harp

Binary Protocol 1.0

Document Version 1.0

1. Introduction

The Harp Binary Protocol is a binary communication protocol created in order to facilitate and unify the interaction between different devices. It was design with efficiency and parse ease in mind.

The protocol is based on addresses. Each address points to a certain position register available in the device. These positions are called registers. Each register should have a defined data type and a meaning/purpose.

Although is not mandatory, usually the Harp Binary Protocol is exchanged between a Master and a Slave. The Master can be a computer or a server and the Slave can be a data acquisition or an actuator device.

The available packets are:

- **Command:** Sent by the Master to the Slave. They allow to change the registers content and to read the registers content.
- **Reply:** Sent by the Slave as an answer to a Command.
- **Event:** Sent by the Slave when an external or internal event happens. An Event carry the content of a register.

Note that the Harp Binary Protocol uses Little-Endian for byte organization.

2. Harp Message

The Harp Message consists on the necessary information to execute a well-informed exchange of data. It follows the next structure.

```
== Harp Message ==
[MessageType] [Length] [Address] [Port] [PayloadType] [Payload] [Checksum]


[MessageType] (1 byte)
 1 – Read  : The device requests the content of the register with address [Address]
 2 – Write : The device is writing the content to the register with address [Address]
 3 – Event : The device is sending the content of the register with address [Address]


[Length] (1 byte)
 The number of bytes in the Harp Message still to be read, i.e., the number of bytes after
the field [Length].


[Address] (1 byte)
 The address to which the Harp Message refers to.
```

[Port] (1 byte)

If the device is a Hub of Harp Messages, this field indicates the origin or destination of the Harp Message.

To point to the device itself this field should be equal to 255.

[PayloadType] (1 byte)

Indicates the type of data available on the [Payload].
The [Payload] can contain:

- An element **T**
- Or a timestamped element **Timestamped<T>**

The next list states the available types of the [Payload]

1 - **T** U8 : Unsigned 8 bits

2 - **T** U16 : Unsigned 16 bits

4 - **T** U32 : Unsigned 32 bits

8 - **T** U32 : Unsigned 64 bits

129 - **T** I8 : Signed 8 bits

130 - **T** I16 : Signed 16 bits

132 - **T** I32 : Signed 32 bits

136 - **T** I64 : Signed 64 bits

68 - **T** Float : Single-precision floating-point 32 bits

16 - **Timestamped<>** : Time information only

17 - **Timestamped<T>** U8 : Timestamped unsigned 8 bits

18 - **Timestamped<T>** U16 : Timestamped unsigned 16 bits

20 - **Timestamped<T>** U32 : Timestamped unsigned 32 bits

24 - **Timestamped<T>** U64 : Timestamped unsigned 64 bits

145 - **Timestamped<T>** I8 : Timestamped signed 8 bits

146 - **Timestamped<T>** I16 : Timestamped signed 16 bits

148 - **Timestamped<T>** I32 : Timestamped signed 32 bits

152 - **Timestamped<T>** I64 : Timestamped signed 64 bits

84 - **Timestamped<T>** Float : Timestamped Single-precision floating-point 32 bits

If the Types is a **Timestamped<T>**, the first 6 bytes contains the time information and is divided into [TimestampSeconds] (4 bytes) and [TimestampMicroseconds] (2 bytes).

[TimestampSeconds] - Unsigned 32 bits containing the seconds.

[TimestampMicroseconds] - Unsigned 16 bits containing the microseconds divided by 32.

The time information can be retrieved using the formula:

$$\text{Timestamp(s)} = [\text{TimestampSeconds}] + [\text{TimestampMicroseconds}] * 32 * 10^{-6}$$

[Payload] (? byte(s))

The content.

[Checksum] (1 byte)

The U8 (unsigned 8 bits) sum of all bytes of the Harp Data.

The receiver of the package should compute himself this checksum and compare with the one present on the Harp Message. The Harp Message should be discarded if both do not match.

2.1 Features and Code Examples

Some of the fields described on the previous chapter have special features. These are presented next.

[MessageType] (1 byte)

The field [Command] has an Error flag on the 4th least significant bit. When this bit is set it means that an error occur.

Examples of possible errors cane be *a)* when Master tries to read a register that doesn't exist, *b)* Master tries to write unacceptable data to a certain register, *c)* [PayloadType] doesn't match with the register [Address] type, etc.

A simple code in C to check for error will be:

```
int errorMask = 0x08;

if (Command & errorMask)
{
   printf("Error detected.\n");
}
```

[Length] (1 byte)

If one byte is not enough to express the length of the Harp Message, use [Length] equal to 255 and add after an unsigned 16 bits word with the Harp Message length.

Replace the [Length] with:

[255] (1 byte) [ExtendedLength] (2 bytes)

[PayloadType] (1 byte)

For the definition of the [PayloadType] types, a C# code is presented.
Note that the time information can appear without an element **Timestamp<>**.

```
int isUnsigned = 0x00;
int isSigned = 0x80;
int isFloat = 0x40;
int hasTimestamp = 0x10;

enum PayloadType
{
    U8  = (isUnsigned | 1),
    S8  = (isSigned   | 1),
    U16 = (isUnsigned | 2),
    S16 = (isSigned   | 2),
    U32 = (isUnsigned | 4),
    S32 = (isSigned   | 4),
    U64 = (isUnsigned | 8),
    S64 = (isSigned   | 8),
    Float = (isFloat  | 4),
    Timestamp = hasTimestamp,
    TimestampedU8  = (hasTimestamp | U8),
    TimestampedS8  = (hasTimestamp | S8),
    TimestampedU16 = (hasTimestamp | U16),
    TimestampedS16 = (hasTimestamp | S16),
    TimestampedU32 = (hasTimestamp | U32),
    TimestampedS32 = (hasTimestamp | S32),
    TimestampedU64 = (hasTimestamp | U64),
    TimestampedS64 = (hasTimestamp | S64),
    TimestampedFloat = (hasTimestamp | Float)
}
```

[PayloadType] (1 byte)

The field [PayloadType] has a flag on the 5th least significant bit that indicates if the time information is available on the Harp Message. For some reasons, the time information may not make sense to appear on the Harp Message.

A simple code in C to check if the time information is available will be:

```
int hasTimestamp = 0x10;

if (PayloadType &  hasTimestamp )
{
   printf("The time information is available on the Harp Message's Payload.\n");
}
```

[Checksum] (1 byte)

Example on how to calculate the [Checksum] in C language.

```
unsigned char Checksum = 0;
int i = 0;

for (; i < Length + 1; i++ )
{
   Checksum += HarpMessage(i);
}
```

2.2 Payload and Arrays

The [payload]'s element can contain a single value or an array of values. To find the amount of values a simple code can be applied using the information contained on the [Length] and the [PayloadType].

Example to calculate the number of values on the [Payload]'s element in C language:

```
int arrayLength;
int hasTimestamp = 0x10;
int sizeMask = 0x0F;

if (PayloadType & hasTimestamp )
{
   /* Harp Message has time information
   arrayLength = (Length – 10) / (PayloadType & sizeMask )
}
else
{
   /* Harp Message doesn't have time information
   arrayLength = (Length – 4) / (PayloadType & sizeMask )
}
```

3. Typical Usage

3.1 Commands

Usually the Slave device that runs this protocol receives Write and Read commands from the Master and sends Events to the Master.

Some Harp Messages are shown here to demonstrate the typical usage. Note that, from the Master to the Slave, the time information is not added to the Harp Message since this information is not necessary

```
Note: [CMD] is a Command, [RPL] is a Reply and [EVT] is an Event.

== WRITE ==
From Master to Slave. Slave replies.
[CMD] Master: 2  Length Address Port PayloadType T Checksum
[RPL] Slave:  2  Length Address Port PayloadType Timestamp<T> Checksum      OK
[RPL] Slave:  10 Length Address Port PayloadType Timestamp<T> Checksum      ERROR
The time information contains the time when the register with Address was updated.

== READ ==
From Master to Slave. Slave replies.
[CMD] Master: 1 4      Address Port PayloadType Checksum
[RPL] Slave:  1 Length Address Port PayloadType Timestamp<T> Checksum      OK
[RPL] Slave:  9 10     Address Port PayloadType Timestamp<> Checksum       ERROR
The time information contains the time when the register with Address was read.

== EVENT ==
Always form Slave to Master.
[EVT] Slave:  3 Length Address Port PayloadType Timestamp<T> Checksum
The time information contains the time when the register with Address was read.
```

Version Control

V0.1
First draft.

V0.2
Changed Event Command to 3.

V0.3
Cleaned up document and added C code examples.
First release.

V1.0
Updating naming of the protocol fields, etc, to latest naming review.
Major release.

V1.1
Corrected [PayloadType] list on page 2.