MEGAHURTS VISUAL-TFT TUTORIAL PROJECT FOR PIC32 BASIC

CUSTOM V-TFT DISPLAY GADGET AND PROJECT PROGRAMMING EXAMPLES FOR BEGINNERS TO INTERMEDIATE SKILLS

INTRODUCTION:

Version; 1.0 10/10/2013

Hi reader(s),
I am an Electronics Technician and embedded digital controller systems designer.
I use *MikroElektronika's* development hardware and software to do a lot of the work with and as a way to say thanks for the great tools and community support based around the world, I took some time to make this example **V-TFT** project and tutorial on some important basics about making a project in **Visual-TFT** and **mBASIC** Pro for **PIC32** language compiler.

Users of the other compilers (**mC** and **mP**), can still benefit from examining how this project was done. The **V-TFT** components themselves can be used by all three compilers and I worked hard to keep the design and code as simple and small as possible for everyone's ease to use.

This project is not a fully dressed out application as the result of keeping it sleek and simple. But it can be used as the starting point for anyone to expand it more. That was intentional planning also.
I guess it could be thought of being kind of like a science experimenter kit many of us have grown up with. The parts are here (*mostly*), you just have to finish putting it together the way you want.

This *PDF* file document is a **bonus** supplement to the actual tutorials that are embedded in the project as comments. The information in the comments has the advantage of being where it is important to have it. The comments and code from the project's **Main-program** and **Events-module** files are copied here, but may be different from the actual project files. Most differences should only be there's more information here in the *PDF* document.

I hope you find this information useful and that it helped you get better results from **V-TFT**.

Robert T. (*MegaHurts*)

VISUAL-TFT TIPS AND TRICKS

First, 2 rules you should know before, and while programming anything:

I call these RULE #1 and RULE #2 of programming.

RULE #1- NO program YOU write will EVER run and DO what YOU wanted it to DO, it will ALWAYS run and do EXACTLY what YOU told it to DO!

RULE #2- If data is corrupted, it will still run and do EXACTLY what it was told to DO, if it can, but NOT what YOU told it to DO.

So learning how programmable digital systems actually operate to follow a programs instructions will be one of the best tools you will use when writing your own programs. I did not make the rules, they are a natural result and inherent to all programmable digital devices.

V-TFT USER MANUAL?

As of this writing, there is no **V-TFT** users manual published *yet*. There is its included *help* file [**F1**], and the **forum** and **MikroE's** support desk for serious problems to be handled promptly by staff members. The forum is a great and valuable resource for solving most issues you might have while using **V-TFT** or their other **HW** and **SW**. Questions will be answered if anybody knows the answer. So use it when needed and maybe you will avoid many frustrations others have already *had* and *solved*.

V-TFT examples to consider also:

There are a great many program examples available for all of the hardware and software MikroElektronika makes. Some examples you may not consider to check is because you do not have that hardware, but you might pass up the exact or helpful example if you do not consider them. For example; you must have one of the **TFT** products if you are even reading this, but have you gotten and looked over all of the examples for any of their products that include or can have connected a **TFT** display?

If you are using one of their **MMB's** for **PIC**, but do not have the **mikromediaWorkStation** development system, you should still *get all* **examples**, **libraries** and the **documentation** for it. Now you would have schematics for wiring the **MMB** to a lot of other external **HW** *AND* **examples** to run or examine for how to interface to such stuff.

There is a good chance that there is something just like you *want* to do or very close, to use as an example that can help you with your project. This applies to any common core **HW** and **SW** product lines you use of theirs.

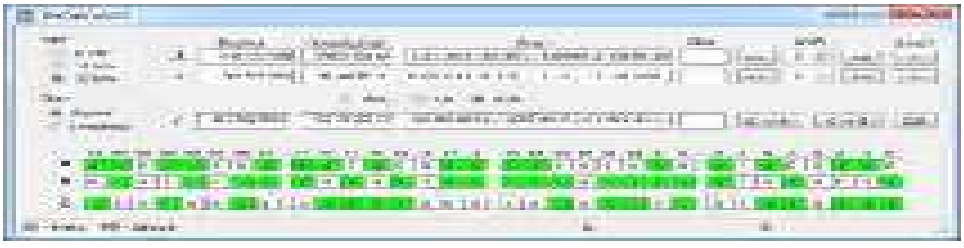
Do not short yourself of what is available to aid you.

Start Simple to test devices configurations FIRST:

Before jumping in and trying to make the multi-screen **GUI** you have been thinking about for a long time now, you should do a very simple **1** screen test project from scratch to make sure you have selected the correct device **HW** configuration settings that *work* and you can get the device to show the **TP** calibration (*if equipped*), screen and the first screen of the **V-TFT** test project with at least **1** (*active*) **button** on it, so you can test that **TP** works at desired touch pressures and can be accurately calibrated when programmed and powered up to run the program.

Simple test setup = simpler troubleshooting to do if needed.

Get the "**BitCalc**" tool that Aleksandar Vukelic made and can be downloaded at
http://www.libstock.com/projects/view/666/bitcalc



It will aid you in learning about **Bit Masking** and using **Bit** *operators* in programming **+ much more!** I consider it an ***essential*** tool for programming and analyzing others code to see how logical operations programmed get the right results (*or wrong*).

When designing in V-TFT, it is usually better to make the least complicated interface at first, then test it, and if that works, then consider what to change to add more elaborate features,

Try to do the project in discreet small chunks of functions, checking the functionality of each as you go.

Use, Use, Use your compilers comment feature to give yourself guidelines and reasons why that code is there, and what it is supposed to do – **remember RULE #1?**

You do not have to comment everything just like I did for this example project, but try to help your *future* self with what you are doing *now* in your code.

Take notice of how I named the **V-TFT** button objects in this example. Change the names of your objects **before** you assign any events to them.

If you save the job of assigning events *(creating event subroutines in event module)*, to the objects for last, and you have more than one screen in your project, you can assign events one screen at a time and the created routines will be grouped by screen together in the events file. If that confused you, do not worry, you will understand it when you make your first project with more than one screen.

It is *easier* to **copy** an object that has a lot of *properties* set like you want another one to have, than configure a new one added from the **component palette**.

The keyboard "**SHIFT**" key when held down, allows you to **left-click** on objects to *add* or *subtract* them *to/from* multiple selected objects for grouping or moving or deleting.

For your own **V-TFT** projects, it is best that you save them in a **different** folder than the **'Projects'** folder in the **Visual-TFT** install folder. If you have to uninstall **V-TFT** before an **update** can be installed, your projects may get **deleted *if in that folder***.

You can use **Layers** to group same object types together by layer or have all objects used to make a custom display or control on its own layer so easier to move or hide on the screen.

Example of using layers for complex screen layout designs. From a project of mine still working on. Did you notice the Odometer? The inspiration for making this example – tutorial.

OK, that is all for the TIPS and TRICKS BONUS material at this time everyone. Hope you liked it and find it helpful. On the next page the complete Main-program code starts, and that is where you start also. Look over everything first, then examine in-depth any part you want to. More later, Robert.

program pic32mmb_v370_event_counter_example_main

```
' *
' * Project name:
'     pic32mmb_v370_event_counter_example.vtft
' * Generated by:
'     Visual TFT
' * Date of creation
'     8/21/2013
' * Time of creation
'     11:09:10 PM
' * Test configuration:
'     MCU:            P32MX460F512L
'     Dev.Board:      MikroMMB_for_PIC32_hw_rev_1.10
'                     http://www.mikroe.com/mikromedia/pic32/
'     Oscillator:     80000000 Hz
'     SW:             mikroBasic PRO for PIC32
'                     http://www.mikroe.com/mikrobasic/pic32/
' * V-TFT Tutorial comments done by Robert Townsley (MegaHurts) 2013
'
' * Program Description:
'   This is a V-TFT example project by Robert Townsley (MegaHurts), to show how
'   to make a simple lab instrument event counter that can count up to 9,999,999
'   events. It can be modified to count higher by any body that has a need to do so.
'   This V-TFT example project also shows how to use multiple components to create
'   what looks like a single screen indicator object. The indicator object was designed
'   to look like a mechanical multi-wheel numerical counter. The indicator object
'   uses seven Button components to display a single digit each of the events total
'   counts in order to get the appearance of a mechanical wheel counter.
'   I did not attempt to program any wheel digit rolling to the next higher digit value
'   in this example project. That is beyond the scope of this demo/example.
'   (The wheel counter looks pretty good as is anyway)
'
'   This V-TFT project also shows how to make a proper user code single pass routine get called
'   from within the main endless loop that V-TFT sets up in this module to drive
'   its TP checking and event handling core code. Or you could call it a 'How to share the loop'
'   example. It demonstrates how to do this and have it manage the RESET buttons visual effects
'   to show confirmation is required before a counter reset gets done by calling a
'   routine located in the events modules "User Code" area from the loop here when
'   it is needed to do so. It also shows how to add confirmation safety to any button press method.
'
'   You are invited to examine this projects code and screen components in V-TFT
'   and a compiler in order to see how it all gets done, and use the knowledge you
'   get from it in your own projects or you can take the simple work done here and
'   use it as the base for making a more complex feature filled application to suit
'   your needs or just to experiment more on. Your choices, you do as you want with it.
'
'   I have also added a lot of comments to the "main" file and the "events" module
'   for new V-TFT users (or anybody that needs the information), about how V-TFT
'   projects are organized and where the user can place their code safely, without
'   getting compiler errors for rookie mistakes. So I hope you take the time to read
'   all of the comments, and that they help you to a better understanding and usage
'   of MikroElektronika's Visual TFT software. Best Regards to all, Robert.
'
' * Note to users of MikroC and MikroPASCAL languages:
'   Sorry, but I do not use those compilers so MikroBASIC versions are all I can
'   make for now. I may someday add PASCAL to my arsenal, but not C. Sorry, but
'   I just do not like that language, and never have (any version). But you users
'   of those others should be able to understand what is going on in BASIC in this
'   project without too much trouble, and more so for you PASCAL users, not much
'   difference between them really. The coding is simple and the comments apply
'   to all languages as for how V-TFT is organized.
```

program pic32mmb_v370_event_counter_example_main

```
' Users code can go any where in this module as long as its placement follows the
' compilers rules for a programs organization layout that is always in effect.
' New users to V-TFT may at first be confused by this very plain, bare almost,
' MAIN program file and its lack of all the normal coding you were expecting.
' With V-TFT projects, the main file is left bare so users can still have a place
' to do their coding in, but it will have to be done a little differently than
' some of you are familiar with in order to work without breaking how the V-TFT
' generated stuff works. Please read all of the comments in this file for more
' detailed instructions about this.


' User Symbol defines can go here, but they will only work in this module.
'  (see compiler help file or manual for more information on symbols and program organization)
'--------------------------------------------------------------------------------



' Users Variables declarations can go here.(global)
' (see compiler help file or manual for more information on variables)
'--------------------------------------------------------------------------------



' Users Constants defines can go here.(global)
' (see compiler help file or manual for more information on constants)
'--------------------------------------------------------------------------------



' Users subroutines coding can go here.
' (see compiler help file or manual for more information on subroutines)
'--------------------------------------------------------------------------------



' Users Function routines can go here.
'  (see compiler help file or manual for more information on function routines)
'--------------------------------------------------------------------------------




' Start of main program body. Program execution begins at the first instruction
' after the "main:" label marker and proceeds downwards towards the "end." label.
main:
' User code here gets executed one time only on power up or after a reset because of endless loop below.

' User variables initialize. (these variables are declared in events module - user declarations area.)
  COUNTER_VALUE = 0      ' set value to zero
  RESET_FLAG    = 0      ' clear flag state
  BLINK_TIMER   = 0      ' set value to zero


  Start_TP()   ' V-TFT generated routine call to initialize HW and V-TFT screen(s) and objects.
               ' The routine called is in the 'driver' module and it calls on other routines in the driver
               ' module before returning program execution flow back here. It is only called once per
               ' device power-up or after a hard or soft restart (reset).


while TRUE     ' main program endless loop section begin
               ' V-TFT makes this endless loop as part of a V-TFT project. It is a important
               ' part of how a V-TFT program executes. Once program execution flow enters
```

```
    ' this loop, it never can exit it, and should not, so do not change the loop set up
    ' or put code that will cause program execution to exit it out the bottom. If it does,
    ' the application will enter a continuous NOP loop state (at the 'end.' statement), and
    ' become 'locked up' and unresponsive to any input. A RESET will be required to restart the
    ' application every time it does so.
    '*  ANY code put here MUST be of the single pass execution design and allow program execution
    '   to continue on back here in a timely manner or the program might appear locked up or actually
    '   be locked up and non responsive.


Check_TP()
    ' This routine is what will call your code in a screen object event handler routine when TP activity
    ' triggers the need to be handled by a predefined event condition you set in V-TFT for a object.
    ' Once all of the code this routine has execute and any event handler code you program in is done,
    ' program execution flow returns back here to the next instruction after this one. That may be your own
    ' routine call or if no user code is present, the "wend" statement will cause another pass of this loop
    ' to start again, and so on and so on and so on.....
    '*  This routine call must be allowed to execute repeatedly and as quickly as possible so TP activity
    '   can be detected. So be discreet with your code you place in this endless loop.
    '   If your program seems to be locked up or not responding correctly to TP input,
    '   check any code you have placed here.

    ' Routine(s) can be called that are either coded above or in the events module
    ' or in a user made module from within this loop that will perform HW controlling
    ' or checking the MCU's HW Analog or Digital Inputs for changes or new data and
    ' if needed, call other routines to do actions based on what has changed or the new data in.
    ' But they must allow program execution to resume back here after doing the task programmed.
    ' By placing your routine call(s) or other code here in this loop, you are 'sharing'
    ' CPU processing time with the V-TFT core code, so share nicely.
    ' The time taken by the V-TFT Check_TP() routine will vary depending on TP activity and number of screen objects
      to test
    ' and the code YOU program in the event handling routines and so on. . .
    ' So any code you program in the event handling routines if done wrong can also
    ' cause an application lock up if it does not allow execution to return here for another pass of the loop.

    ' For this example project, if anyone wants to add the ability for the
    ' counter to be triggered by a PORT pin set as input instead of the UP & DOWN
    ' buttons on the screen, you place a call to a routine that reads the
    ' PORT here in this loop, after you make the routine that does the PORT
    ' reading. Also in that routine, you have it add 1 to the variable
    ' "COUNTER_VALUE" and call the routine I made that updates the screen
    ' counter object "Event_Counter()", and it will update the screen to
    ' the new value after it checks for roll-over condition @ 9,999,999 counts.
    ' My update routine will return program execution flow to the next statement
    ' in your routine after the call to it when done.
    ' Your routine for reading a PORT can be located above (where I indicated with a comment, area for user
      subroutines and functions),
    ' or in the "User Code" area in the 'events' module file where indicated for User Code
    ' or in a module file of your own making.

    ' The following code is an example of using this loop in a sharing way with the V-TFT core code.
    ' The code here calls a routine coded in the events module located in the "User Code" implements area.
    ' On every pass of the loop, the variable RESET_FLAG is checked to see if it is not equal (<>) to zero value.
    ' If it is equal to zero - nothing else happens, program execution goes to the while loops 'wend' instruction
    ' and the looping starts over again at the first instruction after the 'while true' statement.
    ' Event handler routine code in events module will change RESET_FLAG's value when the RESET button on the
    ' screen has been clicked. If RESET_FLAG's value is zero (0), it will be changed to a one (1) value, or if it
    ' is at the value of one (1), it will be changed to a value of two (2). A value of 1 or 2 will cause the test code
    ' below to evaluate as TRUE, so the routine Reset_Counter() will be called every time the loop repeats while
    ' RESET_FLAG is not equal to zero. In the codes logic design, the Reset_Counter() routine can be called many
      times
    ' when RESET_FLAG = 1 because a waiting condition for a second click of the RESET button exists then. So the
      Reset_Counter()
```

```
' routine will blink the RESET button red to indicate the waiting-for-confirmation-click state while RESET_FLAG = 1.
' But after user clicks the reset button a second time, RESET_FLAG is set to a value of 2 by the RESET buttons
  on_Click()
' event handler routine and when the Reset_Counter() routine is called again, it will detect the next state change and
' clear the counters counting variable (0), clear the RESET_FLAG variable (0), and set all 7 wheel digits to "0" and
' reset the blinking timer variable for the next time the RESET button is clicked on, resulting with only a single call to
' Reset_Counter() routine when the RESET_FLAG variable is equal to two (2). After the call, the RESET_FLAG will
  be back to
' zero value again and the "if-then-end if" conditional test below will fail and not call the Reset_Counter() routine
  again until RESET_FLAG changes.
' This setup will result in a varying amount of time the User Code will take of the processor, and may become
  apparent with
' some of the screen updates and/or responsiveness of the TP to inputs when different amounts of user code is
  being executed in this way.
' To balance this and keep everything looking and acting like there is no imbalance of V-TFT core code vs User
  Code, the 'if-then-end if'
' test below could be modified so that when RESET_FLAG = 0, a small delay instruction is executed that simulates
  the amount
' of time the Reset_Counter() routine would take if it were being called. This is probably not required for this
  application and
' is not the best method to keep things looking and acting balanced. I am not going to go into those other methods
  for this
' example project, but wanted to point out that there are timing factors to be considered and managed once you
  start sharing
' the processors time for V-TFT's core code by putting your code in this loop. A lot of tasks can be done by this
  method and
' some may require it to be done this way while the controlling of some MCU's hardware will require a more
  advanced method
' using timer interrupts to keep the timing of the code as required for the HW controlling to work.
' I will try to make an example project that demonstrates that method in the maybe near future or sooner if a lot of
  requests for it I get.

  '* user code for sharing V-TFT endless loop to get repeating passes of execution on some user code that is of single-pass
      execution design.

if (RESET_FLAG > 0) then
   Reset_Counter()     ' call this routine if the RESET_FLAG variable is not equal to zero
end if                 ' which is the indicator that it (screen RESET button), has been clicked
                       ' at least once.
                       ' On first click, flag is set to 1 and Reset_Counter() routine will cause a message to appear and
                       ' the RESET button to blink* between its normal colors and red and maroon colors until it is clicked
                       ' again to confirm reset wanted (RESET_FLAG now set to 2), or user presses MINUS (-)
                       ' button to cancel (RESET_FLAG set to 0 - cleared) and the count total is not lost.
                       '* See the code and comments in Reset_Counter() subroutine to see how this effect was done.*

wend                ' end of while loop, because configured as endless loop, this makes program execution flow go to the next
                      instruction after the 'while true' statement.


' End of main program code and file.
End.
```

module pic32mmb_v370_event_counter_example_events_code

```
module pic32mmb_v370_event_counter_example_events_code

include pic32mmb_v370_event_counter_example_objects
include pic32mmb_v370_event_counter_example_resources
include pic32mmb_v370_event_counter_example_driver

sub procedure ButtonRound_MINUS_OnClick()
sub procedure ButtonRound_PLUS_OnClick()
sub procedure CircleButton_RESET_OnClick()
'-------------------- Externals --------------------'
sub procedure Reset_Counter()    ' forward declaration so call in main program loop to it works


'----------------- End of Externals ------------------'

'-------------- User code declarations ---------------'

' This area is where users variables and constants are declared and defined
' that will be used by users executable code in users own routines or the
' routines V-TFT generates for screen objects touched event handling in
' the area below marked as " ' Event Handlers" because users must supply the code
' to be executed in those event handler routines.
' V-TFT just makes empty routine declarations for you, the user/programmer.

' user code variables (global)
dim COUNTER_VALUE as longword     ' variable to hold counted events total
    RESET_FLAG    as byte         ' variable for tracking RESET button clicks and Reset_Counter() routine operations
    BLINK_TIMER   as byte         ' variable for counting number of passes in Reset_Counter as time control for blinking colors
of RESET button.


' user code constants (global)
const bTRUE       as byte = 1     ' constant for setting or testing a logical One (1) True state
        bFALSE      as byte = 0     ' constant for setting or testing a logical Zero (0) False state
         BLINK      as byte = 50    ' change value to adjust rate RESET button blinks colors. Lower = faster, Higher = slower
                                    ' when BLINK_TIMER value gets greater than this constant, RESET button colors change.



'----------- End of User code declarations -----------'

implements

'--------------------- User code ---------------------'


' User code that is located below the "implements" statement must be of a sub procedure
' or sub function routine nature. Program execution is not allowed to just follow the
' 'TOP-to-BOTTOM' flow like it does in the Main program file. All code here is contained
' within either a sub procedures or a sub functions routine starting and its ending declarations
' and is called on from either V-TFT core code (event handler routine), or user
' code in a event handler routine or from user code in the 'main' program file.
' Program execution flow inside the routines DOES flow 'TOP-to-BOTTOM', but must
' encounter a 'end routine' statement so execution point goes back to the
' instruction after the ONE that called the routine.
' (If your routines coding (logic) does not allow program execution flow to exit
'  the routines in a timely manner, your application might appear to be hung-up or
'  slow to respond to inputs or other HW you are trying to control does not work
'  like it did in a different example.)
'
```

```
' The code a user places in this area is safe from V-TFT overwriting as long as nothing
' is done to the beginning/ending User Code comment markers.
' * Other modules may require that any routines coded here also have an associated
'   'Forwards' code line placed in the 'externals' user code area above in order to be
'   'visible' to that and other modules. All V-TFT generated event handler routines
'   will have a 'forwards' declaration automatically added to 'externals' area above
'   the 'Users Externals' area, so users only have to declare a routines forward if
'   they write their own and it needs to be visible to other modules in the project.
'   V-TFT will take care of this automatically for all routines it generates and needs to.


' User code subroutines and/or function routines
'-------------------------------------------------------------------------------

'+  routine to take the value in COUNTER_VALUE and convert it to string
'   characters (7) and update the Buttons captions with 1 digit each of current value

'* Note from the Author; Robert Townsley (MegaHurts)
' V-TFT Users, You can use the objects that make up the counter (copy them to your project)
'  and copy this entire sub procedures code also to your V-TFT project to easily
'  add this custom numerical wheels display to your design, as is, or modify it
'  to meet your needs of more or less digits displayed.
'  Just make sure the naming matches between code and objects and you will need
'  to also declare a global Variable named "COUNTER_VALUE" (longword type), to
'  hold the value you want to be displayed when you call this routine, (the way it is coded now)
'  or change this sub procedures declaration so value is passed as argument to it
'  and you can use what ever variable name in your projects code to accumulate a value
'  you want displayed by this custom wheel digital display.
'
'  I hope you reading this and looking this example project over have found it
'  useful and easy to understand how it works or at least gotten something useful
'  from it to spark your own creative design ideas to do in V-TFT.
'  This is the best way I have found to share or acquire custom components so far for V-TFT.
'  Use any combination of objects to make a custom display gadget or TP input/control/indicator gadget
'  and include any routines needed to implement it, and make a simple V-TFT project to
'  hold and show it off with and post it for others to download and check out.
'  (please include enough instructions or comments for others, to keep it at least easy to use in their own projects)
'
'  I invite you to use any part of the user code or methods I demonstrated in your
'  own projects and if you do put your project up for downloading by others to have as your example,
'  please have comments by my code you used stating that you got it from this example,
'  so others can trace back to get original material for use if they want it.
'  Thank you for any passing and sharing of methods, ideas, solutions, examples you have or may do with the community.

sub procedure Event_Counter()
' create local temp variables
dim TEMP_STRING as string[7]   ' temp string conversion result holder
    SHRT_STR    as string[1]   ' temp string for holding extracted characters

' prep temp variables
  TEMP_STRING = ""
  SHRT_STR    = ""

' test to make sure value is valid (0 <= COUNTER_VALUE <= 9,999,999)
  if (COUNTER_VALUE > 9999999) then
     COUNTER_VALUE = 0         ' out of range - roll over to zero(s)
  end if

' convert the COUNTER_VALUE variables value directly to a 7 character string with leading zeros (to match counter display
methodology)
  LongWordToStrWithZeros(COUNTER_VALUE, TEMP_STRING)
  ltrim(TEMP_STRING)
```

```
  ' ltrim(string_variable) is used to ensure NO spaces end up in the conversion string,
  ' because we only want the digits (characters), of 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 and 9
  ' to be in the result to use for updating the counter display.
  ' Example: if COUNTER_VALUE has value of 36272, then TEMP_STRING would contain
  ' the characters "0036272" + null character after conversion.
  ' Then individual access of the strings characters by the code below would result
  ' with:
  ' Button_Million_Caption         = "0"
  ' Button_HundredThousand_Caption = "0"
  ' Button_TenThousand_Caption     = "3"
  ' Button_Thousand_Caption        = "6"
  ' Button_Hundred_Caption         = "2"
  ' Button_Ten_Caption             = "7"
  ' Button_One_Caption             = "2"
'* Button_One is right most digit displayed, and Button_Million is left most digit displayed in event counter display.

' Now pull out each individual character for place holder digit value displayed in counter display
  memcpy(@SHRT_STR, @TEMP_STRING + 0, 1)
  Button_Million_Caption         = SHRT_STR          ' millions place holder digit value update
  memcpy(@SHRT_STR, @TEMP_STRING + 1, 1)
  Button_HundredThousand_Caption = SHRT_STR          ' Hundred Thousands place holder digit value update
  memcpy(@SHRT_STR, @TEMP_STRING + 2, 1)
  Button_TenThousand_Caption     = SHRT_STR          ' Ten Thousands place holder digit value update
  memcpy(@SHRT_STR, @TEMP_STRING + 3, 1)
  Button_Thousand_Caption        = SHRT_STR          ' Thousands place holder digit value update
  memcpy(@SHRT_STR, @TEMP_STRING + 4, 1)
  Button_Hundred_Caption         = SHRT_STR          ' Hundreds place holder digit value update
  memcpy(@SHRT_STR, @TEMP_STRING + 5, 1)
  Button_Ten_Caption             = SHRT_STR          ' Tens place holder digit value update
  memcpy(@SHRT_STR, @TEMP_STRING + 6, 1)
  Button_One_Caption             = SHRT_STR          ' Ones place holder digit value update

' now that each Buttons caption property has been updated, we will simply redraw the whole screen to update it
  DrawScreen(Screen1ScreenID)          ' to make sure there are no pixel artifacts left over.

end sub

'--------------------------------------------------------------------------------

sub procedure Reset_Counter()
' the first block of code in this routine handles the repeating calls made to it
' from the loop in main program module. The repeating calls serve a purpose for timing
' of the RESET buttons blinking effect. This is done by changing the RESET buttons
' Transparent property between True (0) and False (1) after every BLINK (constant value set in User Declarations above),
' + 1 times this routine is called while the RESET_FLAG variable equals 1.
' The variable BLINK_TIMER is used to keep track of the number of execution passes
' (or calls), this routine gets from the main program loop.
' When the RESET buttons Transparent property is set True (0), another circle that
' is behind the button and colored red to maroon gradient will be visible for BLINK + 1
' number of execution passes, then the buttons transparent property will be set to
' False (1) for the same number of execution passes and will repeat until the user
' either presses the RESET button again to confirm reset or presses the Minus [-]
' button to cancel the reset counter state. A text message of instructions is also
' displayed while waiting for confirmation or cancellation. It gets set visible or not by
' the RESET buttons event handler routine and made not visible also by the Minus
' buttons event handler routine if canceling RESET condition.
```

```
  if (RESET_FLAG = 1) then
    if (BLINK_TIMER < BLINK) then
      inc(BLINK_TIMER)          ' not enough passes have happened yet so add to counter
     else
      BLINK_TIMER = 0           ' OK, change RESET buttons colors and start over counting passes
      if (CircleButton_RESET_.Transparent = 1) then
        CircleButton_RESET_.Transparent = 0
       else
        CircleButton_RESET_.Transparent = 1
      end if
      DrawCCircle(@Circle2)                ' draw screen objects updates
      DrawCircleButton(@CircleButton_RESET_)
    end if
  end if

' This second block of code when executed will reset the button objects that display
' the individual digits for the value of what COUNTER_VALUE is currently at and clear
' its value to zero and clear the RESET_FLAG to zero and make sure the RESET buttons
' normal colors are visible. It is coded so that it resets ONE digit at a time,
' starting from least to most significant digit (right to left), with a little delay
' between each digit resetting (change all Delay_ms(xxxx) values to suit).

  if (RESET_FLAG = 2) then
    COUNTER_VALUE = 0
    Delay_ms(100)
    Button_One_Caption            = "0"
    DrawButton(@Button_One)
    Button_Ten_Caption            = "0"
    Delay_ms(50)
    DrawButton(@Button_Ten)
    Button_Hundred_Caption        = "0"
    Delay_ms(50)
    DrawButton(@Button_Hundred)
    Button_Thousand_Caption       = "0"
    Delay_ms(50)
    DrawButton(@Button_Thousand)
    Button_TenThousand_Caption    = "0"
    Delay_ms(50)
    DrawButton(@Button_TenThousand)
    Button_HundredThousand_Caption = "0"
    Delay_ms(50)
    DrawButton(@Button_HundredThousand)
    Button_Million_Caption        = "0"
    Delay_ms(50)
    DrawButton(@Button_Million)
    RESET_FLAG = bFALSE         ' clear the flag
    BLINK_TIMER = bFALSE        ' clear the timer
    CircleButton_RESET_.Transparent = 1  ' make sure the buttons real colors show
    DrawCircleButton(@CircleButton_RESET_)
  end if

end sub

'--------------------------------------------------------------------------


'---------------- End of User code ------------------'

' Event Handlers

' V-TFT places all screen(s) objects touch, press and click initial blank event handling routines
' below the comment "Event Handlers" above, that it also makes in this file (module),
' whenever the user assigns a new event to a screens object.
```

```
' DO NOT MODIFY THAT COMMENT OR PLACE CODE ABOVE IT AND BELOW End of User Code comment line.
'
' All event handler routines are considered "implements" code by compiler.
' All event handler routines are blank when V-TFT creates them and needs the user to put their own code inside each for
' action(s) or task(s) to be performed when Touch Panel activity causes the assigned event to trigger
' a call to the handler routine(s).

'
' User code inside of a event handler routine can be programmed to do all actions needed for the event or
' call a routine in User Code area that does tasks that are common also to other screen objects event handling needs, so
they can call it also.
' Other scenarios can also be implemented in this layout for controls coding, it depends on what
' needs to be done by how a user designs the GUI screens. Many, many possibilities can be done with this
' methodology V-TFT uses, a few can not, as it is for now.
' Changes to V-TFT may come that enable more flexibility or void what I have described here.
' But mostly, there are no limits to what can be done, if done right.

' + PLUS (+) Button event handler - adds 1 to COUNTER_VALUE variable

sub procedure ButtonRound_PLUS_OnClick()

  if (COUNTER_VALUE < 9999999) then
    inc(COUNTER_VALUE)              ' still less than max so add ONE to counter value
   else
    COUNTER_VALUE = 0               ' at max, roll counter over to zero
  end if
  Event_Counter()                   ' call routine to handle updating counter value displayed

end sub
' - PLUS (+) Button event handler - adds 1 to COUNTER_VALUE variable

'--------------------------------------------------------------------------

' + MINUS (-) Button event handler - subtracts 1 from COUNTER_VALUE variable
'   and/or clears the "Reset Counter to zero" waiting for confirmation click condition
sub procedure ButtonRound_MINUS_OnClick()

  if (RESET_FLAG = 1) then              ' test if in the waiting for another RESET button confirmation click condition?
    RESET_FLAG = bFALSE                 ' yes - so clear the waiting for another RESET button confirmation click condition
    Label_MSG.Visible = bFALSE          ' set stuff back to normal
    BLINK_TIMER = 0                   '
    CircleButton_RESET_.Transparent = 1  '
    DrawScreen(Screen1ScreenID)         ' and redraw the screen to make sure there are no pixel artifacts left over.
   else
    if (COUNTER_VALUE > 0) then         ' not in RESET waiting condition so test if subtract ONE from value possible
      dec(COUNTER_VALUE)              ' value still above 0, so subtract ONE from value
     else
      COUNTER_VALUE = 9999999           ' was at zero, so roll over to max value
    end if
    Event_Counter()                   ' call routine to handle updating counter value displayed
  end if

end sub
' - MINUS (-) Button event handler - subtracts 1 from COUNTER_VALUE variable



'--------------------------------------------------------------------------
```

```
' + RESET Button event handler - Resets COUNTER_VALUE to zero (0) when clicked twice
sub procedure CircleButton_RESET_OnClick()
  if (RESET_FLAG = 0) then
    RESET_FLAG = 1                  ' set flag to first stage of reset handling
    Label_MSG.Visible = bTRUE         ' RESET_FLAG = 0: Flag is cleared
   else                             ' RESET_FLAG = 1: Flag state is RESET button has been clicked once
    if (RESET_FLAG = 1) then          ' RESET_FLAG = 2: Flag state is RESET button has been clicked twice
      RESET_FLAG = 2                ' routine call in 'main' program module loop will call Reset_Counter() routine
      Label_MSG.Visible = bFALSE      ' while RESET_FLAG <> 0 value (cleared), and handle blinking the buttons colors
      CircleButton_RESET_.Transparent = 1
      BLINK_TIMER = 0
    end if                          ' while waiting for another click of the RESET button and when (if) it happens,
  end if                            ' it will clear COUNTER_VALUE variable and set all counter digits to 0 (zero).
end sub
' - RESET Button event handler - Resets COUNTER_VALUE to zero (0) when clicked twice

'------------------------------------------------------------------------------
'* Final word(s) from the Author about this example project:
'
'  For those who wanted or expected more features or controls or examples of using different
'  sources for trigger events to select, I have to say 'Sorry, but as you can see,
'  it took me far longer to type the comments and tutorial info than the actual program
'  coding for just what is included in this example project for making custom gadgets,
'  and it is intended to only tutor and show some important BASIC facts of what makes
'  a working V-TFT project and how V-TFT organizes the project parts it makes from
'  a users use of its components. It was meant to be a beginners guide for understanding
'  the parts of a V-TFT project and guide source for how to implement a few cool
'  tricks or practices. I hoped everyone could gain something from this example work, no mater the skill level.
'  (but mainly that beginners could use to help make sure their projects start
'   off better and work like they wanted with out hitting any common mistakes that
'   would make a V-TFT project not work at all on first attempts to make one.)
'
' For you more skilled users, here are some suggestions for what you can add to this
' project to make it more versatile and take it to the next level as a example project.
' If you have the time, skill, ideas and ambition, please contribute on this and
' post your results of additions or changes back on LibStock site.
'
' Some additional controls and features a 'real' event counter could/would have:
' Start/Stop/Hold count controls.
' Threshold settings for Analog ADC sampling trigger/monitor source(s).
' Multiple counters for multiple trigger sources and controls to configure them.
' Controls to select from many digital inputs connected to One or more PORTS as trigger(s).
' Controls to configure timing test conditions to any source. (ex... was Digital input change long or short enough or time
between event triggers equal to setting?)
' Digital inputs that trigger Interrupt events to count.
' Device HW timers configurations for trigger events to count.
' And/Or anything else you can imagine would be a good feature to have added.....
'
' Lastly, I hope you have realized that the counter display can be used as just a
' numerical (or alphanumerical), display for about anything you want displayed in
' the manner it does. You can also add or subtract to the number of digits or places
' it can display as needed. You have the freedom to imagine and change it like you want.
'
' Post or email comments or questions about this example if you have any, and I will
' do my best to respond or answer any questions if I can.
' ~~~~~~ Robert Townsley 2013 U.S.A. (MegaHurts) ~~~~~~ '
'             trak_werks@hotmail.com


' End of Module

end.
```