**A**rduino **S**ervice **I**nterface **P**rotocol Reference Outline

Michael Margolis

Document version 1

Last updated 7 May 2015

This document outlines the ASIP command and event message protocol

1. Definition of the format
2. Messages consist of an ASCII header, ASCII character fields separated by commas, terminated by the newline character.
3. Request messages to the server begin with a single character to indicate the desired service followed by a comma and a single character tag to identify the nature of the request.
4. Requests that contain a parameter are separated from the tag with a comma.
5. Replay messages from the server begin with one of the following characters:
   1. "@" An event message responding to a request or autoevent. These messages have three bytes following the ‘@’ character, a character indicating the service, a comma, and the tag indicating the request that triggered this event
   2. "~" An error message reporting an ill formed request or some other problem affecting the server. These messages contain the service and tag associated with the error followed by an error number and error string.
   3. "!" An informational or debug message consisting of unformatted ASCII text terminated by the newline character.
6. Some reply event messages have a payload with a variable number of fields with the following format:
   1. A numeric value precedes the message body indicating the number of fields in the body.
   2. Curley brackets are used to indicate the start and end of fields in the message body
   3. If a message contains sub fields, these are separated by a colon (see analog pin mapping for an example)
   4. All numeric values are expressed as ASCII text digits and are decimal unless otherwise stated.

**System Messages from client**

**Get Version info**

Request:

|  |  |  |  |
| --- | --- | --- | --- |
| **Header** | **Separator** | **Tag** | **Terminator** |
| ‘#’ | , | ‘?’ | ‘\n’ |

Reply:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Header** | **Tag** | **Major**  **Version** | **Minor Version** | **Micro**  **controller** | **Pins** | **Sketch**  **Name** | **Terminator** |
| ‘@#’ | ‘?’ | Numeric digits | Numeric Digits | Text | Numeric Digits | Text | ‘\n’ |

All fields following header are separated by a comma

Example:

Request: "#,?\n"

Reply: "@#,?,0,2,ATmega328P,20,TestIO\n"

In this example, ASIP version 0.2 is running on an ATmega328P with 20 pins using a sketch named TestIO:

**Get Services on Pins**

Request:

|  |  |  |  |
| --- | --- | --- | --- |
| **Header** | **Separator** | **Tag** | **Terminator** |
| ‘#’ | , | ‘S’ | ‘\n’ |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Header** | **Tag** | **Pin**  **Count** | **Start of**  **Body** | **Body** | **End of**  **Body** | **Terminator** |
| ‘@#’ | ‘S’ | Numeric digits | ‘{‘ | Comma separated characters indicating service ID | ‘}’ | ‘\n’ |

Reply:

All fields following header are separated by a comma. Pin count indicates the number of pins on the server. The body will contain a character indicting the service corresponding to a pin for each pin from 0 to pin count-1.

Example:

Request: "#,S\n"

Reply: "@#,S,20,{@,@,I,I,I,I,I,I,I,I,I,I,I,I,I,I,I,I,S,D}\n"

In this example, pins 0 and 1 are reserved (by the serial connection to the client), pin 18 is used by the servo service, pin 19 is used by the distance service. All other pins are available for use by the I/O service.

**Get Service Names**

Request:

|  |  |  |  |
| --- | --- | --- | --- |
| **Header** | **Separator** | **Tag** | **Terminator** |
| ‘#’ | , | ‘N’ | ‘\n’ |

Reply:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Header** | **Tag** | **Pin**  **Count** | **Start of**  **Body** | **Body** | **End of**  **Body** | **Terminator** |
| ‘@#’ | ‘N’ | Numeric digits | ‘{‘ | Comma separated pairs of colon separated values indicating service ID character and service name | ‘}’ | ‘\n’ |

Example:

Request: "#,N\n"

Reply: "@#,N,3,{I:ASIP core IO,S:Servos,D:Distance}\n"

Three services, ‘I’ is the id for the ASIP core IO, ‘S’ is the service id for Servos , ‘D’ is the id for the distance sensor service.

**Get Pin Services List** *(TODO)*

**System Events to client**

**Error Messages**

These messages report an ill formed request or some other problem affecting the server. These messages contain the service and tag associated with the error followed by an error number and error string.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Header** | **Service id related**  **to this error** | **Request tag**  **related to error** | **Error**  **number** | **Error Text** | **Terminator** |
| ‘~’ | A valid service id | A valid svc tag | Text digits | Text string describing  error | ‘\n’ |

|  |  |
| --- | --- |
| **Error type** | **Number** |
| NO\_ERROR | 0 |
| INVALID\_SERVICE | 1 |
| UNKNOWN\_REQUEST | 2 |
| INVALID\_PIN | 3 |
| MODE\_UNAVAILABLE | 4 |
| INVALID\_MODE | 5 |
| WRONG\_MODE | 6 |
| INVALID\_DEVICE\_NUMBER | 7 |
| DEVICE\_NOT\_AVAILABLE | 8 |
| I2C\_NOT\_ENABLED | 9 |

Error messages begin with a tilde (~) followed by the service ID, a comma, and the tag associated with the error (typically this is the tag in the request that caused the error). The body of the error message is one of the above numeric error codes followed by a brief text description of the error.

**Informational Messages**

|  |  |  |
| --- | --- | --- |
| **Header** | **Message text** | **Terminator** |
| ‘~’ | ASCII character string | ‘\n’ |

An informational or debug message begins with the tilde character and consists of unformatted ASCII text terminated by the newline character.

**IO Service Messages from client**

**Get Port to Pin Mapping**

The mapping of analog pin numbers to digital pin numbers can be queried by issuing the following request

Request:

|  |  |  |  |
| --- | --- | --- | --- |
| **Header** | **Separator** | **Tag** | **Terminator** |
| ‘I ’ | , | ‘M’ | ‘\n’ |

Reply:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Header** | **Tag** | **Pin**  **Count** | **Start of**  **Body** | **Body** | **End of**  **Body** | **Terminator** |
| ‘@I ’ | ‘M’ | Numeric digits | ‘{‘ | Comma separated pairs of colon separated hexadecimal values indicating port number and bit mask. | ‘}’ | ‘\n’ |

Example: A board with a 328 chip such as the Uno

Request: "I,M\n"

Reply: @I,M,20,{4:1,4:2,4:4,4:8,4:10,4:20,4:40,4:80,2:1,2:2,2:4,2:8,

2:10,2:20,3:1,3:2,3:4,3:8,3:10,3:20}\n"

This indicates that there are 20 pins. The colon separated pairs in braces indicate the port number and bit mask associated with each pin. In this example, the state of pin 0 is obtained by masking port 4 with 1, pin 1 is obtained by masking port 4 with 2, pin 19 is port 3 masked with 0x20. Note that the port and mask values for this message are hexadecimal (All other ASIP fields use decimal values).

**Get Analog Pin Mapping**

Provides the mapping of analog pin numbers to digital pin numbers

Request:

|  |  |  |  |
| --- | --- | --- | --- |
| **Header** | **Separator** | **Tag** | **Terminator** |
| ‘I ’ | , | ‘m’ | ‘\n’ |

Reply:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Header** | **Tag** | **Pin**  **Count** | **Start of**  **Body** | **Body** | | **End of**  **Body** | **Terminator** |
| ‘@I ’ | ‘m’ | Numeric digits | ‘{‘ | Comma separated pairs of colon separated values indicating digital pin number followed by the analog pin number | ‘}’ | | ‘\n’ |

Example: A board with a 328 chip such as the Uno

Request: "I,m\n"

Reply: "@I,m,6,{14:0,15:1,16:2,17:3,18:4,19:5}\n”

This indicates that there are 6 analog pins, digital pin 14 is mapped to analog pin 0, digital pin 15 to analog pin 1 etc.

**Get Pin Modes**

Provides a list of comma separated values indicating the current mode of each pin.

Request:

|  |  |  |  |
| --- | --- | --- | --- |
| **Header** | **Separator** | **Tag** | **Terminator** |
| ‘I ’ | , | ‘p’ | ‘\n’ |

Reply:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Header** | **Tag** | **Pin**  **Count** | **Start of**  **Body** | **Body** | **End of**  **Body** | **Terminator** |
| ‘@I ’ | ‘p’ | Numeric digits | ‘{‘ | Comma separated value indicating the mode of each pin | ‘}’ | ‘\n’ |

|  |  |  |
| --- | --- | --- |
| **Mode** | **Value** | **Comment** |
| UNALLOCATED\_PIN\_MODE | 0 | Pin has not been allocated to a service |
| INPUT\_MODE | 1 | digitial input |
| INPUT\_PULLUP\_MODE | 2 | digital input with pull-up resistors enabled |
| OUTPUT\_MODE | 3 | digital output |
| ANALOG\_MODE | 4 | analog input |
| PWM\_MODE | 5 | pwm output (analogWrite |
| RESERVED\_MODE | 6 | pin is used by the server (typically for serial communication) |
| OTHER\_SERVICE\_MODE | 7 | pin is in use by a service |
| INVALID\_MODE | 8 | pin is not valid |

Example:

Request: "I,p\n"

Reply: "@I,p,20,{6,6,0,0,7,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0}\n"

In this example, pins 0 & 1 are reserved (they are used for serial communication), and pins 4 & 5 are reserved by a service running on the server, the other pins are all available

**Get Pin Capabilities**

Enables the capability of each pin to be queried

Request:

|  |  |  |  |
| --- | --- | --- | --- |
| **Header** | **Separator** | **Tag** | **Terminator** |
| ‘I ’ | , | ‘c’ | ‘\n’ |

Reply:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Header** | **Tag** | **Pin**  **Count** | **Start of**  **Body** | **Body** | **End of**  **Body** | **Terminator** |
| ‘@I ’ | ‘c’ | Numeric digits | ‘{‘ | Comma separated bit mask indicating the capability of each pin | ‘}’ | ‘\n’ |

|  |  |
| --- | --- |
| **Pin Capability** | **Mask bit** |
| DIGITAL\_IO | 1 |
| ANALOG\_INPUT | 2 |
| PWM\_OUTPUT | 4 |

Example: An Uno board (or any board with a 328 chip)

Request: "I,c\n"

Reply : "@I,c,20,{1,1,1,5,1,5,5,1,1,5,5,5,1,1,3,3,3,3,3,3}\n"

This indicates that there are 20 pins, all with digital IO capability, pins 3,5,6,9,10,11 are PWM capable, and pins 14 through 19 have analog input capability. The values are derived from a bitfield defined in asip.h:

**Set Autoevent Period**

Sets the period in milliseconds for events to be reported to the client. A period of zero will turn off autoevents for the given service.

Request:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Header** | **Separator** | **Tag** | **Period** | **Terminator** |
| ‘I ’ | , | ‘A’ | Numeric value in milliseconds | ‘\n’ |

Reply:

The format of the autoevent reply messages are service specific. The following is the format for autoevents from the core IO service

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Header** | **Tag** | **Analog Pin**  **Count** | **Start of**  **Body** | **Body** | **End of**  **Body** | **Terminator** |
| ‘@I ’ | ‘A’ | Numeric digits | ‘{‘ | Comma separated pairs of colon separated values indicating analog pin number followed by the analog reading on that pin | ‘}’ | ‘\n’ |

Example: request selected analog pin data every 20 milliseconds

Request: "I,A,20\n"

Reply: (TODO)

Example turn off scheduled messages:

Request: "I,A,0\n"

Reply: none – no autoevent messages will be sent for the I/O service after this request is received

**Digital Port Data**

Change of state of all pins set as digital input is reported using this message. No explicit request is required, setting a pin to digital input automatically enables its status to be reported.

Request: none

Reply:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Header** | **Tag** | **Port** | **Port bit mask** | **Terminator** |
| ‘@I ’ | ‘d’ | Numeric digits | Hex bit mask indicating state of all digital input pins on this port | ‘\n’ |

Example: todo