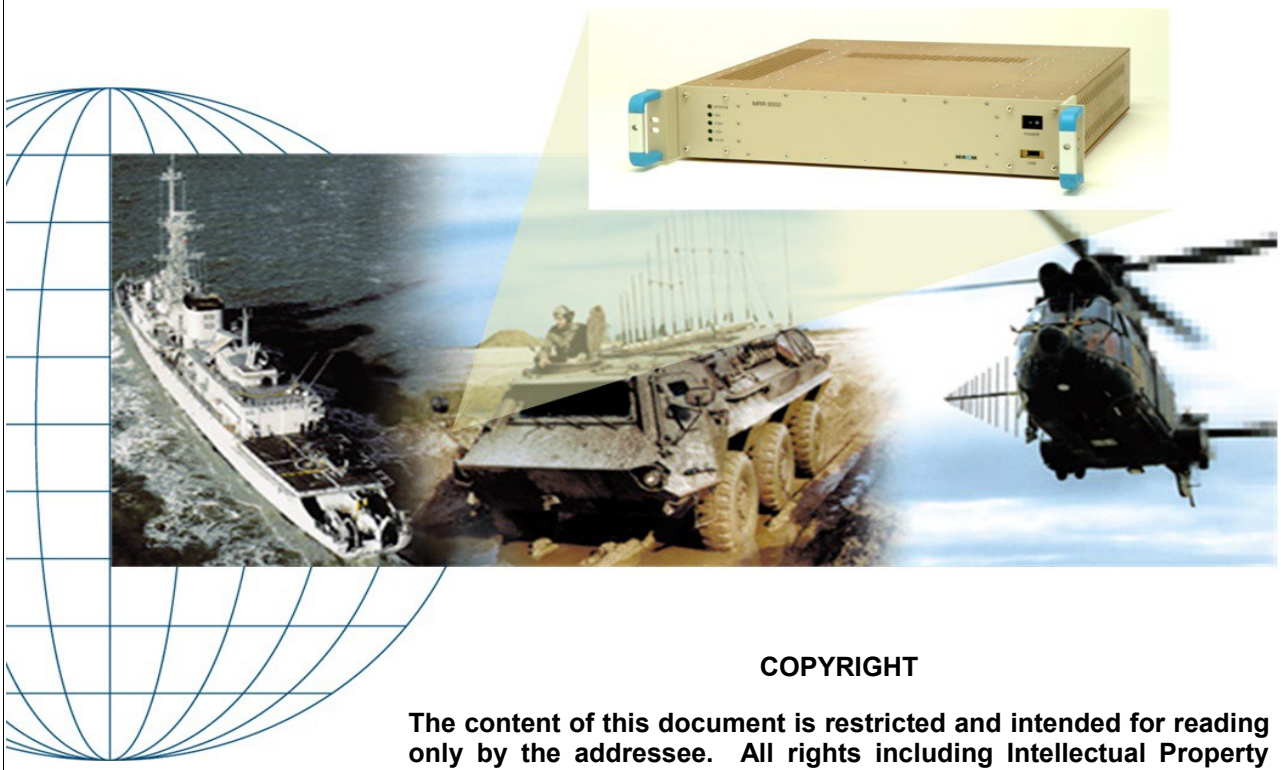
MRCM Products

Interface Requirement Specification for PXGF Streaming and File Format



COPYRIGHT

The content of this document is restricted and intended for reading only by the addressee. All rights including Intellectual Property Rights flowing from, incidental to or contained in this document irrevocably vest in Grintek Ewation (Pty) Ltd (GEW) unless otherwise agreed to in writing.

© GEW 2007

TABLE OF CONTENTS

1 SCOPE....6
1.1 IDENTIFICATION....6
1.2 OVERVIEW....6
1.3 DOCUMENT OVERVIEW....6
2 REFERENCED DOCUMENTS....7
3 PXGF DESCRIPTION....8
3.1 BACKGROUND....8
3.2 THE PXGF CHUNK STRUCTURE....8
3.3 APPLICATION NOTES....9
3.4 DEFINITION OF CHUNKS....10
3.4.1 Single channel Short IQ time data – SSIQ chunk....10
3.4.2 Single channel IQ Packing – SIQP chunk....10
3.4.3 Sample Rate – SR__ chunk....11
3.4.4 BandWidth – BW__ chunk....11
3.4.5 BandWidth Offset Frequency – BWOF chunk....11
3.4.6 Centre Frequency – CF__ chunk....11
3.4.7 dB Full Scale – dBFS chunk....11
3.4.8 dBTotal Gain – dBTG chunk....12
3.4.9 IQ DisContinuity – IQDC chunk....12
3.4.10 Single channel Short Real data - SSR_ chunk....12
3.4.11 Group Short IQ time data – GSIQ chunk....12
3.4.12 Group IQ Packing - GIQP chunk....13
3.4.13 Group Channel BandWidth – GCBW chunk....14
3.4.14 Group Centre Frequencies – GCF_ chunk....14
3.4.15 Start Of File Header – SOFH chunk....14
3.4.16 End Of File Header – EOFH chunk....14
3.4.17 TEXT string – TEXT chunk....14
3.5 PROPOSED EXTENSIONS....15
3.5.1 UTF-8 string – UTF8 chunk....15
3.5.2 IF frequency – IF__ chunk....15
3.5.3 Proposed text chunks....15
3.5.4 Direction data chunk....15
3.6 DEPRECATED CHUNK TYPES....15
3.6.1 Start Of Header – SOF_ chunk. Use SOFH chunk instead....15
3.6.2 End Of Header – EOH_ chunk. Use EOFH instead....16
3.7 SYNCHRONISATION....16
4 ABBREVIATIONS....17

Copyright: © Grintek Ewation 2007

LIST OF TABLES

TABLE 3.1: STRUCTURE OF PXGF CHUNKS....8
TABLE 3.2: THE SSIQ CHUNK....10
TABLE 3.3: THE SIQP CHUNK....10
TABLE 3.4: THE SR__ CHUNK....11
TABLE 3.5: THE BW__ CHUNK....11
TABLE 3.6: THE BWOF CHUNK....11
TABLE 3.7: THE CF__ CHUNK....11
TABLE 3.8: THE DBFS CHUNK....11
TABLE 3.9: THE DBFS CHUNK....12
TABLE 3.10: THE IQDC CHUNK....12
TABLE 3.11: THE SSR_ CHUNK....12
TABLE 3.12: THE GIQP CHUNK....13
TABLE 3.13: THE GIQP CHUNK....13
TABLE 3.14: THE GCBW CHUNK....14
TABLE 3.15: THE GCF_ CHUNK....14
TABLE 3.16: THE SOFH CHUNK....14
TABLE 3.17: THE EOFH CHUNK....14
TABLE 3.18: THE TEXT CHUNK....15
TABLE 3.19: PROPOSED UTF8 CHUNK....15
TABLE 3.20: PROPOSED IF__ CHUNK....15
TABLE 3.21: DEPRECATED SOF_ CHUNK....16
TABLE 3.22: DEPRECATED EOH_ CHUNK....16

Copyright: © Grintek Ewation 2007

LIST OF FIGURES

FIGURE 1: EXAMPLE CHUNK USAGE IN A PXGF STREAM.....9

Copyright: © Grintek Ewation 2007

AMENDMENT RECORD

ISSUE	DESCRIPTION	DATE	NAME
0.01	Initial version	2005-02-11	Robert Crida
0.02	Update after first value management meeting with Trevor, Adrian, Michael and Francois	2005-02-18	Robert Crida
0.03	Added chunk types after discussion with Trevor, Michael, Francois and Richard - channels of int16 IQ	2005-02-18	Robert Crida
0.04	Refined chunk types to support single and multichannel and added type names	2005-02-24	Robert Crida, Michael McGrath
0.05	Revised all bandwidths to micro Hz using int64	2005-02-25	Robert Crida, Michael McGrath
0.06	Added Group Descriptor chunks	2005-03-01	Michael McGrath
0.07	Changed Names of some chunks, changed some types to allow Java compliance	2005-03-02	Michael McGrath
0.08	Changed Names of some chunks, changed definition and name of GIQP chunk	2005-03-03	Robert Crida, Michael McGrath
0.09	Added chunk for bandwidths with frequency offsets. BWOF chunk	2005-03-16	Michael McGrath
0.10	Added "TEXT" chunk, moved "UTF8" and "IF__" chunks to the proposed chunk section.	2005-03-22	Michael McGrath
0.11	Added Purpose of Document header	2005-04-09	Richard Sharrock
0.12	Clarification of Timestamping, and added Use Of Format section	2005-06-21	Richard Sharrock
0.13	SOH_ chunk renamed to SOFH and EOH_ chunk renamed to EOFH to match software library.	2005-06-22	Michael McGrath
0.14	The Type of each header is stored as an int32 rather than a char[4] array, this change affects how the endian swapping is performed.	2005-06-22	Michael McGrath
0.15	SOH_ and EOH_ chunks deprecated, use SOHF and EOHF instead.	2005-06-22	Michael McGrath
0.16	Reformatted document into report format.	2005-10-10	Michael McGrath
0.17	Added chunk usage illustration.	2005-10-10	Richard Sharrock
0.18	Minor formatting changes	2005-10-14	Michael McGrath
0.19	Added real data chunk	2005-10-20	Michael McGrath, Richard Sharrock
0.20	Modified real data chunk	2005-10-20	Michael McGrath
0.21	Clarification of the dBFS value.	2005-12-07	Mark Cammidge
0.22	Changed title and doc number as per GEW request	2007-07-13	David Durrett
1.23	Changed the document number again to a more generic value. Changed the document file name. Changed the document issue number to be based at 1 and not 0.	2007-07-27	Trevor Bartleet
1.24	Updated table of contents and list of tables	2009-01-13	Philip Bliss

SOURCE DOCUMENT

Copyright: © Grintek Ewation 2007

ISSUE	FILE NAME
0.00	FileFormat_SPEQ.html
0.16	PXGF-Streaming Format.odt
1.23	svn://iserver.ct/raid/SVN/repos/EWDev/trunk/com/peralex/Framework/pxgf/documents/design/GSY-0D8-SE_IRS-PXGF-Streaming-Format.odt
1.24	svn://iserver.ct/raid/SVN/repos/EWDev/trunk/com/peralex/Framework/pxgf/documents/design/GSY-0D8-SE_IRS-PXGF-Streaming-Format.odt SVN 36470

Copyright: © Grintek Ewation 2007

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: Referenced Documents

This section lists all referenced documents.

Section 3.1: PXGF Description

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

Section 4: Abbreviations

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

Copyright: © Grintek Ewation 2007

2 REFERENCED DOCUMENTS

The following documents are referenced in this manual. Their applicability is limited to the referenced aspects only.

Ref.	Title	Document No.	Date and / or Issue	Source

Copyright: © Grintek Ewation 2007

3 PXGF DESCRIPTION

The PXGF format was designed to represent sampled data with additional information pertaining to the way in which the data was sampled.

3.1 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.

Copyright: © Grintek Ewation 2007

3.2 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 3.1. 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.

Element	Type	Description
sync	int32	Synchronisation number 0xa1b2c3d4
type	int32	Derived from the chunk name, e.g. "SOFH", " EOFH", "SSIQ".
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.

Table 3.1: Structure of PXGF chunks

- The maximum amount of data in a chunk is limited to 65536 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.

3.3 APPLICATION NOTES

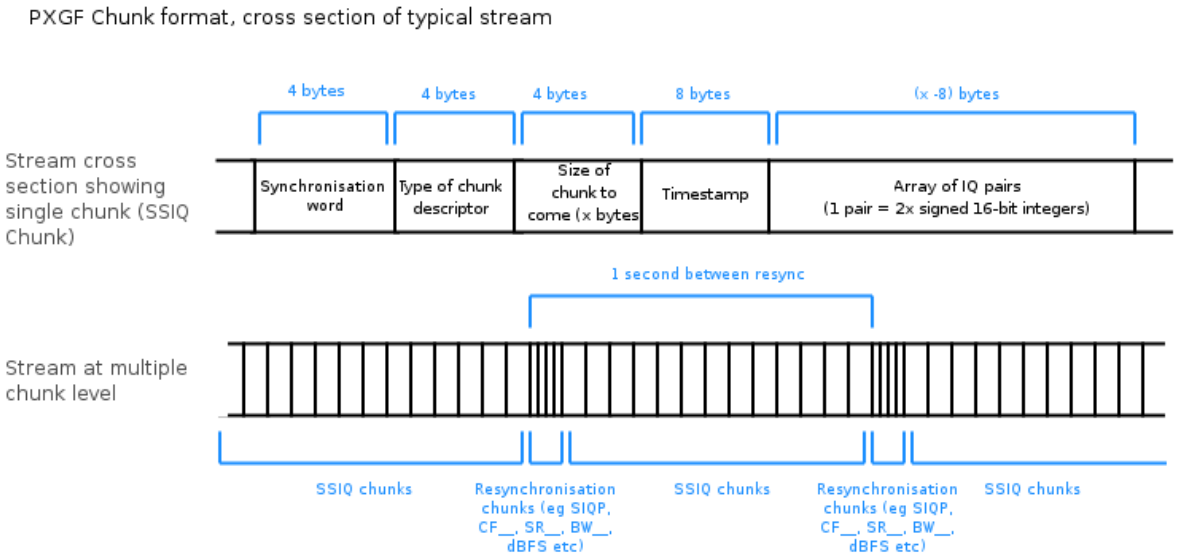


Figure 1: Example chunk usage in a PXGF stream

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 "SSIQ" for single channel data and "GSIQ" 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.

Copyright: © Grintek Ewation 2007

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++ and Java 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 SSIQ chunk the application should wait for a SIQP chunk to determine the packing of the data rather than assuming a particular packing.

3.4 DEFINITION OF CHUNKS

3.4.1 Single channel Short IQ time data – SSIQ chunk

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.

Element	Type	Description
lTimestamp	int64	The timestamp is stored as a 64 bit signed number, representative of the time of capture of the first sample in the chunk block, in microsecond resolution. It is stored as the number of microseconds since beginning of the epoch (i.e. 1st January 1970 midnight).
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.

Table 3.2: The SSIQ chunk

3.4.2 Single channel IQ Packing – SIQP chunk

The information in this chunk is required to parse the data in the SSIQ chunk.

Element	Type	Description
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 SSIQ chunk is an “I” sample.

Table 3.3: The SIQP chunk

Copyright: © Grintek Ewation 2007

3.4.3 Sample Rate – SR__ chunk

Element	Type	Description
lSampleRate_uHz	int64	The number of samples per second that are being recorded by this channel

Table 3.4: The SR__ chunk

3.4.4 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.

Element	Type	Description
lBandwidth_uHz	int64	The bandwidth of the signal in micro Hertz

Table 3.5: The BW__ chunk

3.4.5 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.

Element	Type	Description
lBandwidth_uHz	int64	The bandwidth of the signal in micro Hertz
lOffsetFrequency _uHz	int64	The offset frequency of the signal band from the centre frequency in micro Hertz

Table 3.6: The BWOF chunk

3.4.6 Centre Frequency – CF__ chunk

Element	Type	Description
lCentreFrequency_uHz	int64	The centre frequency of the signal in micro Hertz

Table 3.7: The CF__ chunk

3.4.7 dB Full Scale – dBFS chunk

Element	Type	Description
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. eg. If we are using SSIQ 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.

Table 3.8: The dBFS chunk

Copyright: © Grintek Ewation 2007

3.4.8 dBTotal Gain – dBTG chunk

Element	Type	Description
fGain_dB	float32	The total analogue gain from the input of the receiver (usually an antenna) to the input of the ADC.

Table 3.9: The dBTG chunk

3.4.9 IQ DisContinuity – IQDC chunk

This chunk should be sent as indicator to the reading application to reset it's 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.

Element	Type	Description
N/A	N/A	N/A

Table 3.10: The IQDC chunk

3.4.10 Single channel Short Real data - SSR_ chunk

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.

Element	Type	Description
lTimestamp	int64	The timestamp is stored as a 64 bit signed number, representative of the time of capture of the first sample in the chunk block, in microsecond resolution. It is stored as the number of microseconds since beginning of the epoch (i.e. 1st January 1970 midnight).
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.

Table 3.11: The SSR_ chunk

3.4.11 Group Short IQ time data – GSIQ 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.

Copyright: © Grintek Ewation 2007

Element	Type	Description
lTimestamp	int64	The timestamp of the first sample in microsecond resolution, this is the number of microseconds since beginning of the epoch
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.

Table 3.12: The GSIQ chunk

3.4.12 Group IQ Packing - GIQP chunk

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

Element	Type	Description
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.

Table 3.13: The GIQP chunk

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]

Copyright: © Grintek Ewation 2007

3.4.13 Group Channel BandWidth – GCBW chunk

Element	Type	Description
iChannelBandwidth_uHz	int64	The bandwidth of each channel contained in a GSIQ chunk. Value stored in micro Hertz. 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.

Table 3.14: The GCBW chunk

3.4.14 Group Centre Frequencies – GCF_ chunk

Element	Type	Description
iNumChannels	int32	The number of channels in the group
alCentreFrequencies_uHz	int64[iNumChannels]	The centre frequency of each channel in micro Hertz

Table 3.15: The GCF_ chunk

Copyright: © Grintek Ewation 2007

3.4.15 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.

Element	Type	Description
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. SSIQ or GSIQ.

Table 3.16: The SOFH chunk

3.4.16 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.

Element	Type	Description

Table 3.17: The EOFH chunk

3.4.17 TEXT string – TEXT chunk

Text chunk using ISO-8859-1 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. Text chunks with different names could be created or meta data could be encoded in a single text chunk using XML. See section 3.5.3 for a list of proposed text chunks.

Element	Type	Description
iTextLength	int32	The number of characters in the text message
ayMessage	byte[iTextLength]	Text encoded using ISO-8859-1.
	byte[]	Zero padding to ensure word alignment of chunk.

Table 3.18: The TEXT chunk

3.5 PROPOSED EXTENSIONS

3.5.1 UTF-8 string – UTF8 chunk

Text chunk using UTF-8 encoding. It may be desirable to be able to store text messages using different formats, eg UTF-8 etc.

Element	Type	Description
ayUTF8	byte[]	Text encoded using UTF-8. The byte array is to be padded to ensure word alignment.

Table 3.19: Proposed UTF8 chunk

3.5.2 IF frequency – IF__ chunk

Some systems require data to be played back at the original IF frequency used by the receiver. As the sampled data is already at baseband the IF frequency doesn't provide any information about the signal useful for analysis. This chunk could also cause confusion if the data were not at baseband, but rather offset to some frequency as occurs when using some demodulators in SSB mode. This special case is currently dealt with using the BWOF chunk.

Element	Type	Description
llfFrequency_uHz	int64	The IF frequency of the signal in micro Hertz

Table 3.20: Proposed IF__ chunk

3.5.3 Proposed text chunks

- Location of recording - TLOC.
- Equipment used to generate file - TREQ.
- Name of operator – TNOP.

3.5.4 Direction data chunk

1. Timestamp for start of first sample
2. Frequency of the first bin
3. Bin resolution
4. Number of bins
5. Azimuth/Elevation pairs encoded as shorts

3.6 DEPRECATED CHUNK TYPES

3.6.1 Start Of Header – SOF_ chunk. Use SOFH chunk instead

Used to denote the start of the header in a PXGF file. The presence of this chunk can be used to

Copyright: © Grintek Ewation 2007

identify the file format as a PXGF file. For more information about what type of data is stored, the remaining chunks before the EOH_ chunk should be evaluated. This chunk uses the same chunk type number as the SOFH chunk that replaces it.

Element	Type	Description
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. SSIQ or GSIQ.

Table 3.21: Deprecated SOF_ chunk

3.6.2 End Of Header – EOH_ chunk. Use EOFH instead

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. This chunk uses the same chunk type number as the EOFH chunk.

Element	Type	Description

Table 3.22: Deprecated EOH_ chunk

3.7 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 65536 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 step1.1.

Copyright: © Grintek Ewation 2007

4 ABBREVIATIONS

Abbreviation	Meaning
RIFF	Resource Interchange File Format

Copyright: © Grintek Ewation 2007