BioHarness™ 3 Logging System Interface

Document History

Version	Date	Description
0.0.0.1	2011-12-14	First Draft
1.0.0.0	2011-12-16	Corrections after review

References

Ref #	ID	Description
[1]	9700.0110	BioHarness Bluetooth General Comms Link Specification
[2]		IBM/MS RIFF/MCI Specification V3.0 (April 15, 1994)
[3]	9700.0157	Event Messaging System

Document Notes

All numbers in this document are written in decimal, except hexadecimal numbers which are prefixed by ‘0x’. For example 5436 is decimal, while 0x5436 is hexadecimal.

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

Contents

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

1 Introduction

This document describes the BioHarness 3 Logging System Interface, including the ability to retrieve data from the log file and how the logged data should be interpreted.

2 Overview

The Logging System implemented in the BioHarness 3 allows storing of two separate log files:

- Log File 0 – Logging of periodic data such as General Data, ECG, Accelerometer, etc.
- Log File 1 – Logging of events such as button press, jump detect, etc.

Data can be read from the BioHarness only through the USB interface when it is inserted into a cradle and connected to a PC. Reading of the log files should not be attempted via a Bluetooth connection because logging is usually taking place at the time and the log cannot be read until the log session is closed, which happens when the BioHarness is switched off or placed into a cradle.

The commands required to Read Data from logging memory are detailed in the BioHarness™ Bluetooth Comms Link Specification [1] document.

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

3 Data Format

Both Data and Event log files use the standard RIFF file format [2] for storing information. This groups the data contents into separate chunks, each containing its own header and data bytes. The chunk header specifies the type and size of the chunk data bytes. This organization method allows programs that do not use or recognize particular types of chunks to easily skip over them and continue processing following known chunks.

3.1 Chunk structure

Chunks are, in general, arranged as shown in Table 3-1 General Chunk Structure.

Part	Description
ckID	A four-character code that identifies the representation of the chunk data. A program reading a RIFF file can skip over any chunk whose chunk ID it doesn't recognize; it simply skips the number of bytes specified by ckSize plus the pad byte, if present.
ckSize	A 32-bit unsigned value identifying the size of ckData. This size value does not include the size of the ckID or ckSize fields or the pad byte at the end of ckData.
ckData	Binary data of fixed or variable size. The start of ckData is word-aligned with respect to the start of the RIFF file. If the chunk size is an odd number of bytes, a pad byte with value zero is written after ckData. Word aligning improves access speed (for chunks resident in memory) and maintains compatibility with EA IFF. The ckSize value does not include the pad byte.

Table 3-1 General Chunk Structure

There are 2 special types of chunk which may contain sub-chunks, these are the RIFF chunk, which encapsulates the whole file, and a LIST chunk. (Note, JUNK, RIFF and LIST are the only ckIDs which are permitted to contain upper case letters.)
These chunks still comply with the above structure; however ckData has a specific format. The format of the ckData segment of the RIFF and LIST chunks is illustrated in Table 3-2 LIST or RIFF chunk ckData format.

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

Part	Description
Form Type	A four-character code value identifying the data representation within this RIFF or LIST chunk. The number of sub-chunks, mandatory and /or optional types of sub-chunk or order of sub-chunks may be defined for a particular form type.
Sub-Chunk	A sub-chunk, which may be a standard chunk or a LIST chunk
Sub-Chunk	A sub-chunk, which may be a standard chunk or a LIST chunk
…	…

Table 3-2 LIST or RIFF chunk ckData format

An example of the type of structure that is possible using this technique is illustrated in Figure 3-1 Example Structure.

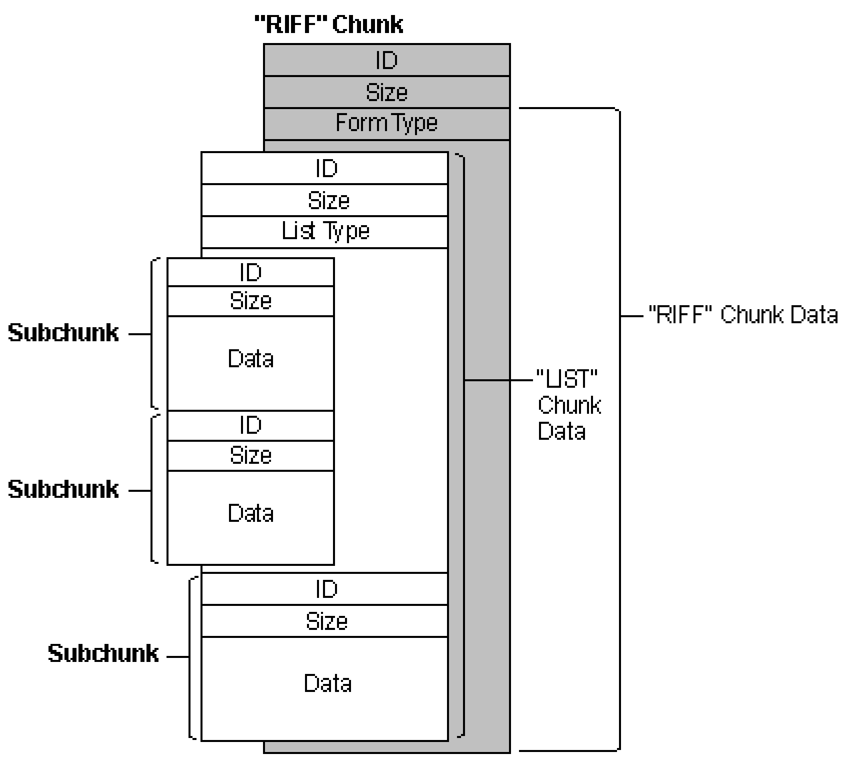


Figure 3-1 Example Structure

4 Data Log File Structure

This chapter describes the file structure of the Data Log, which is accessed by reading Log File 0.

4.1 RIFF File Identifier

The RIFF Chunk Descriptor is the first information resident in the RIFF file, firstly specifying the file is of "RIFF" type and secondly the format of the RIFF file. This enables the PC parsing to process the file based on known sub-chunks.
In the current implementation, there is only one "RIFF" chunk.

Field Value	Size (bytes)	Description
"RIFF"	4	Chunk ID
<file size>	4	Size of chunk (bytes)
"BIO "	4	Format

Table 4-1 RIFF Chunk Descriptor

For the BioHarness the RIFF file format is always "BIO ".

4.2 Zephyr File Identifier

The Zephyr File Identifier sub-chunk is the first sub-chunk chunk in the "RIFF" chunk and contains information relating to the formatting of the file data.

In the current implementation there is only one Zephyr File Identifier chunk in the RIFF file.

Field Value	Size (bytes)	Description
"zphr"	4	Sub-chunk ID
<chunk size>	4	Size of chunk (bytes)
"bhns"	4	File Type Identifier
<version>	4	File Version

Table 4-2 Zephyr File Identifier Sub-Chunk

The File Type Identifier and Version enables the application to determine what type of data is stored within the file. For a standard BioHarness Data Log the File Type Identifier is always set to "bhns" and the File Version identifies one of the logging formats defined in Appendix B – BioHarness™ Log Record Formats.

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

4.3 Log Header

The Log Header sub-chunk is written for each new log session within the log file and contains information relating to the time of the session, the logging period (the time between each log record stored in the file) and the size of the data for each log record (in number of words).

Field Value	Size (bytes)	Description
“logh”	4	Sub-chunk ID
<chunk size>	4	Size of chunk (bytes)
<timestamp>	8	Log session start time
<logging period>	4	Period in milliseconds
<record size>	2	Number of 16-bit words per log record
<”logr” pad size>	2	Padding bytes in ”logr” sub-chunk
<spare>	488	Unused data

Table 4-3 Data Log Header Sub-Chunk

The application uses this information to parse the logging data chunks which follow. Note that the logging data sub-chunk specifies record size as “words” which are essentially 16-bit data items. Therefore the size of each log record in bytes is equal to the number of words multiplied by 2.

<table>
<tr><th colspan="3">Time Stamp</th></tr>
<tr><th>Description</th><th colspan="2">Length</th></tr>
<tr><td>Year (low byte)</td><td colspan="2">1</td></tr>
<tr><td>Year (high byte)</td><td colspan="2">1</td></tr>
<tr><td>Month</td><td colspan="2">1</td></tr>
<tr><td>Day of Month</td><td colspan="2">1</td></tr>
<tr><td rowspan="4">Milliseconds since midnight</td><td rowspan="4">4</td><td>LS byte</td></tr>
<tr><td></td></tr>
<tr><td></td></tr>
<tr><td>MS byte</td></tr>
<tr><td>Total</td><td colspan="2">8 bytes</td></tr>
</table>

Table 4-4 Time Stamp Packing Format

The time stamp identifies the start time of the log to the nearest millisecond.

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

4.4 Split Log Header 1 & Split Log Header 2

The Split Log Header 1 and Split Log Header 2 sub-chunks are in exactly the same format as the Log Header sub-chunk, but denotes a partitioned session.

Field Value	Size (bytes)	Description
"log1"	4	Sub-chunk ID
<chunk size>	4	Size of chunk (bytes)
<timestamp>	8	Log session start time
<logging period>	4	Period in milliseconds
<record size>	2	Number of 16-bit words per log record
<logr pad size>	2	Padding bytes in "logr" sub-chunk
<spare>	488	Unused data

Table 4-5 Split Log Header 1 Sub-Chunk

Field Value	Size (bytes)	Description
"log2"	4	Sub-chunk ID
<chunk size>	4	Size of chunk (bytes)
<timestamp>	8	Log session start time
<logging period>	4	Period in milliseconds
<record size>	2	Number of 16-bit words per log record
<logr pad size>	2	Padding bytes in "logr" sub-chunk
<spare>	488	Unused data

Table 4-6 Split Log Header 2 Sub-Chunk

If a session recording reaches the end of the RIFF file, the session is finalized, but written as a Split Log Header 1 sub-chunk instead of the usual Log Header sub-chunk to indicate that this is the first of two session partitions.

The second partition is written at the start of the RIFF file as a Split Log Header 2 sub-chunk, but the PC logging parser should process the Split Log Header 1 and Split Log Header 2 chunks as one session of data.

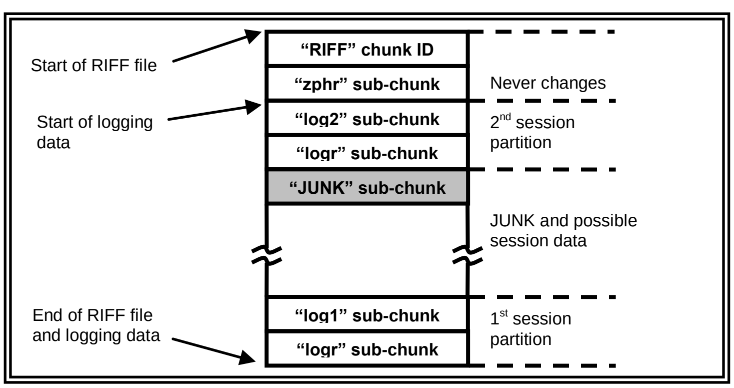


Figure 4-1 Session partitions

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

4.5 Log Raw Data

The Log Raw Data sub-chunk is written to memory for each new logging session within the logging file and contains the log data for a single session.

Field Value	Size (bytes)	Description
"logr"	4	Sub-chunk ID
<chunk size>	4	Size of chunk (bytes)
<spare padding>	(see log header for padding size)	Padding (spare) bytes
<logging data>	<chunksize – padding size>	Raw logging session data

Table 4-7 Log Raw Data Sub-Chunk

The Log Raw Data sub-chunk contains data for one logging session.
The number of records, and hence the length of the recording can be calculated based on the number of words indicated in the associated log header chunk and the size of the Raw Data sub-chunk.

The arrangement of data in each log record is detailed in section Appendix B – BioHarness™ Log Record Formats. This includes all the different log formats supported by BioHarness as identified by the Project Version in the Zephyr File Identifier sub-chunk.

4.6 "JUNK" Sub-chunk

The "JUNK" sub-chunk is used to specify areas of memory that the PC can ignore e.g. padding bytes. These are used for 2 main reasons:

- Padding bytes due to the minimum "page" write size associated with the storage media.
  For example, if a session is completed halfway through a page, because a page is the minimum area that can be written, JUNK must be written for the remainder of the page so the PC does not process the remainder of the data.
- Because the size of the RIFF file on the memory device is typically of the order of several hundreds of Mega-Bytes, when a session has been completed, a JUNK sub-chunk can be written to the end of the file (or until the oldest recorded session data) so the PC doesn't need to process any more data (it ignores JUNK data sub-chunks).

Field Value	Size (bytes)	Description
"JUNK"	4	Sub-chunk ID
<chunk size>	4	Size of chunk (bytes)
<unused data>	<chunk size>	Junk data (ignored)

Table 4-8 "JUNK" Sub-Chunk

Essentially, the PC parsing of the RIFF data will find a JUNK sub-chunk and jump to the end of the chunk to process the next chunk.

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

4.7 Example Data Log File

This section provides examples to indicate the storage of data within the RIFF file and is intended to aid understanding and design of a RIFF file parser for the logging data.

Offset (bytes)	File Entry	Description
0x000000	"RIFF"	RIFF chunk descriptor
0x000004	0x0E09FFF8	File size (minus first 8 bytes).
0x000008	"BIO "	RIFF file format
0x00000C	"zphr"	Sub-chunk ID
0x000010	0x00000008	Size of sub-chunk
0x000014	"bhns"	Project identifier
0x000018	"0003"	Project version
0x00001C	"JUNK"	JUNK sub-chunk
0x000020	0x0001FFDC	JUNK chunk size (up to 0x00020000)
0x020000	"logh"	Sub-chunk ID
0x020004	0x000001F8	Size of sub-chunk
0x020008	0x07DA, 0x01, 0x04, 0x00CE32BF	Timestamp (2010, Jan, 4th, 3:45:13)
0x020010	0x000003E8	Logging Period (1000 milliseconds)
0x020014	0x0040	Number of words per record (64)
0x020016	0x01F8	"logr" padding size (504 bytes).
0x020018	<spare data>	488 spare bytes up to 0x20200
0x020200	"JUNK"	JUNK sub-chunk
0x020204	0x0001FBF8	JUNK chunk size (up to 0x0003FE00)
0x03FE00	"logr"	Sub-chunk ID
0x03FE04	0x000003F8	Size of sub-chunk
0x040000	<raw logging data>	Note – starts after "logr" padding.
0x044B00	"JUNK"	JUNK sub-chunk after session stopped.
0x044B04	0x0001B4F8	JUNK chunk size (up to 0x00060000)
0x060000	"JUNK"	JUNK sub-chunk after session stopped.
0x060004	0x0E03FFF8	JUNK chunk size (up to 0x0E0A0000)

Table 4-9 Example RIFF file (one session recording).

The table above shows a single session recording started at 3:45:13 on the 4th of January 2010. The session is short, but is for illustration purposes only. Note that the final JUNK chunk covers the remainder of the file. As the logging data has yet to wrap around to the beginning again, the JUNK chunk is written to cover until the end of the file. To be precise, the final JUNK chunk is written after the newest session and covers the file up until the beginning of the oldest session. Each sub-chunk is outlined to clearly identify the data within.

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

5 Event Log File Structure

This chapter describes the file structure of the Event Log, which is accessed by reading Log File 1.

5.1 RIFF File Identifier

The RIFF Chunk Descriptor is the first information resident in the RIFF file, firstly specifying the file is of “RIFF” type and secondly the format of the RIFF file. This enables the PC parsing to process the file based on known sub-chunks.
In the current implementation, there is only one “RIFF” chunk.

Field Value	Size (bytes)	Description
“RIFF”	4	Chunk ID
<file size>	4	Size of chunk (bytes)
“BIO ”	4	Format

Table 5-1 RIFF Chunk Descriptor

For the BioHarness the RIFF file format is always “BIO “.

5.2 Zephyr File Identifier

The Zephyr File Identifier sub-chunk is the first sub-chunk chunk in the “RIFF” chunk and contains information relating to the formatting of the file data.
In the current implementation there is only one Zephyr File Identifier chunk.

Field Value	Size (bytes)	Description
“zphr”	4	Sub-chunk ID
<chunk size>	4	Size of chunk (bytes)
“bhel”	4	File Type Identifier
“0001”	4	File Version

Table 5-2 Zephyr File Identifier Sub-Chunk

The project identifier and version enables the PC application to process the RIFF data within the log.
The PC application may process log data from different projects and versions and therefore this information is used to parse the data correctly.

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

5.3 Log Header

The Log Header sub-chunk is written for each new logging session within the logging file and contains information relating to the time of the session and the number of bytes written to the log.

Field Value	Size (bytes)	Description
“logh”	4	Sub-chunk ID
<chunk size>	4	Size of chunk (bytes)
<timestamp>	8	Log session start time
<unused>	4	Not used in event log
<packet size>	2	Number of bytes per event data packet
<”logr” pad size>	2	Padding bytes in ”logr” sub-chunk
<spare>	488	Unused data

Table 5-3 Event Log Header Sub-Chunk

The PC application uses this information to parse the logging data chunks which follows.

Time Stamp		
Description	**Length**	
Year (low byte)	1	
Year (high byte)	1	
Month	1	
Day of Month	1	
Milliseconds since midnight	4	LS byte
		MS byte
Total	8 bytes	

Table 5-4 Time Stamp Packing Format

The time stamp identifies the start time of the log to the nearest millisecond.

5.4 Split Log Header 1 & Split Log Header 2

The Split Log Header 1 and Split Log Header 2 sub-chunks are in exactly the same format as the Log Header sub-chunk, but denotes a partitioned session.

Field Value	Size (bytes)	Description
“log1”	4	Sub-chunk ID
<chunk size>	4	Size of chunk (bytes)
<timestamp>	8	Log session start time
<logging period>	4	Period in milliseconds
<number of channels>	2	Number of bytes
<logr pad size>	2	Padding bytes in ”logr” sub-chunk
<spare>	488	Unused data

Table 5-5 Split Log Header 1 Sub-Chunk

Field Value	Size (bytes)	Description
“log2”	4	Sub-chunk ID
<chunk size>	4	Size of chunk (bytes)
<timestamp>	8	Log session start time
<logging period>	4	Period in milliseconds
<number of channels>	2	Number of bytes
<logr pad size>	2	Padding bytes in ”logr” sub-chunk
<spare>	488	Unused data

Table 5-6 Split Log Header 2 Sub-Chunk

If a session recording reaches the end of the RIFF file, the session is finalized, but written as a Split Log Header 1 sub-chunk instead of the usual Log Header sub-chunk to indicate that this is the first of two session partitions.
The second partition is written at the start of the RIFF file as a Split Log Header 2 sub-chunk, but the PC logging parser should process the Split Log Header 1 and Split Log Header 2 chunks as one session of data.

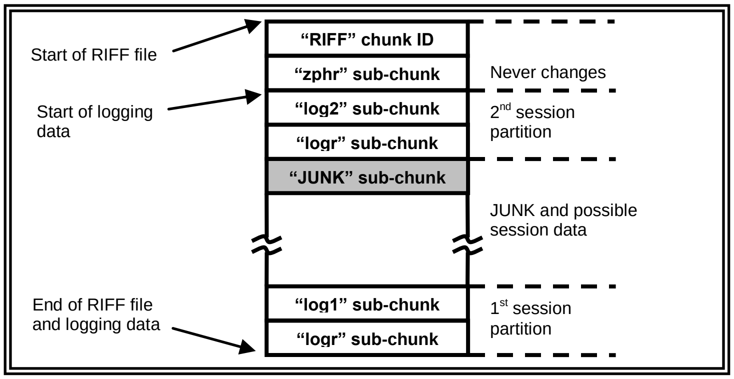


Figure 5-1 Session partitions

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

5.5 Log Raw Data

The Log Raw Data sub-chunk is written to memory for each new logging session within the logging file and contains the log data for a single session.

Field Value	Size (bytes)	Description
“logr”	4	Sub-chunk ID
<chunk size>	4	Size of chunk (bytes)
<spare padding>	(see log header for padding size)	Padding (spare) bytes
<logging data>	<chunksize – padding size>	Raw logging session data

Table 5-7 Log Raw Data Sub-Chunk

The Log Raw Data sub-chunk contains data for one logging session.
The number of event packets, and hence the length of the recording can be calculated based on the number of bytes per packet indicated in the associated log header chunk and the size of the Raw Data sub-chunk.

The arrangement of data in the event packets is detailed in Appendix C – BioHarness™ Event Data Packet Format.

5.6 “JUNK” Sub-chunk

The “JUNK” sub-chunk is used to specify areas of memory that the PC can ignore e.g. padding bytes. These are used for 2 main reasons:

- Padding bytes due to the minimum “page” write size associated the storage media.
  For example, if a session is completed halfway through a page, because a page is the minimum area that can be written, JUNK must be written for the remainder of the page so the PC does not process the remainder of the data.
- Because the size of the RIFF file on the memory device is typically of the order of several hundreds of Mega-Bytes, when a session has been completed, a JUNK sub-chunk can be written to the end of the file (or until the oldest recorded session data) so the PC doesn’t need to process any more data (it ignores JUNK data sub-chunks).

Field Value	Size (bytes)	Description
“JUNK”	4	Sub-chunk ID
<chunk size>	4	Size of chunk (bytes)
<unused data>	<chunk size>	Junk data (ignored)

Table 5-8 “JUNK” Sub-Chunk

Essentially, the PC parsing of the RIFF data will find a JUNK sub-chunk and jump to the end of the chunk to process the next chunk.

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

5.7 Example Event Log File

This section provides examples to indicate the storage of event data within the RIFF file and is intended to aid understanding and design of a RIFF file parser for the logging data.

Offset (bytes)	File Entry	Description
0x000000	"RIFF"	RIFF chunk descriptor
0x000004	0x018DFFF8	File size (minus first 8 bytes).
0x000008	"BIO "	RIFF file format
0x00000C	"zphr"	Sub-chunk ID
0x000010	0x00000008	Size of sub-chunk
0x000014	"bhel"	Project identifier
0x000018	"0001"	Project version
0x00001C	"JUNK"	JUNK sub-chunk
0x000020	0x0001FFDC	JUNK chunk size (up to 0x00020000)
0x020000	"logh"	Sub-chunk ID
0x020004	0x000001F8	Size of sub-chunk
0x020008	0x07DA, 0x01, 0x04, 0x00CE32BF	Timestamp (2010, Jan, 4th, 3:45:13)
0x020010	0x00000000	Unused
0x020014	0x0020	Number of bytes per event packet (32)
0x020016	0x01F8	"logr" padding size (504 bytes).
0x020018	<spare data>	488 spare bytes up to 0x020200
0x020200	"JUNK"	JUNK sub-chunk
0x020204	0x0001FBF8	JUNK chunk size (up to 0x0003FE00)
0x03FE00	"logr"	Sub-chunk ID
0x03FE04	0x00000238	Size of sub-chunk
0x040000	<raw logging data>	Note – starts after "logr" padding.
0x040040	"JUNK"	JUNK sub-chunk after session stopped.
0x044B04	0x0001FFB8	JUNK chunk size (up to 0x00060000)
0x060000	"JUNK"	JUNK sub-chunk after session stopped.
0x060004	0x0187FFF8	JUNK chunk size (up to 0x018E0000)

Table 5-9 Example RIFF file (one session recording).

The table above shows a single session recording started at 3:45:13 on the 4th of January 2010. The session is short, but is for illustration purposes only. Note that the final JUNK chunk covers the remainder of the file. As the logging data has yet to wrap around to the beginning again, the JUNK chunk is written to cover until the end of the file. To be precise, the final JUNK chunk is written after the newest session and covers the file up until the beginning of the oldest session. Each sub-chunk is outlined to clearly identify the data within.

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

6 Appendix A – PC Log Import Example Implementation

This section provides an example implementation of how to import logging data with a PC.

The log data can be processed by implementing the “Read Logging Data” and “Delete Log File” commands. This document does not detail the communication protocol used to access the log file, for a description of these commands see [1].

6.1 Verifying the RIFF Header

The following example C# code demonstrates how an application can read the RIFF file header to correctly identify a Zephyr BioHarness Log file:

```
private void LogDownloadFile( byte fileNum )
{
    /* Read first 12 bytes of the log header at Offset 0 */
    byte[] data  = deviceConnection.ReadLogData( fileNum, 0, 12 );

    /* Check if it's a valid Zephyr BioHarness RIFF file */
    if ( Encoding.ASCII.GetString(data, 0,  4).Equals("RIFF") &&
         Encoding.ASCII.GetString(data, 8,  4).Equals("BIO ") )
    {
        /* Get the file size (RIFF chunk size + 8) */
        UInt32 fileSize = BitConverter.ToUInt32(data, 4) + 8;

        /* We can now read the log sessions after the RIFF header */
        LogReadSessions(fileNum, 12, fileSize);
    }
}
```

In the example above we initially read the first 12 bytes of the RIFF file. We make sure that the file begins with a “RIFF” chunk. We also ensure that the file format at offset 8 is “BIO “, and calculate the file size based on the “RIFF” chunk.

6.2 Downloading the Log Sessions

Once we have correctly identified the log file we can proceed by parsing all the chunks in the log file (skipping any “JUNK” chunks or unrecognized chunks) and finally downloading the actual log session information. The example C# code below implements a simple loop which steps through all chunks in the file and processes the data within the chunks based on the Chunk ID. The actual session data within the “logr” chunk is downloaded in 128-byte blocks because that is the maximum that can be retrieved with a single “Read Log File” command.

To keep the example simple the Time Stamp and Period in the “logh” header are not stored and partitioned log1/log2 sessions are not joined. Also in most cases an application will not download the log data for all sessions as it is done in this example, instead it would store information about all sessions in a list and let the user choose which log session(s) should be downloaded.

```
private void LogReadSessions(byte fileNum, UInt32 offset, UInt32 fileSize)
{
    string fileFormat = null;
    string fileVersion = null;
    UInt16 padBytes = 0;
    UInt16 recordSize = 0;
    List<byte> logData = new List<byte>();      /* list to store logged data */

    /* Process all chunks until end of file is reached */
    while (offset < fileSize)
    {
        /* Read ID and size of next chunk */
        byte[] data = deviceConnection.ReadLogData(fileNum, offset, 8);
        offset += 8;    /* Set offset after chunk header */
        string chunkId = Encoding.ASCII.GetString(data, 0, 4);
        UInt32 chunkSize = BitConverter.ToUInt32(data, 4);

        /* Process chunks based on Chunk Id */
        switch (chunkId)
        {
            case "zphr":
                /* Determine the log file format and version */
                data = deviceConnection.ReadLogData(fileNum, offset, 8);
                fileFormat = Encoding.ASCII.GetString(data, 0, 4);
                fileVersion = Encoding.ASCII.GetString(data, 4, 4);
                break;

            case "logh":
            case "log1":
            case "log2":
                /* Read and store logh/log1/log2 header data */
                data = deviceConnection.ReadLogData(fileNum, offset, 24);
                recordSize = BitConverter.ToUInt16(data, 12);
                padBytes = BitConverter.ToUInt16(data, 14);
                /* Should also store Time Stamp and Period here */
                break;

            case "logr":
                /* Read session data in 128-byte blocks */
                UInt32 dataSize = chunkSize - padBytes;
                UInt32 dataOffset = offset + padBytes;
                for (UInt32 i = 0; i < dataSize; i += 128)
                {
                    UInt32 len = (dataSize - i) < 128 ? (dataSize - i) : 128;
                    byte[] nextData = deviceConnection.ReadLogData(fileNum, dataOffset, len);
                    logData.AddRange(new List<byte>(nextData));
                }
                /* Process data based on Log File Format/Version */
                LogProcessDataSession(logData.ToArray(), fileFormat, fileVersion, recordSize);
                break;

            default:
                /* We can ignore all other chunks (such as "JUNK") */
                break;
        }
        /* Set offset to beginning of next Chunk Id */
        offset += chunkSize;
    }
}
```

6.3 Processing Logging Data

When all data for a log session has been downloaded, the application needs to process each record within the data based on the information previously obtained from the log header. The following C# example shows how the application can iterate through each record in a data log and event packet in an event log:

```
private void LogProcessDataSession(byte[] logData, string fileFormat, string fileVersion,
                                   UInt16 recordSize)
{
    if (fileFormat == "bhns")       /* Standard BioHarness Data Log */
    {
        /* Convert record size from number of words to number of bytes */
        recordSize *= 2;
        /* Process each record within the data log */
        for (int offset = 0; offset < logData.Length; offset += recordSize)
        {
            LogProcessDataLogRecord(logData, offset, fileVersion);
        }
    }
    else if (fileFormat == "bhel")  /* BioHarness Event Log */
    {
        /* Process each packet within the event log */
        for (int offset = 0; offset < logData.Length; offset += recordSize)
        {
            LogProcessEventLogRecord(logData, offset, fileVersion);
        }
    }
}
```

The data for each individual log record in case of a data log can then be interpreted according to the data formatting as described in Appendix B – BioHarness™ Log Record Formats.

The data for each individual event packet in case of an event log can be interpreted according to the data formatting as described in Appendix C – BioHarness™ Event Data Packet Format.

7 Appendix B – BioHarness™ Log Record Formats

7.1 General Data Only Log Record Format (version 0.0.0.3)

Item	Byte Offset	Length	Description
Heart Rate	0	2	0 – 240 Beats Per Minute (in 1 BPM units)
Respiration Rate	2	2	0 – 170 Breaths Per Minute (in 0.1 BPM units)
Skin Temperature	4	2	10 – 60°C (in 0.1°C units)
Posture	6	2	0 – 90° (in 1° units)
Vector Magnitude	8	2	(in 0.01g units)
Peak Acceleration	10	2	(in 0.01g units)
Battery Voltage	12	2	0 – 4.2v (in mV units)
Not Currently Used	14	2	
GSR Level	16	2	(in 1nS units)
ECG Amplitude	18	2	(in 1µV units)
ECG Noise	20	2	(in 1µV units)
Not Currently Used	22	2	
X Acceleration Min	24	2	(in 0.01g units)
X Acceleration Peak	26	2	(in 0.01g units)
Y Acceleration Min	28	2	(in 0.01g units)
Y Acceleration Peak	30	2	(in 0.01g units)
Z Acceleration Min	32	2	(in 0.01g units)
Z Acceleration Peak	34	2	(in 0.01g units)
Breathing Data 1 - 18	36	36	(0 – 4095) 18 samples (raw ADC data) 32767 denotes no data, sample should be ignored.
R to R time 1 - 18	72	36	18 x R to R times (ms) 32767 denotes no data, sample should be ignored
Not Currently Used	108	20	
Total Bytes		**128**	

Table 7-1 Version 0.0.0.3 Log record

7.1.1 General Data Packing

All items are in Little Endian format (Least significant Byte first) and are 16-bit signed numbers. Each entry is 2 bytes in size.

Byte/Bit	7	6	5	4	3	2	1	0
0	Bit7							Bit 0
1	Bit15							Bit 8

Table 7-2 General Data Item Byte Packing

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

7.1.2 Log Period

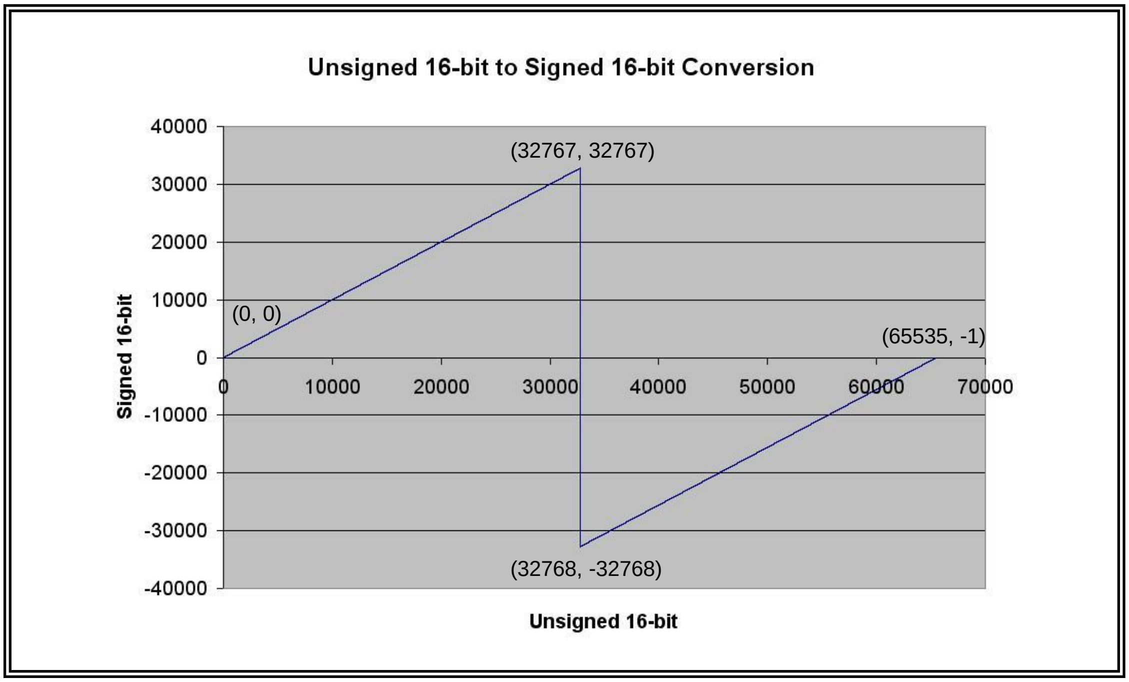
This version of log record holds 18 Breathing Data samples and 18 R to R times and each of these parameters is written to the log record every 56ms. All other data items are written to the log record once per log period. Earlier versions of BioHarness used a log period of 1008ms and in this case all 18 breathing Data samples & R to R times were populated but more recent versions of BioHarness use a log period of 1000ms. In this case most log records will have all 18 data items populated but 1 in every 7 records will only have 17 items populated and the 18^{th} Breathing data sample and 18^{th} R to R time will both be filled with 32767 (0x7FFF) to indicate that the sample contains no data and should be ignored.

7.1.3 Decoding R to R Data

Each R to R time in a log record contains the most recent R to R period at the time it is written to the log record (every 56ms) and has 1ms resolution. This means that it is common to get the same R – R time written several times in succession because the time between heart beats is normally much greater than 56ms.
When a Heart beat is detected, the indicated R to R period is updated. New R to R periods are signified by a change in the sign of the value logged, enabling new R to R values to be recognised even if the new period is the same as the old. For example, if a pulse is received by the unit and the new R to R is the same as the last one, values in the log could be:

876, 876, 876, -876, -876, -876

The change in sign indicates a new R to R period (old and new R to R = 876ms).



Unsigned 16-bit to Signed 16-bit conversion – the data in the log must be converted to a signed 16-bit number before being processed.

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

7.2 General Data + 250Hz ECG Log Record Format (version 0.0.0.4)

Item	Byte Offset	Length	Description
Heart Rate	0	2	0 – 240 Beats Per Minute (in 1 BPM units)
Respiration Rate	2	2	0 – 170 Breaths Per Minute (in 0.1 BPM units)
Skin Temperature	4	2	10 – 60°C (in 0.1°C units)
Posture	6	2	0 – 90° (in 1° units)
Vector Magnitude	8	2	(in 0.01g units)
Peak Acceleration	10	2	(in 0.01g units)
Battery Voltage	12	2	0 – 4.2v (in mV units)
Not Currently Used	14	2	
GSR Level	16	2	(in 1nS units)
ECG Amplitude	18	2	(in 1µV units)
ECG Noise	20	2	(in 1µV units)
Not Currently Used	22	2	
X Acceleration Min	24	2	(in 0.01g units)
X Acceleration Peak	26	2	(in 0.01g units)
Y Acceleration Min	28	2	(in 0.01g units)
Y Acceleration Peak	30	2	(in 0.01g units)
Z Acceleration Min	32	2	(in 0.01g units)
Z Acceleration Peak	34	2	(in 0.01g units)
Breathing Data 1 - 18	36	36	(0 – 4095) 18 samples (raw ADC data) 32767 denotes no data, sample should be ignored.
R to R time 1 - 18	72	36	18 x R to R times (ms) 32767 denotes no data, sample should be ignored
ECG Data Samples	128	378	Up to 252 samples (Bit Packed: 2 samples in every 3 bytes)
Not Currently Used	506	6	
Total Bytes		**512**	

Table 7-3 Version 0.0.0.4 Log record

The first 128 Bytes of this log record format are identical to the General Data Only Log Record Format (version 0.0.0.3) so refer to this section for details of this part of the Log record. The remainder of the log record contains ECG data as detailed below.

7.2.1 250 Hz ECG Data

The ECG data samples are written to the log record at a rate of 250Hz meaning that one sample is written every 4ms. There are up to 252 samples in each log record and the exact number of samples can be calculated by dividing the log period by 4ms, i.e. for a log period of 1008ms, there will be 252 samples and for a log period of 1000ms, there will be 250 samples. In the latter case, the final two samples will not be populated.

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

7.2.2 ECG Data Packing

Each ECG Data sample is 12 bits long and is bit-packed to minimise the amount of space used in the log record. The data is packed in the format shown in Table 7-4 where the bit-packing pattern repeats every 3 bytes.

Byte/Bit	7	6	5	4	3	2	1	0
128								Sample1-Bit 0
129				Sample 2-Bit 0	Sample 1-Bit 11			
130	Sample 2-Bit 11							
131								Sample3-Bit 0
132				Sample 4-Bit 0	Sample 3-Bit 11			
133	Sample 4-Bit 11							

Table 7-4 ECG Data Packing Format

The ECG Data values are not in any units but are raw ADC values.

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

7.3 General data + 100Hz Accelerometer Magnitude Log Record Format (version 0.0.0.8)

Item	Byte Offset	Length	Description
Heart Rate	0	2	0 – 240 Beats Per Minute (in 1 BPM units)
Respiration Rate	2	2	0 – 170 Breaths Per Minute (in 0.1 BPM units)
Skin Temperature	4	2	10 – 60°C (in 0.1°C units)
Posture	6	2	0 – 90° (in 1° units)
Vector Magnitude	8	2	(in 0.01g units)
Peak Acceleration	10	2	(in 0.01g units)
Battery Voltage	12	2	0 – 4.2v (in mV units)
Not Currently Used	14	2	
GSR Level	16	2	(in 1nS units)
ECG Amplitude	18	2	(in 1µV units)
ECG Noise	20	2	(in 1µV units)
Not Currently Used	22	2	
X Acceleration Min	24	2	(in 0.01g units)
X Acceleration Peak	26	2	(in 0.01g units)
Y Acceleration Min	28	2	(in 0.01g units)
Y Acceleration Peak	30	2	(in 0.01g units)
Z Acceleration Min	32	2	(in 0.01g units)
Z Acceleration Peak	34	2	(in 0.01g units)
Breathing Data 1 - 18	36	36	(0 – 4095) 18 samples (raw ADC data) 32767 denotes no data, sample should be ignored.
R to R time 1 - 18	72	36	18 x R to R times (ms) 32767 denotes no data, sample should be ignored
Accelerometer Magnitude Samples	128	100	100 samples (1 byte each)
Not Currently Used	254	2	
Total Bytes		**256**	

Table 7-5 Version 0.0.0.8 Log record

The first 128 Bytes of this log record format are identical to the General Data Only Log Record Format (version 0.0.0.3) so refer to this section for details of this part of the Log record. The remainder of the log record contains Accelerometer Magnitude data as detailed below.

7.3.1 100 Hz Accelerometer Magnitude Data

The Accelerometer Magnitude samples are written to the log record at a rate of 100Hz meaning that one sample is written every 10 ms. There are 100 samples in each log record for a log period of 1000 ms.

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

7.3.2 Accelerometer Magnitude Data Packing

Each Accelerometer Data sample is 8 bits long so that there is one sample per byte as shown in Table 7-6.

Byte/Bit	7	6	5	4	3	2	1	0
128	Sample 1							
129	Sample 2							
:	:							
:	:							
227	Sample 100							

Table 7-6 Accelerometer Data Packing Format

Accelerometer data samples are in 100mg units meaning that the value must be divided by 10 to convert to g's.

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

7.4 Summary Data Only Log Record Format (version 0.0.1.0)

This logging format contains Summary Data, R-to-R Data and B-to-B data.

Data Item	Byte Offset	Range	Resolution	Notes
Version	0	-	-	Currently always set to 2
Heart Rate	1	0…240	1	Invalid = 65535 BPM
Respiration Rate	3	0…70 bpm	0.1	Invalid = 6553.5 BPM
Skin Temperature	5	10…60 °C	0.1	Invalid = -3276.8 °C
Posture	7	-180…180 °	1	Invalid = -32768°
Activity	9	0…16 g	0.01	Invalid = 655.35 g
Peak Acceleration	11	0…16 g	0.01	Invalid = 655.35 g
Battery Voltage	13	0…4.2 V	0.001	Invalid = 65.535 V
Battery Level	15	0..100 %	1	Invalid = 255
Breathing Wave Amplitude	16	-	-	Invalid = 65535
Breathing Wave Noise	18	-	-	Invalid = 65535
Breathing Rate Confidence	20	0..100 %	1	Invalid = 255
ECG Amplitude	21	0…0.05 V	0.000001	Invalid = 0.065535 V
ECG Noise	23	0…0.05 V	0.000001	Invalid = 0.065535 V
Heart Rate Confidence	25	0..100 %	1	Invalid = 255 %
Heart Rate Variability	26	0..65534 ms	1	Invalid = 65535
System Confidence	28	0..100 %	1	Invalid = 255 %
GSR	29	0..65534 nS	1	Invalid = 65535 nS
ROG	31			Invalid = 0
Vertical Axis Acceleration Min	33	-16..16 g	0.01	Invalid = -327.68 g
Vertical Axis Acceleration Peak	35	-16..16 g	0.01	Invalid = -327.68 g
Lateral Axis Acceleration Min	37	-16..16 g	0.01	Invalid = -327.68 g
Lateral Axis Acceleration Peak	39	-16..16 g	0.01	Invalid = -327.68 g
Sagittal Axis Acceleration Min	41	-16..16 g	0.01	Invalid = -327.68 g
Sagittal Axis Acceleration Peak	43	-16..16 g	0.01	Invalid = -327.68 g
Internal Device Temperature	45	-40..80 °C	0.1	Invalid = -3276.8 °C
Status Info	47			
Link Quality	49	0..254	1	Invalid = 255
RSSI	50	-127..128 dB	1	Invalid = -128
Tx Power	51	-128..128	1	Invalid = -128
Estimated Core Temperature	52	33..41 °C	0.1	Invalid = -3276.8 °C
Auxiliary ADC Channel 1	54	0..4095	-	Invalid = 65535
Auxiliary ADC Channel 2	56	0..4095	-	Invalid = 65535
Auxiliary ADC Channel 3	58	0..4095	-	Invalid = 65535
Not Currently Used	60			20 x unused bytes
R-to-R times	80	1…65535 ms	1	6 x R-to-R times (2 bytes each). Invalid = 0
B-to-B times	92	1…65535 ms	1	2 x B-to-B times (2 bytes each). Invalid = 0

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

Data Item	Byte Offset	Range	Resolution	Notes
Not Currently Used	96			32 x unused bytes
Total Bytes	128			

Table 7-7: Summary Log Format

R-to-R data consists of anything between 0 and 6 time values with each value corresponding to a detection indicating the elapsed time since the previous detection. The number of R-to-R times in the packet depends on the number of detections over the 1.000 second epoch with the remaining values set to 0 which should be ignored. Maximum of 360bpm allowed.

B-to-B data consists of anything between 0 and 2 time values with each value corresponding to a detection indicating the elapsed time since the previous detection. The number of B-to-B times in the packet depends on the number of detections over the 1.000 second epoch with the remaining values set to 0 which should be ignored.

7.5 Summary and Waveform Log Record Format (version 0.0.1.1)

This logging format contains Summary Data, R-to-R Data, B-to-B Data and all waveform data including 12-bit 250Hz ECG Data, 24-bit 25Hz Breathing Data and 12-bit 100Hz 3-Axis Accelerometer data.

Data Item	Byte Offset	Range	Resolution	Notes
Summary, R-to-R and B-to-B Data	0	-	-	See Table 7-7
ECG Data	96	-	-	250 x 12 bits packed into 375 bytes
Breathing Data	471	-	-	25 x 24 bits packed into 75 bytes
3-Axis Accelerometer Data	546	-	-	100 x 3 x 12 bits packed into 450 bytes. Data is unsigned, centered around 2048.
Accelerometer Data 1g value	996			16-bit unsigned equivalent of 1g in accelerometer waveform data
Not Currently Used	998			26 x unused bytes
Total Bytes	1024			

Table 7-8: Summary and Waveform Log Format

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

7.6 Summary and Development Log Record Format (v0.0.1.2)

This logging format contains Summary Data, R-to-R Data, B-to-B Data and waveform data required for algorithm development which includes 12-bit 1kHz ECG Data, 24-bit 25Hz Breathing Data and 10-bit 100Hz 3-Axis Accelerometer data.

Data Item	Byte Offset	Range	Resolution	Notes
Summary, R-to-R and B-to-B Data	0	-	-	See Table 7-7.
ECG Data	96	-	-	1000 x 12 bits packed into 1500 bytes
Breathing Data	1596	-	-	25 x 24 bits packed into 75 bytes
3-Axis Accelerometer Data	1671	-	-	100 x 3 x 10 bits packed into 375 bytes. Data is unsigned, centered around 512.
Accelerometer Data 1g value	2046			16-bit unsigned equivalent of 1g in accelerometer waveform data
Total Bytes	2048			

Table 7-9: Summary and Development Log Format

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com

8 Appendix C – BioHarness™ Event Data Packet Format

This section specifies the format used to store event data in the log file. Only the log packet format is described. For a detailed list of event codes and event data descriptions see [3].

8.1 Standard Event Log (version 0.0.0.1)

Data Item	Byte Offset	Size (bytes)	Range	Notes
Event Data Size	0	1	0..255	Determines number of bytes stored in 'Event Data' field
Event Time Stamp	1	8	yyyy:mm:dd:ms	16-bit year, 8-bit month, 8-bit day, 32-bit milliseconds
Event Code	9	2	0..65535	See [3]
Event Data	11	0..21	-	See [3]
Unused Data Space	?	?		unused bytes (filled with 0)
Total Bytes	32			May increase in future implementations

Table 8-1: Standard Event Log (version 0.0.0.1) Format

Ph: (443) 569-3603 Fax: (443) 926-9402 Web: www.zephyr-technology.com