Formatting Variables in C-Max 2.0

One of the many new features in C-Max 2.0 is the enhanced formatting now available for variables. This new capability is available in two distinct areas of "variable" usage: in embedded screen text objects on a Leopard screen and in ASCII strings, where embedded variables were not even possible before. This second possibility also means that Ocelot users can take advantage of this new feature. In the case of ASCII strings, embedded variables and formatting can be used with the master controller's serial port, a slave's serial port, and the serial bobcat. Note: the serial bobcat needs to have firmware version 6 or higher to support embedded variables.

This new feature is also the subject of a minor backward compatibility issue if you have an existing (pre C-Max 2.0) Leopard application that has embedded variables in screen text objects. You will need to add a "u" after the field width number to duplicate the former display behavior (ie: a "%4" must be modified to "%4u"). Failure to do that will result in a blank field where the variable should be, but before doing just that, you will want to look at the new formatting capabilities presented here and possibly optimize the way your variables are displayed.

This application note will show two sample programs where variables are formatted to obtain the desired representation or display. The first example will show formatting of Leopard screen variables while the second example will embed and format variables in ASCII strings, which can be used with a Leopard, Ocelot, or serial bobcat. But first, let's take a closer look at variables themselves.

Controller Variables

What exactly do we mean by formatting variables? To help you better understand we will start by a short description of your controller's variables in their natural form. You will then understand the formatting capabilities more easily.

If you are already familiar with your controller's variables, you will have noticed that they are always an integer with values ranging from 0 to 65535. This apparently odd range is because they are internally represented by 16 bit binary memory locations. In a binary system, each bit (from right to left) represents an increasing value of a power of two. Thus the first bit is the 1's , then the 2's , the 4's , the 8's and so on until we get to the 16th bit which is the 32768's. This is just like the more familiar decimal system where we have the 1's, the 10's, the 100's, etc. The maximum value that can be represented is when we have each bit set. Therefore if we add 1 + 2 + 4 + 8 +....+ 32768 = 65535.

One thing you may have noticed is that in C-Max, there are no negative numbers. The above description of your controller's variables is called an "unsigned 16 bit integer". In the past, this was sometimes annoying when we wanted to display things like temperatures in an embedded variable on a Leopard screen...if the temperature went below zero. If you did try to make a variable go below zero by subtracting or decrementing it past zero, you might have noticed that it "rolled over" to 65535 and then continued to decrement downwards. This is very similar to a mechanical car odometer that counts up to 99999 and then rolls over to 00000. If you then turn it backwards it will show 99999 again and keep decreasing. In binary number usage, there is a convention that allows us to consider a 16 bit number as a "signed 16 bit integer". In this case, the highest bit indicates whether the number is positive or negative (1 = negative, 0 = positive) and the remaining 15 bits give us the number's value. We still have the same total range but it is now "shifted" to represent numbers from –32768 to 3276. In that system, called "two's complement", the bit pattern for 65535 is equal to "–1", 65534 = "-2" and so on. This means that The rolling over of your controller's variables from 0 to 65535 when subtracting or decrementing is already the correct behavior for signed 16 bit integers, but there was no means to visually represent them on your Leopard screen until now. This signed representation is obtained when you use the "d" formatting option (as described in the next section).

Although you can now interpret and display a negative number on your Leopard screen or in an ASCII string, they are still positive-only in the C-Max instruction logic. This can cause a few surprises if you are not careful with "greater than/less than" type instructions. For example, if you are looking for a temperature as being between –10 and +10 deg F for a true condition, you will need to consider it logically as "IF < 11 OR > 65525". You cannot use an AND because you are looking for a number that is possibly at either end of the positive number scale.

Formatting Options

Here is a complete list of the formatting options available for screen objects and ASCII strings (Fig 1). If you are familiar with the "C" programming language and the *printf* statement, then these options will already be familiar to you.

```
%d    signed decimal
%u    unsigned decimal
%o    unsigned octal
%x    unsigned hexadecimal with lowercase for letters
%X    unsigned hexadecimal with uppercase for letters
%c    single ASCII character
%%    to display a "%" sign literally
```

Fig. 1

The above formats can be modified with one or more of the following options (these are inserted between the % and the letter) (Fig. 2):

```
-        left justify
+        always display a sign (+ or -)
0 (zero) pad with leading zeros
```

Fig. 2

Finally, you specify the width of the field reserved for the variable (including any + or – sign) using a number between the % sign (or after one of the above modifiers) and the format letter. You can also specify a minimum number of displayed digits (padded if necessary with leading zeros) by adding a decimal point and a second digit.

One thing to remember about field width specifications: these specify a minimum width, padded with blanks or zeros as specified. If the variable should happen to be or become wider then the specified width number, the field will automatically expand and push any text after it to the right. However, if the variable then becomes shorter again, you might get unwanted characters displayed past the end of the text object. You can easily avoid this by simply specifying a field width wide enough to accommodate the largest variable value that can be expected.

Here are a few examples of format specifications, a variable value, and what it would look like once formatted (Fig. 3).

Format	Variable	Appearance
%6u	1234	1234
%-6u	1234	1234
%3d	65535	-1
%+3d	12	+12
%03d	7	007
%5.3u	17	017
%c	65	A
%3u%%	68	68%
%4X	65535	FFFF
%4x	65535	ffff

Fig. 3

Screen Display Example

This first example will be of a Leopard screen display showing various types of information using formatting to optimize each type of data presented. Let's create a general information screen that will be displayed when the Leopard's home control menus are not being utilized by the user. We will show the current time, date, inside and outside temperatures, and relative humidity (these last three values obtained from temperature and humidity bobcats). Let's examine each type of data and the formatting that will be applied:

Time – We want to use the largest font available to display the time in a digital clock format. Since the largest font can show numbers only, we will use two such fields side by side so that a "colon" can be created in between by using two single character text objects each with a lowercase "o". The left side number field will show the hours using a simple "%2u" format to allow for two characters. The right number will be the minutes and will use a "%02u" format string to allow for two characters and also pad the leftmost column with a zero if the minutes value is a single digit. This is to get a display of "12:02" instead of "12:   2".

Date – We want the date to show in the form "Nov 9,2002". The Leopard does not have variable text string functions but we can simulate it by creating single character fields side by side and then loading the ASCII value of the character for each letter we want to make up the string. We will use three "%c" format strings and overlap them in the screen definition file to get the desired effect. Note that even though creating the format string needs two characters ("%c") in the touchscreen editor, we know this will display as only one character, so it is ok to overlap the text objects in the editor. The day of the month uses a "%2u," format to reserve two character positions and display the comma. The year uses a "%4u" format for 4 characters.

Temperature – This is a more classic format string using text with the variable format specification inserted at the appropriate location. For the inside temperature, we use "Ins. Temp:%3d F." for up to three digits (if it gets really hot...). For the outside temperature we use "Out. Temp:%+4d F." because we want up to three digits as well as a "+" or "-" displayed, which uses up one digit space.

Relative Humidity - This is expressed as a percentage so we will use a string in the form "R.H. %3u%%" which is really a 3 digit display (for up to 100%) followed by the percent symbol itself.

Here is a list summarizing all the properties of our text objects (Fig. 4):

Obj#	Format string	Font Size	Embedded Variable
1	%2u	Lrg # only	10
2	%02u	Lrg # only	11
3	o	Large	(none) used to make "colon" top
4	o	Large	(none) used to make "colon" bottom
5	%c	Large	12
6	%c	Large	13
7	%c	Large	14
8	%2u,	Large	15
9	%4u	Large	16
10	Ins. Temp:%3d F.	Large	17
11	Out. Temp:%+4d F.	Large	18
12	R.H. %3u%%	Large	19

Fig. 4

Here is a screen capture of the touchscreen editor showing the relative positioning of each text object (Fig. 5). Getting the best looking display sometimes requires experimentation and testing with the actual Leopard. Notice that overlapping causes some of the objects to be only partially visible:

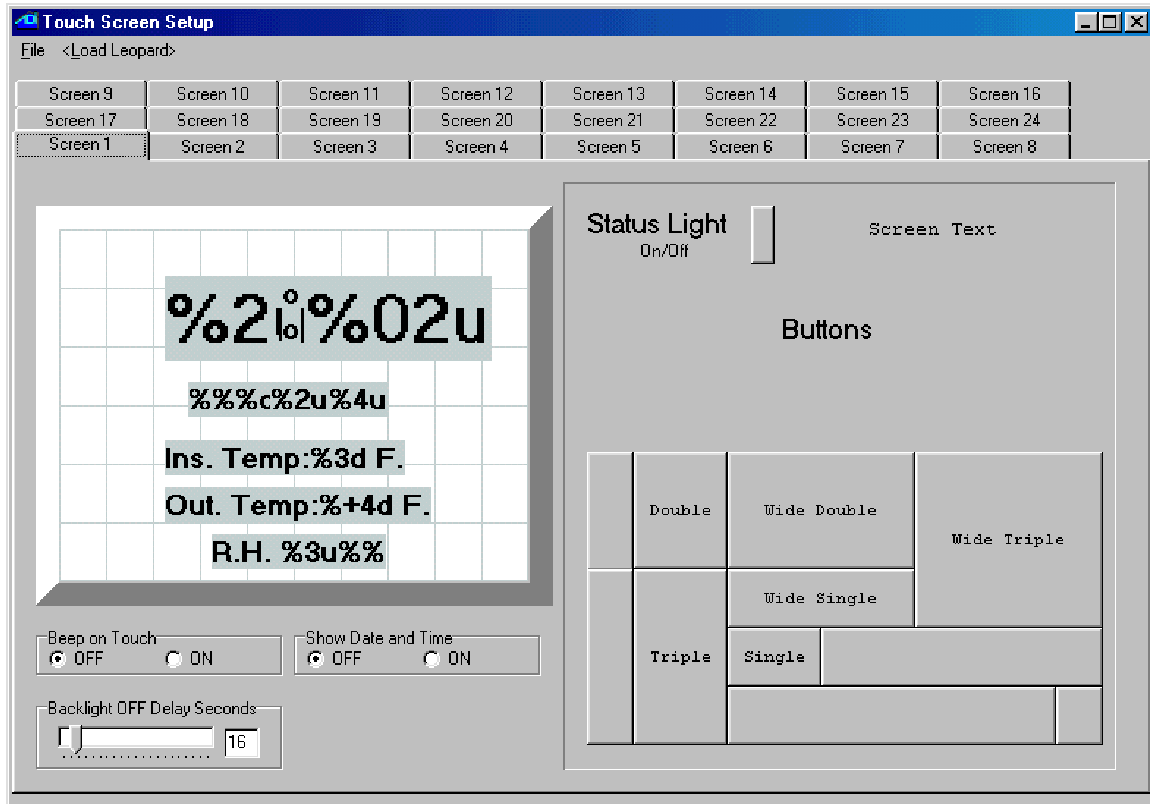

Fig. 5

The C-Max program used to produce the various data values displayed is shown below (Fig. 6). Only the relevant program lines are shown, omitting the other screens and buttons used to switch to the information screen:

```
0001 -                                       // ** Time of Day **
0002 - IF Time of Day is > 00:00             // whether time = 0:00
0003 -     THEN  Load Data to:  Variable #0  // or not, store in 0
0004 -     ELSE  Load Data to:  Variable #0  // (minutes since 0:00)
0005 - IF Variable #0 is = Variable #0       // on every pass
0006 -     THEN Variable #1 = Variable #0    // copy time in var 1
0007 -     THEN Variable #1 / 60             // calculate hours
0008 -     THEN Variable #10 = Variable #1   // put result in var 10
0009 -     THEN Variable #0 % 60             // calculate minutes
0010 -     THEN Variable #11 = Variable #0   // put result in var 11
0011 -                                       // ** Month **
0012 - IF  Month is = January (1)            // If January
0013 -     THEN Variable #12 = 74            // ascii "J"
0014 -     THEN Variable #13 = 97            // ascii "a"
0015 -     THEN Variable #14 = 110           // ascii "n"
0016 - IF  Month is = February (2)           // If February
0017 -     THEN Variable #12 = 70            // ascii "F"
0018 -     THEN Variable #13 = 101           // ascii "e"
0019 -     THEN Variable #14 = 98            // ascii "b"
0020 - IF  Month is = March (3)              // If March
0021 -     THEN Variable #12 = 77            // ascii "M"
0022 -     THEN Variable #13 = 97            // ascii "a"
0023 -     THEN Variable #14 = 114           // ascii "r"
0024 - IF  Month is = April (4)              // If April
0025 -     THEN Variable #12 = 65            // ascii "A"
0026 -     THEN Variable #13 = 112           // ascii "p"
0027 -     THEN Variable #14 = 114           // ascii "r"
0028 - IF  Month is = May (5)                // If May
```

```
0029 -      THEN Variable #12 = 77               // ascii "M"
0030 -      THEN Variable #13 = 97               // ascii "a"
0031 -      THEN Variable #14 = 121              // ascii "y"
0032 - IF  Month is = June (6)                   // If June
0033 -      THEN Variable #12 = 74               // ascii "J"
0034 -      THEN Variable #13 = 117              // ascii "u"
0035 -      THEN Variable #14 = 110              // ascii "n"
0036 - IF  Month is = July (7)                   // If July
0037 -      THEN Variable #12 = 74               // ascii "J"
0038 -      THEN Variable #13 = 117              // ascii "u"
0039 -      THEN Variable #14 = 108              // ascii "l"
0040 - IF  Month is = August (8)                 // If August
0041 -      THEN Variable #12 = 65               // ascii "A"
0042 -      THEN Variable #13 = 117              // ascii "u"
0043 -      THEN Variable #14 = 103              // ascii "g"
0044 - IF  Month is = September (9)              // If September
0045 -      THEN Variable #12 = 83               // ascii "S"
0046 -      THEN Variable #13 = 101              // ascii "e"
0047 -      THEN Variable #14 = 112              // ascii "p"
0048 - IF  Month is = October (10)               // If October
0049 -      THEN Variable #12 = 79               // ascii "O"
0050 -      THEN Variable #13 = 99               // ascii "c"
0051 -      THEN Variable #14 = 116              // ascii "t"
0052 - IF  Month is = November (11)              // If November
0053 -      THEN Variable #12 = 78               // ascii "N"
0054 -      THEN Variable #13 = 111              // ascii "o"
0055 -      THEN Variable #14 = 118              // ascii "v"
0056 - IF  Month is = December (12)              // If December
0057 -      THEN Variable #12 = 68               // ascii "D"
0058 -      THEN Variable #13 = 101              // ascii "e"
0059 -      THEN Variable #14 = 99               // ascii "c"
0060 -                                           // ** Day **
0061 - IF  Day of Month is > 0                  // If day > 0 (always)
0062 -      THEN  Load Data to:  Variable #15    // put in var. 15
0063 -                                           // ** Year **
0064 - IF  Year is > 0                          // if year >0 (always)
0065 -      THEN  Load Data to:  Variable #0     // store in var 0
0066 -      THEN Variable #0 + 2000              // convert to 4 digits
0067 -      THEN Variable #16 = Variable #0      // copy to var 16
0068 -                                           // ** Inside temp **
0069 - IF Module #1 is < 256                    // If temp < 156 deg
0070 -      THEN  Load Data to:  Variable #0     // store in var 0
0071 -      THEN Variable #0 - 100               // subtract 100
0072 -      THEN Variable #17 = Variable #0      // store in var 17
0073 -                                           // ** Outside temp **
0074 - IF Module #2 is < 256                    // If temp < 156 deg
0075 -      THEN  Load Data to:  Variable #0     // store in var 0
0076 -      THEN Variable #0 - 100               // subtract 100
0077 -      THEN Variable #18 = Variable #0      // store in var 18
0078 -                                           // ** Relative Hum. **
0079 - IF Module #3 is < 256                    // If hum < 156 deg
0080 -      THEN  Load Data to:  Variable #0     // store in var 0
0081 -      THEN Variable #0 - 100               // subtract 100
0082 -      THEN Variable #19 = Variable #0      // store in var 19
```

Fig. 6

The program is quite straightforward and needs little explanation, but the following comments might be useful to help you understand some areas:

- Lines 1,2, and 3 look to see if the time is equal to midnight or not, and captures it's value in Variable #0 in either case. This is just a way to capture the current time into a variable. The time value obtained is in

minutes since midnight, not the base 60 time we are used to. The conversion to hours and minutes is done in line 5 to 10.

- Line 7 takes a copy of the time and divides it by 60 to get the number of hours.

- Line 9 gets the remainder (modulo) of the time divided by 60 to get the minutes. The modulo function is also new in C-Max 2.0 and is quite useful for this type of base conversion.

- Lines 12 through 15 (as well as the eleven other program segments following them) copy the ASCII values of the letters we want to display for the month names.

- Line 65 captures the year into a variable in a manner similar to the way the time was captured. The year value will be a two digit year (ie: 12 for the year 2012). Line 66 adds 2000 to the year to convert it to a 4 digit value (...please remember to adjust this line in the year 3000...).

- Lines 69 through 72 (as well as the subsequent two similar segments) reads the bobcat's value as an "extended variable" which is easier to work with then using an "IF bobcat..." (more suited for a direct comparison instead of capturing in a variable for screen display). Note that such raw bobcat values need to have 100 subtracted to get the actual reading. For the outside temperature bobcat, a temperature below zero will cause the final value (once 100 is subtracted) to roll over and be in the 65000 range, but the variable formatting string ending with a "d" will ensure that it is interpreted as a signed integer and a "-" sign will be displayed if the temperature is indeed below zero.

Fig. 7 shows an actual Leopard screen photo of the resulting display produced by this program:

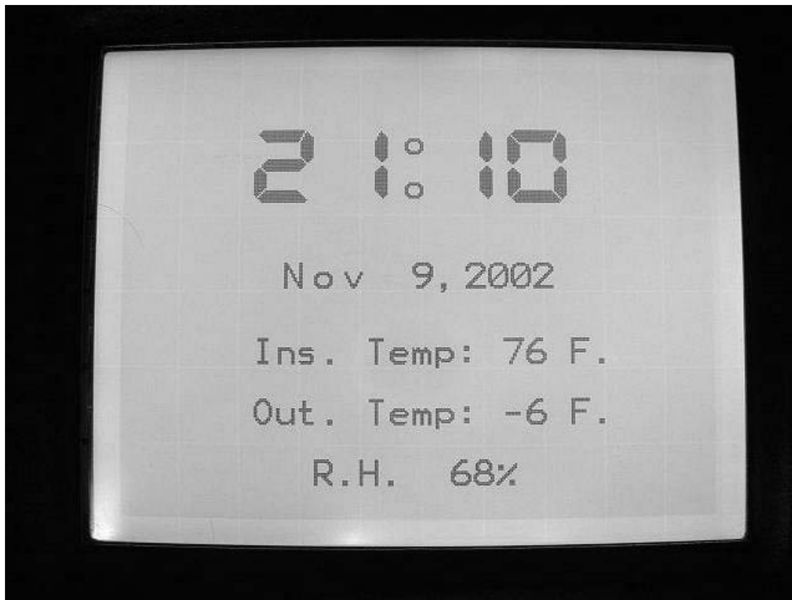

Fig. 7

Formatting Variables in ASCII Strings

Our second example will show how we can use variable formatting in ASCII strings. As previously stated, embedding variables (in any form) in ASCII strings was not available prior to C-Max 2.0. This example will show an application where we want to log a temperature reading from a temperature bobcat every half-hour (along with the date and time of the sample) to the serial port of the controller. It is assumed that the controller's serial port will be connected to a PC or other device capable of recording serial data in a file for later examination.

As mentioned at the beginning, you could use either your master controller's serial port, a slave's serial port, or a serial (ASCII) bobcat to output the strings (the module's number or map name is selected when programming the "Transmit ASCII Message" command). If you use a serial bobcat, follow the instructions received with the bobcat for loading the ASCII string definitions to the bobcat. If you want to use a slave controller to output the strings (this is also a new capability in C-Max 2.0) you must load the string definitions into that slave using it's own serial port; you cannot load them over the bus from the master like for a serial bobcat. You must also make sure that both the master and the slave(s) are running the new executive version that comes with C-Max 2.0

We will create the format definitions and C-Max code necessary to produce a text log of the temperature readings with the following appearance (Fig. 8):

MM/*DD*/*YY* *hh*:*mm* Outside Temperature: +/- *xx* F.

Where:

MM = month
DD = day
YY = year
hh = hour
mm = minute
xx = temperature

Fig. 8

Just like format definitions are entered in screen text objects for Leopard screens, we use the "ASCII Messages" editing utility to first define each string along with the formatting information for the embedded variable. This utility is found under the "Serial Messages" heading after attaching to the controller. The ASCII messages are defined as text strings with a maximum length of 32 characters, and up to 128 such strings can be defined. Note that characters such as carriage return are entered as "^013" and this counts as 4 characters in the ASCII string. Since we cannot have more then one embedded variable per ASCII string, we will need to define several such strings which will then be output in succession so that they concatenate to form one long string in the log. Only the last string will thus have the carriage return and linefeed characters to end the line. We will define the following five ASCII strings (Fig. 9). The quotes shown are to help you see the spaces in some strings and not actually entered:

```
"%02d/"     used for the month and day (with leading zero if 1 digit)
"%02d "     used for the year and minute (with leading zero if 1 digit)
"%02d:"     used for the hour (with leading zero if 1 digit)
"Outside Temperature: "      literal text
"%+4d F.^013^010"      temperature always with sign and line end chars
```

Fig. 9

Here are the strings as they are entered into the ASCII strings definition utility (Fig. 10)

Ascii Messages

File

	Message Text	Message Size
Message #0	%02d/	5
Message #1	%02d	5
Message #2	%02d:	5
Message #3	Outside Temperature:	21
Message #4	%+4d F.^013^010	15
Message #5		
Message #6		
Message #7		
Message #8		
Message #9		
Message #10		
Message #11		
Message #12		

Enter text exactly as you wish it to be sent. To send non-printable characters enter the decimal value of the character preceded by a caret ^. For example, to send a carriage return, enter ^013 Note that 3 numbers must always follow a ^.

Send To: Master Unit 0 Close

Fig. 10

Once you are done entering the strings, you download them to your controller by selecting the appropriate module and then clicking on the "Send To:" button. The Controller Access utility window will show the strings being downloaded in the status window. Once this is done, we are ready to create the C-Max program that will use these strings. Here is the program text used for our example (Fig. 11). If you looked at the preceding example with the Leopard screen, you will find that the first program lines are quite similar, since that example also involved obtaining the current date and time into variables:

```
0001 -                                                  // ** Time of Day **
0002 - IF Time of Day is > 00:00                        // whether time is 0:00
0003 -     THEN  Load Data to:  Variable #0             // or not put in var 0
0004 -     ELSE  Load Data to:  Variable #0             // (mins since 0:00)
0005 - IF Variable #0 is = Variable #0                  // on every pass
0006 -     THEN Variable #1 = Variable #0               // copy time in var 1
0007 -     THEN Variable #1 / 60                        // calculate hours
0008 -     THEN Variable #10 = Variable #1              // put result in var 10
0009 -     THEN Variable #0 % 60                        // calculate minutes
0010 -     THEN Variable #11 = Variable #0              // put result in var 11
0011 -                                                  // ** Day **
0012 - IF  Day of Month is > 0                          // If day > 0 (always)
0013 -     THEN  Load Data to:  Variable #12            // put in var. 12
0014 -                                                  // ** Month **
0015 - IF  Month is > January (1)                       // If month > 1
0016 -     THEN  Load Data to:  Variable #13            // put in var. 13
0017 -     ELSE  Load Data to:  Variable #13            // whether jan or not
0018 -                                                  // ** Year **
0019 - IF  Year is > 0                                  // if year >0 (always)
0020 -     THEN  Load Data to:  Variable #14            // copy to var 14
0021 -                                                  // ** Outside temp **
```

```
0022 - IF Module #1 is < 256                                // If temp < 156 deg
0023 -     THEN  Load Data to:  Variable #0                 // store in var 0
0024 -     THEN Variable #0 - 100                           // subtract 100
0025 -     THEN Variable #15 = Variable #0                  // store in var 15
0026 -                                                      // ** trigger event **
0027 - IF Variable #11 becomes =  0                         // if on the hour
0028 -   OR Variable #11 becomes =  30                      // or on the half hour
0029 -     THEN Send Ocelot Message 0 w/ Variable #13       // transmit month
0030 -     THEN Send Ocelot Message 0 w/ Variable #12       // transmit day
0031 -     THEN Send Ocelot Message 1 w/ Variable #14       // transmit year
0032 -     THEN Send Ocelot Message 2 w/ Variable #10       // transmit hour
0033 -     THEN Send Ocelot Message 1 w/ Variable #11       // transmit minute
0034 -     THEN Send Ocelot Message 3 w/ Variable #0        // xmit "Out. temp.."
0035 -     THEN Send Ocelot Message 4 w/ Variable #15       // xmit temper. +eol
0036 - End Program                                          //
```

Fig. 11

If you need help in understanding how lines 1 to 25 work, you can view the notes for the Leopard screen text example shown earlier in this document (see Fig. 6 and the text following it) since the code usage is almost identical. In our current example, the events that trigger a series of logging strings to be transmitted (to make up one log entry) are shown on lines 27 and 28; when the minutes component of the time is 0 and 30. Here is an actual sample of a log file produced with our logging program (Fig. 12):

```
11/12/02 21:00 Outside Temperature: +60 F.
11/12/02 21:30 Outside Temperature: +60 F.
11/12/02 22:00 Outside Temperature: +60 F.
11/12/02 22:30 Outside Temperature: +58 F.
11/12/02 23:00 Outside Temperature: +57 F.
11/12/02 23:30 Outside Temperature: +55 F.
11/13/02 00:00 Outside Temperature: +54 F.
11/13/02 00:30 Outside Temperature: +52 F.
11/13/02 01:00 Outside Temperature: +49 F.
11/13/02 01:30 Outside Temperature: +47 F.
11/13/02 02:00 Outside Temperature: +47 F.
11/13/02 02:30 Outside Temperature: +46 F.
11/13/02 03:00 Outside Temperature: +47 F.
11/13/02 03:30 Outside Temperature: +47 F.
11/13/02 04:00 Outside Temperature: +46 F.
11/13/02 04:30 Outside Temperature: +46 F.
11/13/02 05:00 Outside Temperature: +49 F.
11/13/02 05:30 Outside Temperature: +50 F.
11/13/02 06:00 Outside Temperature: +50 F.
11/13/02 06:30 Outside Temperature: +51 F.
11/13/02 07:00 Outside Temperature: +51 F.
```

Fig. 12

Note that any single triggering event could be used as a trigger, such as a received X-10 command, Leopard button press, SECU16 input turning on/off, etc. You could also have several types of events in the same program, each one re-using a copy of lines 29 to 33 to transmit the date and time followed by their own customized ASCII string to identify the event or value. You could keep track of events like when your alarm system is armed/disarmed , HVAC run times, etc…anything that translates to a single event. Here is an example where you would want to log when your alarm system is armed and disarmed, supposing that arming produces an X-10 M/1 On command and disarming an X-10 M/1 Off. We added messages 5 and 6 to the ASCII messages with the following text (without the quotes) (Fig. 13):

```
Message #5: "Alarm ARMED^013^010"
Message #6: "Alarm DISARMED^013^010"
```

Fig. 13

Then line 36 (the END statement) of the program in Fig. 11 was removed and the following code was added (Fig. 14):

```
0036 - IF X-10 House M / Unit 1, ON Command Pair        // if alarm armed
0037 -     THEN Send Ocelot Message 0 w/ Variable #13   // transmit month
0038 -     THEN Send Ocelot Message 0 w/ Variable #12   // transmit day
0039 -     THEN Send Ocelot Message 1 w/ Variable #14   // transmit year
0040 -     THEN Send Ocelot Message 2 w/ Variable #10   // transmit hour
0041 -     THEN Send Ocelot Message 1 w/ Variable #11   // transmit minute
0042 -     THEN Send Ocelot Message 5 w/ Variable #0    // xmit "Armed" msg.
0043 - IF X-10 House M / Unit 1, OFF Command Pair       // if alarm disarmed
0044 -     THEN Send Ocelot Message 0 w/ Variable #13   // transmit month
0045 -     THEN Send Ocelot Message 0 w/ Variable #12   // transmit day
0046 -     THEN Send Ocelot Message 1 w/ Variable #14   // transmit year
0047 -     THEN Send Ocelot Message 2 w/ Variable #10   // transmit hour
0048 -     THEN Send Ocelot Message 1 w/ Variable #11   // transmit minute
0049 -     THEN Send Ocelot Message 6 w/ Variable #0    // xmit "Disarmed" msg.
0050 - End Program
```

Fig. 14

Here is what a log from this new program could look like (Fig. 15):

```
11/12/02 21:00 Outside Temperature: +60 F.
11/12/02 21:14 Alarm ARMED
11/12/02 21:30 Outside Temperature: +60 F.
11/12/02 22:00 Outside Temperature: +60 F.
11/12/02 22:30 Outside Temperature: +58 F.
11/12/02 23:00 Outside Temperature: +57 F.
11/12/02 23:30 Outside Temperature: +55 F.
11/13/02 00:00 Outside Temperature: +54 F.
11/13/02 00:30 Outside Temperature: +52 F.
11/13/02 00:41 Alarm DISARMED
11/13/02 01:00 Outside Temperature: +49 F.
11/13/02 01:30 Outside Temperature: +47 F.
11/13/02 02:00 Outside Temperature: +47 F.
```

Fig. 15

As we have seen by these examples, variable embedding and formatting allows the creation of customized display and serial data that can be optimized for many user specific applications where visual appeal and/or proper organization can make the end result look really professional.