AcqKnowledge File Format

AS COMPLEX AS THE OCEAN IS DEEP 😊

Mike Davison | Independent Developer | 2017

Copyright and Disclaimer

NOTE: THIS DOCUMENT IS WORK IN PROGRESS. NOT FOR GENERAL DISTRIBUTION.

Acknowledgements

“If I have seen a little further it is by standing on the shoulders of Giants."

Isaac Newton 1676

I cannot claim to creating this information in isolation. Many thanks to the following sources for information and inspiration.

- Application Note 156 from BIOPAC.
- BIOREAD developed by Nate Vack and John Ollinger.
- BLINKY developed by Mike Davison for Murdoch University.

Many non-thanks to BIOPAC for not publishing the file formats of later versions. We can only hope…

Table of Contents

Introduction

BIOPAC's AcqKnowledge™ software saves its files in a complex binary file format that follows some order of structure, however much of the details of structures are undocumented and mysterious. Older file versions up to 3.9.1 were partially documented in BIOPAC's Application Note 156. As of writing this document AcqKnowledge™ is now up to release version 5.0.1, so there is a huge gap in information for later versions. Fortunately, trailblazers before me made inroads by spending hours analyzing the file structures using hex editors so there I more information now available. I have myself messed around trying to fathom the structures and have collated the information created by myself and others, along with blind guessing into one source document. Please note that this document is work in progress so I strongly encourage you to test the structure yourself.

The most frustrating aspect of interpreting the file structure is that it appears to be created by "committee" with little defined standards for data structures. By data structures I mean the configuration and parameter data structures as well as the raw collected data. In addition, there is a mixture of configurations, collected data, attachments, but not necessarily in any logical order.

Another bugbear is having no consistency over the structure of some fields such as text strings and boolean values.

For some reason string information is stored in one of three ways:

- Fixed length. A set number of bytes are stored.
- Variable length null terminated. These strings are stored as bytes terminated by a zero byte.
- Declared length (with or without null termination). A string length is specified usually as an int16 or int32 prior to the text followed by utf-8 text bytes. Sometimes string length includes the null termination zero byte, other times the length excludes the trailing zero byte, and other times there is no zero byte.

Boolean values are equally a challenge. Even in application notes app151 there is evidence of inconsistency way back to early version 3 releases. Boolean values are either 2 byte short or a 4 byte int. In most cases a true is represented by the value 1 and false by the value 0. However, there are cases where the boolean value false is sometimes represented by -1 (0xFF).

The next challenge is that in earlier versions of files numbers are stored in little endian order whereas later versions are stored big endian. The byte order also changes depending on what platform the application runs on (windows, mac etc).

And finally, in later versions some configuration data is stored as binary whilst others are stored as XML.

After all of the hurdles I think we have sufficient understanding of the structure to provide reasonably accurate interpretation. If you do use this information to write your own file reader application, hopefully it will be of use, but I strongly recommend you spend a lot of time testing and debugging.

Again, I repeat…. NO GUARANTEES OR WARRANTIES EXPRESSED OR IMPLIED AS TO THE ACCURACY OR USABILITY OF THIS INFORMATION

Onwards!!

ENDIAN or endian

Before we get into the data structures I would like to touch on the topic of Big/Little Endian file formatting and an EASY way to find out which one the file is.

To cater for different platforms the files are stored as either BIG EDIAN or LITTLE ENDIAN depending on version and also operating system that created the file. Earlier version 3 files all seem to be LITTLE on the Windows platform.

To correctly read the structure you need to establish the endianness of the file. Best wy to do this is via the version field in the graph header.

The file structure begins with:

2 bytes – int16 ← header length on older files, o on newer files.

4 bytes – int32 ← file version number.

To work out byte order:

1. Read version number.
2. Change endian of reader and read version number again.
3. Lower of the two read values will tell the endianness of the file.

A list of current known version numbers based on experimentation:

Number in file	Version number	Comment
30	2.0a	
31	2.0b	
32	2.0r	
33	2.07	
34	3.0r	
35	3.03	
36	3.5x	
37	3.6x	
38	3.70	
39	3.73	
41	3.81	
42	3.7P	
43	3.82	
44	3.8P	
45	3.90	
61	4.00B	
68	4.00	
76	4.01	

78	4.02	Best guess match
80	4.1a	Best guess match
83	4.10	
84	4.11	
108	4.20	Best guess match
121	4.2x	
124	4.30	
128	4.40	
132	5.01	

General File Structure

The conceptual structure of the file consists of elements (blocks of data) that contains configuration information and/or data. Each element is either fixed size, or have a variable size defined by information within the element data. The location of sizing information differs across element types. Sections in this document contains details of element layout and clearly identifies how to calculate the size of each element. All versions have elements listed in the sequence shown in the COMMON table below. From version 4 the files have added elements storing configuration and other additional elements.

COMMON ELEMENTS

Seq	Type	Description	Count
1	Graph Header	File data metrics plus User Interface (UI) configuration parameters.	1 per file only
2	Unknown	Unknown block in version 4.3 and above. Consists of int32 byte count (always 40) followed by unknown data (36 bytes)	1 per file.
3	Channel Header	Channel data metrics plus Channel User Interface (UI) configuration parameters.	1 per channel. Multiple per file.
4	Foreign Data	Contains external configuration information or data. Contains details of the data size plus the actual data. Version 3 has data however version 4 does not. Based on analysis I believe that the “Foreign Data” described in application note 156 from BIOPAC is actually some of the configuration data structures similar to that described in ver 4 elements.	1 per file only.

Seq	Type	Description	Count
5	Channel Type Header	Describes the type and configuration of channel.	1 per channel. Multiple per file.
6	Channel Data	Uncompressed data in an interleaved array. Note: on compressed files all data is stores as a “Snapshot” see the Snapshot element description.	1 per file only.
7	Marker Header	Marker data metrics, count plus marker User Interface configuration parameters.	1 per file only.
8	Marker Item	Marker style and location information.	1 per marker. Multiple per file.
9	Marker Header 2	Count and UI interface configuration for later version 3 files. This is extra data for each marker.	1 per file only.
10	Extra Marker Item Configuration	Additional marker configuration information. Seems to be for version 3.8 and 3.9 files only.	1 per marker. Multiple per file.
11	Journal	Journal header and text information. Version 3 raw text. Version 4 HTML. Version 4.2 also supports embedded images. The images are stored immediately after the journal.	1 per file only.
12	Journal Objects	Images stored as binary PNG objects. Version 4.2 onwards only.	Multiple per file
13	Unknown	Unknown purpose Version 3.8 and 3.9 files only.	1 per file.
14	Snapshots	Compressed data files have the data stored as a snapshot here. Later versions can also have additional snapshots stored regardless of whether primary data I compressed or not.	Header plus multiple snapshots.
15	Measurement UI Header	Defines the setup of the graph measurement UI display stored as XML. Version 4 onwards.	1 per file only.
16	Automarker Header	From version 4 you can set up hotkey and automated markers. This appears to be the header for auto-marker information. Version 4 onwards.	1 per file only.
17	Automarkers	Configuration of each auto-marker	Multiple per file.
18	Unknown	Unknown purpose version 4.2 onwards	1 per file only.
19	Unknown	Unknown purpose version 4.2 onwards	1 per file only.
20	Unknown	Unknown purpose version 4.3onwards	1 per file only.
21	Unknown	Unknown purpose version 4.3 onwards	1 per file only.
22	Unknown	Unknown purpose version 4.3 onwards	1 per file only.

Seq	Type	Description	Count
23	PDF Header	From version 4.4 you can attach pdf files to be viewed alongside the journal. This section contains a header for embedded pdf files	1 per file
24	PDF	From version 4.4 embedded pdf files stored as binary images.	Multiple per file
25	Unknown	Version 4.4 unknown purpose	1 per file only.
26	Key Values	Stores key values as XML. Version 4 onwards	1 per file only.
27	Text Annotations	Configuration and count of text annotations stored as XML. Version 4 onwards.	1 per file only.
28	Unknown	Version 4.11 onwards. Unknown purpose	1 per file only.
39	Media Header	Appears to be details of any media (audio/ video) associated with data collection stored as XML. Version 4.11 onwards.	1 per file only.
30	Graph History	Configuration and count of historical actions on graph data. Stores data modification history , eg transformations. Stored as XML. From 4.11 onwards.	1 per file only.
31	Unknown	Version 4.2 onwards. Unknown purpose. Seems to be just a zero integer on its own.	1 per file only.
32	Unknown	Version 4.2 onwards. Unknown purpose	1 per file only.
33	Focus Area Header	Configuration and count of focus areas. Stores details of any focus areas attached to the graph. Stored as XML. Version 4.3 onwards.	1 per file only.
34	Timers	Configuration of timers in XML format. Version 4.3 onwards.	1 per file only.
35	Output Control Presets	Stores output control presets. Stored as XML. Version 4.3 onwards	1 per file only.
36	Foreign Data	Large data block with general configuration information. Version 4 onwards. NOTE: it appears that because of the extra data requirements of foreign data in later versions, the data has been moved to the end of the file. THEORY ONLY.	1 per file only.

UNKNOWN FIELDS

Whilst every effort has been made to identify each byte of configuration there are still many fields and chunks of data within elements that need further work. In such cases these chunks are listed as unknown fields or just as chunks of unknown byte data. Any ideas would be greatly appreciated.

UNKNOWN BLOCKS

Unknown blocks are areas where their purpose is not known. This information is provided to assist in future development of our understanding of the blocks. Usually where there are chunks of unknown purpose bytes between known elements I have tried to group them into logical blocks to make it easier for future understanding.

Here is what I do know about the unknown block structures. The unknown ones come in either 4, 8,10, or 12 byte blocks. They appear to be headers for information not stored or configured in the particular file but act as place markers. THEORY ONLY SO NO GUARANTEES.

The breakdown of the unknown blocks appears to be:

4 byte blocks.

- A single int32 usually 0x00000000 indicating nothing in the block. I have only used this type to properly align the known blocks to make it easier to read the file.

8 Byte Blocks.

- An int32 possibly for number of bytes in the block (if populated).
- A 4 byte, or int32 with a block id. Many of the known blocks seem to have a unique ID to indicate or make sure correct data is read from the block. So assuming this is the same for empty unknown blocks. Binary dumps of the blocks support this theory.

10 Byte Blocks.

- An int32 possibly for number of bytes in the block (if populated).
- A 4 byte, or int32 with a block id.
- A short (int16 or 2 byte Boolean) could be to flag whether there is additional data or not or a count of sub elements within the block.

12 Byte Blocks.

- An int32 possibly for number of bytes in the block (if populated).
- A 4 byte, or int32 with a block id.
- A int32 (or 4 byte Boolean) could be to flag whether there is additional data or not or as an indicator of amount of additional data (if populated).

COMPRESSED DATA

Data is compressed with zlib compression, and is little-endian regardless of the endianness of the rest of the file. Snapshots consist of a series of zlib compressed structures.

EMBEDDED FILES AND OBJECTS

Later versions of the file can have embedded files which are images for the journal and pdf files attached to the journal. The embedded objects are binary compatible with how they would be stored as a separate file.

PDF's – same structure as a file with .pdf extension.

Images – same structure as a file with .png extension.

Graph data

UNCOMPRESSED DATA

In uncompressed files, the data is *interleaved* in the sequence as the channels appear in the headers , at the sample rate matching the frequency divider. That is; for files where all channels are sampled at the same rate the interleaving is in order. If you have say 4 channels the data sequence would be laid out as

1 2 3 4 1 2 3 4 1 2 3 4 ….. patterns repeating

Mapping this to the data in the file. You would see the following sequence:

S#*1	1	2	3	4	5	6	7	8	9	10	11	12	13	…
Ch*2														
1	P1*3				P2				P3				P4	
2		P1				P2				P3				P4
3			P1				P2				P3			
4				P1				P2				P3		

*1. S# = Data read sequence number *2. Ch = Channel *3. Px = Data read point number

Mapping this to a timeline of data:

Time	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	…
Channel														
1	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	etc
2	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	etc
3	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	etc
4	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	etc

However the file can also contain different sample rates for each channel and is specified by the channels frequency_divider parameter. The frequency divider can be powers of 2 only.

If you say have 3 channels with the frequency dividers of 1,2 and 4 respectively then the channels with a frequency divider > 1 will have gaps in their data (when measuring across time). For the above example of 3 channels the data layout is:

1 2 3 1 1 2 1 1 2 3 1 1 2 1 1 2 3 1 1 2 1 1 2 3 1 1 2 1 ….patterns repeating

Mapping this to the data in the file. You would see:

S#*1	1	2	3	4	5	6	7	8	9	10	11	12	13	…
Ch*2														
1	P1			P2	P3		P4	P5			P6	P7		P8
2		P1				P3			P5				P7	
3			P1							P5				

*1. S# = Data read sequence number *2. Ch = Channel *3. Px = Data read point number

Mapping this to a timeline of data:

Time	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	…
Channel														
1	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	etc
2	P1	Gap	P3	Gap	P5	Gap	P7	Gap	P9	Gap	P11	Gap	P13	etc
3	P1	Gap	Gap	Gap	P5	Gap	Gap	Gap	P9	Gap	Gap	Gap	P13	etc

*Gap = no data available for that particular timeslice.

Data channels can also have differing data types. Data types can be either 2 byte int16 (e.g java short) , or 8 byte IEEE floating point (e.g java double) numbers. Be aware of the endianness of the file when reading data.

To obtain the actual value for 2 byte data you need to multiply by the channel's amplitude value then add the channel's offset value.

COMPRESSED DATA

When the file is flagged "compressed" this means that the channel data is stored in a "snapshot". See the snapshot section for a breakdown of snapshot structures.

Each channel of data is stored in its own zlib compressed data chunk. You still need to know whether the channel is int16 or double(32) data. Also be aware that all compressed data is little endian.

If the file only contains one snapshot this this snapshot is you primary data.

If the file contains multiple snapshots then the last snapshot is you most recent view of primary data. Prior snapshots are historical views of the data.

Data Types

These are the different data types that appear to be used in the files. Data types are:

Type	Size in bytes	Similar to
Int16	2	Int16, short
Int32	4	Int32, long
double	8	F32, double
byte	1	byte, char
bool4	4	int32
bool2	2	Int16, bool, boolean
Fixed Length String	Preset by config	char[n]. Strings have a predefined size. Any unused characters are filled with null (ox00)
Variable length string	n	Size of the string is stored in another field.

Data Structures

This section provides details of the structure of header and item data structures. Fields are listed in the order they appear in the file. When new versions of Acqknowledge were released, additional fields were added to the header structure. The Version Notes tells you which version additional fields are added.

GRAPH HEADER (SEQUENCE 1)

Purpose

General information about the file and configuration.

Structure

Field	Description/ Purpose	Data Type	Version Notes
itemHeaderLen	Unused	int16	All
version	Version number.	int32	All
extITemHeaderLen	Header Length	int32	All
numChannels	Number of channels stored	int16	All
horizAxisType	Horizontal scale type [1]	int16	All

Field	Description/ Purpose	Data Type	Version Notes
currChannel	UI Currently selected channel	int16	All
sampleTime	The number of milliseconds per sample.	double	All
timeOffset	The initial time offset in milliseconds.	double	All
timeScale	UI time scale in milliseconds per division.	double	All
timeCursor1	UI Cursor 1 time position in milliseconds	double	All
timeCursor2	UI Cursor 2 time position in milliseconds	double	All
chartWindow	The chart's size and position relative to the AcqKnowledge client area. When each RECT field is set to 0, the chart is displayed with default a size and position.	8 bytes	All
measurement	Describes the currently selected measurements. Note: set to zero in V4 files because measurements are defined I XML in v4 files.	6 * int16	All
highlight	Gray non-selected waveforms: 0 = Don't gray 1 = Gray	int16	All
firstTimeOffset	Initial time offset in milliseconds. If not starting at 0.	double	All
rescale	Autoscale after transforms: 0 = Don't autoscale 1 = Autoscale	int16	All
horizUnits1	Horizontal units text.	40 bytes fixed length string	All
horizUnits2	Horizontal units text (abbreviated).	10 bytes fixed length string	All
inMemory	Keep data file in memory: 0 = Keep 1 = Don't keep	int16	All
grid	Enable grid display. 0 = Enable	int16	All

Field	Description/ Purpose	Data Type	Version Notes
	1 = Disable		
markers	Enable marker display. 0 = Enable 1 = Disable	int16	All
plotDraft	Enable draft plotting. 0 = Enable 1 = Disable	int16	All
displayMode	Display mode: 0 = Scope 1 = Chart.	int16	All
reserved	Reserved.	int16	All
showToolbar	Show Toolbar 0 = Hide 1 = Show	int16	3.0 to 3.9
showChanButt	Show channel select buttons 0 = Hide 1 = Show	int16	3.0 to 3.9
showMeasurement	Show Measurements 0 = Hide 1 = Show	int16	3.0 to 3.9
showMarker	Show Markers 0 = Hide 1 = Show	int16	3.0 to 3.9
showJournal	Show Journal 0 = Hide 1 = Show	int16	3.0 to 3.9
curXChannel	Current selected channel	int16	3.0 to 3.9
mmtPrecision	Number of decimal places in measurements.	int16	3.0 to 3.9
measurementRow	Number of measurement rows	int16	3.0.2 to 3.9
mmt	Measurement functions displayed 1 per measurement	40 * int16	3.0.2 to 3.9
mmtChan	Channel each measurement is acting on 1 per measurement	40 * int16	3.0.2 to 3.9
mmtCalcOpnd1	First operand of measurement calc per measurement	40 * int16	3.5 to 3.9
mmtCalcOpnd2	Second operand of measurement calc per measurement	40 * int16	3.5 to 3.9
mmtCalcOpt	Measurement calc operation per measurement	40 * int16	3.5 to 3.9

Field	Description/ Purpose	Data Type	Version Notes
mmtCalcConstant	Constant used in calculation per measurement	40 * double	3.5 to 3.9
newGridMinor	New grid with minor lines.	bool4	3.7 to 3.9
colorMajorGrid	Major grid colour	4 byte COLORREF	3.7 to 3.9
colorMinorGrid	Minor grid colour	4 byte COLORREF	3.7 to 3.9
majorGridStyle	Major Grid Style [3]	int16	3.7 to 3.9
minorGridStyle	Minor Grid Style [3]	int16	3.7 to 3.9
majorGridWidth	width of line in Pixels	Int16	3.7 to 3.9
minorGridWidth	width of line in Pixels	Int16	3.7 to 3.9
fixedUnitsDiv	Locked/Unlocked grid lines	bool4	3.7 to 3.9
midRangeShow	show gridlines as MidPoint and Range	bool4	3.7 to 3.9
startMiddlePoint	Startpoint to draw grid	double	3.7 to 3.9
offsetPoint	Offset of VERTICAL value per channel	60 * double	3.7 to 3.9
hGrid	Horizontal grid spacing	double	3.7 to 3.9
vGrid	Vertical grid spacing per channel	60 * double	3.7 to 3.9
enableWaveTool	Enable Wavetools during acquisition	bool4	3.7 to 3.9
horizPrecision	digits of precision for units in Horizontal Axis	Int16	3.7.3 to 3.9
reserved	RESERVED	20 * byte	3.8.1 to 3.9
overlapMode	Overlap Mode	bool4	3.8.1 to 3.9
showHardware	Hardware visibility	bool4	3.8.1 to 3.9
xAutoPlot	Autoplot during acquisition	bool4	3.8.1 to 3.9
xAutoScroll	Autoscroll during acquisition	bool4	3.8.1 to 3.9
startButtVisible	Start button visibility	bool4	3.8.1 to 3.9
compressed	The file is compressed	bool4	3.8.1 to 3.9
alwaysStartButtVisible	Always show start button	bool4	3.8.1 to 3.9
pathVideo	Path to playback video file	260 byte string	3.8.2 to 3.9
optSyncDelay	use sync delay between start of video file and graph start.	bool4	3.8.2 to 3.9
syncDelay	Value of sync delay (ms)	double	3.8.2 to 3.9
hrpPasteMeasurement	paste measurements to journal (when Hold Relative Position is selected)	bool4	3.8.2 to 3.9

Field	Description/ Purpose	Data Type	Version Notes
graphType	Type of the graph. The graph type identifies the source of the graph and whether any special transformations apply.	Int32	3.8.1 to 3.9
mmtCalcExpr	Measurements parameters: holds the expression entered by the user.	40 Fixed length strings 256 characters	3.8.1 to 3.9
mmtMomentOrder	Measurements parameters: the order of the moment for moment measurements.	40 * int32	3.8.1 to 3.9
mmtTimeDelay	Measurements parameters: the time delay to use for the computation in sample intervals	40 * int32	3.8.1 to 3.9
mmtEmbedDim	Measurements parameters: the embedding dimension for a measurement.	40 * int32	3.8.1 to 3.9
mmtMiDelay	Measurements parameters: the delay for which the mutual information should be computed.	40 * int32	3.8.1 to 3.9
V4blob	Byte array of all remaining bytes in header. See *note below.	Byte[]	V4 onwards

Special Notes

*Note: As of this time the information in the header after reserved (from version 3.0 onwards) is unknown and differs from Version 4 sequence. The field extITemHeaderLen contains the size of the header element. Until the information structure is resolved you should use the following sequence:

- read until “reserved” field.

- skip 822 bytes

- read the compressed (int32 4 byte Boolean) to determine if file is compressed.

- skip forward to the length of the header.

1	horizAxisType: 0 = Time in seconds 1 = Time in HMS format 2 = Frequency 3 = Arbitrary
2	measurement: 0 = No measurement 1 = Value Absolute voltage 2 = Delta Voltage difference 3 = Peak to peak voltage 4 = Maximum voltage 5 = Minimum voltage 6 = Mean voltage 7 = Standard deviation 8 = Integral 9 = Area 10 = Slope 11 = LinReg 13 = Median 15 = Time 16 = Delta Time 17 = Freq 18 = BPM 19 = Samples 20 = Delta Samples 21 = Time of Median 22 = Time of Max 23 = Time of Min 25 = Calculation 26 = Correlation
3	Grid Styles (TYPEDEF) PS_SOLID, PS_DASH, PS_DOT, PS_DASHDOT, PS_DASHDOTDOT

UNKNOWN (SEQUENCE 2)

This is the 40 bytes immediately after the graph header.

Field	Description/ Purpose	Data Type	Version Notes
itemLength	Length of element block	Int32	4.3 on
data	Unknown data	Byte[36*]	All

* itemLength minus 4 bytes. Check itemLength field in case this changes.

CHANNEL HEADER (SEQUENCE 3)

Each channel has a header containing configuration information. 1 per channel, multiple per file.

Field	Description/ Purpose	Data Type	Version Notes
ChanHeaderLen	Length of channel header.	Int32	All
Channelnumber	Channel number (in sequence).	Int16	All
name	Comment text. Channel name.	40 bytes string	All
rgbColor	Color.	Byte[4]	All
DisplayOption	Display option.	Int16	All
VoltOffset	Amplitude offset (volts).	double	All
VoltScale	Amplitude scale (volts/div).	double	All
UnitsText	Units text.	20 byte string	All
BufLength	Number of data samples in channel	Int32	All
AmplScale	Units/count. Used to calculate when using int data	Double	All
AmplOffset	Units. Used to calculate when using int data	Double	All
ChanOrder	Displayed channel number.	Int16	All
DispSize	Channel partition size.	Int16	All
unknown	Unknown block in Ver4 files. sampleDivider follows immediately after.	40 bytes	4.0 on
plotMode	Plot mode	Int16	3.0 to 3.9
vMid	Mid point on graph	double	3.0 to 3.9
description	Description of channel	128 byte string	3.7 to 3.9
sampleDivider	Divider for frequency of data	Int16	3.7 on
vertPrecision	Precision of vertical scale	Int16	3.73 to 3.9
segmentColor	Active segment colour	4 bytes RGBA	3.82 to 3.9
segmentStyle	Style of segment line	Int32	3.82 to 3.9

Special Notes

*Note: As of this time the information in the header after reserved (from version 3.0 onwards) is unknown and differs from Version 4 sequence. The field ChanHeaderLen contains the size of the header element. Until the information structure is resolved you should use the following sequence:

- read until “dispSize” field.

- skip 40 bytes

- read the sampleDivider (int32).

- skip forward to the length of the header.

FOREIGN DATA (SEQUENCE 4)

In version 3 files foreign data comes after the channel headers and has the following structure.

Field	Description/ Purpose	Data Type	Version Notes
length	Length of foreign data block including header	Int16	3.0-3.9
id	Foreign data id	Int16	3.0-3.9
data	Foreign data	Byte[length -4	3.0-3.9

In version 4 files foreign data is slightly different and always appear to be empty after the header with 4 bytes of 0. I think this is because the larger foreign data for V4 is now at end of file. See Sequence 37 block.

Field	Description/ Purpose	Data Type	Version Notes
length	Length of foreign data including header	Int32	4.0 on
data	Foreign data	Byte[length -4]	4.0 on

CHANNEL TYPE HEADER (SEQUENCE 5)

For each channel there is a header describing the type of data stored for the channel.

Field	Description/ Purpose	Data Type	Version Notes
dataPointSize	Size of each data point. 8 if double and 2 if int16.	Int16	All
dataPointType	Tells you what type of data. 1 = double, 2 = integer. * See note.	Int16	All

*Warning. In some files the dataPointType is 0. This means it double data so your test should include a test for 0.

CHANNEL DATA (SEQUENCE 6)

If the file is not compressed data is stored here as interlaced data. See the Graph Data section for how the data is laid out.

To calculate the total bytes of all data:

Sum (For each channel : bufLength * dataPointSize).

MARKER HEADER (SEQUENCE 7)

There are 3 sections required to read markers. The Marker Header section tells you how many markers. One header per file.

Field	Description/ Purpose	Data Type	Version Notes
markersByteLength	Size of all markers including header	Int32	All
markersCount	Count of markers	Int32	All

MARKER ITEM (SEQUENCE 8)

Marker Items. One per marker. Multiple per file. Version 3 markers are only global.

Version 3 marker structure.

Field	Description/ Purpose	Data Type	Version Notes
sampleIndex	Location of marker. This is the datapoint from the start of the data	Int32	3.0 to 3.9
selected	Boolean defines if this particular marker is selected.	Bool2	3.0 to 3.9
textLocked	Boolean defines if this particular marker text locked.	Bool2	3.0 to 3.9
positionLocked	Boolean defines if this particular marker is position locked.	Bool2	3.0 to 3.9
markerTextLength	Length of text	Int16	3.0 to 3.9
markerText	Text comment attached to the marker.	n byte string length defined by markerTextLength + 1 (there is a null terminator).*	3.0 to 3.9

- Note Ap151 says string length includes null but this is not the case. You need to add 1 to the length.

Version 4 marker structure.

From version 4 markers can be associated with a specific channel. The new structure is:

Field	Description/ Purpose	Data Type	Version Notes
sampleIndex	Location of marker. This is the datapoint from the start of the data	Int32	V4 on
unknown	Unknown int32	Int32	V4 on
channelNumber	Channel this marker belongs to . -1 means global marker.	Int16	V4 on
markerType	A 4 byte sting of the marker type. See marker type table below.	Bool2	V4 on
channelNumber2	Appears to be second copy of channel number	Int16	Only V4.3 on
colour	RGBA colour of marker	Int32 4 byte colour	Only V4.3 on
Unknown	Short unknown reason	Int16	Only V4.3 on
Unknown	2 x int32 unknown reason	2 x Int32	Only V4.4 on
markerTextLength	Length of text	Int16	V4 on
markerText	Text comment attached to the marker.	n byte string length defined by markerTextLength (this time includes null in length field)	V4 on

Version 4 Marker Types

Type	Description
"apnd"	Append
"defl"	Default
"wfon"	Waveform Onset
"wfof"	Waveform End
"nois"	Change in Signal Quality
"rhyt"	Change in Rhythm
"recv"	Recovery
"max "	Maximum
"min "	Minimum
"rset"	Reset
"cmlb"	Communication Lost Begin
"cmle"	Communication Lost End

Type	Description
"ansh"	Short Arrow
"anmd"	Medium Arrow
"anlg"	Long Arrow
"flag"	Flag
"star"	Star
"usr1"	User Type 1
"usr2"	User Type 2
"usr3"	User Type 3
"usr4"	User Type 4
"usr5"	User Type 5
"usr6"	User Type 6
"usr7"	User Type 7
"usr8"	User Type 8
"usr9"	User Type 9
"qrsb"	QRS Onset
"qrs "	QRS Peak
"qrse"	QRS End
"tbeg"	T-wave Onset
"t "	T-wave Peak
"tend"	T-wave End
"pbeg"	P-wave Onset
"p "	P-wave Peak
"pend"	P-wave End
"q "	Q-wave Peak
"s "	S-wave Peak
"u "	U-wave Peak
"pq "	PQ Junction
"jpt "	J-point
"stch"	ST Segment Change
"tch "	T-wave Change
"nrml"	Normal Beat
"pace"	Paced Beat
"pfus"	Fusion of Paced and Normal Beat
"lbbb"	Left Bundle Branch Block Beat
"rbbb"	Right Bundle Branch Block Beat
"bbb "	Bundle Branch Block Beat
"apc "	Atrial Premature Beat
"aber"	Aberrated Atrial Prematuire Beat
"npc "	Nodal Premature Beat

Type	Description
"svpb"	Supraventricular Premature Beat
"pvc "	Premature Ventricular Contraction
"ront"	R-on-T Premature Ventricular Contraction
"fusi"	Fusion of Ventricular and Normal Beat
"aesc"	Atrial Escape Beat
"nesc"	Nodal Escape Beat
"sves"	Supraventricular Escape Beat
"vesc"	Ventricular Escape Beat
"syst"	Systole
"dias"	Diastole
"edp "	End Diastolic Pressure
"aptz"	A-point
"bptz"	B-point
"cptz"	C-point
"xptz"	X-point
"yptz"	Y-point
"optz"	O-point
"plat"	Plateau
"upst"	Upstroke
"vfon"	Start of Ventricular Flutter
"flwa"	Ventricular Flutter Wave
"vfof"	End of Ventricular Flutter
"pesp"	Pacemaker Artifact
"arfc"	Isolated QRS-like Artifact
"napc"	Non-conducted P-wave
"base"	Baseline
"dose"	Dose
"wash"	Wash
"apon"	Spike Episode Begin
"apof"	Spike Episode End
"rein"	Inspire Start
"reot"	Expire Start
"reap"	Apnea Start
"stim"	Stimulus Delivery
"resp"	Response
"scr "	Skin Conductance Response
"sscr"	Specific SCR
"ctr1"	Cluster 1
"ctr2"	Cluster 2

Type	Description
"ctr3"	Cluster 3
"ctr4"	Cluster 4
"ctr5"	Cluster 5
"ctr6"	Cluster 6
"ctr7"	Cluster 7
"ctr8"	Cluster 8
"ctr9"	Cluster 9
"ctrn"	Cluster n
"cend"	End Cluster
"outl"	Outlier
"tran"	Training Set
"cut "	Cut
"vb "	Paste Begin
"ve "	Paste End
"selb"	Selection Begin
"sele"	Selection End
"steb"	Start of Eye Blink Artifact
"eneb"	End of Eye Blink Artifact
"sexc"	Start of Excursion Artifact
"eexc"	End of Excursion Artifact
"ssat"	Start of Saturation Artifact
"esat"	End of Saturation Artifact
"sspk"	Start of Spike Artifact
"espk"	End of Spike Artifact
"semg"	Start of EMG Artifact
"eemg"	End of EMG Artifact
"wles"	Workload - EMG Start
"wlee"	Workload - EMG End
"ipss"	Workload - Invalid PSD Start
"ipse"	Workload - Invalid PSD End
"ddst"	Dummy Data Start
"dded"	Dummy Data End
"idst"	Misaligned Data
"bprs"	Button Pressed
"leho"	Left Eye Hit Object
"reho"	Right Eye Hit Object
"smis"	SMI Stimulus Image Has Been Presented to the Subject
"mors"	Start Out of Range
"more"	End Out of Range;

MARKER HEADER 2 (SEQUENCE 9)

V3 files from version 3.81 (41) to version 3.9 (45) have extra marker information in a separate section immediately following the primary marker sections. It consists of a marker header followed by extra marker items.

The extra header is:

Field	Description/ Purpose	Data Type	Version Notes
headerId	4 byte block id	Int32	V3.81 to V3.9
itemCount	Count of extra marker item info. Always seem to match number of primary markers	Int32	V3.81 to V3.9
unknown	Unknown headerdata	76 bytes	V3.81 to V3.9

EXTRA MARKER ITEM CONFIGURATION (SEQUENCE 10)

V3 files from version 3.81 (41) to version 3.9 (45) have extra marker information in a separate section immediately following the primary marker sections. It consists of a marker header followed by extra marker items.

The extra items structure is:

Field	Description/ Purpose	Data Type	Version Notes
unknown	Unknown purpose	Int32	V3.81 to V3.9
markerNumber	A sequential number that starts at 1 for the first marker and increments for each marker item. I think this I used to link back to the primary marker configuration.	Int32	V3.81 to V3.9
unknown	Unknown purpose	3 x Int32	V3.81 to V3.9
colour	RGBA colour of marker	Int32	V3.81 to V3.9
markerTag	Id of marker Type Id tag	Int16	V3.81 to V3.9
markerTypeId	Index Number of the type of marker	Int16	V3.81 to V3.9

JOURNAL (SEQUENCE 11)

Version 3 Journal

Version 3 journals are just a simple structure of:

Field	Description/ Purpose	Data Type	Version Notes
headerId	Id of journal header. It always seems to be 0x44332211	Int32	V3.0 to V3.9
showJournal	Boolean seems to be repeat of showJournal flag in main file graph header.	Int16	V3.0 to V3.9
textLength	Length of journaltext including any null terminator	Int32	V3.0 to V3.9
text	Journal Text	textLength byte string	V3.0 to V3.9

Version 4 Early Journal

Version 4 journals are much more complicated. Journals are stored as HTML and can contain additional objects such as images and embedded pdf documents. I have put information about embedded PDFs in a separate section because they appear much later in the file structure.

There are also differences in structure between early (up to 4.11) and later (4.2 onwards).

Field	Description/ Purpose	Data Type	Version Notes
journalLength	Length of all journal information including header	Int32	V4.0 to 4.11
journalCount	This could be a flag or a count. If 0x00 the journal is not present. Do not read beyond this field.	Int16	V4.0 to 4.11
journalUILength	Length of journal UI configuration block	Int32	V4.0 to 4.11

Field	Description/ Purpose	Data Type	Version Notes
	including this length indicator		
journalUIBytes	Block of data with the UI configuration of journal part of program	Byte[jornalUILength] -4	V4.0 to 4.11
journalText	HTML structured Journal text. Occupies the remainder of the journalLength.	Calculated string length. See note below.	V4.0 to 4.11

The early header does not have the actual length of HTML text. You need to calculate text length.

textLength = journalLength – 6 – journalUILength.

Version 4 Late Journal

From version 4.2 onwards the structure changes again. The journal is stored as HTML and supported embedding images. Images are referenced within the journal HTML by a unique ID. Example:

```
<img src="{ee768e11-0848-4835-b6a7-afaf8db2a1f3}" />
```

An int32 immediately after the journal text gives a count of the embedded objects. Each image is stored as a binary object with identifying information. So far I have not found anything that indicates the number of bytes within the object.

Field	Description/ Purpose	Data Type	Version Notes
journalLength	Length of all journal information including header	Int32	V4.2 on
journalCount	This could be a flag or a count. If 0x00 the journal is not present. Do not read beyond this field.	Int16	V4.2 on
journalUILength	Length of journal UI configuration block excluding this length indicator	Int32	V4.2 on
journalUIBytes	Block of data with the UI configuration of journal part of program	Byte[journalUILength]	V4.2 on

Field	Description/ Purpose	Data Type	Version Notes
journalTextCount	This could be a flag or a count. If 0x0000 the journal text is not present. Do not read beyond this field.	Int32	V4.2 on
journalTextLength	Length of journal HTML text excluding null terminator	Int32	V4.2 on
journalTextLength0	Length of journal HTML text INCLUDING null terminator. Use this to get correct length	Int32	V4.2 on
journalText	HTML structured Journal text.	Text of length journalTextLength0 .	V4.2 on
objectCount	Number of objects embedded into journal	Int32	V4.2 on

The early header does not have the actual length of HTML text. You need to calculate text length.

textLength = journalLength – 6 – journalUILength.

JOURNAL OBJECTS (SEQUENCE 12)

From V4.0 onwards the journal is stored as HTML and supported embedding images. Images are referenced within the journal HTML by a unique ID. Example:

An int32 immediately after the journal text gives a count of the embedded objects. Each image is stored as a binary object with identifying information. So far I have not found anything that indicates the number of bytes within the object.

Field	Description/ Purpose	Data Type	Version Notes
objectType	Type of embedded object. Image type = 1. Other types unknown.	Int32	4.00 on
id	Hexadecimal id.	Byte[16]	4.00 on
object	For images: PNG binary. Refer to PNG file structure for a breakdown of the bytes. Note: there is no marker to tell you how many bytes in each image.	Byte[]	4.00 on

Field	Description/ Purpose	Data Type	Version Notes
	You need to read the PNG structure to work out size.		

UNKNOWN HEADER (SEQUENCE 13)

This is an extra 8 bytes that appeared in version 3.8 and 3.9 files just after the Journal. Structure is:

Field	Description/ Purpose	Data Type	Version Notes
headerLength	Length of header including this field. (seem to be 0x00000008)	Int32	V3.8 to V3.9
itemCount	Count of objects in this section. (seems to be 0x00000000)	Int32]	V3.8 to V3.9

SNAPSHOTS (SEQUENCE 14)

Snapshots are sets of compressed data. Snapshots are used for two purposed.

Firstly, from version 3.8 compression is supported. if the file is compressed, there is no interleaved data. The primary data is stored as a snapshot.

Second, the concept of taking point in time snapshots of data is also supported.

Both reasons are stored in the same manner. The first snapshot is always the compressed primary dataset representing the current data if the file is flagged as compressed.

Snapshots have header information that differs between version 3 and version 4 files.

The snapshot area also has a master header telling you how many snapshots there are.

Snapshot Master Header

One per file.

Field	Description/ Purpose	Data Type	Version Notes
snapshotsLength	Length of all snapshots including this field.	Int32	V3.8 on
snapshotsSetId	Id of this header 0xDEADED3D	Int32	V3.8 on
snapshotsCount	Count of number of snapshot sets	Int16	V3.8 on

Snapshot Header

One per snapshot.

Field	Description/ Purpose	Data Type	Version Notes
snapHeaderLength	length (including this int) of snap header.	Int32	V3.8 on
snapHeadertId	Id of this header	Int32	V3.8 on
snapshotsCount	Count of number of snapshot sets	Int16	V3.8 on
unknown	Unknown purpose	Int32	V3.8 on
channelCount	Number of channel datasets in this snapshot	Int16	V3.8 on
timestampLength	Length of timestamp text	Int32	V3.8 on
descriptionLength	Length of description text	Int32	V4.0 on
markerName	4 byte name of a marker	Text 4 bytes long	V4.0 on
unknown	Unknown info	Byte[16]	V4.0 on
unknown	Unknown shorts	3 x Int16	V4.2 on
timestampText	timestamp	Text of length timestampLength	V3.8 on
descriptionText	description	Text of length descriptionLength	V4.0 on
Data Datasets	One data dataset per channel	Dataset, one per channel	V3.8 on
GraphInfo	* Set of configurations for snapshot	5 x GraphInfo	V4.0 on
GraphInfo	* extra configuration for snapshot	1 x GraphInfo	V4.3 on
GraphInfo	* extra configuration for snapshot	1 x GraphInfo	V4.4 on

*Note: From version 4 the snapshots have additional compressed dataset items which appear to be graph info snapshots to go with the data snapshot set.

Ver 4 files have 5 graphInfo snapshots

Ver 4.3 has an additional graphInfo snapshot (for a total of 6)

Ver 4.4 has an additional graphInfo snapshot (for a total of 7)

Each configuration snapshot has compressed version of different information items such as text annotations etc.

Dataset Compression Header

One per dataset (channel).

Field	Description/ Purpose	Data Type	Version Notes
compressionHeaderLength	length (including this int) of header.	Int32	V3.8 on
compHeadertId	Id of this header	Int32	V3.8 on
unknown	Unknown short (ver 3 files only)	Int16	V3.8 to Ver 3.9
channelNumber	Channel number	Int16	V3.8 on
unknown	Header info still to be analysed	32 x bytes in version3, 34 x bytes in version 4	V3.8 on
channelLabelLength	Length of channel label text	Int32	V3.8 on
unitLabelLength	Length of unit label text	Int32	V3.8 on
uncompressedLength	Length of uncompressed data	Int32	V3.8 on
compressedLength	Length of compressed data	Int32	V3.8 on
ChannelLabelText	ChannelLabel	text	V3.8 on
UnitlabelnText	unitLabel	text	V3.8 on
Data	Compressed zlib structure.	Byte[compressedLength]	V3.8 on

Compressed Graph Information Header

One per configuration Up to 7per snapshot.

Field	Description/ Purpose	Data Type	Version Notes
compressionHeaderLength	length (including this int) of header.	Int32	V3.8 on
compHeadertId	Id of this header	Int32	V3.8 on
uncompressedLength	Length of uncompressed data	Int32	V3.8 on
compressedLength	Length of compressed data	Int32	V3.8 on
Data	Compressed zlib structure.	Byte[compressedLength]	V3.8 on

MEASUREMENT (SEQUENCE 15)

Defines the setup of the graph measurement UI display stored as XML. Version 4 onwards.

Field	Description/ Purpose	Data Type	Version Notes
textLength	length of text.	Int32	V4.0 on
headerId	Id of this header	Int32	V4.0 on
textLength2	length of text. Seems to be second copy of length	Int32	V4.0 on
text	XML text	Byte[textLengh]	V4.0 on

AUTOMARKER HEADER (SEQUENCE 16)

From version 4 you can set up hotkey and automated markers. This appears to be the header for auto-marker information.

Field	Description/ Purpose	Data Type	Version Notes
markersLength	length of all markers including this header	Int32	V4.0 on
headerId	Id of this header	Int32	V4.0 on
markersCount	AutoMarkers	Int32	V4.0 on
Automarkers	Automarkers.See structure below	Automarkers x markersCount	V4.0 on

AUTOMARKER (SEQUENCE 17)

From version 4 you can set up hotkey and automated markers. This appears to be the structure of auto-marker information.

Field	Description/ Purpose	Data Type	Version Notes
markerId	Id of marker or type of marker	Int16	V4.0 on

Field	Description/ Purpose	Data Type	Version Notes
markerTypeText	Type of marker	4 byte string	V4.0 on
Automarkers	Automarkers.See structure below	Automarkers x markersCount	V4.0 on
markerChannel	This s a guess think it's the channel that the marker should be associated with . Default is -1	Int16	V4.0 on
markerData	Data for marker. Yet to be decoded	Byte[]. For Ver 4.0 – 4.1 260 bytes For Ver 4.2 270 bytes For Ver 4.3 on 528 bytes	V4.0 on

UNKNOWN BLOCKS (SEQUENCE 18-22)

There are 5 small unknown blocks that most likely are headers for information not yet worked out. See section on structure of unknown blocks.These are:

Sequence 18 – 12 bytes. Header Id 0xDDBB55FF. Ver 4.2 onwards

Sequence 19 – 8 bytes. Header Id 0x73F519BD. Ver 4.2 onwards.

Sequence 20 – 8 bytes. Header Id 0x036E94A3. Ver 4.3 onwards.

Sequence 21 – 12 bytes. Header Id 0x0598916D. Ver 4.3 onwards.

Sequence 22 – 4 bytes. Header Id 0x00000000. Ver 4.3 onwards.

PDF HEADER (SEQUENCE 23)

From version 4.4 you can attach pdf files to be viewed alongside the journal. This section contains a header for embedded pdf files. One per file.

Field	Description/ Purpose	Data Type	Version Notes
pdfLength	Total size of all pdfs including header	Int32	V4.4 on
headerId	Id for this header	Int32	V4.4 on
pdfCount	Number of pdf objects	Int32	V4.4 on
pdfs	Pdf objects	See below	V4.4 on

PDF OBJECTS (SEQUENCE 24)

Each PDF has a header followed by the binary of the pdf. The binary is exactly what you would see in a .pdf file.

Field	Description/ Purpose	Data Type	Version Notes
pdfHeader	Header information about the pdf. Yet to be analysed.	Byte[264]	V4.4 on
pdfLength	Length in bytes of the pdf binary	Int32	V4.4 on
pdf	PDF binary	Byte[pdfLength]	V4.4 on

UNKNOWN BLOCK (SEQUENCE 25)

There is small unknown blocks that most likely are headers for information not yet worked out. See section on structure of unknown blocks. These are:

Sequence 25 – 12 bytes. Header Id 0x08911DDB. Ver 4.4 onwards

KEY VALUES (SEQUENCE 26)

Stores key values as XML. Version 4 onwards.

Field	Description/ Purpose	Data Type	Version Notes
textLength	length of text.	Int32	V4.0 on
headerId	Id of this header	Int32	V4.0 on
textLength2	length of text. Seems to be second copy of length	Int32	V4.0 on
text	XML text	Byte[textLengh]	V4.0 on

TEXT ANNOTATIONS (SEQUENCE 27)

Stores graph text annotations as XML. Version 4 onwards.

Field	Description/ Purpose	Data Type	Version Notes
textLength	length of text.	Int32	V4.0 on
headerId	Id of this header	Int32	V4.0 on
textLength2	length of text. Seems to be second copy of length	Int32	V4.0 on
text	XML text	Byte[textLengh]	V4.0 on

UNKNOWN BLOCK (SEQUENCE 28)

There is small unknown blocks that most likely are headers for information not yet worked out. See section on structure of unknown blocks. These are:

Sequence 28 – 8 bytes. Header Id 0x50726F70. Ver 4.11 onwards.

MEDIA (SEQUENCE 29)

From 4.11 you can attach and sequence media such as video. Version 4.11 onwards. This XML header contains configuration for any media associated with the graph file. The XML appears to show video as external links, not embedded binary. THEORY.

Field	Description/ Purpose	Data Type	Version Notes
textLength	length of text.	Int32	V4.11 on
headerId	Id of this header	Int32	V4.11 on
textLength2	length of text. Seems to be second copy of length	Int32	V4.11 on
text	XML text	Byte[textLengh]	V4.11 on

GRAPH HISTORY (SEQUENCE 30)

From 4.11 you can store details of the history of major transformations to graph data using filters etc. This XML information shows historically what filters were applied. It does not track the resulting data change. That is the job for snapshots.

Field	Description/ Purpose	Data Type	Version Notes
textLength	length of text.	Int32	V4.11 on
headerId	Id of this header	Int32	V4.11 on
textLength2	length of text. Seems to be second copy of length	Int32	V4.11 on
text	XML text	Byte[textLengh]	V4.11 on

UNKNOWN BLOCK (SEQUENCE 31)

There is small unknown blocks that most likely are headers for information not yet worked out. See section on structure of unknown blocks. These are:

Sequence 31 – 4 bytes. Header Id 0x000000. Ver 4.2 onwards.

UNKNOWN BLOCK (SEQUENCE 32)

There is small unknown blocks that most likely are headers for information not yet worked out. See section on structure of unknown blocks. These are:

Sequence 32 – 8 bytes. Header Id 0x 53434F20. Ver 4.2 onwards.

FOCUS AREA (SEQUENCE 33)

From 4.3 you can store details of focus areas marking specific areas to graph data. This XML information shows the configuration of focus areas.

Field	Description/ Purpose	Data Type	Version Notes
textLength	length of text.	Int32	V4.11 on
headerId	Id of this header	Int32	V4.11 on
textLength2	length of text. Seems to be second copy of length	Int32	V4.11 on
text	XML text	Byte[textLengh]	V4.11 on

TIMERS (SEQUENCE 34)

From 4.3 you can associate timers to graph data. This XML information shows the configuration of timers.

Field	Description/ Purpose	Data Type	Version Notes
textLength	length of text.	Int32	V4.11 on
headerId	Id of this header	Int32	V4.11 on
textLength2	length of text. Seems to be second copy of length	Int32	V4.11 on
text	XML text	Byte[textLengh]	V4.11 on

OUTPUT CONTROL PRESETS (SEQUENCE 35)

From 4.3 you can setup output control presets. This XML information shows the configuration of presets.

Field	Description/ Purpose	Data Type	Version Notes
textLength	length of text.	Int32	V4.11 on
headerId	Id of this header	Int32	V4.11 on
textLength2	length of text. Seems to be second copy of length	Int32	V4.11 on
text	XML text	Byte[textLengh]	V4.11 on

FINAL BLOCK FOREIGN DATA

Purpose

Large data block with general configuration information. Version 4 onwards. NOTE: it appears that because of the extra data requirements of foreign data in later versions, the data has been moved to the end of the file. THEORY ONLY.

Structure

Field	Description/ Purpose	Data Type	Version Notes
blockLength	Block length in bytes including this field.	Int32	4.00 on
data	Size = blockLength - 4	Byte[]	4.00 on