

USING THE AMULET STARTER KIT WITH A PARALLAX BASIC STAMP AS MASTER

INTRO

The Amulet Starter Kit not only works well for rapid prototyping, but also fast product implementation and easy, seamless development. In combination with another microprocessor and a peripheral I/O device, the Amulet GUI is a powerful technology. In this experiment, we will be using a Basic Stamp 2 as the externally interfaced microprocessor (because of its popularity and wide spread use in the embedded industry) with a Dallas Semiconductor DS1620 Digital Thermometer. The DS1620 is not particularly relevant to the basic communication between the Amulet GUI and the Stamp, it is used as an example of an application to execute the commands sent by the Amulet. This app note will specifically highlight the hardware setup and software communications protocol, not the user interface created to demo this experiment. It is set up with the Stamp acting as master and the Amulet as the slave.

HARDWARE CONNECTIONS

The easiest way to set up this system is to use a Basic Stamp Activity Board with the Basic Stamp (preferably a Basic Stamp 2 or faster) to talk to the Amulet module via a RS232 (serial) cable. This setup allows for easy connection of the DS1620 and Activity Board to a PC for debugging and programming in addition to connecting to the Amulet unit once the Stamp has been programmed. The schematic for the connection between the Stamp and the board is shown below.

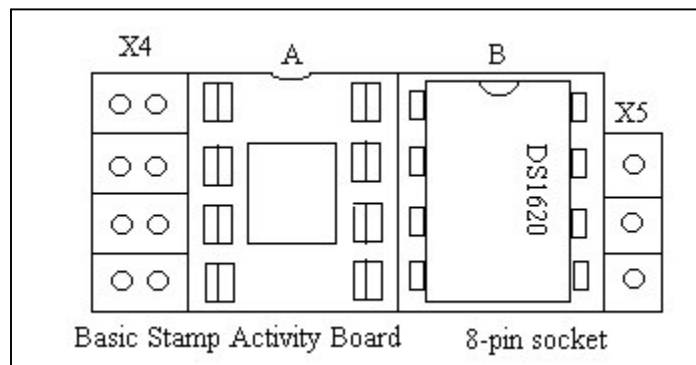


Figure 1. Schematic of Stamp connection to the DS1620 Digital Thermometer

AMULET PROTOCOL (taken from <http://www.amulettechnologies.com/>)

The communications protocol is half-duplex, with the Stamp acting as master. The Stamp can send fourteen different types of messages:

- A "Get Internal RAM byte variable" request.
- A "Get Internal RAM word variable" request. (word = 2 bytes)
- A "Get Internal RAM string variable" request.
- A "Get Internal RAM RPC buffer" request.
- A "Set Internal RAM byte variable" command.
- A "Set Internal RAM word variable" command.
- A "Set Internal RAM string variable" command.
- A "Set Internal RAM byte variable array" command.
- A "Set Internal RAM word variable array" command.
- A "Draw line" command.
- A "Draw rectangle" command.
- A "Draw filled rectangle" command.
- A "Draw pixel" command.
- A "Jump to specific page" command.

If the message is valid, the Amulet should either return the requested data (if a "Get" request) or acknowledge the message (if an "Invoke" or "Set" command). If the Amulet receives an invalid message, it will not respond at all.

Tables 1 and 2 in [Appendix A](#) define the fourteen types of messages that can be sent between the Stamp and the Amulet. With the Stamp acting as Master, the RPC flags sent can be one of two values: 0x00 for the Amulet to send all the RPCs in the buffer and then flush it or 0xFF to flush the buffer. The valid range for byte variable values returned by the Amulet (in response to the "Get Byte variable" request) is 0-0xFF. The valid range for word variable values returned from the Amulet (in response to the "Get Word variable" request) is 0-0xFFFF. String and label variable values returned from the Amulet can have a maximum of 252 ASCII (0x20-0x7E) characters plus a null termination character (0x00). The length of the variable array used to store the stream of data (see code example below) should be large enough to hold the largest message the Amulet will be sending for that program.

The Amulet uses an ASCII protocol. This means that the variable 0x1A would be transmitted as 0x31, 0x41, where 0x31 is the ASCII representation of the high nibble "1" and 0x41 is the ASCII representation of the low nibble "A".

NOTE: The Amulet will respond to every Stamp command by echoing the command if a "Set" or "Invoke" and echoing the command, followed by the requested data if a "Get" command. In order to configure the Amulet to respond to any "Set" or "Invoke" commands with a simple acknowledge (0xF0), use the "SlaveAckRsp" Meta tag (<META NAME="Amulet" Content="SlaveAckRsp.project">). To configure the

Amulet to not respond at all, use the “SlaveNoRsp” Meta tag (<META NAME="Amulet" Content="SlaveNoRsp.project">).

Synchronization--As master, the Stamp initiates all communications by sending a message to the Amulet. All valid messages from the Stamp to the Amulet start with one of fourteen command bytes: [0xA0], [0xD0], [0xD1], [0xD2], [0xD4], [0xD5], [0xD6], [0xD7], [0xD9], [0xDA], [0xDB], [0xDC], [0xDF], or [0xF2] -- these are considered the Client Start Of Message (CSOM) characters, with the stamp being the client. **NOTE:** These fourteen CSOM bytes ALWAYS signify the start of a message and they are not allowed in the body of any message. The only valid characters in the body of a message are: ASCII 0-9 (0x31-0x39), and A-F (0x41-0x46), except in the body of the “Get string” and “Set string” responses, where all ASCII characters from ‘ ‘ through '~' (0x20-0x7E) are valid. If the Amulet receives any character other than those specified, the message will be considered errant, and the Amulet will start over hunting for a new CSOM character.

All Amulet responses will start with the counterpart of the CSOM character that began the message that is being responded to. The valid Server (Amulet) Start Of Message (SSOM) bytes are: [0xE0], [0xE1], [0xE2], [0xE4], [0xE5], [0xE6], [0xE7], [0xE9], [0xEA], [0xEB], [0xEC], [0xEF], or [0xF3]. The body of the response message starts with the counterpart SSOM and is then followed by any optional response data (in ASCII format).

As noted earlier, the Amulet will echo the message and then send any data requested by the command. To have the Amulet instead send an ‘acknowledge’, or no response at all, use the appropriate “SlaveAckRsp” or “SlaveNoRsp” Meta tags.

The Amulet should be set to a baud rate of 9600. The Amulet also needs to null terminate each response so that the Stamp can determine where to interpret the end of each response. The BASICStamp Meta tag should be used to both null terminate each message and set the communication protocol to an appropriate interbyte delay. The Amulet’s BASICStamp Meta tags are as follows:

```
<META NAME="Amulet" Content="BASICStamp.project">  
<META NAME="Amulet" Content="baud.project=9600">
```

SOFTWARE INTERFACE

An abbreviated version of the communication protocol used to make the Amulet, Stamp, and thermometer communicate is shown below. The software used to program the Stamp can be found at the following link:

http://www.parallax.com/html_pages/downloads/software/software_basic_stamp.asp

The following code snippets were taken from an actual implementation based on the Basic Stamp 2 micro controller. The code consists of a main loop that calls subroutines

associated with each possible command the Stamp can send out. The loop can call as few or as many of the subroutines as needed. It assumes that the Amulet is connected and has been initiated using the BASIC Stamp, 9600, and SlaveAckRsp Meta tags. The entire code can be found in [Appendix B](#).

```
serial_out:
```

```
'*****
'* This function executes an example of every possible command. Of the Stamp
'* Message sent out, the command/request is the first byte, the HIGH nibble of
'* the variable being used is always the second, and the LOW nibble of the variable
'* being used is always the third. Some Stamp messages will have more than three
'* bytes. Those are dealt with specifically in the section of code pertaining
'* to that command. Using the BASICStamp Meta in your html page automatically
'* null terminates every answer sent by the Amulet.
'* The first segment of this code calls the name of the subroutine for the
'* commands the code will invoke (currently it is set to invoke an example of
'* each type of command). Consequently, this code is set up to exist solely as
'* an example of the implementation of the Stamp as Master. This code assumes
'* that the Amulet has been set to SlaveAckRsp using a Meta tag, so that
'* after every valid 'set' command it responds with an ACK (F0)
'*****
```

```
GOSUB getByte          ' jumps to subroutine pertaining to GOSUB command.
GOSUB getWord         ' Depending on the needs of your program the
GOSUB getString       ' needed command code could be pasted directly
GOSUB getRPC          ' into the main function of the program or,
GOSUB setByte         ' if is used multiple times, kept as a
GOSUB setWord         ' subroutine and return
GOSUB setString
GOSUB setByteArr
GOSUB setWordArr
GOSUB drawLine        ' For a more in-depth explanation of the
GOSUB drawRect        ' graphic primitives and jump page instruction,
GOSUB drawFilledRect  ' see the Amulet online help documentation
GOSUB drawPixel       ' on UART protocol and graphic primitives
GOSUB jumpPage
```

```
GOTO serial_out
```

```
'*****
'* Handler for a getByte command
'* Format of command get byte = 0xD0xx where xx=the variable byte being used
'* Amulet returns 0xE0xxNN, where NN equals the HEX data of the variable xx
'*****
```

```
getByte:
Variable = $00          ' byte number to be requested - 0 in this example
SEROUT 16, 84, [208,HEX2 Variable] ' send 'get byte' command. 208 is D0 in decimal
SERIN 16, 84, 3000, getByte, [STR AmuletMsg \5] ' wait for five bytes of response.
' if Amulet doesn't respond in 3 sec, send message again
B0.HIGHNIB = AmuletMsg(3) ' store the two bytes of the value sent by the
B0.LOWNIB = AmuletMsg(4) 'Amulet into byte 0
RETURN
```

```
'*****
'* Handler for a getRPC command
'* Format of command get RPC = 0xD4xx, where xx = index of label variable
'* Returns 0xE4xx to Stamp
'* Returns requested data back to the screen
'*****
```

```

getRPC:
  SEROUT 16, 84, [212,"00"]          ' RPC flag 0x00 sent in this example which
                                     ' requests all the stored RPCs
  SERIN 16, 84, 3000, getRPC, [STR AmuletMsg \9 \0] ' wait for either 3RPC's or null
  ' terminator. If it doesn't happen in 3 sec, resend the command
  FOR Counter = 3 TO 7 STEP 2        ' go through each RPC received
    Variable.HIGHNIB = AmuletMsg(Counter) ' Store RPC in variable
    Variable.LOWNIB = AmuletMsg(Counter + 1)
    IF Variable = 1 THEN RPCOne      ' execute code for current RPC
    IF Variable = 2 THEN RPCTwo
    IF Variable = 3 THEN RPCThree
    IF Variable = 4 THEN RPCFour
  incRPC: NEXT
  GOTO doneRPC

RPCOne:
  ' if RPC 1 comes back then execute this code
  DEBUG "RPC 1 executing"
  HIGH RST                          ' activate the DS1620
  SHIFTOUT DQ,CLK,LSBFIRST, [WhiT]  ' set the "max" temp before the AC kicks
  SHIFTOUT DQ,CLK,LSBFIRST, [%000111100] ' in to 30.0 degrees C
  LOW RST                            ' deactivate the DS1620
  PAUSE 50                           ' pause to give EEPROM time to program
  GOTO incRPC

RPCTwo:
  ' if RPC 2 comes back then execute this code
  DEBUG "RPC 2 executing"
  HIGH RST                          ' activate the DS1620
  SHIFTOUT DQ,CLK,LSBFIRST, [WloT]  ' set the "min" temp before the heater
  SHIFTOUT DQ,CLK,LSBFIRST, [%000100100] ' kicks in to 18 degrees C
  LOW RST                            ' deactivate the DS1620
  PAUSE 50                           ' pause to give EEPROM time to program
  GOTO incRPC

RPCThree:
  ' if RPC 3 comes back then execute this code
  DEBUG "RPC 3 executing"
  GOTO incRPC

RPCFour:
  ' if RPC 4 comes back then execute this code
  DEBUG "RPC 4 executing"
  GOTO incRPC

doneRPC: RETURN

'*****
'* Handler for a setByte command
'* Format of command string = 0xD5xxNN, where xx = variable to be set
'* and NN = HEX data
'* Returns 0xF0 (ACK) to Stamp
'*****

setByte:
  HIGH RST                          ' Activate the '1620
  SHIFTOUT DQ,CLK,LSBFIRST, [Rtemp] ' Request to read temperature
  SHIFTOUT DQ,CLK,LSBPRE, [DSdata\9] ' Get the temperature reading
  LOW RST                            ' Deactivate
  DSdata = DSdata/2
  SEROUT 16, 84, [213,"00",HEX DSdata] ' send out command to set byte 0
                                     ' to temperature reading in Celsius
  SERIN 16, 84, 3000, setByte, [STR AmuletMsg \1] ' accept the Amulet's response
  Variable = AmuletMsg
  IF Variable <> $F0 THEN setByte      ' if wasn't 'acknowledge', resend
  RETURN

```

```

'*****
'* Handler for a drawLine command
'* Format of string = 0xD9XXXXYYYYxxxxyyyyPW, where
'* XXXX = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* x-coordinate of first endpoint
'* YYYY = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* y-coordinate of first endpoint
'* xxxx = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* x-coord of second endpoint
'* yyyy = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* y-coordinate of second endpoint
'* P = line Pattern
'* W = Line weight
'* Returns 0xF0 (ACK) to Stamp
'*****

drawLine:
' The following SEROUT tells the Amulet to draw a line from (20,30) to (300,180)
' with a line pattern of 0 (black) and a line width of 4
SEROUT 16, 84, [217,"0014","001E","012C","00B4","0","4"]
SERIN 16, 84, 3000, drawLine, [STR AmuletMsg \1] ' accept the Amulet's response
Variable = AmuletMsg
IF Variable <> $F0 THEN drawLine ' if did not get ACK, resend
RETURN

```



Amulet Technologies, LLC

GUI Engines for Embedded Systems

Appendix A

Message	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte N
Stamp Get Byte Variable	0xD0	Variable Hi Nibble	Variable Lo Nibble	None	None	None	None	None
Amulet Response	0xE0	Variable Hi Nibble	Variable Lo Nibble	Value Hi Nibble	Value Lo Nibble	None	None	None
<hr/>								
Stamp Get Word Variable	0xD1	Variable Hi Nibble	Variable Lo Nibble	None	None	None	None	None
Amulet Response	0xE1	Variable Hi Nibble	Variable Lo Nibble	----MS Byte---- Value Hi Value Hi Hi Nibble Lo Nibble		----LS Byte---- Value Lo Value Lo Hi Nibble Lo Nibble		None
<hr/>								
Stamp Get String Variable	0xD2	Variable Hi Nibble	Variable Lo Nibble	None	None	None	None	None
Amulet Response	0xE2	Variable Hi Nibble	Variable Lo Nibble	ASCII char	ASCII char	ASCII char	ASCII 0x00 char	
<hr/>								
Stamp Get Remote Procedure Calls (RPC)	0xD4	RPC flag Hi Nibble	RPC flag Lo Nibble	None	None	None	None	None
Amulet Response	0xE4	RPC flag Hi Nibble	RPC flag Lo Nibble	RPC #1 Hi Nibble	RPC #1 Lo Nibble	RPC #2 Hi Nibble	RPC #2 0x00 Lo Nibble	
<hr/>								
Stamp Set Byte Variable	0xD5	Variable Hi Nibble	Variable Lo Nibble	Value Hi Nibble	Value Lo Nibble	None	None	None
Amulet Response	0xE5	Variable Hi Nibble	Variable Lo Nibble	Value Hi Nibble	Value Lo Nibble	None	None	None
<hr/>								
Stamp Set Word Variable	0xD6	Variable Hi Nibble	Variable Lo Nibble	----MS Byte---- Value Hi Value Hi Hi Nibble Lo Nibble		----LS Byte---- Value Lo Value Lo Hi Nibble Lo Nibble		None
Amulet Response	0xE6	Variable Hi Nibble	Variable Lo Nibble	----MS Byte---- Value Hi Value H Hi Nibble Lo Nibble		----LS Byte---- Value Lo Value Lo Hi Nibble Lo Nibble		None
<hr/>								
Stamp Set String Variable	0xD7	Variable Hi Nibble	Variable Lo Nibble	ASCII char	ASCII char	ASCII char	ASCII 0x00 char	
Amulet Response	0xE7	Variable Hi Nibble	Variable Lo Nibble	ASCII char	ASCII char	ASCII char	ASCII 0x00 char	
<hr/>								
Stamp Set Byte Variable Array	0xDF	Variable Hi Nibble	Variable Lo Nibble	Value Hi Nibble	Value 0x00 Lo Nibble		None	None
Amulet	0xEF	Variable	Variable	Array Cnt	Array Cnt	None	None	None

Response		Hi Nibble	Lo Nibble	Hi Nibble	Lo Nibble			
Stamp Set Word Variable Array	0xF2	Variable Hi Nibble	Variable Lo Nibble	----MS Byte---- Value Hi Value Hi Hi Nibble Lo Nibble		----LS Byte---- Value Lo Value Lo 0x00 Hi Nibble Lo Nibble		
Amulet Response	0xF3	Variable Hi Nibble	Variable Lo Nibble	Array Cnt Hi Nibble	Array Cnt Lo Nibble	None	None	None
Stamp Jump Page	0xA0	0x02	MSB Internal #	LSB Internal #	Checksum	None	None	None
Amulet Response	None	None	None	None	None	None	None	None

Table 1. Ten types of messages that can be sent from the Stamp to the Amulet.

Message	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
"Draw" Line Primitive	0xD9	Pnt 1 X Hi Byte Hi Nibble	Pnt 1 X Hi Byte Lo Nibble	Pnt 1 X Lo Byte Hi Nibble	Pnt 1 X Lo Byte Lo Nibble	Pnt 1 Y Hi Byte Hi Nibble	Pnt 1 Y Hi Byte Lo Nibble	Pnt 1 Y Lo Byte Hi Nibble	Pnt 1 Y Lo Byte Lo Nibble
Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15	Byte 16	Byte 17	Byte 18	Byte 19
Pnt 2 X Hi Byte Hi Nibble	Pnt 2 X Hi Byte Lo Nibble	Pnt 2 X Lo Byte Hi Nibble	Pnt 2 X Lo Byte Lo Nibble	Pnt 2 Y Hi Byte Hi Nibble	Pnt 2 Y Hi Byte Lo Nibble	Pnt 2 Y Lo Byte Hi Nibble	Pnt 2 Y Lo Byte Lo Nibble	Line Pattern	Line Weight
	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
Amulet Response	0xE9	Pnt 1 X Hi Byte Hi Nibble	Pnt 1 X Hi Byte Lo Nibble	Pnt 1 X Lo Byte Hi Nibble	Pnt 1 X Lo Byte Lo Nibble	Pnt 1 Y Hi Byte Hi Nibble	Pnt 1 Y Hi Byte Lo Nibble	Pnt 1 Y Lo Byte Hi Nibble	Pnt 1 Y Lo Byte Lo Nibble
Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15	Byte 16	Byte 17	Byte 18	Byte 19
Pnt 2 X Hi Byte Hi Nibble	Pnt 2 X Hi Byte Lo Nibble	Pnt 2 X Lo Byte Hi Nibble	Pnt 2 X Lo Byte Lo Nibble	Pnt 2 Y Hi Byte Hi Nibble	Pnt 2 Y Hi Byte Lo Nibble	Pnt 2 Y Lo Byte Hi Nibble	Pnt 2 Y Lo Byte Lo Nibble	Line Pattern	Line Weight
	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
Message	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
"Draw" Rectangle Primitive	0xDA	Pnt 1 X Hi Byte Hi Nibble	Pnt 1 X Hi Byte Lo Nibble	Pnt 1 X Lo Byte Hi Nibble	Pnt 1 X Lo Byte Lo Nibble	Pnt 1 Y Hi Byte Hi Nibble	Pnt 1 Y Hi Byte Lo Nibble	Pnt 1 Y Lo Byte Hi Nibble	Pnt 1 Y Lo Byte Lo Nibble
Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15	Byte 16	Byte 17	Byte 18	Byte 19
Delta X Hi Byte Hi Nibble	Delta X Hi Byte Lo Nibble	Delta X Lo Byte Hi Nibble	Delta X Lo Byte Lo Nibble	Delta Y Hi Byte Hi Nibble	Delta Y Hi Byte Lo Nibble	Delta Y Lo Byte Hi Nibble	Delta Y Lo Byte Lo Nibble	Line Pattern	Line Weight
	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
Amulet Response	0xEA	Pnt 1 X Hi Byte Hi Nibble	Pnt 1 X Hi Byte Lo Nibble	Pnt 1 X Lo Byte Hi Nibble	Pnt 1 X Lo Byte Lo Nibble	Pnt 1 Y Hi Byte Hi Nibble	Pnt 1 Y Hi Byte Lo Nibble	Pnt 1 Y Lo Byte Hi Nibble	Pnt 1 Y Lo Byte Lo Nibble

Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15	Byte 16	Byte 17	Byte 18	Byte 19
Delta X Hi Byte Hi Nibble	Delta X Hi Byte Lo Nibble	Delta X Lo Byte Hi Nibble	Delta X Lo Byte Lo Nibble	Delta Y Hi Byte Hi Nibble	Delta Y Hi Byte Lo Nibble	Delta Y Lo Byte Hi Nibble	Delta Y Lo Byte Lo Nibble	Line Pattern	Line Weight
<hr/>									
Message	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
"Draw" Fill Rectangle Primitive	0xDB	Pnt 1 X Hi Byte Hi Nibble	Pnt 1 X Hi Byte Lo Nibble	Pnt 1 X Lo Byte Hi Nibble	Pnt 1 X Lo Byte Lo Nibble	Pnt 1 Y Hi Byte Hi Nibble	Pnt 1 Y Hi Byte Lo Nibble	Pnt 1 Y Lo Byte Hi Nibble	Pnt 1 Y Lo Byte Lo Nibble
Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15	Byte 16	Byte 17	Byte 18	Byte 19
Delta X Hi Byte Hi Nibble	Delta X Hi Byte Lo Nibble	Delta X Lo Byte Hi Nibble	Delta X Lo Byte Lo Nibble	Delta Y Hi Byte Hi Nibble	Delta Y Hi Byte Lo Nibble	Delta Y Lo Byte Hi Nibble	Delta Y Lo Byte Lo Nibble	Fill Pattern	Line Weight
<hr/>									
Message	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
Amulet Response	0xEB	Pnt 1 X Hi Byte Hi Nibble	Pnt 1 X Hi Byte Lo Nibble	Pnt 1 X Lo Byte Hi Nibble	Pnt 1 X Lo Byte Lo Nibble	Pnt 1 Y Hi Byte Hi Nibble	Pnt 1 Y Hi Byte Lo Nibble	Pnt 1 Y Lo Byte Hi Nibble	Pnt 1 Y Lo Byte Lo Nibble
Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15	Byte 16	Byte 17	Byte 18	Byte 19
Delta X Hi Byte Hi Nibble	Delta X Hi Byte Lo Nibble	Delta X Lo Byte Hi Nibble	Delta X Lo Byte Lo Nibble	Delta Y Hi Byte Hi Nibble	Delta Y Hi Byte Lo Nibble	Delta Y Lo Byte Hi Nibble	Delta Y Lo Byte Lo Nibble	Fill Pattern	Line Weight
<hr/>									
Message	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
"Draw"Pixel Primitive	0xDC	Pixel X Hi Byte Hi Nibble	Pixel X Hi Byte Lo Nibble	Pixel X Lo Byte Hi Nibble	Pixel X Lo Byte Lo Nibble	Pixel Y Hi Byte Hi Nibble	Pixel Y Hi Byte Lo Nibble	Pixel Y Lo Byte Hi Nibble	Pixel Y Lo Byte Lo Nibble
Byte 10	Byte 11								
Fill Pattern	Line Weight								
Message	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9
Amulet Response	0xEC	Pixel X Hi Byte Hi Nibble	Pixel X Hi Byte Lo Nibble	Pixel X Lo Byte Hi Nibble	Pixel X Lo Byte Lo Nibble	Pixel Y Hi Byte Hi Nibble	Pixel Y Hi Byte Lo Nibble	Pixel Y Lo Byte Hi Nibble	Pixel Y Lo Byte Lo Nibble
Byte 10	Byte 11								
Fill Pattern	Line Weight								

Table 2. Four types of graphic primitive commands that can be sent from the Stamp to the Amulet



Amulet Technologies, LLC

GUI Engines for Embedded Systems

Appendix B

```
*****
*The main loop of this code sends a valid Client Start of Message (CSOM) character
*and interprets any returning information appropriately
*****

'{$STAMP BS2}
*****
* Pin Setup - pins to communicate with the DS1620
*****
DQ          CON      15      'Data input/output line
CLK         CON      14      'Synchronizing Clock
RST         CON      13      'Reset/select (high=active)
Wconfig     CON      $0C     'Protocol for 'Write Configuration'
CPU         CON      %10     'Config bit: serial thermometer mode
Cont        CON      %00     'Config bit: continuous conversions after start
StartC      CON      $EE     'Protocol for 'Start Conversion'
Rtemp       CON      $AA     'Protocol for 'Read Temperature'
WhiT        CON      $01     'Protocol for 'Write High-Temperature Setting'
WloT        CON      $02     'Protocol for 'Write Low-Temperature Setting'

*****
* Variable declarations
*****
* Variables for Stamp-Thermostat communication
DSdata      VAR      Word     'Word variable holds 9-bit temperature
Thigh       VAR      Word     'Word variable holds 9-bit "max" temp before AC turns on
Tlow        VAR      Word     'Word variable holds 9-bit "min" temp before heat turns on
* Variables for Stamp-Amulet communication
Counter     VAR      Byte
Variable    VAR      Byte     'Byte that holds variable being requested
AmuletMsg   VAR      Byte(11) 'array that response from Amulet
Str0        VAR      Byte(7)  'string array to hold response from Amulet
AmuletMsg(10) = NULL
NULL        CON      0        'Null character for end of String

*****
*Configure the DS1620
*****
LOW RST     'Deactivate '1620 for now
HIGH CLK    'Put clock in starting state
PAUSE 100   'Let things settle down a moment
HIGH RST    'Activate the '1620 and set it for continuous temp conversion
SHIFTOUT DQ,CLK,LSBFIRST, [Wconfig,CPU+Cont]
LOW RST     'Done-deactivate
PAUSE 50    'Wait for EEPROM to self-program
HIGH RST    'Now activate it again and send the start-conversion protocol
SHIFTOUT DQ,CLK,LSBFIRST,[StartC]
LOW RST     'Done-deactivate

serial_out:

*****
* This function executes an example of every possible command. Of the Stamp
* Message sent out, the command/request is the first byte, the HIGH nibble of
* the variable being used is always the second, and the LOW nibble of the variable
* being used is always the third. Some Stamp messages will have more than three
* bytes. Those are dealt with specifically in the section of code pertaining
* to that command. Using the BASICStamp Meta in your html page automatically
* null terminate every answer sent by the Amulet.
* The first segment of this code calls the name of the subroutine for the
* commands the code will invoke (currently it is set to invoke an example of
* each type of command). Consequently, this code is set up to exist solely as
* an example of the implementation of the Stamp as Master. This code assumes
* that the Amulet has been set to slaveAckRsp using a Meta tag, so that
* after every valid 'set' command it responds with an ACK (F0)
*****
```

```

GOSUB getByte          ' jumps to subroutine pertaining to GOSUB command.
GOSUB getWord         ' Depending on the needs of your program the
GOSUB getString       ' needed command code could be pasted directly
GOSUB getRPC          ' into the main function of the program or,
GOSUB setByte         ' if is used multiple times, kept as a
GOSUB setWord        ' subroutine and return
GOSUB setString
GOSUB setByteArr
GOSUB setWordArr
GOSUB drawLine        ' For a more in-depth explanation of the
GOSUB drawRect        ' graphic primitives and jump page instruction,
GOSUB drawFilledRect ' see the Amulet online help documentation
GOSUB drawPixel       ' on UART protocol and graphic primitives
GOSUB jumpPage

GOTO serial_out

'*****
'* Handler for a getByte command
'* Format of command get byte = 0xD0xx where xx=the variable byte being used
'* Amulet returns 0xE0xxNN, where NN equals the HEX data of the variable xx
'*****
getByte:
  Variable = $00          ' byte number to be requested - 0 in this example
  SEROUT 16, 84, [208,HEX2 Variable] ' send 'get byte' command. 208 is D0 in decimal
  ' wait for five bytes of response. if Amulet doesn't respond in 3 sec,
  ' send message again
  SERIN 16, 84, 3000, getByte, [STR AmuletMsg \5]

  B0.HIGHNIB = AmuletMsg(3)      ' store the two bytes of the value sent by the
  B0.LOWNIB = AmuletMsg(4)      ' Amulet into byte 0
  RETURN

'*****
'* Handler for a getWord command
'* Format of command get word = 0xD1xx, where xx = variable being requested
'* Returns 0xE1xxPPNN, where PP = high byte of variable xx, and NN = low byte of
'* variable xx
'*****
getWord:
  Variable = $01          ' word number to be requested - 1 in this example
  SEROUT 16, 84, [209,HEX2 Variable] ' send 'get word' command
  ' wait for seven bytes of response if doesn't happen, send message again
  SERIN 16, 84, 3000, getWord, [STR AmuletMsg \7]
  W1.HIGHBYTE.HIGHNIB = AmuletMsg(3) ' store the four bytes of the value sent by
  W1.HIGHBYTE.LOWNIB = AmuletMsg(4)  ' the Amulet into word 0
  W1.LOWBYTE.HIGHNIB = AmuletMsg(5)
  W1.LOWBYTE.LOWNIB = AmuletMsg(6)
  RETURN

'*****
'* Handler for a getString command
'* Format of command get string = 0xD2xx, where xx = index of string variable
'* Returns 0xE2xxString+Null from Amulet
'* If a string of greater than 6 characters, need to change Str0 declaration
'*****
getString:
  Variable = $00          ' string number to be requested - 0 in this example
  SEROUT 16, 84, [210,HEX2 Variable] ' send 'get string' command
  ' wait for either 7 bytes or null terminator, if it doesn't happen in 3 sec,
  ' resend command
  SERIN 16, 84, 3000, getString, [STR AmuletMsg \7 \0]
  FOR Counter = 0 TO 5      ' store incoming string in Str0
    Str0(Counter) = AmuletMsg(Counter + 3)
  NEXT
  Str0(6) = 0
  RETURN

```

```

*****
'* Handler for a getRPC command
'* Format of command get RPC = 0xD4xx, where xx = index of label variable
'* Returns 0xE4xx to Stamp
'* Returns requested data back to the screen
*****
getRPC:
SEROUT 16, 84, [212,"00"]          ' RPC flag 0x00 sent in this example
' which requests all the stored RPC's. wait for either 3RPC's or
' null terminator. if it doesn't happen in 3 sec, resend the command
SERIN 16, 84, 3000, getRPC, [STR AmuletMsg \9 \0]
FOR Counter = 3 TO 7 STEP 2        ' go through each RPC received
  Variable.HIGHNIB = AmuletMsg(Counter) ' Store RPC in variable
  Variable.LOWNIB = AmuletMsg(Counter + 1)
  IF Variable = 1 THEN RPCOne      ' execute code for current RPC
  IF Variable = 2 THEN RPCTwo
  IF Variable = 3 THEN RPCThree
  IF Variable = 4 THEN RPCFour
incrRPC: NEXT
GOTO doneRPC

RPCOne:                          ' if RPC 1 comes back then execute this code
  DEBUG "RPC 1 executing"
  HIGH RST                        ' activate the DS1620
  SHIFTOUT DQ,CLK,LSBFIRST, [WhiT]' set the "max" temp before the AC kicks in to
  SHIFTOUT DQ,CLK,LSBFIRST, [%000111100] ' 30.0 degrees C
  LOW RST                          ' deactivate the DS1620
  PAUSE 50                          ' pause to give the EEPROM time to program
GOTO incrRPC

RPCTwo:                          ' if RPC 2 comes back then execute this code
  DEBUG "RPC 2 executing"
  HIGH RST                        ' activate the DS1620
  SHIFTOUT DQ,CLK,LSBFIRST, [WloT]' set the "min" temp before the heater kicks in
  SHIFTOUT DQ,CLK,LSBFIRST, [%000100100] ' to 18 degrees C
  LOW RST                          ' deactivate the DS1620
  PAUSE 50                          ' pause to give the EEPROM time to program
GOTO incrRPC

RPCThree:                        ' if RPC 3 comes back then execute this code
  DEBUG "RPC 3 executing"
GOTO incrRPC

RPCFour:                         ' if RPC 4 comes back then execute this code
  DEBUG "RPC 4 executing"
GOTO incrRPC

doneRPC: RETURN

*****
'* Handler for a setByte command
'* Format of command string = 0xD5xxNN, where xx = variable to be set
'* and NN = HEX data
'* Returns 0xF0 (ACK) to Stamp
*****
setByte:
HIGH RST                          'Activate the '1620
SHIFTOUT DQ,CLK,LSBFIRST, [Rtemp]  'Request to read temperature
SHIFTOUT DQ,CLK,LSBPRE, [DSdata\9]  'Get the temperature reading
LOW RST                             'Deactivate
DSdata = DSdata/2
SEROUT 16, 84, [213,"00",HEX DSdata] ' send out command to set byte 0
' to temperature reading in Celsius
SERIN 16, 84, 3000, setByte, [STR AmuletMsg \1] ' accept the Amulet's response
Variable = AmuletMsg
IF Variable <> $F0 THEN setByte      ' if didn't get 'acknowledge', resend
RETURN

```

```

'*****
'* Handler for a setWord command
'* Format of request string = 0xD6xxPPNN, where xx = variable to be set
'* PP = high byte of HEX data and NN = low byte of HEX data
'* Returns 0xF0 (ACK) to Stamp
'*****
setWord:
    SEROUT 16, 84, [214,"00","0783"]           ' set word 0 to decimal 783
                                                ' (0x30, 0x33, 0x00, 0x46)
    SERIN 16, 84, 3000, setWord, [STR AmuletMsg \1] ' accept the Amulet's response
    Variable = AmuletMsg
    IF Variable <> $F0 THEN setWord           ' if did not get ACK, resend
    RETURN

'*****
'* Handler for a setString command
'* Format of request string = 0xD7xxP{PNNQQZ}0, where xx = variable to be set
'* followed by the HEX data (up to 7 bytes in this case) and then a Null terminator
'* Returns 0xF0 (ACK) to Amulet
'*****
setString:
    ' send out command to set string 0 to "Amulet"
    SEROUT 16, 84, [215,"00","Amulet",NULL]
    ' accept the Amulet's response
    SERIN 16, 84, 3000, setString, [STR AmuletMsg \1]
    Variable = AmuletMsg
    IF Variable <> $F0 THEN setString         ' if did not get ACK, resend
    SS2:
    SEROUT 16, 84, [215,"01","Hello",NULL]    ' send command to set string 1 to "Hello"
    SERIN 16, 84, 3000, SS2, [STR AmuletMsg \1] ' accept the Amulet's response
    Variable = AmuletMsg
    IF Variable <> $F0 THEN SS2              ' if did not get ACK, resend
    RETURN

'*****
'* Handler for a setByteArr command
'* Format of string = 0xDFxxNN{PPZZ...}0 where xx=the variable byte being used
'* Returns 0xF0 (ACK) to Stamp
'*****
setByteArr:
    ' send out byte array 0 to be 2, 4, 6, 8
    ' (0x30, 0x32, 0x30, 0x34, 0x30, 0x36, 0x30, 0x38)
    SEROUT 16, 84, [223,"00",HEX2 2,HEX2 4,HEX2 6, HEX2 8,NULL]
    ' accept the Amulet's response
    SERIN 16, 84, 3000, setByteArr, [STR AmuletMsg \1]
    Variable = AmuletMsg
    IF Variable <> $F0 THEN setByteArr       ' if did not get ACK, resend
    RETURN

'*****
'* Handler for a setWordArr function request
'* Format of string = 0xF2xx, where xx = variable being requested
'* Returns 0xF0 (ACK) to Stamp
'*****
setWordArr:
    ' send out word array 0 to be 200,400,600,800
    ' (0x30, 0x30, 0x43, 0x38 / 0x30, 0x31, 0x39, 0x30 / 0x30, 0x32, 0x35, 0x38 /
    ' 0x30, 0x33, 0x32, 0x30)
    SEROUT 16, 84, [242,"00","00C8","0190","0258","0320",NULL]
    ' accept the Amulet's response
    SERIN 16, 84, 3000, setWordArr, [STR AmuletMsg \1]
    Variable = AmuletMsg
    IF Variable <> $F0 THEN setWordArr       ' if did not get ACK, resend
    RETURN

```

```

'*****
'* Handler for a drawLine command
'* Format of string = 0xD9XXXXYYYYxxxxyyyyPW, where
'* XXXX = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* x-coordinate of first endpoint
'* YYYY = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* y-coordinate of first endpoint
'* xxxx = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* x-coord of second endpoint
'* yyyy = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* y-coordinate of second endpoint
'* P = line Pattern
'* W = Line weight
'* Returns 0xF0 (ACK) to Stamp
'*****
drawLine:
' The following SEROUT tells the Amulet to draw a line from (20,30) to (300,180)
' with a line pattern of 0 (black) and a line width of 4
SEROUT 16, 84, [217,"0014","001E","012C","00B4","0","4"]
' accept the Amulet's response
SERIN 16, 84, 3000, drawLine, [STR AmuletMsg \1]
Variable = AmuletMsg
IF Variable <> $F0 THEN drawLine ' if did not get ACK, resend
RETURN

'*****
'* Handler for a drawRect command
'* Format of string = 0xDAXXXXXYYYYxxxxyyyyPW, where
'* XXXX = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* x-coordinate of upper left corner
'* YYYY = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* y-coordinate of upper left corner
'* xxxx = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* length of rectangle in x direction (x-delta)
'* yyyy = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* length of rectangle in y direction (y-delta)
'* P = line Pattern
'* W = Line weight
'* Returns 0xF0 (ACK) to Stamp
'*****
drawRect:
' The following SEROUT tells the Amulet to draw a rectangle with its left corner
' at (5,80) that is 40 pixels in the x-direction and 80 pixels in the y-direction
' with a line pattern of 1 (gray) and a line width of 2
SEROUT 16, 84, [218,"0005","0050","0028","0050","1","2"]
' accept the Amulet's response
SERIN 16, 84, 3000, drawRect, [STR AmuletMsg \1]
Variable = AmuletMsg
IF Variable <> $F0 THEN drawRect ' if did not get ACK, resend
RETURN

'*****
'* Handler for a drawFilledRect command
'* Format of string = 0xDBXXXXYYYYxxxxyyyyPW, where
'* XXXX = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* x-coordinate of upper left corner
'* YYYY = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* y-coordinate of upper left corner
'* xxxx = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* length of rectangle in x direction (x-delta)
'* yyyy = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* length of rectangle in y direction (y-delta)
'* P = Fill Pattern
'* W = Line weight
'* Returns 0xF0 (ACK) to Stamp
'*****
drawFilledRect:
' The following SEROUT tells the Amulet to draw a rectangle with its left corner
' at (80,30) that is 60 pixels in the x-direction and 20 pixels in the y-direction

```

```

' with a fill pattern of 6 (bricks) and a line width of 3
SEROUT 16, 84, [219,"0050","001E","003C","0014","6","3"]
' accept the Amulet's response
SERIN 16, 84, 3000, drawFilledRect, [STR AmuletMsg \1]
Variable = AmuletMsg
' if did not get ACK, resend
IF Variable <> $F0 THEN drawFilledRect
RETURN

```

```

*****
'* Handler for a drawPixel command
'* Format of string = 0xDCXXXXYYYYPW, where
'* XXXX = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* x-coordinate of pixel
'* YYYY = MSB Hi nibble, MSB Lo nibble, LSB Hi nibble, LSB Lo nibble for
'* y-coordinate of pixel
'* P = Fill Pattern
'* W = Line weight
'* Returns 0xF0 (ACK) to Stamp
*****

```

```

drawPixel:
' The following SEROUT tells the Amulet to draw a pixel at (200,180)
' with a fill pattern of 0 (black) and a line width of 10
SEROUT 16, 84, [220,"00B8","00B4","0","7"]
' accept the Amulet's response
SERIN 16, 84, 3000, drawPixel, [STR AmuletMsg \1]
Variable = AmuletMsg
IF Variable <> $F0 THEN drawPixel ' if did not get ACK, resend
RETURN

```

```

*****
'* Handler for a jumpPage command
'* Format of string = 0xA002Pps, where
'* 0xA0,0x02 = command to jump page
'* P = MSB of internal number of page
'* p = LSB of internal number of page
'* s = checksum of first four bytes so that LSB of sum being 0x00
'* The internal number can be found by opening the Amulet .map file created when
'* downloading the Amulet html page to Flash
*****

```

```

jumpPage:
' The following SEROUT tells the Amulet to jump to internal page 0x106
' Note that this command does not expect a response
SEROUT 16, 84, ["A0","02","01","06","39"]
RETURN

```

END



Amulet Technologies, LLC

GUI Engines for Embedded Systems