USB Sabertooth Packet Serial

Reference Manual

Table of Contents

Plain Text or Packet Serial?

USB-enabled Sabertooth motor drivers support two serial protocols, Plain Text Serial and Packet Serial:

Plain Text Serial
An extremely easy way to interact with the Sabertooth from a microcontroller or PC.

For example, if you open up a terminal at 9600 baud (8-N-1), all of the following are valid commands you can type:

```
M1: 0
M1: 2047
M1: -2047
M2: 500
M1: GET
M1: GET B
M1: GET C
M1: GET T
M1: SHUT DOWN
M1: START UP
```

Mixed mode (tank-style differential drive) is similarly easy with drive and turn:

```
MD: 0
MT: 0
MT: 2047
```

Power outputs (P1 and P2, if configured as controllable outputs in DEScribe), freewheeling (Q1 and Q2), ramping (R1 and R2), and current limit (T1 and T2) can also be controlled.

Plain Text Serial commands can, *optionally*, even be protected by a checksum.
Sum the uppercase form of all non-space characters. Append a +, and then the hexidecimal form of the lower eight bits of the sum:

```
M1:-2047+B2
```

If this interests you, see the manual's section on Plain Text Serial.

Packet Serial
Designed for high reliability even on noisy signal lines.

Among Packet Serial's advantages, it uses checksums or optionally CRCs, its command packets are more compact, and it can share a S1 line with other SyRen and Sabertooth motor drivers as well as Kangaroo motion controllers running in Packet Serial mode. On the other hand, Packet Serial is more difficult than Plain Text Serial to implement correctly.

If you decide to use Packet Serial, this documentation has all the necessary details. Alternatively, we've made libraries for several platforms.

Software Libraries

We provide Packet Serial libraries for several platforms:

Arduino
http://www.dimensionengineering.com/arduino

C#, VB.NET, and other .NET Framework languages
http://www.dimensionengineering.com/dotnet

Packet Format

Packet Serial commands can be sent protected by either a checksum or a CRC:

checksum
- good for most applications
- easier to implement
- faster updates (uses one less byte per command than a CRC)

CRC
- good for safety-critical applications with noisier wiring
- harder to implement
- slower updates (uses one more byte per command than a checksum)
- provides a Hamming distance of 4

Checksum-protected Packet Serial commands have the following format:

Address (1 byte)
The Sabertooth address. By default, this is 128. Address bytes have the high bit set.

Command (1 byte)
The command number.

Value (1 byte)
The command value.

Checksum (1 byte)
The low 7 bits of the sum of the address, command, and value bytes.

Data (n bytes)
Extra data the command needs, if any.

Data Checksum (1 byte)
The low 7 bits of the sum of the data bytes. If there is no extra data, you do not need to send this.

See the Example Code (Checksum) section for sample implementations.

CRC-protected Packet Serial commands have the following format:

Address (1 byte)
The Sabertooth address (by default, 128), plus 112.
Adding 112 signifies that the command is CRC-protected.

Command (1 byte)
The command number.

Value (1 byte)
The command value.

CRC (1 byte)
See the Example Code (CRC) section.

A 7-bit CRC computed from the address, command, and value bytes.
The polynomial is 0x5B in Koopman notation.

Data (n bytes)

Extra data the command needs, if any.

Data CRC (2 bytes)
See the Example Code (CRC) section.

A 14-bit CRC computed from the data bytes. If there is no extra data, you do not need to send this.
The lower 7 bits are written into the first byte, and the upper 7 bits into the second byte.
The polynomial is 0x21E8 in Koopman notation.

See the Example Code (CRC) section for sample implementations.

Example Code (Checksum)

```
#include <stdint.h>

/*! Writes a Packet Serial command into a buffer.
\param address The address of the Sabertooth. By default, this is 128.
\param command The command number.
\param value   The command value.
\param data    Extra data.
\param length  The number of bytes of extra data.
\param buffer  The buffer to write into.
\return        How many bytes were written. This always equals 5 + length,
               unless length is 0, in which case it equals 4. */
uint8_t writeSabertoothCommand(uint8_t address,
                               uint8_t command, uint8_t value,
                               const uint8_t* data, uint8_t length,
                               uint8_t* buffer)
{
  uint8_t i; uint8_t dataChecksum;

  buffer[0] = address;
  buffer[1] = command;
  buffer[2] = value;
  buffer[3] = (address + command + value) & 127;

  if (length == 0)
  {
    return 4;
  }
  else
  {
    dataChecksum = 0;
    for (i = 0; i < length; i ++)
    {
      buffer[4 + i] = data[i];
      dataChecksum += data[i];
    }
    buffer[4 + length] = dataChecksum & 127;

    return 5 + length;
  }
}

/*! Writes a Set command into a buffer.
\param address      The address of the Sabertooth. By default, this is 128.
\param setType      0 to set the value, 16 to send a keep-alive,
                    32 to set the shutdown state, or 64 to set the serial
timeout.
\param targetType   'M' for a motor output, 'P' for a power output, etc.
\param targetNumber 1 or 2, or a Simplified Serial character like '1' or '2'.
\param value        The value to set to.
\return             How many bytes were written. This always equals 9. */
uint8_t writeSabertoothSetCommand(uint8_t address, uint8_t setType,
                                  uint8_t targetType, uint8_t targetNumber,
```

```
                                    int16_t value,
                                    uint8_t* buffer)
{
  uint8_t data[4];
  data[2] = targetType;
  data[3] = targetNumber;

  if (value < 0)
  {
    value = -value;
    data[0] = (uint8_t)((value >> 0) & 127);
    data[1] = (uint8_t)((value >> 7) & 127);

    return writeSabertoothCommand(address, 40, setType + 1, data, 4, buffer);
  }
  else
  {
    data[0] = (uint8_t)((value >> 0) & 127);
    data[1] = (uint8_t)((value >> 7) & 127);

    return writeSabertoothCommand(address, 40, setType, data, 4, buffer);
  }
}
```

Example Code (CRC)

```
#include <stdint.h>

/*! Computes a 7-bit CRC.
\param data   The data to compute the CRC of.
\param length The length of the data.
\return       The CRC. */
uint8_t crc7(const uint8_t* data, uint8_t length)
{
  uint8_t crc = 0x7f; uint8_t i, bit;

  for (i = 0; i < length; i ++)
  {
    crc ^= data[i];
    for (bit = 0; bit < 8; bit ++)
    {
      if (crc & 1) { crc >>= 1; crc ^= 0x76; }
      else         { crc >>= 1;              }
    }
  }

  return crc ^ 0x7f;
}

/*! Computes a 14-bit CRC.
\param data   The data to compute the CRC of.
\param length The length of the data.
\return       The CRC. */
uint16_t crc14(const uint8_t* data, uint8_t length)
{
```

```
  uint16_t crc = 0x3fff; uint8_t i, bit;

  for (i = 0; i < length; i ++)
  {
    crc ^= data[i];
    for (bit = 0; bit < 8; bit ++)
    {
      if (crc & 1) { crc >>= 1; crc ^= 0x22f0; }
      else         { crc >>= 1;                }
    }
  }

  return crc ^ 0x3fff;
}

/*! Writes a CRC-protected Packet Serial command into a buffer.
\param address The address of the Sabertooth. By default, this is 128.
\param command The command number.
\param value   The command value.
\param data    Extra data.
\param length  The number of bytes of extra data.
\param buffer  The buffer to write into.
\return        How many bytes were written. This always equals 6 + length,
               unless length is 0, in which case it equals 4. */
uint8_t writeCRCSabertoothCommand(uint8_t address,
                                  uint8_t command, uint8_t value,
                                  const uint8_t* data, uint8_t length,
                                  uint8_t* buffer)
{
  uint8_t i; uint16_t crc;

  buffer[0] = address + 112;
  buffer[1] = command;
  buffer[2] = value;
  buffer[3] = crc7(buffer, 3);

  if (length == 0)
  {
    return 4;
  }
  else
  {
    for (i = 0; i < length; i ++)
    {
      buffer[4 + i] = data[i];
    }

    crc = crc14(buffer + 4, length);
    buffer[4 + length] = (uint8_t)((crc >> 0) & 127);
    buffer[5 + length] = (uint8_t)((crc >> 7) & 127);

    return 6 + length;
  }
}

/*! Writes a CRC-protected Set command into a buffer.
\param address      The address of the Sabertooth. By default, this is 128.
```

```
\param setType      0 to set the value, 16 to send a keep-alive,
                    32 to set the shutdown state, or 64 to set the serial
timeout.
\param targetType   'M' for a motor output, 'P' for a power output, etc.
\param targetNumber 1 or 2, or a Simplified Serial character like '1' or '2'.
\param value        The value to set to.
\return             How many bytes were written. This always equals 10. */
uint8_t writeCRCSabertoothSetCommand(uint8_t address, uint8_t setType,
                                     uint8_t targetType, uint8_t targetNumber,
                                     int16_t value,
                                     uint8_t* buffer)
{
  uint8_t data[4];
  data[2] = targetType;
  data[3] = targetNumber;

  if (value < 0)
  {
    value = -value;
    data[0] = (uint8_t)((value >> 0) & 127);
    data[1] = (uint8_t)((value >> 7) & 127);

    return writeCRCSabertoothCommand(address, 40, setType + 1, data, 4,
buffer);
  }
  else
  {
    data[0] = (uint8_t)((value >> 0) & 127);
    data[1] = (uint8_t)((value >> 7) & 127);

    return writeCRCSabertoothCommand(address, 40, setType, data, 4, buffer);
  }
}
```

Commands

USB Sabertooth Commands
The USB-enabled Sabertooth command set exposes features such as 12-bit motor outputs, power outputs, control over freewheeling, motor current read-back, and User Mode variables. If you do not need these features, and want your code to be compatible with non-USB Sabertooth/SyRen motor drivers as well, you may want to use the legacy command set instead.

Set

Sets a value on the motor driver.

Command
The command number for Set is 40.

Command Value
The command value depends on what you are trying to set. The possibilities are:

> **0 - Value**
> Sets the value.
>
> **16 - Keep-Alive**
> A keep-alive will reset the serial timeout without taking any action.
>
> **32 - Shutdown**
> A positive value shuts down the motor channel.
> A negative or zero value starts up the motor channel again.
>
> **64 - Timeout**
> Sets the serial timeout, in milliseconds.
>
> A zero value uses the DEScribe setting.
> A negative value disables the serial timeout.

If the value you are trying to set is negative, add 1 to this command value.

Command Data
The data for a Set is 4 bytes long:

> **Value (2 bytes)**
> The value to set the target to.
>
> The lower 7 bits are written into the first byte, and the upper 7 bits into the second byte.
>
> **Target (2 bytes)**
> The target.
>
> This can be a particular channel, such as M1, M2, MD (drive), MT (turn), P1, P2, Q1, Q2, R1, R2, T1, or T2.
> Alternatively, you can target all channels of a type: M*, P*, Q*, R*, or T*. M* does not include MD or MT.
>
> For the channel's number, if you use decimal 1 or 2, the channel will set regardless of Plain Text Serial address.
> If you use the character '1' or '2', the channel will set based on the motor driver's Plain Text Serial

address (settable via DEScribe).

Get

Gets a value on the motor driver.

Command
The command number for Get is 41.

Command Value
The command value depends on what you are trying to set. The possibilities are:

0 - Value
Gets the value.

16 - Battery
Gets the battery voltage, in tenths of a volt.

32 - Current
Gets the motor current, in amps.

64 - Temperature
Gets the temperature, in degrees Celsius.

Command Data
The data for a Get is 2 bytes long:

Source (2 bytes)
The source.

This can be S1, S2, A1, A2, M1, M2, P1, or P2.

For the channel's number, if you use decimal 1 or 2, the channel will get regardless of Plain Text Serial address.
If you use the character '1' or '2', the channel will get based on the motor driver's Plain Text Serial address (settable via DEScribe).

Reply
The reply number is 73.

Reply Value
The reply value will match the command value, or the command value plus one if the value is negative.

Reply Data
The reply is 4 bytes long:

Value (2 bytes)
The value the source is set to.

The lower 7 bits are in the first byte, and the upper 7 bits are in the second byte.

Source (2 bytes)
The source.

On the Web

Click the following link to visit our main website:
Dimension Engineering

If you have questions, we can be reached via either of the following:
Help Desk
Contact Us

Also, here are some software links you may find useful:
DEScribe
Libraries for Arduino
Libraries for C# and .NET

Revision History

December 17, 2014:

Fixed the CRC example code.

December 9, 2014:

Revised some of the copy.

September 12, 2013:

Initial public version of the Packet Serial Reference.

Index