Computrainer Serial Connectivity Analysis

Stephen Done
March 2021
stephenrdone@gmail.com

Introduction

Scope

This document is an analysis of the serial protocol used by the now discontinued Computrainer bike ergometer.

Terms Used

Controller: The software communicating with the Computrainer. Usually running on a PC or Mac.
Packet: The blocks of serial bytes that are sent and received.
Message: The data conveyed within the packet, once the bits relating to serial transmission have been removed and the message has been assembled from what is left.

Motivation

I (still!) own and use a Computrainer for all my indoor bike training. I don't have an excuse to buy a new one because it just won't die! I was interested in creating a Bluetooth interface to the Computrainer. I came upon Mark Liversedge's work and decided to take a look under the hood. As I was probably going to rewrite the code in another language, I wanted to fully understand what was going on, rather than just try to copy what someone else had done. I find things more interesting that way, plus it is easier to debug when you actually understand fully what is going on.

This work has been done in order to make the now quite dated Computrainer work with more modern software. I.e. So that existing software that works with other turbo trainers can be made to work with the Computrainer too, hopefully saving Computrainer owners from scrapping what is a mechanically good system due to outdated software. I have also been contacted by someone who wanted to do the reverse – they wanted to make their own turbo trainer for fun and needed to give it a serial protocol that existing control software understood.

Analysis Methods

Information in this document has been attained or verified by watching the communication between a Computrainer and a controller. Where this has proved insufficient, either the Computrainer or the Controller has been emulated to watch the behaviour of the other.

Basis

This document builds on the excellent analysis work done by Mark Liversedge and contributors for the GoldenCheetah project some time ago. His analysis can be found here:

https://github.com/GoldenCheetah/GoldenCheetah/blob/master/doc/contrib/Computrainer%20Serial%20Protocol%20v0.3%20draft.doc

Mark's code, resulting from his analysis can be found here:

https://github.com/GoldenCheetah/GoldenCheetah/tree/master/src/Train

Divergence from Mark's Document

Inevitably, my work and style will differ slightly from Mark's. Also, since our document are based on various forms of analysis, we are going to have slightly different opinions on the way things work. For readers who have also read Mark's document (and I suggest that you do!), I will try to keep a list below of the areas where my document differs…

Bit and Byte Numbering

In my dim and distant past, I did quite a bit of work on 8051 microcontrollers (as used by the Computrainer). So, in writing this document, I have tried to loosely stick to the bit and byte description style used in the chip manuals. That is just personal preference, as it makes these notes easier for me to understand. This means that bits in a byte are numbered from 7 down to 0, as opposed to 1 up to 8. Arrays of bytes begin at index zero.

Dealing with the Protocol in Layers

I have tried to deal with the protocol in layers. This is why in this document I have I use concepts of packet and message. The 7-byte blocks that appear on the serial line I have named packets. The 6-byte and one bit of data contained within these blocks I have named messages . I found that once the packet layer has been peeled away, the message within is much easier to understand and explain. For this reason, I do not talk in terms of individual bit functions relative to the packet layout, only in the message within, once reconstructed.

What the Z bit actually is

Turns out it's sum on the bytes of the message… in both directions.

Controller Message Checksum

I have a different take on the checksum contained within packets sent from the controller to the Computrainer. It's a checksum of the 3 message bytes that contain data.

Computrainer Wheel Speed

It is simply mph * 50.

Computrainer Button Field

I have discovered more about the bit field that contains the Computrainer Keypad State. The reset button is in bit 0. Bit 7 is a parity bit.

Computrainer Sensor Presence

I have discovered more about the bits that are set when sensors are present. This is important if you want to emulate a Computrainer as well as just talk to one.

Controller Messages

I have documented the messages that are sent by the controller to the Computrainer.

Outstanding Work

No, I don't mean this is a great piece of work! This is a list of work that still needs doing to fully flesh out the analysis…

Computrainer Modes

Further work needs to be done to understand all the Computrainer modes that can be set by the controller.

Unknown Computrainer Parameters

There are several parameters returned by the Computrainer that are not yet understood. Parameter 8 contains the value 0xC7/199dec and I am still unsure what this is. The rest are 0x000 or 0xFFF, so probably not as interesting.

Unknown Controller Parameters

There is one parameter sent by the controller that is not yet understood. Namely parameter 7. If anyone knows what this does, please let me know.

Computrainer Serial Hardware and Port Settings

Connector

The connector used for serial data is a stereo 3.5mm jack.

Voltage Levels

Communication is performed over an RS232 link. Note that the link is not TTL level (0 and 5 volts) as you would find with many hobby microcontroller boards, but full RS-232. So, if you decide to connect with a USB to serial cable, make sure you get the right type, or you'll kill the cable. If the cable ends in a DB9 connector, the chances are that the cable is right.

Signal lines

The stereo jack has only 3 connections. These are used for Ground, Transmit Data and Receive Data. This is 3 wire RS-232.

Port Settings

2400 Baud, 8 data bits, no parity, 1 stop bit.
Hardware Flow Control: None.

Note that hardware flow control is not used. It cannot be, as the link is only 3 wires. Hardware flow control requires CTS and RTS lines i.e. 5 wires.

No further port settings should be necessary.

Computrainer Protocol

Protocol Overview

- First of all, the controller must determine if there is Computrainer attached to the serial port in question – this is **Device Detection**.
- Once a Computrainer has been detected, the controller sends a continuous stream of **Controller Messages** to the Computrainer.
- Shortly after the controller begins sending messages, the Computrainer will response with its own continuous stream of **Computrainer Messages**.It appears that the Computrainer must receive 7 packets from the controller before it starts to stream data back.
- 1.5 seconds after the controller stops sending **Controller Messages**, the Computrainer will also stop sending **Computrainer Messages**.
- Once the message stream has ceased in both directions, it may be reinitiated by repeating the process, beginning with **Device Detection**.

Device Detection

	Controller Challenge	Computrainer Handshake
ASCII Bytes	R a c e r M a t e	L i n k U p
HEX Bytes	52 61 63 65 72 4D 61 74 65	4C 69 6E 6B 55 70

To determine if a Computrainer is connected to a serial port, the controller sends the 9-byte string ‘RacerMate’. A Computrainer that receives this string will respond with the 6-byte string ‘LinkUp’.

Control Protocol

Once it has been determined that a Computrainer is present on the serial port, the controller and Computrainer exchange packets containing messages encoded in a common format.

Packet Format

Each packet consists of 7 bytes of serial data. These packets each convey a 6 byte plus 1 bit (49 bits in total) message. In the diagram below, the full bytes of the message are labelled A-F. The lone message bit is labelled Z.

Each row in the table below shows 1 of the 7 bytes of the packet, labelled 0-6 in the first column. Each cell in subsequent columns shows 1 bit of that byte. By convention, bit 7 is the most significant bit (value 0x80 or 128), while bit 0 is the least significant bit (value 1).

The contents of each bits are as follows:

- Where a bit has value 0 or 1, this is a fixed value that does not change between packets.
- The bits labelled A-F & 0-7 are the individual bits of the message within the packet.
- The bit labelled Z is a single, leftover bit value, not attributed to any full message byte.

		Bit							
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Byte	**0**	0	A7	A6	A5	A4	A3	A2	A1
	1	0	B7	B6	B5	B4	B3	B2	B1
	2	0	C7	C6	C5	C4	C3	C2	C1
	3	0	D7	D6	D5	D4	D3	D2	D1
	4	0	E7	E6	E5	E4	E3	E2	E1
	5	0	F7	F6	F5	F4	F3	F2	F1
	6	1	Z	A0	B0	C0	D0	E0	F0

Data Stream Synchronisation

Bit 7 of all packet bytes is used for synchronisation of the data stream. Once 6 bytes are received with Bit 7 clear, a seventh byte with bit 7 set indicates a complete packet that should be decoded into a message. Should any data be lost (due to interference or a dodgy cable), the receiver can simply wait for 7 bytes matching these criteria and know that it has received a full packet.

Message Format

The message data is formed by extracting and rearranging the bits from the packet as formatted above.

		Bit							
		7	6	5	4	3	2	1	0
Byte	0	A7	A6	A5	A4	A3	A2	A1	A0
	1	B7	B6	B5	B4	B3	B2	B1	B0
	2	C7	C6	C5	C4	C3	C2	C1	C0
	3	D7	D6	D5	D4	D3	D2	D1	D0
	4	E7	E6	E5	E4	E3	E2	E1	E0
	5	F7	F6	F5	F4	F3	F2	F1	F0
Lone Bit		Z							

Z Bit Function

The Z bit is a single bit sum of all six messages bytes conveyed within the packet. In other words, it is set if the sum of the six bytes is odd and is cleared otherwise.

In pascal (as it is less cryptic than C for the casual reader), the message data can be extracted from a packet as follows…

```
z       := Boolean(                          packet[6] AND $40  SHR 6);
data[0] := ((packet[0] AND $7F) SHL 1) OR ((packet[6] AND $20) SHR 5);
data[1] := ((packet[1] AND $7F) SHL 1) OR ((packet[6] AND $10) SHR 4);
data[2] := ((packet[2] AND $7F) SHL 1) OR ((packet[6] AND $08) SHR 3);
data[3] := ((packet[3] AND $7F) SHL 1) OR ((packet[6] AND $04) SHR 2);
data[4] := ((packet[4] AND $7F) SHL 1) OR ((packet[6] AND $02) SHR 1);
data[5] := ((packet[5] AND $7F) SHL 1) OR ((packet[6] AND $01)      );
```

Conversely, the message data can be converted to a packet as follows…

```
packet [0] := (data[0] SHR 1) AND $7F;
packet [1] := (data[1] SHR 1) AND $7F;
packet [2] := (data[2] SHR 1) AND $7F;
packet [3] := (data[3] SHR 1) AND $7F;
packet [4] := (data[4] SHR 1) AND $7F;
packet [5] := (data[5] SHR 1) AND $7F;

packet [6] := $80 // Sync bit always 1
           OR (  Ord(Z)           SHL 6 )
           OR ( (data[0] AND $01) SHL 5 )
           OR ( (data[1] AND $01) SHL 4 )
           OR ( (data[2] AND $01) SHL 3 )
           OR ( (data[3] AND $01) SHL 2 )
           OR ( (data[4] AND $01) SHL 1 )
           OR ( (data[5] AND $01)       );
```

Controller Message Format

This section details what bytes 0..5 are used for in messages sent by the controller.

Overview

Controller messages convey 3 pieces of information from controller to Computrainer:

- Computrainer Mode
- Parameter Type
- Parameter Value

		Bit							
		7	6	5	4	3	2	1	0
Byte	0	Checksum							
	1	Not Used							
	2	Not Used							
	3	Computrainer Mode							
	4	Parameter Type				12-bit Param Value Hi Order Nibble			
	5	12-bit Parameter Value Lo Order Byte							

Computrainer Mode

Byte 3 sets the mode in which the Computrainer runs, such as Incline, Ergo or Calibrate.

Parameter Type

The high nibble of byte 4 indicates a parameter type. I.e. Which parameter value the message contains.

```
ParameterType  :=   data[4] SHR 4;
```

Parameter Value

The low nibble of byte 4 combined with byte 5 is a 12-bit parameter value.

```
ParameterValue := ( data[4] AND $0F ) SHL 8
               OR   data[5];
```

Checksum

This is an inverted byte sum of bytes 3, 4 and 5…

```
Checksum := NOT( Packet.data[3] + Packet.data[4] + Packet.data[5] );
```

I am not convinced that the checksum is validated by the Computrainer, as most 3rd party software gets it wrong, but still seems to work.

Lone Bit Z

Messages sent by the controller must have this bit be set for the Computrainer to accept the message. Since this is a sum bit, you might expect its value to vary between controller packets. However, since the checksum of the message is contained within the message, the sum result is always an odd number, and so the Z bit is always set.

Computrainer Modes

The table below lists all the Mode values that have been seen. There may be others.

Mode	Name	
0x10	Ergo Pause	Paused Ergo (Power) Workout
0x14	Ergometer	Ride at a constant work rate in Watts
0x1C	Calibrate	Calibrate the rolling resistance
0x28	Incline Pause	Paused Incline (Resistance) Workout
0x2C	Incline	Ride at a constant resistance or 'incline'. SpinScan data is reported in this mode.

Calibration Mode

Selecting mode 0x1C puts the Computrainer into calibration mode. In this mode, the Computrainer only reports wheel speed. It still outputs the standard sequence of 13 packets, but the power and cadence parameters appear to be zero. If you need to know whether the user actually performs a calibration, you would need to do one or a combination of the following:

- Watch for the Computrainer Keypad 'F3' key being pressed to save the rolling resistance value.
- Watch for the 'Calibration Required' bit being cleared, assuming the Computrainer was not already calibrated before entering calibration mode.
- Watch for a change in the RRC rolling resistance value reported.

Controller Message Parameters

Param Type	Parameter Name	Data Type	Units	Value Scaling
0x0	*Not Seen*			
0x1	Gradient (incline mode)	Signed		`Gradient := Value / 10; -15..+15`
0x2	Wind Speed	Signed	Mph	`WindSpeed := Value;`
0x3	Rider Weight	Unsigned	Lbs	`RiderWeight := Value`
0x4	HR Warning Limit Low	Unsigned	BPM	`Low Limit := $800 OR Value`
0x5	HR Warning Limit High	Unsigned	BPM	`High Limit := $800 OR Value`
0x6	Drag Factor	Unsigned		`DragFactor := (Value * 100)/2048`
0x7	Seen but Unknown			`Seen 0x0F7. Likely unsigned value.`
0x8	*Power (ergo mode)*	Unsigned		`Power := Value;`
0x9	*Not Seen*			
0xa	*Not Seen*			
0xb	*Not Seen*			
0xc	*Not Seen*			
0xd	*Not Seen*			
0xe	*Not Seen*			
0xf	*Not Seen*			

Signed Data Values

Where a data value is signed, the 12-bit number needs to be sign extended into whatever integer data type is being used to store the integer on the controller. If this is not done correctly, negative values will not be correct.

Unknown Parameters

The following parameters have been seen, but their purpose is not understood.

Parameter 0x7

During RacerMate One startup, the values 0x0E6 and 0x0F7 have been seen. Zwift sets this parameter to 0x120. These hex values correspond to decimal values 230, 247 and 288.

Computrainer Message Format

This section details what bytes 0..5 are used for in messages sent by the Computrainer.

Overview

Messages are able to carry a portion of the SpinScan data, the status of the buttons on the Computrainer keypad, as well as a single data parameter, such as cadence or power. The full picture of the Computrainer state (cadence, wheel speed, power, heart rate etc) is built up over the course of several consecutive messages sent back by the Computrainer.

		Bit							
		7	6	5	4	3	2	1	0
Byte	0	SpinScanDataByte[0]							
	1	SpinScanDataByte[1]							
	2	SpinScanDataByte[2]							
	3	Parity	SpinScan	Minus	F2	Plus	F3	F1	Reset
	4	Parameter Type				12-bit Param Value Hi Order Nibble			
	5	12-bit Parameter Value Lo Order Byte							

SpinScan Data

A complete set of SpinScan data consists of 24 byte values. A single message can only convey 3 bytes of SpinScan data in the first 3 bytes of the message.

Status Bits

SpinScan Data

When the SpinScan bit is set, this indicates that the message contains bytes 0..2 of the 24 SpinScan bytes. The rest of the SpinScan data will be sent in 7 future messages. See SpinScan section.

Button State

Each of the six button state bits are set when the corresponding button is depressed.

Parity Bit

Bit 7 is an even parity bit. This means that bit 7 is set when there are 0, 2, 4 or 6 other bits set. This is why bit 7 is clear when there are no buttons pressed.

Parameter Type

The high nibble of byte 4 indicates a parameter type. I.e. Which parameter value the message contains.

```
ParameterType  :=   data[4] SHR 4;
```

Parameter Value

The low nibble of byte 4 combined with byte 5 is a 12-bit parameter value.

```
ParameterValue := ( data[4] AND $0F ) SHL 8
                  OR   data[5];
```

Lone Z Bit Value

This varies for every message, depending on the sum of the bytes. So if you want to emulate a Computrainer, you must calculate this value correctly, otherwise RacerMate (and any other software that properly understands the protocol) is going to reject the message as being corrupted.

Computrainer Message Parameters

Param Type	Parameter Name	Units	Value Scaling
0x0	*Not seen*		
0x1	Speed	mph	`Speed := Value / 50`
0x2	Power	Watts	`Power := Value`
0x3	Heart Rate	BPM	`HeartRate := Value AND $FF`
	Heart Rate Sensor Active	Boolean	`Active := Value AND $100`
0x4	Unknown		`Seen Value 0x000`
0x5	Unknown		`Seen Value 0x000`
0x6	Cadence	RPM	`Cadence := Value AND $FF`
		Boolean	`SensorConnected := Value AND $800 <> 0`
0x7	Unknown		`Seen Value 0x000`
0x8	Constant		`Constant Value Returned: $C7 or 199 dec`
0x9	Push-on Pressure ('RRC')	lb ft	`(Value AND $7FF) / $100`
	Calibrated	Boolean	`(Value AND $800) <> 0`
0xa	Unknown		`Seen Value 0x000`
0xb	Sensor Connected	Boolean	`Cadence = ( Value AND $800 ) <> 0`
	Indicates that a jack cable is plugged into the heart rate or cadence sensor socket.	Boolean	`HeartRate = ( Value AND $400 ) <> 0`
0xc	Firmware Version	No Units	`Controller Version 45.43 Returns $FFF`
0xd	*Not seen*		
0xe	*Not seen*		
0xf	*Not seen*		

Once the Computrainer is returning data, the controller is not able to influence what parameters are returned. The Computrainer returns a continuous stream of messages containing parameters from 1 to 12. The sequence repeats every 13 packets, with speed being sent twice – probably because it is deemed more important to be updated frequently.

The 13-message sequence is **1** 2 3 4 5 6 **1** 7 8 9 A B C.

Unseen Message Parameters

Message Parameter Types 0, 13, 14 and 15 have not been seen. It is assumed that only parameters 1-12 exist.

Unknown Message Parameters

Message parameters 4, 5, 7, 8 & 10 have been seen, though their purpose is not yet understood. Perhaps these relate in some way to the VeloTron – the luxury relative of the Computrainer.

Emulating the Computrainer

The 'Computrainer Message Parameters' table details how to extract parameters from the message parameter fields. But bear in mind that the value field is a 12-bit field. When emulating a Computrainer, as opposed to reading it, it is important to set certain bits within these 12 bits, that are not used for the actual value. Otherwise official software may not accept the data that your emulator sends.

Sensor Presence

The $800 and $400 bits for Cadence and Heart Rate must be OR'ed with $301 in order to report an identical value as a Computrainer. I.e. if both Heart Rate and Cadence sensors are present, the 12-bit value reported by a Computrainer is $F01. I have not determined what the other bits signify, but it would be reasonable to assume that this is a bitmask of features or hardware.

Heart Rate

Bit 8 of the heart rate parameter value must be set, as this indicated that a working heart rate chest strap is being picked up by the receiver.

```
Value := HeartRate OR $100;
```

Cadence

Bit 11 of the cadence parameter must be set. This is a duplicate of the cadence bit in the Sensor Connected parameter. I.e. It indicates that a cadence sensor has been plugged into the jack socket.

```
Value := Cadence OR $800;
```

SpinScan

SpinScan in Gradient/Incline Mode Only

SpinScan data is only reported in gradient/Incline mode. This is likely because of the way that the SpinScan data is calculated. The Computrainer looks for the cadence sensor signal in order to know the location of the crank. It will then look for minor variations in wheel speed during a single rotation of the crank. These changes in speed (acceleration and deceleration) correspond to peaks and troughs in the polar SpinScan plot. For this data to make sense, the resistance presented to the rider must be constant, and this is only the case in gradient mode. In ergo mode, the Computrainer is constantly varying the resistance presented to the rider, which in itself would cause changes in wheel speed, hence SpinScan is not available in ergo mode. This assessment is based on my best guess of how things work.

Assembling the SpinScan Data

A full 24 bytes of SpinScan data is returned every 2 seconds. To gather a full 24-byte set of SpinScan data, 8 records each containing 3 bytes of SpinScan data must be received. The first of the 8 records is signified by a message with the SpinScan flag set in the status bits byte of the message. Any subsequent message where any of the first 3 bytes are non-zero is the next 3 bytes of SpinScan data. There may be several messages in between the messages carrying the SpinScan data. Once the last of the 8 SpinScan records is received, there will then be more non-SpinScan messages before the sequence repeats, starting with another message with the SpinScan flag set.

Interpreting the SpinScan Data

The 24 SpinScan values represent a complete revolution of the crank. Therefore each value corresponds to a 15-degree arc of the revolution (360/24=15 degrees).

Here's a one guess of how to interpret the data. This one is Mark's:

- Swap nibbles of each byte.
- Invert each byte.
- AND with $7F.

Here's another guess (mine). No better than Mark's – just another possibility:

- Rotate the byte right three bits.
- Invert the byte.

The only difference here is that the bit that Mark masks off, I am instead using as the least significant bit of the value and adding double Mark's value.

The scale is currently unknown and I don't know if either guess of the SpinScan data format is correct.

Notes to self

Computrainer message parameter 8 might be related to SpinScan in some way. Parameter 8 always has decimal value 199, which is 0xC7 in hex. When in SpinScan mode and the crank is not turning, the SpinScan values returned are all 0xF7. These hex values are similar. I wonder if 199/0xC7 is some sort of bit mask.
Also, controller parameter 7 is often set to 0xF7. This is the element value returned in spinscan data when there is no SpinScan data. Perhaps this is also related or perhaps it is a co-incidence.

Appendix 1 – Seen Controller Message Parameters

This section includes examples of logged data, including parameters that are not yet understood.

RacerMate One

Gradient Mode

```
01 0x000: Gradient              0.0%
02 0x000: Wind Speed            0.00 mph
03 0x0A1: Rider Weight          161.00 lbs
04 0x800: HR Low alarm          0
05 0x8FF: HR High alarm         255
06 0x800: Drag Factor           100.0%
07 0x0F7: ???                   247.00
08 0x000: Power                 0 Watts
```

RacerMate One Ergo Mode Computrainer Data Stream

The stream repeats every 13 messages. There are 12 unique parameters sent, but speed is repeated once, mid-way through, hence 13 messages. Z is set only on the message conveying parameter 8, with constant value 199.

```
00 00 00 80 10 00  z:False
00 00 00 80 20 00  z:False
00 00 00 80 3C 00  z:False
00 00 00 80 40 00  z:False
00 00 00 80 50 00  z:False
00 00 00 80 68 00  z:False
00 00 00 80 10 00  z:False
00 00 00 80 70 00  z:False
00 00 00 80 80 C7  z:True
00 00 00 80 9A 7A  z:False
00 00 00 80 A0 00  z:False
00 00 00 80 BB 01  z:False
00 00 00 80 CF FF  z:False
```

RacerMate One Incline Mode Computrainer Data Stream

```
F7 F7 F7 40 BB 01  z:True
F7 F7 F7 80 CF FF  z:True
F7 F7 F7 80 10 00  z:True
F7 F7 F7 80 20 00  z:True
F7 F7 F7 80 3C 00  z:True
F7 F7 F7 80 40 00  z:True
F7 F7 F7 80 50 00  z:True
00 00 00 80 68 00  z:False
00 00 00 80 10 00  z:False
00 00 00 80 70 00  z:False
00 00 00 80 80 C7  z:True
00 00 00 80 9A 7A  z:False
00 00 00 80 A0 00  z:False
00 00 00 80 BB 01  z:False
F7 F7 F7 80 CF FF  z:True
00 00 00 80 10 00  z:False
00 00 00 80 20 00  z:False
00 00 00 80 3C 00  z:False
00 00 00 80 40 00  z:False
00 00 00 80 50 00  z:False
00 00 00 80 68 00  z:False
00 00 00 80 10 00  z:False
00 00 00 80 70 00  z:False
00 00 00 80 80 C7  z:True
00 00 00 80 9A 7A  z:False
00 00 00 80 A0 00  z:False
00 00 00 80 BB 01  z:False
00 00 00 80 CF FF  z:False
00 00 00 80 10 00  z:False
00 00 00 80 20 00  z:False
00 00 00 80 3C 00  z:False
00 00 00 80 40 00  z:False
00 00 00 80 50 00  z:False
00 00 00 80 68 00  z:False
00 00 00 80 10 00  z:False
00 00 00 80 70 00  z:False
00 00 00 80 80 C7  z:True
00 00 00 80 9A 7A  z:False
00 00 00 80 A0 00  z:False
00 00 00 80 BB 01  z:False
00 00 00 80 CF FF  z:False
00 00 00 80 10 00  z:False
00 00 00 80 20 00  z:False
00 00 00 80 3C 00  z:False
00 00 00 80 40 00  z:False
00 00 00 80 50 00  z:False
00 00 00 80 68 00  z:False
00 00 00 80 10 00  z:False
00 00 00 80 70 00  z:False
00 00 00 80 80 C7  z:True
00 00 00 80 9A 7A  z:False
00 00 00 80 A0 00  z:False
```

Re-Assembled SpinScan Records

The following 24-byte blocks of SpinScan data were generated by pedalling… in slippers. So no comments about (lack of) smoothness please! These are 'as received' and have not been decoded in any way. The number with a decimal point is the time at which the data was logged.

```
SpinScan: 44 E3 9B 6B 6B 5B 9B 6B 3B 53 FB 44
159.546   0C E3 AB B3 9B 43 2B 33 4B 6B B3 14

SpinScan: F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7
161.515   F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7

SpinScan: 4C FB D3 9B 53 63 6B 73 7B A3 DB 1C
163.562   1C FB C3 93 6B 5B 73 6B BB AB F3 64

SpinScan: 7C 34 DB AB 7B B3 7B 43 7B BB E3 14
165.468   4C 1C E3 93 5B 4B 5B 6B 73 A3 E3 24

SpinScan: 6C 54 04 CB 9B 8B 7B 7B BB 83 C3 2C
167.296   7C 24 D3 9B 73 A3 43 2B 63 D3 1C 64

SpinScan: 84 34 E3 B3 9B 83 7B 73 83 93 D3 EB
169.031   1C FB C3 8B 73 73 7B BB 93 B3 FB 5C

SpinScan: 6C 44 FB E3 D3 A3 7B 8B A3 CB E3 0C
170.843   24 04 BB 93 73 73 63 83 9B CB EB 2C

SpinScan: 5C 5C 14 D3 BB 9B 9B B3 6B 8B EB 44
172.578   3C 0C CB B3 AB 5B 6B 9B B3 DB 0C 4C

SpinScan: 84 84 6C 64 5C 4C 34 2C 34 4C 84 9C
174.484   BC D4 25 06 E7 F7 F7 F7 F7 F7 F7 F7

SpinScan: F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7
176.515   F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7 F7
```