It is not permissible to mix chunks of different endian format within a stream or file.

- When the PXGF format is used to store information in a file, there must be a global header at the beginning of the file to aid identification of the file format and the data stored in the file. This is necessary due to the potentially large size of files.
- Nested sub-chunks are not supported as this would unnecessarily complicate synchronisation.
- The implication of the previous point is that all chunks are at root level and are interpreted entirely sequentially. The parser must know which chunks need to be identified before it can use other chunks. The only constraint here is that files must start with a “SOFH chunk”. Due to the sequential nature of parsing and the inability to nest chunks, a separate global chunk is needed to identify the end of the file header, namely the "EOFH" chunk.

2.3 APPLICATION NOTES

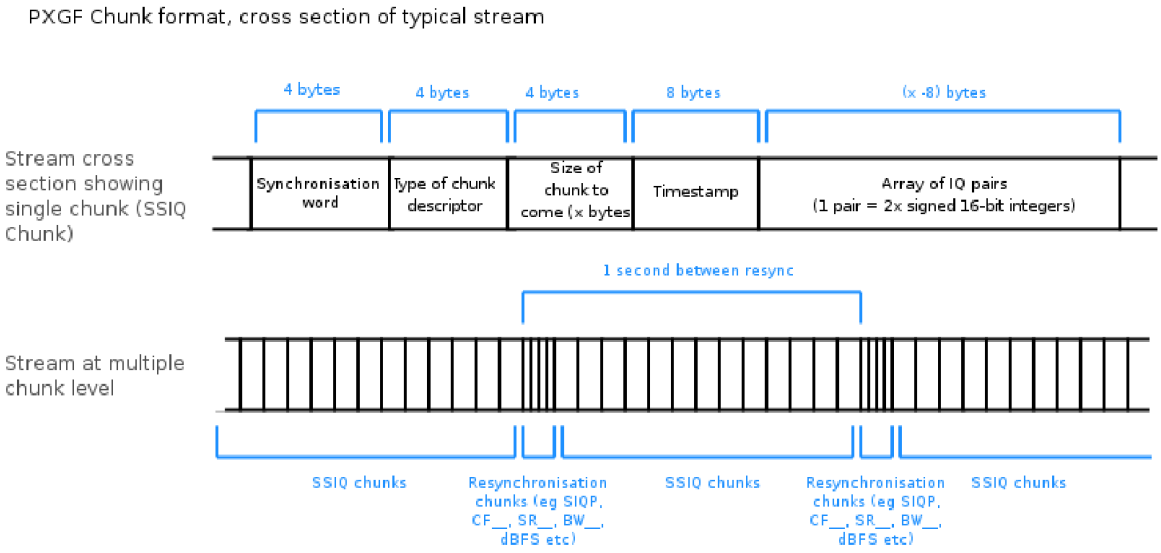

Figure 2.1: Example chunk usage in a PXGF stream.

Above is a graphical illustration showing how PXGF chunks of different type are ordered in a stream/file. Below are some notes to the developer to keep in mind when using the PXGF format in an application.

1. The PXGF framework for streaming and storage is designed to be extensible. Different applications require different information and if this information is not available in a stream, then that application will not be able to process that stream successfully. Just because an application uses the PXGF format doesn't mean that it will be able to process all PXGF streams or files. For a particular project care should be taken to ensure that all necessary chunks are included.
2. It is recommended that meta data like the sample rate and packing description be sent every second. This allows state information to be recovered if synchronisation is lost and makes it possible to process large files from the middle of the file.
3. Only data from one data source and of one format must be included in each stream or file. Current formats include "SSNC" for single channel data and "GSNC" for multi-channel data. The format used in files should be indicated using the SOFH chunk. The format name may

also be used for the file extension to allow visual discrimination of different files.

4. State information is accumulated by an application by reading different chunks sequentially. If synchronisation is lost, state information needs to be reset. This is why it is essential to resend meta data every second.
5. It is necessary to be able to distinguish between continuous data and block data where only part of the time data is available. Data chunks contain timestamps to enable detection of discontinuities. A chunk has also been defined to indicate discontinuities in the time data, namely the "IQDC" chunk.
6. Playback control is essential for the off-line analysis of files, however due to the stream based design of the PXGF format, playback control is not easily supported. The PXGF format uses data chunks supported by a number of meta chunks that describe the state of the data stream. Before processing data chunks it is necessary to obtain sufficient state information, like the sample rate, by processing the necessary chunks in the data stream.

   The use of an index file has been proposed as a possible solution to the problem of playback control. By reading an index file an application could determine over what period the recording was made and determine where to start processing the stream to play back a particular section.
7. C++ libraries have been developed for the writing and reading of PXGF streams. The libraries take care of synchronisation and formatting issues; they do not provide or dictate the communication medium.
8. The PXGF streaming format does not provide any mechanism for communication between the source of the data stream and the application receiving the data stream. The PXGF stream therefore represents a *unidirectional* flow of information from the source to the sink of the stream.
9. Applications that process PXGF input streams should not make assumptions about the data. For example, if the sample data were being sent using the SSNC chunk the application should wait for a SIQP chunk to determine the packing of the data rather than assuming a particular packing.

2.4 DEFINITION OF CHUNKS

2.4.1 Single channel Short Nanosecond-timestamped Complex time data – SSNC chunk

The SSNC chunk contains IQ data for single channel of data.

Data is assumed to be continuous when using this data format. If the data is blocky, an IQDC chunk must be sent after every block of continuous data. The in-phase and quadrature (IQ) samples from an Analogue to Digital Converter (ADC) with a maximum of 16 bit resolution are stored directly into the awIQData element of this chunk. Samples should not be scaled or modified before they are stored in the awIQData element. All ADCs with 16 bits or lower resolution should use this chunk. ADCs with more than 16 bit resolution should use the SFNC chunk in conjunction with the FFS_ chunk.

Table 2.2: The SSNC chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "SSNC".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
lTimestamp	int64	The timestamp is stored as a 64 bit signed number, representing the time of capture of the first sample in the chunk, in nanoseconds since beginning of the epoch (i.e. 1970-01-01T00:00:00Z).
awIQData	int16[length of IQ data array]	IQ pairs of signed int16 short numbers. Note that regardless of the number of valid bits, the most significant bits in each short should be used. This allows us to specify the full-scale level without needing to specify the number of bits.

2.4.2 Single channel IQ Packing – SIQP chunk

The information in this chunk is required to parse the data in any of the data carrying chunks such as the SSNC and SFNC chunks.

Table 2.3: The SIQP chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "SIQP".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
iIsIQPacked	int32	Value 1 for IQ ordering and value 0 for QI ordering. For example: a value of 1 will indicate that the first sample in the element awIQData of a SSNC chunk is an “I” sample.

2.4.3 Sample Rate – SR__ chunk

Table 2.4: The SR__ chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "SR__".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
lSampleRate_uHz	int64	The number of samples per second (in microhertz) that are being recorded by this channel.

2.4.4 BandWidth – BW__ chunk

The bandwidth centred about the centre frequency. If the bandwidth is not centred about the centre frequency use the BWOF chunk instead.

Table 2.5: The BW__ chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "BW__".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
lBandwidth_uHz	int64	The bandwidth of the signal in microhertz

2.4.5 BandWidth Offset Frequency – BWOF chunk

In some cases the signal bandwidth will not be centred about the centre frequency. Such cases may occur when demodulating SSB signals.

Table 2.6: The BWOF chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "BWOF".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
lBandwidth_uHz	int64	The bandwidth of the signal in microhertz
lOffsetFrequency _uHz	int64	The offset frequency of the signal band from the centre frequency in microhertz

2.4.6 Centre Frequency – CF__ chunk

Table 2.7: The CF__ chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "CF__".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
lCentreFrequency_uHz	int64	The centre frequency of the signal in microhertz

2.4.7 dB Full Scale – dBFS chunk

Table 2.8: The dBFS chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "dBFS".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
fFullScaleLevel_dBm	float32	The analogue input level to the ADC in dBm which will produce maximum full scale digital samples for the current IQ time data chunk integer type. e.g. If we are using SSNC chunks, then a dBFS chunk will indicate the analogue input level that will yield a maximum digital sample swing of +-(2^{15}-1). Note that this value may be different from the full scale value of the ADC.

2.4.8 dB Total Gain – dBTG chunk

Table 2.9: The dBTG chunk

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "dBTG".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
fGain_dB	float32	The total analogue gain from the input of the receiver (usually an antenna) to the input of the ADC.

2.4.9 IQ DisContinuity – IQDC chunk

This chunk should be sent as indicator to the reading application to reset its history. Discontinuities may be caused by samples being dropped, changes in sample rate, changes in bandwidth, changes in centre frequency or changes in receiver gain. In systems where continuous data is expected from a stream an IQDC chunk should not be expected unless a parameter change has forced a discontinuity. In some applications it may be necessary to send an IQDC chunk if the writing application doesn't support the necessary chunk to identify an obvious discontinuity, for instance if CF__ chunks were not supported but it was known that the centre frequency had changed an IQDC chunk could be send.

This chunk has zero size.

Table 2.10: The IQDC chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "IQDC".
size	int32	Zero.

2.4.10 Single channel Short Nanosecond-timestamped Real data – SSNR chunk

The SSNR chunk carries real data for a single channel.

Data is assumed to be continuous when using this data format. If the data is blocky, an IQDC chunk should be sent after every block of continuous data. This chunk can be used to send audio data.

Table 2.11: The SSNR chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "SSNR".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
lTimestamp	int64	The timestamp is stored as a 64 bit signed number, representing the time of capture of the first sample in the chunk, in nanoseconds since beginning of the epoch (i.e. 1970-01-01T00:00:00Z).
awRealData	int16[length of real data array]. The length of the array must be a multiple of 2.	Real signed int16 short numbers. Note that regardless of the number of valid bits, the most significant bits in each short should be used. This allows us to specify the full-scale level without needing to specify the number of bits. The number of real int16 shorts in the array must be a multiple of 2.

2.4.11 Group Short Nanosecond-timestamped Complex time data – GSNC chunk

The Group IQ chunk came out of the need to send multiple channels worth of time data sampled from several adjacent channels in the frequency domain. These channels are often slightly overlapped in the frequency domain and can be used to create FFT information of a wider bandwidth than what is contained in a single channel.

Data is assumed to be continuous when using this data format. If the data is blocky, an IQDC chunk should be sent after every block of continuous data.

Table 2.12: The GSNC chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "GSNC".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
lTimestamp	int64	The timestamp is stored as a 64 bit signed number, representing the time of capture of the first sample in the chunk, in nanoseconds since beginning of the epoch (i.e. 1970-01-01T00:00:00Z).
awIQData	int16[(length of the group IQ data array]	IQ pairs of signed int16 short numbers. Note that regardless of the number of valid bits, the most significant bits in each short should be used. This allows us to specify the full-scale level without needing to specify the number of bits. The packing used is described by the GIQP chunk.

2.4.12 Group IQ Packing – GIQP chunk

Since the GSNC chunk supports many variations in contents, the content specific information is supplied by the GIQP chunk, and is required to parse the chunk correctly.

Table 2.13: The GIQP chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "GIQP".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
iNumChannels	int32	The number of channels in the group
iIsIQPacked	int32	Value 1 for IQ ordering and value 0 for QI ordering. The first int 16 in the awIQData array mentioned above is an “I” sample if the value is 1.
iIncrement	int32	The number of samples to increment to read the next sample for a particular channel. The value of iIncrement will be 1 or iNumChannels.
aiChannelOffset	int32[iNumChannels]	The channel offset to the start of each channel given in samples where a sample is an IQ pair.

Examples:

Take a stream which contains 4 channels of IQ data A,B,C and D. If the data were packed as follows (where N = number of samples per channel):

- A[0] A[1] A[2] ... A[N-1] B[0] B[1] B[2] ... B[N-1] C[0] C[1] C[2] ... C[N-1] D[0] D[1] D[2] ... D[N-1]

  For this packing scheme: iIncrement = 1 and aiChannelOffset = [0 N 2N 3N]

- A[0] A[1] A[2] ... A[N-1] B[0] B[1] B[2] ... B[N-1] D[0] D[1] D[2] ... D[N-1] C[0] C[1] C[2] ... C[N-1]

  For this packing scheme: iIncrement = 1 and aiChannelOffset = [0 N 3N 2N]

- A[0] B[0] C[0] D[0] A[1] B[1] C[1] D[1] A[2] B[2] C[2] D[2] ... A[N-1] B[N-1] C[N-1] D[N-1]

  For this packing scheme: iIncrement = 4 and aiChannelOffset = [0 1 2 3]

2.4.13 Group Channel BandWidth – GCBW chunk

Table 2.14: The GCBW chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "GCBW".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
iChannelBandwidth_uHz	int64	The bandwidth of each channel contained in a GSNC chunk. Value stored in microhertz. It is assumed that all of the channels are sampled at the same sample rate, and therefore the bandwidth of all channels in the group is equal.

2.4.14 Group Centre Frequencies – GCF_ chunk

Table 2.15: The GCF_ chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "GCF_".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
iNumChannels	int32	The number of channels in the group.
alCentreFrequencies_uHz	int64[iNumChannels]	The centre frequency of each channel in microhertz.

2.4.15 Group Relative Gains – GRG_ chunk

Table 2.16: The GRG_ chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "GRG_".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
iNumChannels	int32	The number of channels in the group.
fGains_dB	float32[iNumChannels]	The total analogue gain in dB of each channel from the input of the receiver (usually an antenna) to the input of the ADC, relative to the dBTG chunk, which would be the total analogue gain of one of the channels. Therefore, the total analogue gain for a particular channel is equal to the sum of the value contained in this chunk and the value contained in the dBTG chunk.

2.4.16 Start Of File Header – SOFH chunk

All files must be started with an instance of the SOFH chunk. The presence of this chunk can be used to identify the file format as a PXGF file. For more information about what type of data is stored, the remaining chunks before the EOFH chunk should be evaluated. This chunk must only appear once at the start of a file.

Table 2.17: The SOFH chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "SOFH".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
iFormat	int32	Identifier for the format used in a file. It is recommended that the numeric value of the data chunk name used in the file be used for this, e.g. SSNC or GSNC.

2.4.17 End Of File Header – EOFH chunk

This chunk must contain an empty data block, i.e. the size must be 0. It is used to indicate the end of the header at the start of a file.

Table 2.18: The EOFH chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "EOFH".
size	int32	Zero.

2.4.18 TEXT string – TEXT chunk

Text chunk using UTF-8 encoding. Each character is stored as a byte. This chunk can be used to store meta data. It is suggested that this information appear in the header section of files.

Table 2.19: The TEXT chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "TEXT".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
iTextLength	int32	The number of bytes in the text message
ayMessage	byte[iTextLength]	Text encoded using UTF-8.
	byte[]	Zero padding to ensure word alignment of chunk.

2.4.19 Single channel Float Nanosecond-timestamped Complex time data – SFNC chunk

The SFNC chunk carries IQ data for a single channel with 32 bit floating point resolution.

Data is assumed to be continuous within a chunk when using this data format. If the data contains discontinuities, an IQDC chunk must be sent after every block of continuous data. This chunk is used when an ADC is used with more than 16 bit resolution or when scaling of samples is implemented. The packing for the SFNC chunk (i.e. IQ or QI) is determined by the SIQP chunk. The IQ samples from an ADC are stored directly into the afIQData element of this chunk. Samples from ADCs with 16 bits or lower resolution should use the SSNC chunk. ADCs with more than 16 bit resolution should use the SFNC chunk in conjunction with the FFS_ chunk. When scaling has been done on the samples, the FFS_ chunk should be calculated to indicate the scaling factor as explained in section 0.

Table 2.20: The SFNC chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "SFNC".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
lTimestamp	int64	The timestamp is stored as a 64 bit signed number, representing the time of capture of the first sample in the chunk, in nanoseconds since beginning of the epoch (i.e. 1970-01-01T00:00:00Z).
afIQData	float32[length of IQ data array]	IQ pairs of float32 numbers.

2.4.20 Single channel Float Nanosecond-timestamped Real data – SFNR chunk

The SFNR chunk carries real data for a single channel with 32 bit floating point resolution.

Data is assumed to be continuous when using this data format. If the data is blocky, an IQDC chunk should be sent after every block of continuous data. This chunk can be used to send audio data. This chunk should be used in conjunction with the FFS_ chunk.

Table 2.21: The SFNR chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "SFNR".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
lTimestamp	int64	The timestamp is stored as a 64 bit signed number, representing the time of capture of the first sample in the chunk, in nanoseconds since beginning of the epoch (i.e. 1970-01-01T00:00:00Z).
afRealData	float32[length of real data array]	Real float32 numbers.

2.4.21 Group Float Nanosecond Complex time data – GFNC chunk

The Group IQ chunk came out of the need to send multiple channels worth of time data sampled from several adjacent channels in the frequency domain. These channels are often slightly overlapped in the frequency domain and can be used to create FFT information of a wider bandwidth than what is contained in a single channel.

Data is assumed to be continuous when using this data format. If the data is blocky, an IQDC chunk should be sent after every block of continuous data.

Table 2.22: The GFNC chunk.

Element	Type	Description
lTimestamp	int64	The timestamp of the first sample in nanosecond resolution, this is the number of nanoseconds since beginning of the epoch.
afIQData	float32[(length of the group IQ data array]	IQ pairs of float32 numbers. The packing used is described by the GIQP chunk. The fullscale positive swing is described by the FFS_ chunk.

2.4.22 Float FullScale – FFS_ chunk

The FFS_ chunk is used to specify what the fullscale value for the SFNC and SFNR chunks is.

An SSNC chunk contains 16-bit signed samples, so that the sample values that can be represented range from -32768 to 32767. When using a 16-bit ADC, the samples from the ADC fit exactly into the SSNC chunks. If an N-bit ADC is used with N smaller than 16, only the N most significant bits of the SSNC chunks are used and the 16-N least significant bits are set to zero, so as not to influence the scaling.

If an ADC with N larger than 16 is used, the data can be represented using SFNC chunks, which contain 32-bit floating point samples. In this case, an FFS chunk must be added to indicate the number of bits in the ADC.

Table 2.23: The FFS_ chunk.

Element	Type	Description
sync	int32	Synchronisation number 0xA1B2C3D4
type	int32	Derived from the chunk name, "FFS_".
size	int32	The number of data bytes in the remainder of the chunk. The value of size must be a multiple of 4.
fFullscale	float32	The float32 value that specifies the full scale (maximum positive swing) value of the samples in the SFNR and SFNC chunks. This is used together with the dBFS chunk to determine what the actual signal level in dBm is.

The formulas and examples below describe how to calculate the value to put in the FFS chunk.

- When recording with an N-bit ADC
  - $FFS\_ = 2^{N-1}$
- When converting from signed integer (SSNC) to floating point (SFNC) samples
  - Choose any scale factor
  - $sfiqSample = ssiqSample \cdot scaleFactor$
  - $FFS\_ = scaleFactor \cdot 2^{15}$
- Example 1: Data from 16-bit ADC
  - $FFS\_ = 2^{15} = 32768$
- Example 2: Data from 24-bit ADC
  - $FFS\_ = 2^{23} = 8388608$
- Example 3: Direct conversion
  - $scaleFactor = 1$
  - $sfiqSample = ssiqSample$
  - $FFS\_ = 2^{15} = 32768$ *(as if it was recorded with a 16-bit ADC)*
- Example 4: Normalization of samples to the range [-1,1]
  - $scaleFactor = 2^{-15} = \frac{1}{32768}$
  - $sfiqSample = \frac{ssiqSample}{32768}$
  - $FFS\_ = 1$

2.5 SYNCHRONISATION

If an application is reading a file from the beginning there will be no trouble synchronising unless the file has become corrupted. However, if the application is connecting to an output stream in this format which has been running since prior to the connection, then it is necessary to ensure that the application becomes synchronised with the stream. Synchronisation is largely hidden from the developer as it is handled by the libraries for reading and writing PXGF streams.

The procedure to obtain synchronisation is as follows:

Step 1. Obtain synchronisation

1. Read the stream until the 4 byte sync pattern is recognised.
2. Reset state information in the application.
3. Move on to step 2.1.

Step 2. Attempt to process a chunk

1. Read the type and length of the chunk.
2. If the length is more than 69632 (0x11000) bytes return to step 1.1.
3. If there is a registered handler for the type then process it otherwise skip over the data of the chunk.
4. Move on to step 3.1.

Step 3. Check synchronisation

1. Read the sync pattern from the stream. If sync pattern matches move to step 2.1 otherwise move back to step 1.1.

3 ABBREVIATIONS

Abbreviation	Meaning
ADC	Analogue to Digital Converter
BW	Bandwidth
BWOF	Bandwidth Offset Frequency
CF	Centre Frequency
dB	Decibel
dBFS	dB Full Scale
dBTG	dB Total Gain
EOFH	End Of File Header
EOH	End Of Header
FFS	Float Fullscale
GCBW	Group Channel Bandwidth
GCF	Group Centre Frequencies
GIQP	Group IQ Packing
GRG	Group Relative Gains
GSNC	Group Short Nanosecond-timestamped Complex time data
IQ	In-phase and Quadrature
IQDC	IQ Discontinuity
RIFF	Resource Interchange File Format
SFNC	Single channel Float Nanosecond-timestamped Complex time data
SFR	Single channel Float Real
SIQP	Single channel IQ Packing
SOFH	Start Of File Header
SOH	Start Of Header
SR	Sample Rate
SSNC	Single channel Short Nanosecond-timestamped Complex time
SSNR	Single channel Short Nanosecond-timestamped Real data