

StampPlot® Pro Developer's Primer

**Released: 12/29/05
Last revised: 01/11/06**

StampPlot is a registered trademark of SelmaWare Solutions.
BASIC Stamp is a registered trademark of Parallax, Inc.

I.	<i>Introduction</i>	3
	Test Circuit	4
	BASIC Stamp Code to Test	5
II.	<i>Adding and Updating an Object</i>	6
	Adding and Setting a Meter Object	6
	Using the UPDATE Feature and Events	7
	Updating and Manually Plotting on Analog Data Arrival	8
III.	<i>Building a Macro</i>	9
IV.	<i>Interactive Control</i>	10
	Silencing the Buzzer with a Checkbox	10
	Controlling the Tone Duration with a Slider	11
	Configuring StampPlot with Controls	12
V.	<i>Graphical Controls and the Background</i>	13
	Silencing the Buzzer with a Virtual Switch	13
	Adding a Virtual Lamp Indicator	15
	Comments on using the Image Control	16
	Using your Own Graphics and Distributing	16
	Setting Background Color or Image	17
	Background Event Code	17
VI.	<i>StampPlot Math and Tips</i>	18
	Macro Values	18
	Example with Math Operations	19
	Boolean Logic	19
	Equality Testing	20
	Running Conditional Code	20
VII.	<i>Various Tips and Tricks</i>	21
	Forming an Interface from the BASIC Stamp	21
	Allow use of ENTER Key to Accept Textbox Data	22
	Setting the Tab Order	23
	Allowing Hot Keys for Buttons and Checkboxes	23
	Manually Logging Data	23
	Storing Temporary Data in StampPlot	24
	Actions on Plot Reset	24
	Saving and Recalling the Configuration	24
VIII.	<i>Conclusion</i>	25
IX.	<i>Revisions:</i>	25

I. INTRODUCTION

This primer is written for those with a Developers License for StampPlot, and those wishing to evaluate StampPlot within the 10 minutes run time allowed. It is recommended that the reader look over the standard user's primer available on the StampPlot home page (www.stampplot.com) as much of the basic information is not covered here.

StampPlot can be an extremely versatile tool as an interface for controllers. The help files for StampPlot are chocked full of good information, but it lacks some basic information on developing your own interfaces. This document highlights how to get started creating interfaces and macros from the developer's perspective, but is far from a complete guide to using StampPlot. We mainly focus on those things we find most important when developing our own interfaces. It is recommended to give help file a quick look after working through this tutorial to expand your knowledge even further and better see how to apply the information in the help files.

This primer uses the BASIC Stamp from Parallax (www.parallax.com) as the controller of choice, though most any controller may be used that can send serial data. We'll rely on you to figure it out, though we will help where we can.

Beyond this tutorial, we invite you to ask further questions on our yahoo group at <http://groups.yahoo.com/group/selmaware>. We also regularly monitor the Parallax Stamps In Class forum and a few other microcontroller groups. You may also Email us at support@selmaware.com

Conventions for fonts used in this text:

StampPlot Code

BASIC Stamp Code

Important things to look for

Thank you for your support,

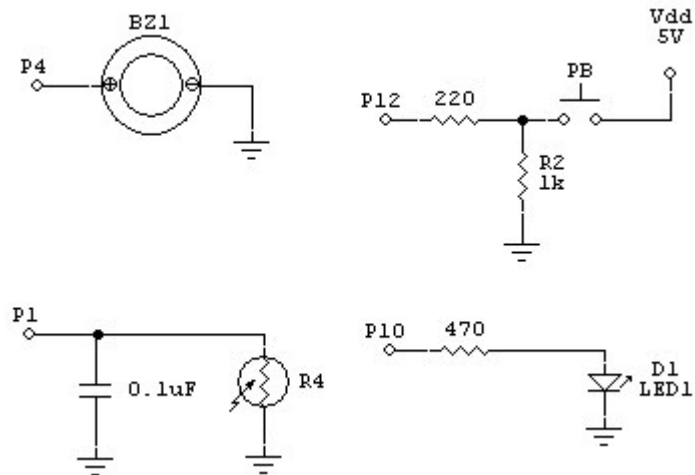
Martin Hebel

SelmaWare Solutions

<http://www.stampPlot.com>

Test Circuit

The basic circuit is shown in Figure 1. These parts are available in many of the Parallax text kits. It consists of an active-high pushbutton, active-high LED, a buzzer, and an RC network using a photo-resistor.



BASIC Stamp Code to Test

This code will test the circuit and illustrate basic plotting with StampPlot. Pressing the Pushbutton will light the LED, and adjusting light level will affect the buzzer's tone.

```
{${STAMP BS2}
{$PBASIC 2.5}

'Declare Pins for hardware
Photo PIN 1
Buzzer PIN 4
LED PIN 10
PB PIN 12

'Declare variables
LED_State    VAR    Bit
PB_State     VAR    Bit
PhotoVal     VAR    Word
Freq         VAR    Word
Dur          Var    Word
Silence      VAR    Bit

'Initialize
HIGH LED
Dur = 100           ' Initial duration of buzzer
PAUSE 1000         ' Allow connection to stabilize
DEBUG CR          ' Send CR to end any old data

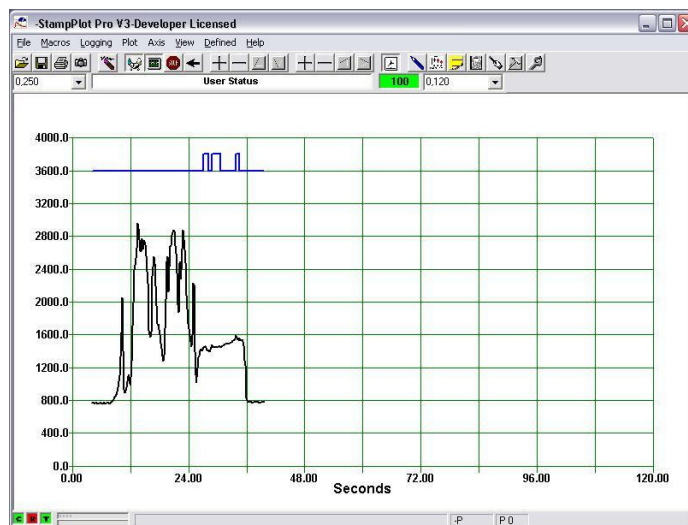
DO
  HIGH Photo       ' Read photo RC
  PAUSE 2
  RCTIME Photo, 1, PhotoVal

  FREQOUT buzzer, Dur, PhotoVal ' Sound tone based on light

  PB_State = PB    ' Read Pushbutton
  led_State = PB   ' Set LED to pushbutton
  LED = LED_State

  DEBUG DEC PhotoVal,CR ' Display photo value
  DEBUG IBIN1 PB_State,CR ' Display pushbutton state as binary
LOOP
```

- Download the code and note an analog value and a binary value are being sent.
- Run StampPlot Pro, and use the *File Menu* to start a *New Plot* (Delete configuration and objects if asked.)
- *Connect* and *Plot*.
- Adjust light level and press the pushbutton.
- *Adjust ranges* to allow full light level to be plotted.




II. ADDING AND UPDATING AN OBJECT

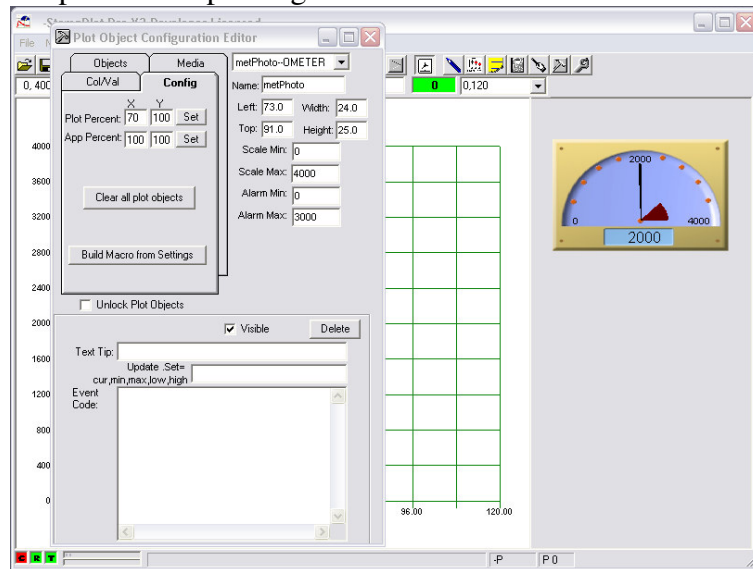
This section discusses building a basic interface to display data information. We will start by adding a virtual meter to your interface. Here are some rules and discussion to guide you:

- The *Object Editor* is used to place and configure control objects.
- Control objects may be placed by *dragging* them onto an *empty background* area of the plot interface. The upper-left corner of the control is used for placement.
- *Object names* are used to refer to the object. Of the name in parenthesis, such as (objName), will returns the objects value. Objects may be updated from controller code with:
DEBUG "objName=", DEC value,CR
- *Update values* are used to update an object when the command **!O Update** is issued. Examples may be an analog value sent from the controller (**AINVAL0**) or other StampPlot values, such as plot time (**PTIME**).
- *Text Tips* are used to give the user some information when their mouse pointer is over the object.
- *Event code* is run when a particular object's event occurs. Examples of events are meters in the alarm range, buttons clicked, text boxes changed, etc.
- The *Analog Object* event code, under *oAnalog*, will automatically run code when data arrives.

Adding and Setting a Meter Object

- *Disconnect* on StampPlot.
- Open the *Object Editor* . Select the *Config* Tab.
- Set the *Plot Percent* to **70** and **100**, click *Set*.
- Note the plot spans 70% of the horizontal area.
- Select the *Objects* tab.
- Position the *Object Editor* so that the open area on the plot background may be seen.
- Drag a *Meter Object* to the open area. *Name* the object **metPhoto**. It is common in programming that the 1st 3 letters of an object's name represents the type of object, in this case a meter.

- Use the *Tab* key so that the *Left* textbox is highlighted. Press the *arrow keys* on your keyboard (left/right/up/down.) Note what the meter moves on the interface.
- Hold down the *shift-key* and use the *arrow keys* again. Note the meter moves by small amounts.
- Use *Tab* key to move to the *Width* box. Test the *arrow keys* again and note they adjust the size.
- *Tab* to *Scale Max* and set the *Scale Max* to **4000**. Set *Alarm Max* to **3000**.
- At the end of the DO-LOOP (before LOOP) in the BASIC Stamp Code, enter **DEBUG "!O metPhoto=", DEC PhotoVal ,CR**
This will send a string similar to: **!O metPhoto= 512**
!O is short hand for **!POBJ** or plot object.
- Download, *close DEBUG Window*, and *connect and plot* with StampPlot, reset if needed. Note the meter is updated and plotting occurs.



- Disconnect on StampPlot.
- Close the Object Editor.
- Hold down the *Shift Key* and *Right-Click* the meter. The object editor should appear with the meter selected for configuration.
Note: Drop-down box objects need to be manually selected with the object-editors' drop-down box, and selecting text boxes may require the user to click on the object editor again.

Using the UPDATE Feature and Events



While multiple objects can be updated individually from controller code, it can be memory consuming and take time. A better way is to set an *Update value*, which will cause the object to be updated whenever a **!O Update** is issued. Along the way, we will do a few other things.

- Check the *Unlock Plot Objects* box, *click and drag* your meter to a new open location on the background.
- Click in the *Text Tip* box and enter **Displays Photo RC Value**. This will appear when you place your mouse over the meter (you may have to click on the main StampPlot window first.)

- Tab to the *Update .Set=* box and enter **(AINVAL0)**. This will update the meter with the 1st analog value when an **!O Update** is issued. If a second analog value were sent, such as 512,105, the 2nd value would be **(AINVAL1)** and so on.
- In the *Event Code*, enter **!BELL**. This will sound the computer's bell whenever the alarm is level is exceeded. This is known as event code, and most objects have an event trigger, such as a button being clicked.
- At the end of the DO-LOOP replace **DEBUG "!O metPhoto=", DEC PhotoVal ,CR** with: **DEBUG "!O Update",CR**
- Download, connect on StampPlot, plot, and test the interface.



Updating and Manually Plotting on Analog Data Arrival

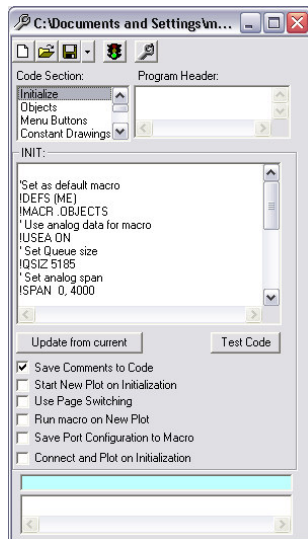
Sometimes we simply have data arriving and cannot modify the controller's code, or we wish to manipulate the data or use it in other ways. The *oAnalog* object code can be set to perform functions when analog data arrives.

- Remove **DEBUG "!O Update",CR** from the BS2 code and download.
- In the *Object Editor's* drop-down box, select *oAnalog*.
- Check *Use Analog data for macro only*. This will prevent analog values from automatically being plotted.
- In the code section, enter:
!O Update
!ACHN 0,(AINVAL0),(BLACK)
- Connect and plot. All should be acting as previously. The instruction **!ACHN** tells StampPlot to plot an analog channel, the channel to use is 0, with the value of **(AINVAL0)** in the color black.
- Let's do a few more things when data arrives, add to the code (note: comments cannot be on code lines):
' Show max value in User Status
!STAT Maximum is (AINMAX0)
' Plot Maximum in Green
!ACHN 1,(AINMAX0),(GREEN)
- You will note that on plot reset, the maximum value does not change.
- Open the *Immediate/Debug Window* .
- Enter **!CLMM**, this will clear the min/max saved values.
- Connect and plot if not doing so.
- While in that window, select each of the debug checkboxes to view what they show.
- Of course, it would be cumbersome to manually enter **!CLMM** every time you reset the plot. The instruction **!CMMR ON** (Clear Min/Max on Reset) may be issued or selected from in the Values Window . In the *Immediate/Debug Window* enter **!CMMR ON**.

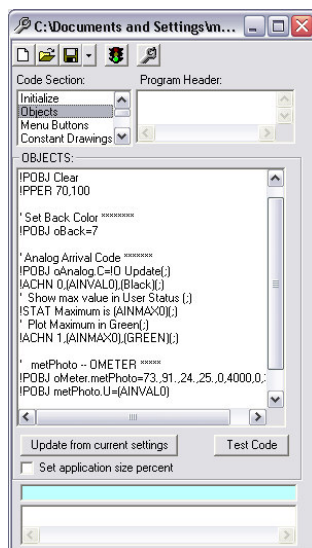
III. BUILDING A MACRO

A macro is a text file that configures StampPlot and may contain routines to be executed. The Macro Editor allows developers to build macros from a plot configuration.



- Open the *Macro Editor* .
- Click the *Build* button  on the *Macro Editor Window*.
- Two areas are of importance for this primer are the *Initialization* and *Objects* sections. The *Initialization* is code that configures the basic plot window and basic settings configured through various StampPlot menus.
- The *Program Header* may be used to enter your name, title, date and version if desired.



- The *Objects* section is called from Initialize, **!MACR .OBJECTS**, and configures the interface objects. Looking through it you can note how the meter is configured and set, and how oAnalog is coded.



- This code section may be *updated or tested* with the buttons. The bottom lines act the same as the Immediate/Debug window to quickly test instructions.

- Save the file by clicking the *Save button* (it also drops down for save-as) .
- Locate your *C drive*, create a folder name *ProPrimer*.
- Open the folder and save your file as *myTest.spm*.
- If asked, have the paths updated and rebuilt. This will be discussed later.
- Click the *Run Current* button  to test the code.
- Ensure the Developer's version of StampPlot (instead of Home/Educ) is the default version by using *File-->Update Associations for *.spm *.plt* on the main plot window
- Close StampPlot.
- Use *Windows Explorer*, locate your folder and *myTest.spm*.
- Double-click to open it.
- If you always want your test interface to open when starting StampPlot, you may select it by using the *Macro--> Select Start-Up Macro* menu choice on the main interface.

IV. INTERACTIVE CONTROL

StampPlot can be used for interactive control with the controller. The general procedure is for the controller to request a value via a **!READ** and to accept the returning data. With the BASIC Stamp, it is important to have a short pause prior to sending new data to allow the controller's echo to clear.

```
DEBUG "!READ (objName)",CR
DEBUGIN DEC myVariable
PAUSE 50
```

Note: DEBUGIN will cause program execution to cease until data arrives. If you need to ensure your code keeps running if there is a problem with returned data, the use of SERIN with a timeout is recommended:

```
SERIN 16, 84, 1000, NoData, [DEC myVariable] ' accept data at 9600 with 1 second timeout
NoData:
```

Silencing the Buzzer with a Checkbox

That buzzer is sure annoying, isn't it? Let's add a *checkbox* to the interface that will silence the sound.

- Drag and drop a Checkbox object on the interface. Configure as follows:
 - Name: chkSilence
 - Text: Silence Buzzer
 - Font: 12, Bold
 Adjust width and position as desired.
- At the top of the DO-LOOP in the BASIC Stamp Code, add the following:


```
DEBUG "!READ (chkSilence)",CR
DEBUGIN DEC Silence
PAUSE 50
```
- Modify the FREQOUT line of code to:

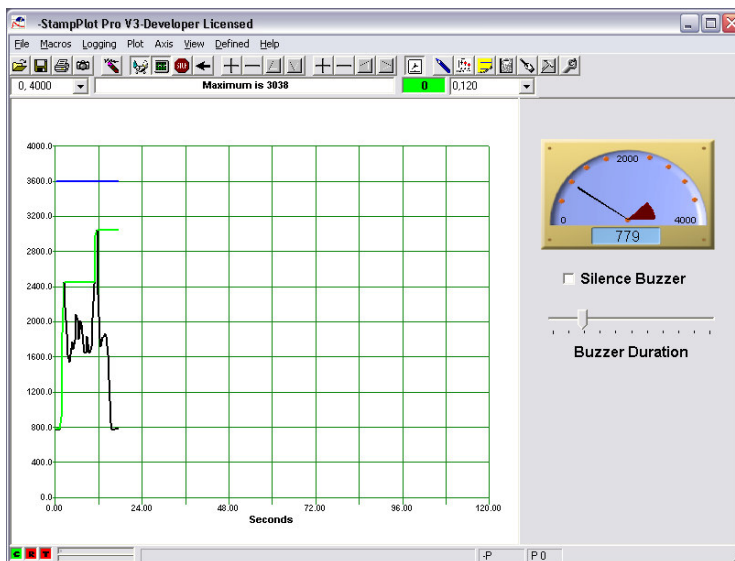

```
IF Silence = 0 THEN FREQOUT buzzer, 100, PhotoVal
```

- Download, connect and test.
- A bit variable is used in the BASIC Stamp code since a checkbox returns a 1 (checked) or a 0 (unchecked).

Controlling the Tone Duration with a Slider

Many other objects can return values to be read by the BASIC Stamp. Let's add a *horizontal slider* to control the duration a tone is played.

- Add a horizontal slider to your plot interface.
 Name: sldDur (keeping names short saves BASIC Stamp code and increases interaction speed).
 Min: 20
 Max: 1000
 Size and position to your liking. If you can't see tick marks, increase the height.
- Add a label below the slider.
 Name: lblDur
 Text: Buzzer Duration
 Size and position to your liking.
- Add BASIC Stamp code after the Silence code:
DEBUG "!READ (sldDur)",CR
DEBUGIN DEC Dur
PAUSE 20
- Test your code.



- These objects may be set for a known configuration by adding the following at the end of the Initialization section of your BASIC Stamp code.

```
DEBUG "!O chkSilence=0",CR      ' Ensure silence off  
DEBUG "!O sldDur=", DEC Dur, CR  ' Set Slider to current
```

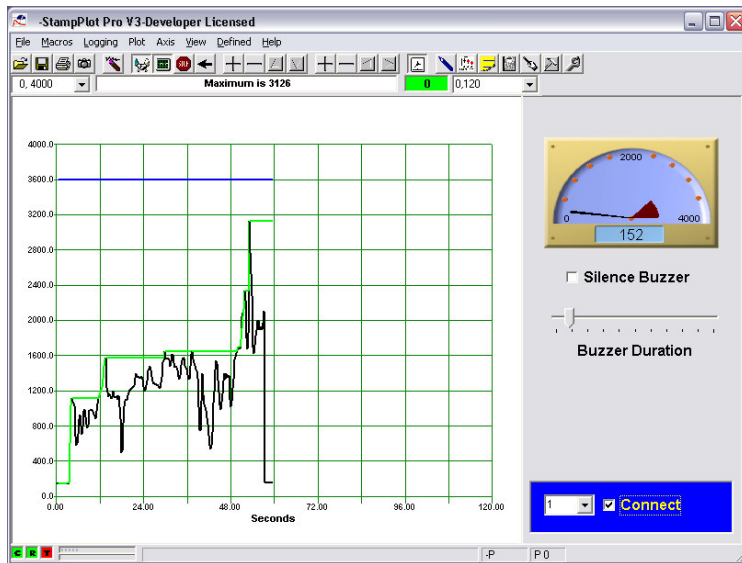
- Download, connect and test.
- Don't forget to rebuild and save your macro!

Configuring StampPlot with Controls

Control Objects may also be used to configure and control StampPlot, such as adding connection controls for the serial port number and a control for opening the connection.

- Add a *Label Object* control, *oLabel*, to the interface to act as a background for other controls.
 - Name it lblConn.
 - Set the text to a space or blank.
 - Set the background to Blue by clicking on the Col/Val Tab and dragging blue to the Back Color Box (it may not update until you tab a few times).
 - Turn the border on (1).
 - Size it so it is fill the bottom 10% of the open interface area.
- Add a *drop-down box* to the empty background.
 - Name it drpConn.
 - Position it so that it is on top of the blue label area.
 - Size it so that it is 7 wide.
 - Click 'Clear' to the clear the contents.
 - In the List Add area, enter 1, press return, 2, return, and so on up to 15, then enter 1 again. *The last value entered will be the default value.*
- Add a *checkbox*.
 - Name it chkConn.
 - Change the text to read Connect.
 - Position so that it is within the blue background.
 - Change the back color to blue.
 - Change the text color to yellow.
 - Change the font to 12, bold.
 - Enter event code:


```
' Ensure connection closed
!CONN OFF
' Set port to value of drpConn.
!PORT (drpConn)
' Set connection state based up value of checkbox.
!CONN ((ME))
```
 - Try connecting, changing ports, and reconnecting, etc.



Within Event Code, using **ME** refers to the object whose code is being run. **(ME)** returns the name of the control, and **((ME))** returns its value. In this case, a checkbox has a value of 0 or 1, so the code **!CONN 1** or **!CONN 0** would be ran. Using 1 and 0 is equivalent to **!CONN ON** and **!CONN OFF**. Try out this code in the event and watch the Status textbox as you click the box:
!STAT (ME) = ((ME))

Note that if an error occurs, there is no automatic function to uncheck the check box. It is up to the user to correct, uncheck and re-check. Another way would be to have a Connect and Disconnect *command button* and code in one to set the port and connect, **!CONN ON**, and the other to disconnect, **!CONN OFF**.

Of course, you may want to also enable plotting at the same time you connect by adding **!PLOT ((ME))** to the checkbox's event code. NOTE: When a connection is opened, the queue, which holds data and instructions to be processed, is cleared out. If **!PLOT ((ME))** is added after **!CONN ((ME))**, the instruction will be deleted prior to being processed.

V. GRAPHICAL CONTROLS AND THE BACKGROUND

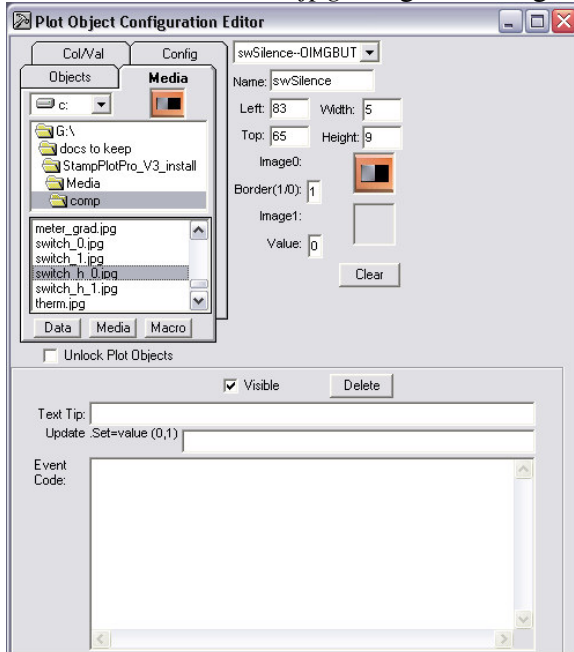
StampPlot includes the ability to use graphical controls as well for indication or interaction. The images may be from the included set of graphics or one you design.

Silencing the Buzzer with a Virtual Switch

We will replace the checkbox for silencing with a virtual slide switch.

- *Shift-Right-Click* the *Silence Buzzer* checkbox on your interface.
- Ensure the *chkSilence* is listed and *Delete* it using the Object Editor.
- From the *Object Tab*, drag and drop an *image-button object* (looks like a vertical switch).
- Name it *swSilence* and position the blank image to an appropriate location.
- Select the *Media Tab*, open the *comp* (components) directory in *Media*.

- Locate the *switch_h_0.jpg* image and drag the image name to the *image0:* location.



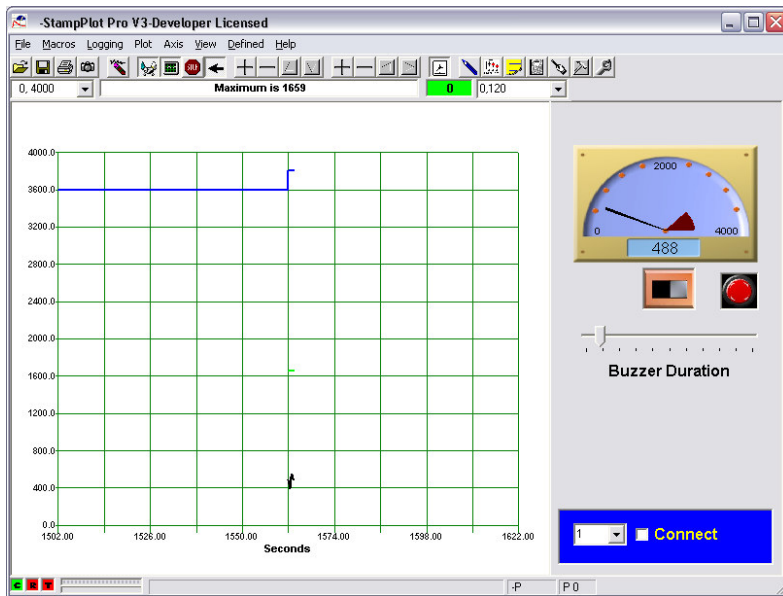
- Locate image *switch_h_1.jpg* and drag to the *Image1:* location.
- In your BASIC Stamp code, modify the following code:
DEBUG "!READ (chkSilence)",CR
to
DEBUG "!READ (swSilence)",CR
- Run your program and test. Note that the selected image (0 or 1) is being read by the BASIC Stamp.
- Note when the macro is saved (update if requested), the path to the images is as follows in the object code:
comp\switch_h_0.jpg,1
The root directory for graphics and sounds is the media directory of StampPlot. We will work with graphics outside of this directory soon.
- Let's add some sound to your switch by coding a WAV file to play with the switch is clicked. In the switches event code, add:
~IWAV stapler

Note: **IWAV** will interrupt any WAV currently being played to play the selected one. **PWAV** will not interrupt one being played. The ~ is typically used with graphics and implies that the action will not be stored or played when the plot refreshes. The installed sounds are all in the media root directory.

Adding a Virtual Lamp Indicator

We'll add a virtual lamp (or LED... sorta) to our interface.

- Drag and drop another *image button* onto the background. Position next to the switch.
Name: ledAlarm
Image0: led_red_0.jpg
Image1: led_red_1.jpg
Update Value: (BIT0)
- Connect, plot and test by pressing the pushbutton on your board. The lamp should change images. (BIT0) refers to the 1st bit of the digital data being received.
- Click the lamp on your interface and note that it will change states due to being an image button. This can be prevented by positioning a label object over the image to act as a transparent cover for the image. Use a space only for text.
- The image also could have been updated from BASIC Stamp code using:
DEBUG "!O ledAlarm=", DEC LED,CR



Comments on using the Image Control

The image control (looks like a green lens in the objects Tab) may be used to have a static image, or images that are selected through code. While we won't formally work this into this tutorial, it's worth a discussion because of the versatility it provides.

Say for example we had multiple graphics to indicate a process. An Image object could be used to display the selected multiple images. For sake of discussion, let's say we have on the interface named `imgTest`. In BASIC Stamp code we could send data to update the image (these don't really exist, but you get the idea):

```
DEBUG "!O imgTest= comp\fill.jpg",CR
```

or

```
DEBUG "!O imgTest= comp\mix.jpg",CR
```

But a more versatile means would be to have a combination of numbers to represent the image to be displayed. For example, in the `comp` directory is a batch of images named `led_bar0.jpg` to `led_bar10.jpg`. We could have in code in the controller to display each:

```
FOR X = 0 to 10  
  DEBUG "!O imgTest=comp\led_bar", DEC X, ".jpg",CR  
  PAUSE 1000  
NEXT
```

Note that a string is sent filling in the value of `X` as the loop progresses. Or, the *Update value* for the object could be set to the following where the value of incoming analog data (0 to 10) is replaced for the proper object name:

```
comp\led_bar(AINVAL0).jpg
```

Using your Own Graphics and Distributing

You are not restricted to the AMAZING graphics we professionals made (drips with sarcasm). But you need to ensure that if you want to distribute your developed macros, that others have access to the graphics as well.

- Using Microsoft Paint ®, create a small image for your Alarm graphics, such our samples:



- Save them to your `C:\ProPrimer` directory as `ok.jpg` and `help.jpg`.
- Edited the image button for the lamp indicator (`ledAlarm`) and locate your graphics in your directory and select them as `image0` and `image1`. If you have saved your macro recently, clicking the Macro button on the Media Tab should bring you there.
- Test with your graphics.
- Build your macro. Note in the Objects code, the full path to the image is given: **C:\ProPrimer\ok.jpg**. This would be unacceptable if the user put the files on their drive D for example.
- Save your macro to the `C:\ProPrimer` directory. When asked, update.

- Check your object code again. The path to the image should now be: **(PATH)ok.jpg**. **(PATH)** is a macro value meaning to use the path from which the macro originates.
- The ProPrimer directory may now be Zipped and sent to another to unzip and run. The images should work properly (do not try to use in compressed format).
- A subdirectory could also have been used to hold all your unique graphics and sounds.

Setting Background Color or Image

The background may be set to a color or to an image that control objects may still be placed on.

To set to a color:

- Open the *Object Editor*.
- Select *oBack* from the *drop-down*.
- Select the *Col/Val* Tab.
- Drag a color to the *Back Color or Image* text box.

To set an image:

- Open the *Object Editor*
- Select *oBack* from the drop-down.
- Select the *Media* Tab.
- Click the *Media* command button.
- Open the *Misc* folder.
- *Drag and drop background.jpg* to the *Back Color or Image* text box.
Note: The Backs folder holds background images that could be used on the plot itself prior to the development of plot objects.
- Or, design your own background, save to your ProPrimer folder, and select that background.

Background Event Code

The *background*, *oBack*, has its own *event code* when clicked. The coordinates of the click can be displayed in the Status textbox with the following event code:

!STAT X = (oBack.x) Y = (oBack.y)

Using macro math and if-then function, a line of code can be executed. An example to run particular code if the background is clicked in the coordinates of X 80 to 90 and Y 20 to 30:

!IFTH [[[oBack.x),>,80]*[(oBack.x),<,90]]*[[oBack.y),>,20]*[(oBack.y),<,30]]],1,!BELL

This allows a graphic to be designed for the background with 'clickable' areas. Math operations used in the code are discussed in the next section.

VI. STAMPLOT MATH AND TIPS

StampPlot has math functions that can be used to manipulate data, use conditionals, and perform Boolean logic. StampPlot processes primarily through replacement, and, while not pretty, math operations were designed for speed by making it easy for StampPlot to recognize and act on math operations.

The primary structures are as follows:

- Macro Values: Used to recall a value or a constant and are indicated by parenthesis ().
- Math operations: Used to manipulate data, indicated by [] and using commas to separate terms and operators.
- Boolean operations: Used to perform Boolean logic of AND - *, OR - +, NOT - ', and XOR - ^.
- Conditionals: Equalities may be processed and conditional code ran.

Use of the !DEBUG instruction is highly recommended for testing your operations. This will show the results of code in the DEBUG/Immediate Window.

!DEBUG [(sldDur),*,10]

Also, this window can be used to manually enter values for testing:

500, 100

%101

Macro Values

Macro Values are used to recall information in StampPlot, be it a control value or other intrinsic values, such as the plot time. When used in code, the value of the requested data is replaced prior to the string being processed.

Here are some examples of values, you can test their values by using the **!DEBUG** instruction in the Debug/Immediate window:

(AINVAL0) to (AINVAL99)	The received analog value (or text as long as 1 st is analog) where 0 to 99 is the place of the value from a values string such as: 0,1,2,3,4,5,6,7,8,10,11,12,13,14,15,hello,bye
(PTIME)	The time into the plot in seconds.
(RTIME)	The real time of the plot in HH:MM:SS.
(objName)	The value of a control object, such as (sldDur).
(objName.T)	The top coordinate of a control object.
(status)	The text of the Status box
(PATH)	The path to the opened macro.

(DATAVAL0) to
(DATAVAL9)

A stored data value using **!SETD num, value** or **!SETD 0,100**
Note: The green text box at the top of the plot holds DATAVAL0, the rest are in the Value Window, Data Tab.

(DIGITAL)

The last digital string received (%10010)

(BIT0) to **(BIT99)**

The bit in the position identified in the binary string.

(AMAX) and **(AMIN)**

The analog (y) maximum and minimum on the plot.

Example with Math Operations

Here are the rules on math operations:

- 1 operation per bracket.
- Each term separated with a comma.
- Operations are from inner bracket out.

Example: $2 * 3 + 5$:

!DEBUG [[2,*,3],+,5]

Math operations can be performed virtually anywhere in code. Here are some examples of use:

!ACHN 0,[(AINVAL0),SIN],(RED)

To plot the SINE value of the 1st analog term in red.

!O metPhoto=[[(AINVAL0),/,3],FORMAT,0.0]

To show the value of the 1st analog value divided by 3 and formatted to 1 decimal place.

Note: The update value of the control object may also be set to:

[[(AINVAL0),/,3],FORMAT,0.00]

DEBUG "[, DEC PhotoVal, ",SIN]",CR

Send the value from BASIC Stamp and have StampPlot take the sine before use.

DEBUG "!READ [(sldDur),MAX,255]",CR
DEBUGIN DEC Dur

Read the value of sldDur with a maximum of 255 to be returned and accepted.

Boolean Logic

Boolean logic performs the logical operations of NOT, AND, OR and XOR on 1 set of data, is enclosed in brackets, and does NOT use commas:

- NOT: **[0']**
- OR: **[1+0]**
- AND: **[1*0]**
- XOR: **[1^0]**

Just like other math, it may be used most anywhere to perform the operations:

To turn the alarm lamp red if only the silence switch is off (NOT high), AND the incoming bit is high:

```
!O ledAlarm=[[swSilence]']*(BIT0)]
```

Or, it may be used as the update value for the ledAlarm configuration: **[[swSilence]']*(BIT0)]**

Equality Testing

Equalities may be used to evaluate a value and return 1 or 0 based on the truth, or used in the **!IFTH** instruction to perform conditional code. Allowable equalities include:

>, <, >=, <=, == (equal to), <> or ! (either for not equal to).

Example: Light *ledAlarm* based on analog value above or below 500:

```
!O ledAlarm=[(AINVAL0),>,500] Or, as an update value: [(AINVAL0),>,500]
```

Combine with Boolean logic to check if within a range – Not between 500 AND 600:

```
[[[(AINVAL0),>,500]*[(AINVAL0),<,600]]']
```

Running Conditional Code

The **!IFTH**, or If-Then, instruction can be used to perform instructions based on the result.

```
!IFTH val1, equality, val2, statement(s)
```

To plot in Red if > 500 and Blue is less or equal to 500:

```
!IFTH (AINVAL0),<=,500,!ACHN 0,(AINVAL0),(BLUE)
```

```
!IFTH (AINVAL0),>,500,!ACHN 0,(AINVAL0),(RED)
```

To run multiple lines of code if true, a (CR), carriage return, may be used between instructions:

```
!IFTH (AINVAL0),>,500,!ACHN 0,(AINVAL0),(RED)(CR)!BELL
```

Another way is to create a non-visible command button with code to be ran and manually run the event. Say for example a button called butAlarm is created and given the event code of:

```
!ACHN 0,(AINVAL0),(RED)
```

```
!BELL
```

It may be triggered using:

```
!IFTH (AINVAL0),>,500,!O butAlarm.Run
```

A third way would be to create a user macro routine in the User Defined area of the Macro Editor and call it using **!MACR .routineName**. The macro needs to be saved first and loaded to ensure the proper macro is being used. The macro file is opened and the routine is read each time it is called. *Note: macro routines are not held in memory and macro build will not create them. The file with original routine must be opened then updated.*

Macro routine:

```
Alarm:
```

```
!ACHN 0,(AINVAL0),(RED)
```

```
!BELL
```

```
ENDMAC
```

To call:

```
!IFTH (AINVAL0),>,500,!MACR .Alarm
```

Note that when using Boolean, the results still must be compared to a value of 1 or 0:

```
!IFTH [(BIT0)],=,1,!BELL
```

NOTE: Errors in math coding is one of the main problems of a macro not loading correctly. If you open the macro and it does not seem to process correctly, check to ensure the event code loaded as expected. If it did not, be sure NOT to save the macro, but open the macro manually with a text editor (NotePad) and try to locate and manually fix the error.

VII. VARIOUS TIPS AND TRICKS

Forming an Interface from the BASIC Stamp

The easiest way to create an interface from the BASIC Stamp is to create it using StampPlot, creating the macro, then copying and pasting the code into the BASIC Stamp Editor and adding DEBUGs. The Objects section of the macro and only important sections of the Initialization code should be used. We'll do a simple interface with a meter, the silence switch, and the alarm LED lamp. The built macro object code:

```
!POBJ Clear  
!PPER 70,100
```

```
' Set Back Color *****  
!POBJ oBack=7
```

```
' Analog Arrival Code *****  
!POBJ oAnalog.C=!POBJ Update(;  
!ACHN 0,(AINVAL0),(Black)(;  
!STAT Maximum is (AINMAX0)(;  
!ACHN 1,(AINMAX0),(GREEN)
```

```
' metPhoto -- OMMETER *****  
!POBJ oMeter.metPhoto=73.,91.,24.,25.,0,4000,0,3000  
!POBJ metPhoto.U=(AINVAL0)
```

```
' swSilence -- OIMGBUT *****  
!POBJ oImgBut.swSilence=82.,65.,7.,12.,comp\switch_h_0.jpg,1,comp\switch_h_1.jpg,1  
'-- Event Code  
!POBJ swSilence.C=~IWAV stapler
```

```
' ledAlarm -- OIMGBUT *****  
!POBJ oImgBut.ledAlarm=90.,64.,7.,10.,comp\led_red_0.jpg,1,comp\led_red_1.jpg,0  
!POBJ ledAlarm.U=(BIT0)
```

Convert this to a DEBUG in the BASIC Stamp Editor:

```
' Declare variables
LED_State VAR Bit
PB_State VAR Bit
PhotoVal VAR Word
Freq VAR Word
Dur VAR Word
Silence VAR Bit

PAUSE 1000
DEBUG CR,"!O Clear",CR,
      "!PPER 70,100",CR,
      ' Set Back Color *****
      "!O oBack=7",CR,
      ' Analog Arrival Code *****
      "!O oAnalog.C=!O Update(;)",CR,
          "!ACHN 0, (AINVAL0), (Black) (;)",CR,
          "!STAT Maximum is (AINMAX0) (;)",CR,
          "!ACHN 1, (AINMAX0), (GREEN)",CR,
      ' metPhoto -- OMEETER *****
      "!O oMeter.metPhoto=73,91,24,25,0,4000,0,3000",CR,
      "!O metPhoto.U=(AINVAL0)",CR,
      ' swSilence -- OIMGBUT *****
      "!O oImgBut.swSilence=82,65,7,12,comp\switch_h_0.jpg,1,comp\switch_h_1.jpg,1",CR,
      '-- Event Code
      "!O swSilence.C=~IWAV stapler",CR,
      ' ledAlarm -- OIMGBUT *****
      "!O oImgBut.ledAlarm=90,64,7,10,comp\led_red_0.jpg,1,comp\led_red_1.jpg,0",CR,
      "!O ledAlarm.U=(BIT0)",CR

PAUSE 2000
```

A few pointers:

- Start with a pause and CR to ensure 1st line is read correctly.
- End each line with a CR.
- Replace !POBJ with !O to save memory.
- Remove unnecessary decimals and spaces.
- Ensure event code is properly formed. Since event code can span several lines a line ends in (;) to indicate a continuation. Ensure the last line in the even code does NOT end with (;), such as with oAnalog.C code above.
- End with a pause to give the interface time to process the object creation.

Allow use of ENTER Key to Accept Textbox Data

The textbox control object allows the user to enter values or other data. To ensure that the user has completed entering data, the text box needs to lose focus, such as the user Tabbing off to highlight another control. This is difficult many times for the user to understand. To allow the user to use the Enter key instead, a hidden button that sets itself to focus when the user presses enter can be used.

- Drag and drop a command button object to the interface.
- For event code, add: **!O (ME).SF**

- The last button created will be used by default when Enter is pressed. The code for this button will set cursor focus to itself.
- Since it may have no actual use, make it hidden by moving its location to behind the plot. Making it non-visible will prevent its use, so that is not an option.
- If other buttons are added later, they will be the default. To ensure your focus button works still, build your macro, cut and paste the button code to the bottom of the Object text. Ensure the code is tested or saved and used before rebuilding again or you will need to repeat.

Setting the Tab Order

The tab order of the controls is in the order that they are created. To adjust the tab order, you will need to cut and paste the object code as required to place the objects in the order you desire. Be sure to test the code before rebuilding.

Allowing Hot Keys for Buttons and Checkboxes

Buttons and checkboxes allow the use of hot keys, that is, allowing the user to use ALT-letter to operate the control. When providing the text for these objects, precede the character you wish to use by & to make that the hot key. Such as &Silence for the checkbox would show as Silence and ALT-S would act as a click.

Manually Logging Data

While StampPlot can be configured to log incoming messages and data to a text file, often times it is desired to log other data or data in special ways. The **!LOGD** instruction can be used to write a string to a file. Automatic logging should NOT be enabled, **!SAVD OFF**, and if you wish to manually insert times, turn time stamping off, **!TSMP OFF**.

To log data, such as when data arrives, place code in the oAnalog event code, or send from the BASIC Stamp:

!LOGD (RTIME), (AINVAL0), (sldDur), (STAT)

This will log the real time, the value of the 1st analog value, the value of the duration slider, and the text in the Status textbox. Math may also be performed during logging.

To allow the user to enable/disable logging, create checkbox such as chkLog. The code would then be (placed in oAnalog or other location – not in chkLog code):

!IFTH (chkLog),=,1, !LOGD (RTIME), (AINVAL0), (sldDur), (STAT)

A textbox can be used to allow the user to enter a name for the file. The event code for the textbox, named txtFileName, would be:

!NAMD ((ME)).txt or !NAMD ((ME)).csv

Note: code to manually rename the file on initialization to ensure it matches should be inserted into the Initialization code after !MACR .Objects

!NAMD (txtFileName).txt

A button could then be used to open the log using the associated application (NotePad or Excel®) with event code of **!APPA (NAMD)**.

Storing Temporary Data in StampPlot

Now and then the need arises to store data in StampPlot for later use, for accumulation of values, or other uses. There are 3 ways to allow this.

- Create hidden textboxes to hold the data.
- Use the 10 (0-9) built in data values to hold the data which are accessible via the 'Value' window, data tab. These values are set using **!SETD num, value** and returned using **(DATAVAL#)**
To save a value:
!SETD 1, (AINVAL0)
or to accumulate:
!SETD 1, [(DATAVAL1),+, (AINVAL0)]
or to set from controller code:
DEBUG " !SETD 1, ", DEC value, CR
To Display:
!STAT The accumulated value is (DATAVAL1)
- The 3rd way is to use the 100 data markers in StampPlot (%m0 to %m99):
To save a value:
!MATH %m1=(AINVAL0)
or to accumulate:
!MATH %m1= [%m1,+, (AINVAL0)]
or to set from controller code:
DEBUG " !MATH %m1=, ", DEC value, CR
Note that in code parenthesis are not needed to read: **!STAT Accumulated value = %m1**

Actions on Plot Reset

On reset it may be required to clear accumulated, reset values, or perform other actions. In the Object Editor's oReset section, write and code you wish to occur on a plot reset:

```
!SETD 1,0  
!MATH %m1=0  
!PWAV flush.wav
```

Saving and Recalling the Configuration

As the end user adjusts setting and other controls, it may be beneficial to provide them a means to save and recall their settings. Using buttons for save and load, the following code can be used to save the configuration and reload it when desired.

```
'Save settings  
!SSET myTestConfig
```

```
'Load settings  
!GSET myTestConfig
```

To query the user to ensure they really intended to save or load the settings, a pop-up request box can be used:

```
' Request user input and store in %m99
!MREQ N,10,Overwrite configuration settings?\nY\N,99
'Verify entry
!IFTH %m99,~,Y,!SSET myTestConfig
```

~ is an equality to compare regardless of case (y or Y).



VIII. CONCLUSION

We hope you found this tutorial instructive and informative. We are sure there many other questions that may be unanswered, so always feel free to contact us concerning your development of macros.

IX. REVISIONS:

- 01/03/06 – **Minor spelling/grammar corrections.**
Page 19: Corrected code for
`!O metPhoto=[[(AINVAL0),/3],FORMAT,0.0]`
- 01/08/06- **More general corrections/clarifications.**
Page 15: Corrected code for:
Image1: led_red_1.jpg
Fixed Table of Contents entry on Storing Temporary Data.
- 01/11/06- **Correct BS2 code for copy & paste in Acrobat Reader.**