Binary Protocol Description Standard

1.0

Rationale

When working with binary protocols the format and details of the documentation have been inconsistent and often hard to follow. The binary protocol description standard (BPDS) tries to fix this, using a standard format that is human readable, intuitive, and text based. The hope is that will improve the readability of the documentation for byte based protocols.

The format was designed to be used by humans to document byte based binary protocols in a simple to understand format. It tries to keep things simple whenever possible, and doesn't try to cover every possible use case. BPDS does not try to represent possible states, message responses, the flow of a protocol, handshaking, or how the protocol should be used. It just tries to explain the structure of a packet of data, not how to work with it.

BPDS was also designed so that a machine can parse a BPDS definition and be able to decode what bytes in a protocol are for what. It cannot interpret the meaning of a byte, only extract the structure of the packet, knowing how to recognize the start and end of the packet and label the parts.

Example

We start with an example to give an idea of how this works.

```
<Header=0xFF><Version><Prop><Cmd><Len:2><Data:Len><Footer=0x77>
```

This is a generic description of a made up protocol. You can see that it starts with a header set to the value 0xFF followed by the version. This is followed by prop that is a bit field (1 byte in size) and then the command, because there is no size provided then these are 1 byte each. The length is next and the :2 tells us that it's 2 bytes. The length is followed by the data and we can see the size is a field 'Len' so there will be 'Len' bytes of data (that we can only determine when we read a packet). The data will be followed by a footer byte that is always set to 0x77.

So you would expect to see a data stream something like:

```
0xFF 0x01 0x00 0x01 0x00 0x08 0x64 0x64 0x10 0x10 0x00 0xFF 0x00 0x00 0x77
```

Documented in Autodoc format (as a C comment):

```
/*******************************************************************************
 * NAME:
 *    Command Protocol
 *
 * SYNOPSIS:
 *    <Header=0xFF><Version><Prop><Cmd><Len:2><Data:Len><Footer=0x77>
 *
 * PARAMETERS:
 *    Header -- Marker used for resyncing.  Always 0xFF.
 *    Version -- What version of the protocol we are using.  Currently only
 *               version 1 is defined.
 *    Prop -- What optional properties are included in this packet.  This is a bit
 *            field where:
 *               0x01 -- Transparency
 *               0x02 -- Fill style
 *               0x04 -- Border
 *    Cmd -- What command we are going to be executing.  See details below
 *           for a description of available commands.
 *    Len -- The length of the 'Data'.  This does not include the 'Footer',
 *           only the data itself.  This is in big endian.
 *    Data -- The data for this command.  The size will depend on the command.
 *    Footer -- Marker used for resyncing.  Always 0x77.
 *
 * FUNCTION:
 *    This is general packet format for sending commands to the graphic
 *    processor system.
 *
 * REPLY:
 *    Replies will be an acknowledge packet.
 *
 * SEE ALSO:
 *
 ******************************************************************************/
```

As you can see when used with an Autodoc style comment it is very effective at explaining the protocol and can be included directly with the handling code.

Goals

The binary protocol description standard has a number of goals:

Human readable	AscII characters are used for the symbols that mark parts of the specification, these are easy to pick out from the text and are used extensively in programming. The field names and labels are human readable strings.
Machine readable	Computers should be able to parse a BPDS definition string to be able to act on a byte stream encoded in that format. This is useful for things like generic highlighters or error checkers.
Intuitive	Someone should be able to look at a BPDS definition string and more or less understand how to interpret the meaning without needing to read a document explaining what BPDS is or how it works.
Single line	A single line should be able to describe a packet. This keeps the description compact and allows it be added to larger documents.
Text based	Sticking with a text based description means it can be easily copied and embedded in other documents like in source code.
Byte based	Most protocols are byte-based. Supporting arbitrary bit-level protocols would significantly increase complexity and is therefore intentionally excluded.

Terminology

Field	A grouping of bytes that form an element of the protocol. This maybe a single byte or multiple bytes together. This identifies a part of the message. For example, the length would be considered a field that indicates the length in the message.
Literal	A constant value that must match this value in the byte stream.
Symbol	A byte that is recognized as having meaning in the BPDS. For example, '<' and '>' mark the start and end of a field.
Field name	The name of a field. This is alpha numeric, starting with a letter.
Label	When a size refers to a previous field name it is called a label.
Value	The value of a field when a literal is used. This maybe a number or a string.
Attribute	An attribute is a symbol that modifies a field. An example is the Size (:) attribute.
Definition	The whole BPDS string that defines a matching set of fields.

Format

Symbols

Symbol	**Name**	**Description**	**Value**
<>	Field	Marks the start and end of a field. A field group multiple bytes together and marks an element in the protocol.	0x3C 0x3E
Literal	Literal value	This byte does not have a name and is just the literal value. If this is a number can uses C number prefixes (0x for hex, 0 for octal, etc) or a string surrounded by quotes (").	
=	Assigned value	The field will be have this value (or set of values). This is the same as a literal but comes after the name of the field.	0x3D
|	OR	Can only be used with values. When you want to use a set of values instead of just one you place an | between the values and it counts as this value or this other value.	0x7C
:	Size	The number of bytes this field is (if not provided defaults to 1, so "cmd" and "cmd:1" are the same).	0x3A
...	Variable size	This is only used as a 'Size' with the size symbol (:). It matches any number of bytes until it finds a match for the next field. This size can match 0 bytes. You must have some kind of terminating field following this as otherwise there is no way to end the stream.	0x2E 0x2E 0x2E
"	Quote	A quote that marks the start and end of a string value. Used with	0x22

Symbol	Name	Description	Value
		string literals.	

Fields

A field starts with the '<' symbol and ends with the '>' symbol. A field is a group of bytes together and makes up a base element in the protocol. A field includes information about the group such as size, literal values, and type information.

If the field is a literal then it is just the literal and does not use any other attributes expect the OR (|) attribute. The literal can be a number or a string (but not both). If it's a number then it uses C number prefixes (0x for hex, 0 for octal, etc) and can be any number of bytes (although going over 8 bytes (64 bit) might make parsers fail).

If it's a string then it will have quotes around it and will have a size the same as the string length. For example, <"Dog"|"Fish"> will match a 3 byte string or a 4 byte string.

If the field is not a literal then it starts with the name of the field followed by any attributes. For example, <Start>, <Start:2>, <Start=0xFF>.

Examples

<0x55>	A literal that must be 55 hex.
<0x55|0xAA>	A literal that must be 55 hex OR AA hex.
<32>	A literal that must the 32 decimal.
<"Cat">	A literal that must match 0x43, 0x61, and 0x74
<"Cat"|"Dog">	A literal that must match 0x43, 0x61, and 0x74 OR 0x44 0x6F and 0x67
<Start>	A field with the name of "Start". It can be any 1 byte value (the value doesn't mater only that it is 1 byte long).
<Start:2>	A field with the name "Start" that is 2 bytes in length. The value doesn't mater, just that it is 2 bytes in length.
<Start=0x55>	A field with the name "Start" that is 1 byte long and must be the value 55 Hex.

Literal Value

A literal value is a constant value that must match. These are numbers or strings.

Numbers use C language literal number prefixes. Supported prefixes:

- Decimal (base 10) -- No prefix is used. The number starts with a non-zero digit. Example 123, 45, etc.
- Octal (base 8) -- The number is prefixed with a single zero (0). Example 076, 012, etc.
- Hexadecimal (base 16) -- The number is prefixed with 0x or 0X. Example 0x2A, 0XFf, etc.

- Binary (base 2) -- The number is prefixed with 0b or 0B. Example 0b1010, 0B10, etc.

Strings are wrapped in quotes. Quotes nest, so this means that quotes are counted, there must always a matching end quote for every quote in the literal. There is no need to escape the quotes inside literals.

Examples

<0xFF>	Must match the value 255
<0xFF|0xEE>	Can match 0xFF OR 0xEE
<"Hello">	Must match 0x48 0x65 0x6c 0x6c 0x6f
<"Hello"|"Bye">	Can match 0x48 0x65 0x6c 0x6c 0x6f, OR 0x42 0x79 0x65
<"Nested"quotes" Here">	Must match 0x4e 0x65 0x73 0x74 0x65 0x64 0x22 0x71 0x75 0x6f 0x74 0x65 0x73 0x22 0x48 0x65 0x72 0x65

Assigned value

The assigned value is the same as a Literal value but with a field name. The literal value is after a equal sign (=) and follows the same rules as a literal.

Examples

<Name=0xFF>	Field has the name "Name" as must match the value 0xFF
<Start=0xFF|0xEE>	Field has the name "Start" and can match 0xFF or 0xEE
<Command="Hello"|"Bye">	Field has the name "Command" and can match 0x48 0x65 0x6c 0x6c 0x6f, OR 0x42 0x79 0x65

OR

The OR symbol (|) is used to say any literal from a set of literal can be a match. These can be numbers or strings (but they cannot be mixed). You list all the values you wish to accept with a pipe bar between them. This is valid in assigned values and literal values.

Examples

<0x55|0xAA|0x00>	A literal that must be 55 hex OR AA hex OR 00 hex.
<Start=0xFF|0xEE>	Field has the name "Start" and can match 0xFF OR 0xEE
<Command="Hello"|"Bye">	Field has the name "Command" and can match 0x48 0x65 0x6c 0x6c 0x6f, OR 0x42 0x79 0x65

Size

The size symbol tells you how many bytes this field uses. The size symbol uses the colon (:) and must follow a field name. If the size symbol is not provided then the size of the field will be 1 byte.

This can also be the field name of a previous field. In this case what is being stated is that this field is variable length and the number of bytes to expect comes from this previous field (label).

This can also be set to variable size (...) in which case it means that the size of this field is variable and may be between 0 and unlimited. The field is terminated by the next field. So for example, if a size is variable size and the next field is a literal 0x0A then all the bytes between this point and the 0x0A fit into this field. See variable size below for more info.

Examples

<Len:2>	The length is 2 bytes
<Data:32>	This field is 32 bytes long
<Start:2=0xDEAD>	The field "Start" is 2 bytes in size and must match the value 0xDEAD
<Other:3="Cat">	The field "Other" is 3 bytes and must match 0x43 0x61 0x74
<More:Prev>	The field "More" uses the value from the previous "Prev" field.

Variable Size

The variable size symbol (...) means match all bytes until the next field is satisfied.

For example, if you have <Data:...><0x0A> this matches all chars until a 0x0A (new line) char is found (the new line will not be part of 'Data'). So this will match (\n = 0x0A):

Stream	**Data Field**	**0x0A Field**	**Description**
Test\n	Test	0x0A	The string "Test" will be in data
A long string\n	A long string	0x0A	The string "A long string" will be in data
\n		0x0A	A blank string will be in data

If the next field is more than 1 byte then all the bytes have to match and will not be part of the field using the variable size symbol.

Examples

<Data:...><0x00>	A zero terminated string
<CmdNum:...><EndOfCmd="END">	A string that must end is the string "END". So 0x31 0x32 0x45 0x4E 0x44 would end up with a CmdNum field = to "12".
<Comment:...><EndOfComment=".">	A comment that ends with a period.
<0xFF><Cmd><Data:2><Note:...><0x00><0x77>	A longer definition with a 'Note' field that is variable size terminated by a NULL (0x00)

	char.

Endian

This standard does not set an endian, you must provide this information in your documentation. Providing an endian symbol would not be intuitive as there isn't a widely known symbol for providing this information.

Reserved Symbols

The following symbols are reserved for future use. Math symbols have not being used for any of the known symbols (these symbols are "+", "-", "/", and "*"). This is keep the option of adding math blocks in the future. It is not clear if adding math would be a good idea or not, so the symbols have been listed as reserved.

Tips

Optional fields

BPDS does not include optional fields, this is because for optional fields there needs context and knowledge of the meaning of the bytes which is out of scope for the BPDS. However you can handle optional fields by using additional BPDS definitions. For example, for a single optional field you can write two BPDS definition. One with the optional field in it and a second version with it missing. You then document the condition in the Autodoc for each definition (see the examples).

Autodoc

This is a blank Autodoc you can use if you decide to use this format for your details documentation.

```
/*******************************************************************************
 * NAME:
 *
 *
 * SYNOPSIS:
 *
 *
 * PARAMETERS:
 *
 *
 * FUNCTION:
 *
 *
 * REPLY:
 *
 *
 * SEE ALSO:
 *
 ******************************************************************************/
```

More Examples

These are more examples of the made up protocol. It shows optional fields, different fields based on command type, and a number of other details.

This is an example of the reply from the draw command described in the "Example" sections above.

```
/*******************************************************************************
 * NAME:
 *    Command reply
 *
 * SYNOPSIS:
 *    <Header=0xFF><Version><Prop=0><Cmd=0xFF><Len:2=0><Footer=0x77>
 *
 * PARAMETERS:
 *    Header -- Part of standard header
 *    Version -- Part of standard header
 *    Prop -- Must be 0x00
 *    Cmd -- 0xFF - ack to last command
 *    Len -- 0x00
 *    Footer -- Part of standard header
 *
 * FUNCTION:
 *    This command is an ack reply to a draw command.
 *
 * REPLY:
 *    NONE
 *
 * SEE ALSO:
 *    Command Protocol, Properties
 ******************************************************************************/
```

This is an example of an optional field. This documents the 'Prop' field and all the options that can used. It provides only the details of the 'Prop' field but does show it in relation to the other fields (the details for the other fields are found elsewhere).

```
/*******************************************************************************
 * NAME:
 *    Properties
 *
 * SYNOPSIS:
 *    <Header=0xFF><Version><Prop><Transparency:4><FillStyle:4><Border>
 *    <Cmd><Len:2><Data:Len><Footer=0x77>
 *
 * PARAMETERS:
 *    Header -- Part of standard header.
 *    Version -- Part of standard header.
 *    Prop -- Props:
 *              Bit     Meaning
 *              0x01    Transparency
 *              0x02    Fill style
 *              0x04    Border thickness
 *    Transparency -- This tells the drawing function how much transparency to
 *                    apply to the draw command.  0 = Fully opaque, 100.0 =
 *                    fully transparency.
 *    FillStyle -- What fill style to use:
 *                      0x00 -- Solid
 *                      0x01 -- Dashed
 *                      0x02 -- Dotted
 *    Border -- How many pixels to make the border
 *    Cmd -- Part of standard header.
 *    Len -- Part of standard header.
 *    Data -- Part of standard header.
 *    Footer -- Part of standard header.
 *
 * FUNCTION:
 *    These are the properties that can be applied.  If the bit is set in
 *    'Prop' then you need to provide the corresponding fields.
 *
 * REPLY:
 *    Depends on 'Cmd'
 *
 * SEE ALSO:
 *    Command Protocol
 ******************************************************************************/
```

This is an example of a draw command. It fills in the details of the <data> field when used with the draw box command.

```
/*******************************************************************************
 * NAME:
 *    Draw box
 *
 * SYNOPSIS:
 *    <Header=0xFF><Version><Prop><Cmd=0x01><Len:2=8><x:2><y:2><width:2>
 *    <height:2><Footer=0x77>
 *
 * PARAMETERS:
 *    Header -- Part of standard header
 *    Version -- Part of standard header
 *    Prop -- Part of standard header
 *    Cmd -- Draw a box (0x01)
 *    Len -- Part of standard header.  Set to 8
 *    x -- The x point on screen for the box
 *    y -- The y point on screen for the box
 *    width -- The number of pixels for the width of the box
 *    height -- The number of pixels for the height of the box
 *    Footer -- Part of standard header
 *
 * FUNCTION:
 *    This command draws a box on the display.
 *
 * REPLY:
 *    Replies will an acknowledge packet.
 *
 * SEE ALSO:
 *    Command Protocol, Properties, Command reply
 ******************************************************************************/
```

This is another example of a different draw command. It shows a draw bitmap command that uses an embedded .gif file.

```
/*******************************************************************************
 * NAME:
 *    Draw Bitmap .gif
 *
 * SYNOPSIS:
 *    <Header=0xFF><Version><Prop><Cmd=0x02><Len:2><x:2><y:2><GraphicData>
 *    <Footer=0x77>
 *
 * PARAMETERS:
 *    Header -- Part of standard header
 *    Version -- Part of standard header.
 *    Prop -- Part of standard header.
 *    Cmd -- Draw a bitmap stored in gif format (0x02)
 *    Len -- Part of standard header.  The length of the 'GraphicData' + 4
 *    x -- The x point on screen for the graphic
 *    y -- The y point on screen for the graphic
 *    GraphicData -- the binary data for a .gif file.  This includes the
 *                   headers and bitmap data (basically the whole .gif file).
 *    Footer -- Part of standard header
 *
 * FUNCTION:
 *    This command draws a bitmap to the display.  The bitmap is stored as
 *    a gif.
 *
 * REPLY:
 *    Replies will an acknowledge packet.
 *
 * SEE ALSO:
 *    Command Protocol, Properties, Command reply
 ******************************************************************************/
```

License