The program, "DEMO.BPE", illustrates the basics of how to do data logging on the OWL2pe, using the PBASIC language on the BASIC Stamp BS2pe microprocessor. This demo has features we like to include in any data logging project, and it can serve as a template that you can freely adapt to build your own application. Note that the demo program evolves over time, and there are different versions of the program to go with different terminal boards and sensors that we supply with the OWL2pe systems. However, the changes are not so great and their import should be quite clear from notes in the program listing.

If you are not already familiar with the BASIC Stamp please refer often to the Parallax manual and other references, of which there are many. The programs here are written in Parallax PBASIC version 2.5, which is part of the IDE (integrated development environment) version 2.1, available as a free download from the Parallax web site in the downloads sections at this URL:

http://www.parallax.com

The BASIC Stamp manual is downloadable as a PDF file, or a hard copy can be purchased online. There are other resources in the downloads and links section, including FAQs, application notes, and several years worth of articles from the Nuts and Volts magazine, and threads from the very active mail list-server. The current list server and archive is reachable at <forums.parallax.com>. And finally, there are tutorials, data sheets and links at the EME Systems web site:,

http://emesystems.com/BS2index.htm

and

http://emesystems.com/OWL2pe.htm

It is important to note that any program that runs on any BASIC Stamp, as described in any of the above sources, will also probably run on the OWL2pe. For example, the following starter program will print "hello" on the debug terminal screen and flash a red/green led attached between i/o pins p2 and p3:

```
' {$STAMP BS2PE}
' {PBASIC 2.5}
DEBUG "Hello",CR  ' message
LOW 2             ' led on p2
LOW 3             ' led on p3
DO
  TOGGLE 2        ' change output state of p2
  PAUSE 250       ' pause 1/4 second
  TOGGLE 3        ' change the output state of p3
  pause 250       ' pause 1/4 second
LOOP              ' do it forever
```

The demo data logging program is longer than that of course, and is spread acros 4 program banks. The OWL2pe uses one of the more advanced multibank chips, whic has 16 banks of memory for program and data. Having the program spread across 4 banks leaves room to grow and add functions in each bank, and there are in

addition 4 empty banks available for program code, and 8 banks above that for special data. The OWL2pe code in each bank is a functional unit.

bank 0: initialization and user menu
bank 1: main program loop for scanning channels and performing actions
bank 2: routines for accessing and maintaining the AT45DB041 data logging memory
bank 3: menu system for entering operating and calibration parameters
banks 4-7: additional code for special sensors or special actions
banks 8-15: program configuration data

Code execution switches from one bank to another using the RUN command, and variables remain intact across the transition. The demo program has the mechanism for interbank program calls in place. This is the mechanism used in all of our data logging programs

The OWL2pe has several peripherals above and beyond the ordinary BASIC Stamp. These include the analog to digital converter, the real time clock, power supplies, and several memory resources, including 256 kwords of logging memory and battery backed parameter RAM. The technical manual goes into detail about the resources. Here they are in summary:

--program and system eeprom memory -- program code and DATA is uploaded to this memory from the PC. Some of this memory that is not used for program code can be used at run time to store auxiliary data, using the STORE READ and WRITE commands. This memory should not be used for rapidly changing data, because it can wear out, albeit after 100000 to 1 million write cycles.
--main RAM memory, 26 bytes that can be written and read without any concern for wearing it out. PBASIC allows very flexible access to this memory as bits, nibbles, bytes, words and as explicit or implicit arrays. There are an additional 12 RAMbytes that shadow the state of the 32 i/o pins. A couple of the 26 bytes have specific names and uses that are related to the overall function of the system, but the demo program uses most of the main ram variables for many different purposes. The scratchpad RAM is used to hold on to the more sensor or task-specific values.
--127 bytes of scratchpad RAM that also can be written and read without any concern for wearing it out. This memory is not as flexible as the main RAM, in that it can only be accessed through PUt AND get commands. Nevertheless, this memory is very important in the operation of the data logger, to serve as a buffer for the data that will be logged from time to time, for the accumulators used for averages, and for other operating parameters. The demo program works one task at a time, to deal with one sensor and then another, It stores the values appropriate for each sensor in the scratchpad RAM, reads them into the main RAM with their task is front and center, and then puts the altered values back in scratchpad when the task for that sensor is finished, and then moves on to the next task. The sensors hardly ever have their own variales in the main RAM.
-- logging memor
-- clock memory

The program listing for each bank is tagged with notations that will be reference points for this document. These are points where you might want to modify the template yourself to make it do what you want, to add more sensors, to change the timing etc. etc.

Overview:

-- The code in bank 0 initializes the variables and sends out a main menu that appears on a terminal screen attached to the Stamp serial port. The menu shows the user a message with information about the program version, the system power supply voltage and temperature, essential settings such as the data logging interval, a key to the data fields, and command prompts. The program in bank 0 processes the choice that the user makes from the keyboard, You may want to change this menu code to add or subtract information from the message or to add special menu options, such as a password. Bank 0 also contains DATA statements that hold parameters that govern the operation of the system, such as the data logging interval, the low battery threshold, the scanning interval, etc. You can change these items directly in the program code, but most of them can also be modified at run time by the user when they choose the "Parameters" or "Clock" options from the menu. That causes the program to jump to bank 3, where the parameter entry menus are located. The memory management options like "Offload" or "NewRun" cause the program to branch to bank 2, where the routines that address the AT45041 logging memory are located. The options that scan the input channels, "Run", "View" and "Zoom" cause the program to branch to bank 1, where the main sensor scanning and control loop is located.

-- The code in bank 1 is the main operating loop that scans the channels and takes data and logs it in memory at intervals. The program will spend most of its operating time in bank 1, with occasional calls to bank 2, where the routines that put the data in the AT45DB041 logging memory are located. Bank 1 is where you will add code in line to process the signal from additional sensors and to implement control functions as needed. Also, you might need to do special timing or tricks performed in the regular course of operation. It is bank 1 that will generally require the most work when modifying the template program. This bank also contains DATA statements that contain certain calibration factors that are needed for the sensors and other operations. It is often useful to be able to change those at run time. The "Parameters" code in bank 3 can be used to do that when it is required. That code, which modifies the parameters stored in bank 1 eeprom, is called from the main user menu in bank 0. Bank 1 makes use of the scratchpad RAM in the BS2pe as a buffer to store the readings that are to be logged from each channel. That can include raw readings as well as processed data like averages as appropriate for each sensor. When the time comes to move the contents of the scrachpad buffer over to the log file, the program in bank 1 makes a call to the Log routine in bank 2, which handles the pointers to the next free location in the log file, puts the data there, and updates the pointers, and then returns control back to the scan routine in bank 1. The sratchpad memory is also used by the bank 1 code to hold

various intermediate values that will not be logged, and also system variables like the logging interval that are used at specific points in the program (getting data from the scratchpad is faster than reading it from the eeprom). Data logging can require quite a few more variables than are avaialble in the BS2pe main RAM. Therefore the stratagy in the demo program is to name locations in the scratchpad memory, When needed in a routine, the current value from a named location is read into a general purpose variable in RAM. The generic variable is used with its generic name, not an alias, because there would simply have to be so many aliases as to be confounding. Usually these usages that assign a location in scratchpad to a generic variable are local within a relatively short segment of code, so that the local meaning of the generic variables is self-documenting. At the end of each scan the program can either start a new scan immediately (Zoom mode), or sleep for a while (View or Log modes), which conserves battery power. The interval between scans is a parameter that can be set by the user if desired. Time to log is determined by matching the clock reading to a logging interval, which sets a program flag bit when the time to log has arrived. At the end of that scan, the logging routine in bank 2 is called, and upon return the scanning routine can update things like averages or minimum or maximum values. In Zoom and View modes, the data readings are displayed in real time on the terminal screen, and that is determined by flag bits that are set by a configuration byte and by choices from the main menu.

-- The code in bank 2 comprises the routines necessary to access the AT45DB041 logging memory. Since the low level routines are rather complicated, the routines in this bank are set up to perform the most commonly required tasks, such as logging a data record, reading back the data records, maintaining the pointers, and so forth. In most cases you should be able to use the template to call those routines. Many of these routines are called from the main menu, to read out the data file, to start a new data file etc. The number of fields in each logged record, and the formats to use for the readout of each field, are stored in DATA statements in bank 0. Those must be updated along with the program as the scheme grows in bank 1. There are format specifiers for example, to indicate that the logged data is an unsigned integer, or a signed fixed point value with two decimal places, etc. The data logged in each field is one word, 16 bit binary without embedded format information. The format is in one-format-to-one-field data correspondence up to the number of fields per record. The one thing you might need to change in this bank would be the formats, if your data needs a format that is not included in the default list. Each format available for the offload routine in bank 2 is assigned a number, and there are plenty of numbers left for your custom formats. The data logging routine in bank 2 is called from the scanning routine in bank 1 when the time comes to log data, and the routine in bank 1 passes the data to be logged in a buffer located in the scratchpad RAM. The data offload routine in bank 2 is called from the Offload option in the main menu in bank 0, and the format codes corresponding to the data fields are passed from bank 0 to bank 2 in the scratchpad memory. (access to scrathpad is much faster than access to the eeprom) Other menu options in bank 0 transfer control to routines in bank 2, to

accomplish chores such as starting a new run or dumping diagnostic data from the logging memory. These routines use pointers to record the position in the memory of the start of the current file and the position for logging the next record. The pointers are maintained in the DS1307 clock chip, which is battery backed so that the pointers will not be lost in the case of a reset or power failure.

-- The code in bank 3 presents the user with a menu that allows them to change the system operating parameters. That includes the logging or scanning interval, and also calibration constants for sensors. You may well need to add or subtract material from this bank. It is quite straightforward coding to read in the old value of the parameter from bank 0 or bank 1, and write back a new value if the user chooses to change it. Bank 3 also holds the routines that allow the user to set the real time clock.

Bank 0 specific notes:

XXX0.0
This line:
'{$STAMP BS2pe, demo-a1.bpe, at45c.bpe, demo-a3.bpe}
is a message to the compiler that this program includes 3 additional banks as part of the same project. When you start a new project, you will give these component programs their own names. I like to include the bank number as part of the program name. Note that the routine at45c.bpe is the exception, as I almost always use the memory routines without modification. However, you can change the name of that bank also. The important thing is that they must be linked in the line with the '{$Stamp directive. See the Parallax documentation for more information about the directive. The line,
'{$PBASIC 2.5}
is also a directive to the compiler, that tells it that we are going to use the latest version of the Parallax tools. The rest of the lines starting with ' are comments, good to help remember what the program is for and important revision notes.

XXX0.1
These DATA statements hold important values for the operation of the program. Comments after each one should make evident their purpose.

Some of these can be changed by the user at run time, by invoking the "Parameters" command on the main menu, which calls routines in bank 3 that prompt the user for choices. If you need a new parameter, you can add it here and allow the user to modify it by following the template in bank 3.

The parameter recsiz0 is the number of words that will be entered in the log for each record. You must take care that this number agrees with the number of channels that are scanned in bank 1. In this example, there are 5 words per record, that is, words consumed each time the OWL2pe logs a data record. Three words are used for the date and time, one word for the system battery voltage

and temperature (packed together into the one word), and one word is used for the voltage from a potentiometer. If you add more fields per record, you will increase this number accordingly.

The formats specify in order how data in the log file (which is stored in binary format), is to be transformed as it is read from the data file for offload. For example, this:

```
  formats  DATA 1    ' date/time format
           DATA 9    ' volts/temperature two bytes packed
           DATA 3    ' volts from potentiometer
           DATA 1    ' Celsius temperature
```

Each record consists of 3 words for the date and time, one word with the battery voltage and system temperature packed in one word, and one channel reading volts from a potentiometer, and one channel of Celsius temperature. When the program in bank 2 reads out the data, it uses the format specifiers to convert the binary data into scientific units, as it sends the data out the serial port as ascii strings that can appear on a standard terminal or be captured in a log file.

Date/time format #1 outputs the date and time information as one field in the form 2004/10/25 13:42. Since Excel can understand that format directly, we have found that it works for most purposes. If you want a special format, you can follow the template in bank 2 to add it and assign it a number. Format 9 is special for the packed battery voltage and temperature that we always include in the data set. The two sensor readings are treated with three decimal places for volts and 1 decimal place for temperature. Here are the numerical formats assigned.

```
   0) signed integer  -xxxxx
   1) signed one decimal place  -xxxx.x
   2) signed two decimal places  -xxx.xx
   3) signed three decimal places  -xx.xxx
   4) signed four decimal places   -x.xxxx
   5) unsigned integer  xxxxx
   6) unsigned hex  hhhh
   7) ascii binary  bbbbbbbbbbbbbbb
   8) null (empty field, TAB separator only)
   9) packed (special) for battery voltage/temperature)
```

XXX0.2

Here is where you declare the use of the i/o pins by name. In this example, pins 4 and 5 are dedicated to a bi-color led indicator. Pin 0 is named "rin" to be the input for a rain gage.

XXX0.3

The RAM variables are assigned according to a scheme that preassigns 8 of the 13 RAM words exactly the same in all of the BS2pe program banks. Those are the variables wsx, flags, wpe, ww, wx, wy, wz, wj. Of those, only "flags" has one

specific purpose, to hold important operating states of the program. "wsx" serves a specific purpose for cross-bank program execution, and locally appears as utility bytes, cat and dog. Variables "wpe" to "wj" serve as general purpose variables. The rationale is this: Data processing and logging often requires quite a few variables, more than the thirteen that are available in the main RAM of the BS2pe. The BS2pe does have 128 additional bytes in the scratchpad ram, and it is there in scrachpad that we assign names for the variables and accumulators for data acquisition. See XXX0.6 and XXX1.5. When the program needs to process data, it reads the necessary values and parameters from the named scratchpad locations into the general purpose words, wpe to wj, and then executes the code using the general purpose names. This code is usually local so that the association between the general purpose variable and the named value or parameter is easy to follow. We felt that this is less confusing and makes it easier for a program to grow, than it would be if the program had to define many many aliases.

After wj there are 5 additional words available for other assignments, the named variables. Here in this bank they are used for char, ADch, ix, jx. You can add other variables necessary for your application. Note that you must give special thought to variables that need to be maintained across bank boundaries. See www.emesystems.com/BS2sx.htm.

The named aliases here redefine general purpose variables as a buffer for the clock variables. You may define aliases here in the same manner. Note that alias names are also defined with the general purpose variables. You can define aliases anywhere you want in a program.

XXX.4
These are system constants. You may need to change the system port baud rate, or the number of seconds that the system port waits for a command before it drops back into logging mode. These constants also define the number of pages and page size for the AT45 logging memory. You would only change these if for some reason you need to reserve some of the logging memory for a special purpose.

XXX.5
These constants give names to the memory available in the clock chip. Here it is used to store the important pointers, to the next free location in the log memory, and the pointer to the start of the log file. It also stores the backup pointers, to the previous log file. You may need other values stored in this RAM. For example, you might need to store the total rainfall or the total operating time of equipment. The battery backed clock ram is a good place to store this kind of variable, the kind that changes quite often, but also needs to be protected against erasure, resets and power failure.

XXX0.6
The scratchpad RAM is used to hold the values acquired by data acquisition. This is the buffer defined starting at location zero. The buffer is not used in

that way in bank 0, so the names are assigned in bank 1. At the top end of the scratchpad are stored several variables that are used often, at particular points within the program. These values are read into the general purpose variables at those points where they are needed. Several of these are copies of the DATA that was covered under XXX0.1. The values are transferred from the eeprom over to the scratchpad at startup. That makes access quicker (GET and PUT commands for the scratchpad are quicker in execution than READ, WRITE and STORE commands to eeprom). You can add additional variables here. Be sure they are defined the same in all of the BS2pe program banks. The lowest address is "stacktop", which defines memory that can be used in the fashion of a stack. (This demo program does not use that capability, though). The location "back" serves as the return address for cross-bank calls.

XXX0.7
The BS2pe/40 has 32 input/output pins, divided into two banks, that are selected by the commands MAINIO and AUXIO. When the BS2pe is powered up or reset, all of the pins come up initially as inputs. The first order of business in program execution is to set those 32 pins to a desired state. Your application may have special requirements that you will want to set here.

The DIRS variable sets the direction of 16 pins as input or output, and the OUTS variable sets the outputstate of pins to either high or low. There are two DIRS and 2 OUTS statements, one each for auxio and mainio, 32 pins total.

The OWL2pe uses 12 of the auxio pins for its internal functions. See the technical manupal for detail about what these pins do. In particular, the initial state of the power supply Vx is set by bit c, in this case it is high, ON. Bit 7 high turns on the power to the analog to digital converter and 4.096 voltage reference. Those power supplies could be left off, in some applications. Note that the Stamp has many ways of controlling the state of the individual pins. For example, the command, "HIGH vxpwr" will turn on the power and it will leave bit c in auxio DIRS and OUTS both equal to one.

Four of the auxio pins, x0 to x3, are available for general purpose use. However, they are not available on the OWL2pe top board, so this program simply initializes them as low outputs .

Similarly, in this program, all the 16 pins on mainio are going to start out as low outputs. This is a point where you will want to alter the initial state to suit the application.

It is important to initalize all unused pins that are not connected to anything. They should be made low outputs. Otherwise the pins "float" and this condition can lead to excess current consumption. It does not harm the OWL2pe, but often it is important to conserve battery life.

We usually use a bicolor, red/green led connected between two pins to allow the program to indicate its internal state. The pins ledred and ledgrn are defined in the pin definitions, see XXX0.2.

XXX0.7.1
Important operating values are transferred from the eeprom (DATA statements) over into the faster and more flexible scratchpad. You may want to add other values to this list.

XXX0.8
This code displays a menu on the terminal screen and waits for user input. If the user does not respond, then after a while, the program reverts to its logging mode.

As part of its intitial message, it reads and displays the clock time, the OWL serial number, the station ID code, the logging interval, and the current number of records stored out of N possible in the log file. It also reads the current logger temperature and battery voltage and displays those. And it also displays a line showing the allowable user responses.

You should at least change the name of the station that appears in the message, from "DEMO-a" to one more descriptive.

XXX0.8.1
If an LCD screen is attached, its initial message would go here.

XXX0.8.3
This is where the input is received from the user. It is possible to add a "password" at this point using the BASIC Stamp "WAIT" modifier to have it wait for a certain string before accepting the command. Note that there is a timeout on the SERIN command, and the destination in case of a timeout is the default logging routine.

The commands are decoded in the line following, using a lookdown and then a branch to the routines selected by user input.

XXX0.9
The remainder of the routines in this bank are subroutines, and the comment with them should provide a good idea of what they do and how.
Many of them are crossbank calls. Here is how the branch to the data offload routine is handled:

```
  PUT back,$00
  lola=$23
  RUN lolarun
```

The RUN command in conjuction with the variable, lola, causes the program to branch to execution in bank $2 at point $3. The bank 3 code also understands

the variable lola, and the pointer to a routine indexed as number $3. The variable location, back, is kind of like a single level stack for return from cross bank calls. When the offload routine finishes, it will use a similar crossbank mechanism to branch to execution in bank 0 at point 0, which is effectively back at the main menu in this bank zero.

BANK 1 specific notes

XXX1.1
Data statements here hold calibration constants for use in sensor processing. For example, when the code to read the temperature is executed, it also reads this calibration constant, TCiCal, and adds it to the raw sensor reading. The value in this case is a word variable that can be either a positive or a negative correction. The number and type of correction factors depends completely on the sensors attached to the OWL. Since the correction factors are entered as DATA (rather than as CONstants), it is possible for the user to modify those factors from the menu that runs in bank 3. The first three data bytes are reserved for the Stache field programming device.

XXX1.2
The auxio and mainio pin assignments here will ususally be identical to the ones in bank 0, and also the general purpose RAM variables from wsx through wj. Time variables (seconds .. year) are alias names for the array of bytes from ww0 to wz1. Many of the actions in this bank will involve getting readings from the AD converter using the "getADC" subroutine. Variable wx will hold the result of the conversion. Other general purpose variables will be used to hold calibration read in from eeprom, and for computations if necessary. The variable wj will hold the number of samples, where averaging is done. The use of the general purpose variables is quite flexible, as outlined above.

XXX1.5
There is a buffer for the readings starting at location 0 in the scratchpad. The first 6 bytes, 3 words, are reserved for the date and time, and following that starting at location 5 are the battery voltage "bat0" and internal temperature "degCi", and following that named locations for the data acquired from other sensors in the course of data logging, "degCair" and "mVpotentiometer". You would modify and add to this list any time you change the configuration of the sensors. Each sensor reading will normally require one word of eeprom storage. When data is to be logged in the AT45DB041 flash memory, a call is made to bank 2, where program code moves the data from the scratchpad buffer in the order that it is there at the time of the call, into the next free location in the log file. It moves the number of words specified by the bank 0 parameter "numrecs". So the number of words reserved in the scratchpad buffer must match up with the numrecs.

XXX1.6
In scratchpad locations after the data buffer, there can be additional locations set aside for intermediate computations accumulations and other factors that are not going to be logged.

XXX.7
The high addresses of scratchpad should match up with the names in the other banks. These locations hold some important system variables like the return address for cross bank calls, and the logging and scanning interval. You can add additional variables if necessary. The lowest address is named "stacktop" to indicate that is a free for general purpose stack type operations, however, this demo program does not do anything of that sort.

XXX.8
The main program in this bank can be entered in one of two ways, either from the main menu after power up, or a user menu option. Or upon return from logging data via a call to code in bank 2. The initial BRANCH instruction distihguishes these two possibilities. The return from bank 2 is distiguished by a value of lolago=1, so the branch instruction takes the program to the label "afterlog". When lolago=0, execution falls right through to the initialization "scan0" and the top of the main scan loop. The initialization reads the clock (which has the effect of synchronizing certain flags that are used by the program), and it sets the flag "first", which will be reset to zero at the end of the first time through the scanning loop.

XXX1.8.1
Starting at label "Scan1" is the main scanning loop that repeats over and over. That loops extends down through label "scan9". Overall, the loop reads the clock, then reads the system battery voltage and internal temperature, then reads the sequence of sensors particular to this data logger, then if it is time to do so, it calls the log data routine to transfer the data buffer into the log file and to re-initializes accumulators, and then it either jumps (at label scan7) right back up to do it all again, or else (at label scan8) it sleeps for a while in a low power state, and then loops back.

In more detail now:

At scan1 the program reads the real time clock, and, if vuflg=1, it displays the clock value on the terminal screen. The subroutine "readclock" checks to see if it is now time to log data, and if if so, it sets the flag "logflg=1" and puts the current date and time in the first 6 locations of the scratchpad. The readclock routine also checks if the clock has rolled over to a new minute, and if so sets the flag, "minflg=1" as certain actions will be taken on the minute.

The routine to grab the battery voltage (Vin supply to the OWL2pe) first selects the AD channel 10, then calls the AD converter subroutine. That returns a value

from 0 to 4096 millivolts in variable wx. On the OWL2pe, this raw value comes from a 1/4 voltage divider, so the raw reading needs to be multiplied times 4 to give 0 to 16000+ millivolts, and then divided by 100 to give a reading from 000 to 161 in units of 1/10 volt. That value from 0 to 161 is stored in the scrachpad data buffer at location "bat0". It is also displayed (if vuflg=1) on the terminal screen, with the format xx.x. The code retrieves from the system area of the scratchpad, the voltage that constitutes a low battery, compares it with the current voltage, and sets or unsets the low battery flag accordingly. This is used by the demo program to flash the red led when the battery is low, or green when it is okay.

BANK 2 specific notes
The AT45DB041 logging memory
The operation of the data logging routines is described in a separate document.