Column #111 July 2004 by Jon Williams:

GUI on a Beam of IR

If you ask my close friends they'll tell you that I'm as stubborn as a mule and yet, I maintain the right to be human, and therefore the right to be wrong. And I'd like to think that when I am wrong I admit it – I try, anyway.

When the FlexiPanel first came out a few months ago I looked at it – for about two seconds – and thought it was wholly uninteresting. It turns out I was wrong; it's actually a very clever device. I think what originally put me off was that I *assumed* I'd have to develop a GUI for my Pocket PC with traditional tools. While that's not too hard, it's certainly not as easy as developing a desktop application. This is especially true when one needs to access serial communications ports (this includes IR). Well, again, I was wrong because I just didn't give the FlexiPanel a fair look the first time through. Let me correct that.

Beam Me Up, GUI

It turns out that we don't have to develop software for our Pocket PC after all – that's already been done for us in the form of the FlexiPanel client software (called VCP for *Virtual Control*

Panel). So what gives? Well, here's what I didn't originally understand: The VCP acts like a specialized browser for the FlexiPanel module, and that it's actually the FlexiPanel module that beams the GUI to the VCP running on our Pocket PC. That's pretty cool, and a very clever idea. What this means is that we could have multiple devices using FlexiPanel modules and not have to worry about separate apps for each on our Pocket PC. Imagine what a hassle the Internet would be if we needed a separate application for every web site we visited.

In fact, the workflow to use the FlexiPanel is about the same as creating and publishing a web site on the Internet. The first step is to create a script file that will produce the desired output in the browser application. That file is uploaded to the server – in our case, to the FlexiPanel module. A client request will cause the file to be downloaded and displayed on the client. Sophisticated web pages will allow the user to provide information to the server through the interface; so does the FlexiPanel VCP.

So where does the BASIC Stamp microcontroller fit in? Its purpose is to serve as the host for the FlexiPanel module. As the host, it can send information to the module that can be displayed in the VCP, and it can also read information from the module (that was provided by the VCP) that is required by its application.

Man, It's Hot!

Summers in north Texas are hot and humid – there's no getting around it. That being the case (and the temperature on my mind), let's demonstrate the FlexiPanel system by creating a handheld UI for an air conditioning controller. I used my own home AC controller as the model for the project. Here are the requirements:

Display current temperature
Display current status
Set/Display temperature setpoint
Set/Display operating mode
Set/Display fan mode

Application logic and control outputs are provided by the BASIC Stamp. It also reads the current temperature from the DS1621 (I2C version of our old friend the DS1620). Since we're using the VCP, we don't have a local display – but we could add one later if we choose as there are plenty of IO pins left when we're done with this project.

Creating the GUI is probably the trickiest aspect of the whole deal. After experimenting with some of the FlexiPanel examples (there are plenty of screen shots in FlexiPanel.PDF) you'll get a good idea of how the VCP handles the various interface controls.

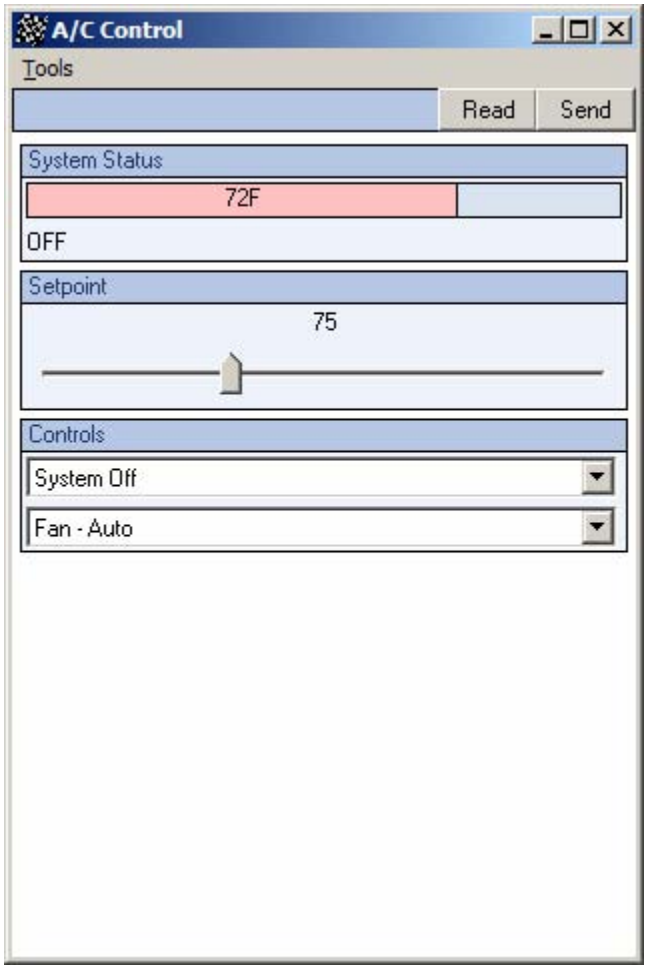

Figure 111.1: VCP in Windows

Once you're comfortable with what it does, sketch your interface and work out the details before starting the GUI script. Let me work backwards and show you the final output before the script that develops it.

Take a look at Figure 111.1. As you can see, all of the requirements for the controller specified earlier are nicely handled by the VCP software. The screen shot is actually from the Windows version of the VCP that is used in development and testing. Once the program is ready to go, it can be downloaded (via IrDA link) to the VCP on a Pocket PC. Figure 111.2 shows our AC controller interface running on an HP iPaq.

Figure 111.2: VCP on an iPaq

It All Starts With a Script

Just as cool web sites are based on a script (HTML file), the VCP GUI is created with a simple script. Let's have a look at the script that generates the UI for our AC controller.

We start with a header that provides device and connection information:

```
PARTNUM          irGUI452M
DEVICENAME       "A/C Control"
SERIAL           RANDOM
RESET
UNASSIGNED_PINS_OUTPUTS

I2C              0x68
I2CADDRESSING    1BYTE_INDEX
```

The PARTNUM parameter describes the device that will hold the GUI information. In our case, we're using the pre-built module which has a part number of `irGUI452M`. DEVICENAME identifies the FlexiPanel module for the VCP (this text will appear in the VCP title bar) and for other devices that are capable of IrDA communications (like our PC).

The SERIAL parameter describes how the serial number is generated for the device. The reason for the serial number is to prevent conflicts between devices that are running the same application code. I used `RANDOM` for this project. Other options for SERIAL are `INCREMENTAL` and manual (user-supplied 4-byte serial number). If we have two devices that serve the same function and we want to exchange information between them (read from one, send to the other), we'll need to match the serial numbers. The easiest way to do this is to provide the four-byte serial number manually.

The next line tells the VCP to RESET after it's configured. This will make sure that all values are initialized properly. The onboard controller has extra IO pins, and for the module we need to tell them to be set as outputs (UNASSIGNED_PINS_OUTPUTS) to reduce current consumption. If using the FlexiPanel irGUI processor in a specialized app, the extra pins may be handled differently.

The next section in the header has to do with connecting to a host controller; in our case, the BS2p. The connection between the host and the FlexiPanel module is via I2C, at the address specified by the I2C parameter. The default (read) address for the FlexiPanel module is 0x68, but we can specify any even address between 0x68 and 0xFC.

The final parameter is called I2CADDRESSING. This specifies how we will address the various controls in the VCP. I think for most applications that 1BYTE_INDEX is going to be the easiest to deal with – so long as we plan our project and then work the plan. Using this method, each control in the VCP is accessed by its position in the definitions (see below) – and using one byte gives us the ability to deal with up to 256 controls. This method is also the cleanest to implement using **I2CIN** and **I2COUT**.

Give Me Some Control

Okay, now we can get into the fun stuff – the VCP controls. If you look at the screen shots in the documentation you'll notice that the controls are always stacked vertically in the VCP window. This strategy keeps the controls scripting very simple; controls will appear on the form in the order defined.

Going back to our requirements, the first thing to display is current temperature. We have a few choices here, but this being a temperature controller, using a numeric output in the form of a progress bar (which suggests a mercury thermometer) seemed the logical choice. Here's the script for this control:

```
CONTROL NUMBER
    ID          ctmp
    FIXEDSTORE  ROM
    VARSTORE    RAM
    STYLE       FIXED
    OPTION      PROGRESS
    TITLE       " System Status"
    UNITS       "F"
    VALUE       72
    CTL_MIN     0
    CTL_MAX     100
```

The type of control we'll use for the current temperature is the NUMBER type. There are options to display numbers, but as we just discussed, we're going to use the PROGRESS type. This STYLE is also set to FIXED which means it cannot be changed by the user running the VCP, though it can be changed by the host. The UNITS parameter allows us to display a string after the value, and CTL_MIN and CTL_MAX allow us to specify the control's value range.

All controls must have a unique, 4-byte ID. I chose to use a 4-byte string to serve as a reminder (that are matched with constants in the PBASIC host program), but any unique 4-byte value will work. The FIXEDSTORE and VARSTORE parameters specify the location of fixed (like the control's title) and variable (like the value) data are stored in the module. The parameters we used are typical, though there are other options.

All VCP controls are surrounded by a grouping box, even when there is just one control. This box has a title string that is specified by the TITLE parameter. We can group controls into the same box, and we'll demonstrate that in just a moment.

Our next control is a TEXT type to display system status. The status text will tell us what's actually going on inside our AC controller (OFF, Idle, Cooling, Heating, etc.). We give it a maximum length with the MAXCHARS parameter and in this case, we are grouping it into the "System Status" box by using APPEND.

```
CONTROL TEXT
    ID          stat
    STYLE       FIXED
    FIXEDSTORE  ROM
    VARSTORE    RAM
    VALUE       "OFF     "
    MAXCHARS    8
    APPEND
```

Now we get to the controls that we'll actually change, and will have an effect on our control program. The first is the temperature setpoint. The controller logic will use this setpoint along with the operating mode and current temperature to determine which control outputs, if any, are active.

```
CONTROL NUMBER
    ID          setp
    STYLE       EDIT
    OPTION      SLIDER
    FIXEDSTORE  ROM
    VARSTORE    RAM
    TITLE       " Setpoint"
    VALUE       75
    CTL_MIN     65
    CTL_MAX     95
```

Notice that the setpoint is (logically) and NUMBER control, but this time it is formatted as a SLIDER, and its STYLE is set to EDIT so we can change the value.

The next two controls mimic slide switches on my home AC controller. Both are SELECTION controls using the DROPDOWN option. This makes them look and behave like a standard Windows dropdown selector. The VALUE parameter selects the current position of the SELECTION control. The ITEM parameters provide strings for each available position in the control.

```
CONTROL SELECTION
    ID          mode
    OPTION      DROPDOWN
    FIXEDSTORE  ROM
    VARSTORE    RAM
    VALUE       0
    TITLE       " Controls"
    ITEM        "System Off"
    ITEM        "Cool"
    ITEM        "Heat"

CONTROL SELECTION
    ID          fctl
    OPTION      DROPDOWN
    FIXEDSTORE  ROM
    VARSTORE    RAM
    VALUE       0
    ITEM        "Fan - Auto"
    ITEM        "Fan - On"
    APPEND
```

Okay, now that we have a VCP script, it's time to give it a run. Let me encourage you to test the VCP in a stand-alone manner before adding any host (BASIC Stamp) interaction with the FlexiPanel module. And if you decide to make changes in your VCP script, be sure you remove the host program so that it doesn't interfere with your FlexiPanel module reprogramming. This is not as tedious as it sounds, and will save you a bit of trouble as you're refining your FlexiPanel projects.

After connecting the circuit shown in Figure 111.3, make sure that the BASIC Stamp is "blank" – that is, there is no program running that attempts to access the FlexiPanel module. The easiest way to do this is to download the following single-line program:

```
END
```

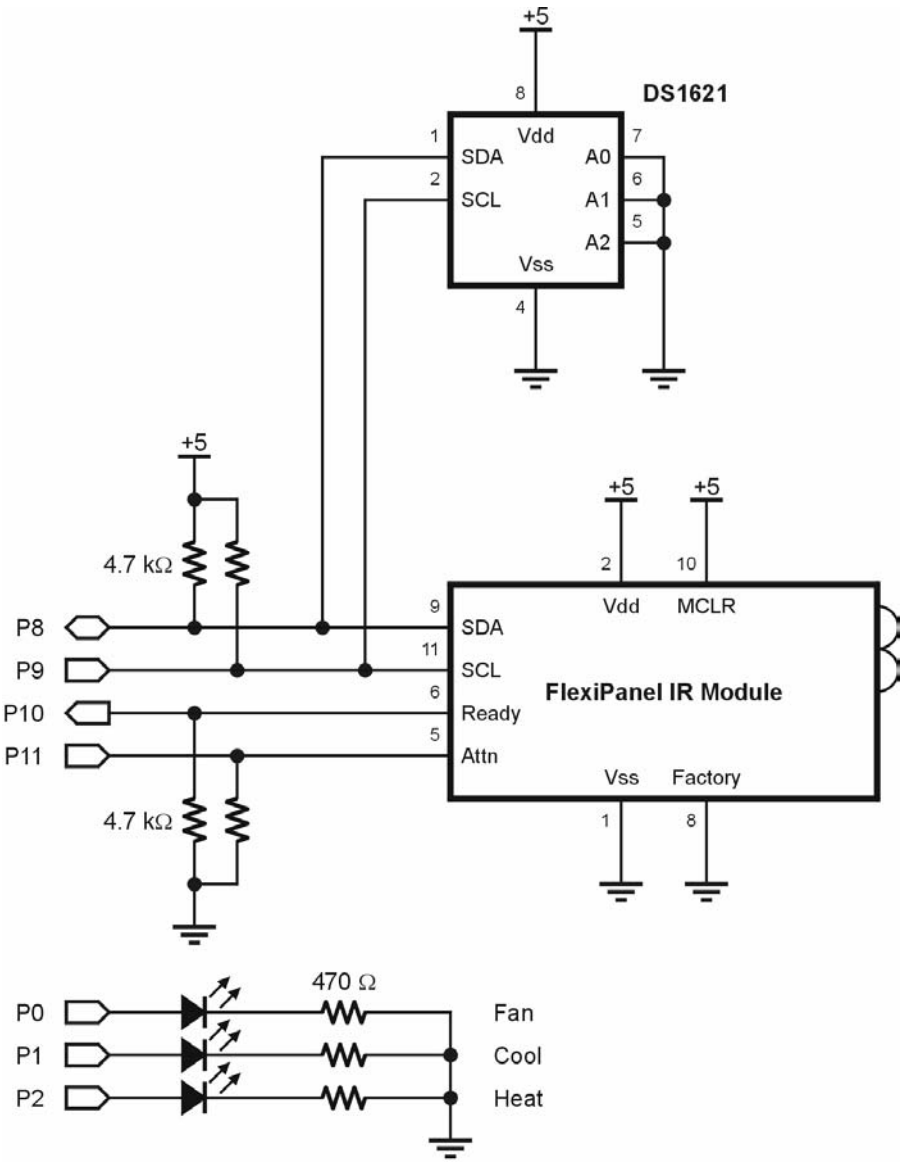

Figure 111.3: AC Controller Schematic

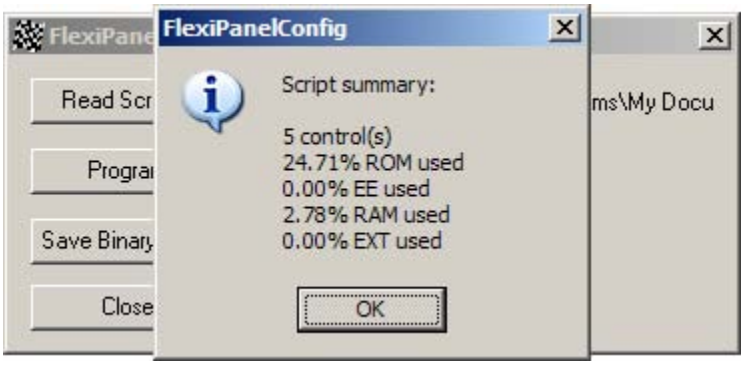

Figure 111.4: VCP Script loaded and ready for VCP

After the circuit is powered-up, bring it into range of your PC's IrDA port. If you don't have an IrDA port built into your PC, you can get an IrDA adapter that plugs into a USB port for about $30 – that's what I did and it works great.

Once the PC acknowledges the presence of the FlexiPanel module, start the program called **FlexiPanelConfig**, and then click the "Read Script" button. A standard *File Open* dialog will be displayed. Select the desired script (.rs file) and click the "Open" button. If all goes well and there are no errors in the script, you'll see an information dialog as in Figure 111.4. Click on the "Program" button to download the script data to the FlexiPanel module. Note that the only indicator provided that the process is finished is the mouse pointer. Leave it over the program dialog and wait for it to change from an hourglass cursor back to the standard pointer.

Now we can test it. Start the program called **FlexiPanel IR**. The working space will initially be blank. Click on the "Read" button and after a brief delay the title and controls will appear as shown in Figure 1. Play with the controls to see how they work and note that it really wasn't very difficult to create a nice functional display. And keep in mind that we're just barely scratching the surface of the capabilities with the VCP software.

The Host with the Most

Our host will, of course, be the BASIC Stamp microcontroller – but if you are have something else that is capable of I2C communications you can us it; you'll just need to adapt the code here for your target.

For this project the BASIC Stamp has three tasks:

Read current temperature from DS1621
Exchange data with FlexiPanel module
Process and update control outputs

Okay, then, first things first. This is a temperature controller we're building, so reading the current temperature is a priority. The DS1621 was selected so that we could take advantage of the I2C bus that is required by the FlexiPanel module.

```
Get_Temperature:
  I2COUT SDA, Wr1621, [RdTemp]
  I2CIN  SDA, Rd1621, [tempIn.BYTE1, tempIn.BYTE0]
  tempIn = tempIn >> 7
  ' Celsius
  tempC = (tempIn / 2) | ($FF00 * sign)
  ' Fahrenheit
  tempF = (tempIn | ($FF00 * sign)) + 110
  tempF = tempF * 9 / 10 - 67
  RETURN
```

As you can see, this is pretty straightforward. We start by sending the RdTemp instruction then read back the current temperature. The value returned by the DS1621 will be in units of 0.5 degrees Celsius, and will be shifted left in the *tempIn* word. Shifting everything to the right by seven bits takes care of the alignment. Conversion to Celsius is really just a matter of removing the half-degree bit, then correcting the upper bits of the *tempC* value if the temperature is negative (sign bit will be 1 when negative).

Converting to Fahrenheit uses an old Scott Edwards trick from the DS1620 which starts by converting the temperature to an absolute value. By doing this, we maintain the proper two's-compliment format for negative numbers. If you decide to add **DEBUG** or a local display, be sure to use the SDEC modifier in the event the temperature is negative (You must live in an igloo if it is!).

With the current temperature in hand we can send it to the FlexiPanel module and receive any controls or setting changes that may have happened since the last access.

```
Process_FlexiPanel:
  HIGH Attn
  DO : LOOP UNTIL (Ready = IsHigh)

  I2COUT SDA, WrFlxPnl, CTmp, [tempF.BYTE0, tempF.BYTE1]
  INPUT SCL
  DO : LOOP UNTIL (SCL = IsHigh)

  SELECT status
    CASE StatOff
      I2COUT SDA, WrFlxPnl, Stat, ["OFF    ", 0]

    CASE StatIdle
      I2COUT SDA, WrFlxPnl, Stat, ["Idle   ", 0]

    CASE StatCool
      I2COUT SDA, WrFlxPnl, Stat, ["Cooling", 0]

    CASE StatHeat
      I2COUT SDA, WrFlxPnl, Stat, ["Heating", 0]

    CASE StatFan
      I2COUT SDA, WrFlxPnl, Stat, ["Fan    ", 0]
  ENDSELECT
  INPUT SCL
  DO : LOOP UNTIL (SCL = IsHigh)

  I2CIN SDA, RdFlxPnl, StPt, [setPoint]
  I2CIN SDA, RdFlxPnl, Mode, [sysMode]
  I2CIN SDA, RdFlxPnl, FCtl, [fanCtrl]

  INPUT Attn
  RETURN
```

Even though the exchange between the BASIC Stamp and the FlexiPanel module is via I2C, we must first get the module's attention before initiating any communication. This is accomplished by taking the Attn pin high. When the module is available it will set the Ready line high.

The first piece of information transmitted is the current temperature. Remember that the current temperature control in the VCP was first; hence it has an index value of 0. When we address controls in the VCP, the address used by **I2CIN** and **I2COUT** will be the control index (0 – n). This keeps us from having to know the location of data within the module.

Storing the temperature may take a moment, so we'll monitor the clock (SCL) line. The FlexiPanel module uses "clock stretching" (pulls the clock line low) to indicate that it's busy, so waiting for the clock line to go high before moving to the next control is a good idea after each write.

The next control is the current system status text. For me, the cleanest implementation was the use of the **SELECT-CASE** structure. To be honest, I'm not a big fan of this structure for embedded control (it's a code hog), but in this case it just makes good sense. Note that each string is terminated with a zero – this is a requirement of the VCP.

Next, we read in the setpoint, mode, and fan controls from the VCP. At this point, all the information has been exchanged, so we can release the module and we can apply the air conditioning logic.

The processing logic is very simple. We start by checking the mode, and if in an active mode (cool or heat), check the current temperature against the setpoint. Here's a case where one might be inclined to use **SELECT-CASE**, but there is so much code involved it can become a bit unwieldy. No problem, **BRANCH** works really well here.

```
Process_Temp:
  BRANCH sysMode, [System_Off, Cool_On, Heat_On]

System_Off:
  IF (fanCtrl = FanAuto) THEN
    status = StatOff
    Fan = IsOff
  ELSE
    status = StatFan
    Fan = IsOn
  ENDIF
  Cool = IsOff
  Heat = IsOff
  RETURN
```

As you can see, **BRANCH** selects the code segment that corresponds with the operating mode. Off is the easiest to deal with; turn off the control outputs -- unless the user has placed the fan in manual mode for circulation.

The logic for cooling and heating is identical, so we'll just look at cooling. Really, this code is so easy that I didn't even put comments in my original listing.

```
Cool_On:
  IF (tempF > setPoint) THEN
    status = StatCool
    Fan = IsOn
    Cool = IsOn
    Heat = IsOff
  ELSE
    GOTO Manual_Fan
  ENDIF
  RETURN
```

It should make perfect sense: if the temperature is above the setpoint, turn on the fan and cooling control outputs, otherwise jump to the Manual_Fan section. Let's have a look at that.

```
Manual_Fan:
  Cool = IsOff
  Heat = IsOff
  IF (fanCtrl = FanOn) THEN
    status = StatFan
    Fan = IsOn
  ELSE
    status = StatIdle
    Fan = IsOff
  ENDIF
  RETURN
```

This code turns off the cooling and heating control outputs, and then checks the fan control setting for automatic or manual mode. If the fan has been set to manual it gets activated. The status called "idle" indicates that cooling or heating is selected, but at the moment the system is at rest.

Okay, I have run myself out of space, but let me finish with this. The FlexiPanel is a very unique device, and can add a sophisticated user interface to embedded projects by using the Pocket PC you may already own. Be sure to download all of the documentation and examples and spend some time with them. As you'll see, there's a whole lot more that can be done than we did here – some really fun stuff. We'll be back at this next month with a Bluetooth version of the FlexiPanel. It brings additional features and removes the requirement to be right on top of the embedded device.

Until then, Happy Stamping.

(additional) Resources

Société HOPTROFF
www.hoptroff.com

```
' =============================================================================
'
'   File...... FlexiPanel_AC_Control.BSP
'   Purpose... AC Controller Using FlexiPanel irGUI Module
'   Author.... Jon Williams
'   E-mail.... jwilliams@parallax.com
'   Started...
'   Updated... 07 MAY 2004
'
'   {$STAMP BS2p}
'   {$PBASIC 2.5}
'
' =============================================================================


' -----[ Program Description ]---------------------------------------------


' -----[ Revision History ]------------------------------------------------


' -----[ I/O Definitions ]-------------------------------------------------

SDA             PIN     8                       ' I2C data IO
SCL             PIN     9                       ' I2C clock
Ready           PIN     10                      ' <-- FlexiPanel
Attn            PIN     11                      ' --> FlexiPanel

Fan             PIN     0                       ' control outputs
Cool            PIN     1
Heat            PIN     2


' -----[ Constants ]------------------------------------------------------

Wr1621          CON     $90                     ' write to DS1621
Rd1621          CON     $91                     ' read from DS1621

RdTemp          CON     $AA                     ' read temperature
RdCntr          CON     $A8                     ' read counter
RdSlope         CON     $A9                     ' read slope
StartC          CON     $EE                     ' start conversion
StopC           CON     $22                     ' stop conversion
AccTH           CON     $A1                     ' access high temp limit
AccTL           CON     $A2                     ' access low temp limit
AccCfg          CON     $AC                     ' access config register

WrFlxPnl        CON     $68                     ' write to FP module
RdFlxPnl        CON     $69                     ' read from FP module
```

```
CTmp            CON     0                       ' FP control index
Stat            CON     1
StPt            CON     2
Mode            CON     3
FCtl            CON     4

SysOff          CON     0
SysCool         CON     1
SysHeat         CON     2

StatOff         CON     0                       ' System Off
StatIdle        CON     1                       ' temp is okay
StatCool        CON     2                       ' cooling in progress
StatHeat        CON     3                       ' heating in progress
StatFan         CON     4                       ' fan only

FanAuto         CON     0                       ' fan mode control
FanOn           CON     1

IsHigh          CON     1
IsLow           CON     0

IsOn            CON     1                       ' active high controls
IsOff           CON     0


#DEFINE DebugMode = 1                           ' show internal values


' -----[ Variables ]-------------------------------------------------------

tempIn          VAR     Word                    ' raw temp from DS1621
sign            VAR     tempIn.BIT8             ' - sign (after alignment)
halfC           VAR     tempIn.BIT0             ' half-degree C bit
tempC           VAR     Word                    ' temp in Celsius
tempF           VAR     Word                    ' temp in Fahrenheit

setPoint        VAR     Byte                    ' thermostat setting
status          VAR     Byte                    ' current status
sysMode         VAR     Byte                    ' Off, Cool, Heat
fanCtrl         VAR     Byte                    ' Auto, On


' -----[ EEPROM Data ]-----------------------------------------------------


' -----[ Initialization ]--------------------------------------------------

Reset:
  I2COUT SDA, Wr1621, [AccCfg, %10]             ' continuous conversion
  I2COUT SDA, Wr1621, [AccTH, 0, 0]             ' set high threshold
```

```
  I2COUT SDA, Wr1621, [AccTL, 0, 0]             ' set low threshold
  PAUSE 10                                      ' allow EE write
  I2COUT SDA, Wr1621, [StartC]                  ' start continuous

  LOW Fan                                       ' initialize control outs
  LOW Cool
  LOW Heat


' -----[ Program Code ]----------------------------------------------------

Main:
  DO
    GOSUB Get_Temperature                       ' read from DS1621
    GOSUB Process_FlexiPanel                    ' update FP module
    GOSUB Process_Temp                          ' handle temp/controls

    #IF DebugMode #THEN                         ' show internal values
      DEBUG HOME,
            SDEC ? tempF, CLREOL,
            DEC ? status, CLREOL,
            DEC ? setPoint, CLREOL,
            DEC ? sysMode, CLREOL,
            DEC ? fanCtrl, CLREOL
    #ENDIF

    PAUSE 500
  LOOP
  END


' -----[ Subroutines ]------------------------------------------------------

' Read temperature from DS1621

Get_Temperature:
  I2COUT SDA, Wr1621, [RdTemp]
  I2CIN  SDA, Rd1621, [tempIn.BYTE1, tempIn.BYTE0]
  tempIn = tempIn >> 7                          ' correct bit alignment
  ' Celsius
  tempC = (tempIn / 2) | ($FF00 * sign)
  ' Fahrenheit
  tempF = (tempIn | ($FF00 * sign)) + 110       ' convert to absolute T
  tempF = tempF * 9 / 10 - 67                   ' convert to F
  RETURN


' Send current temperature and status to FlexiPanel
' Accept new setpoint and mode settings

Process_FlexiPanel:
```

```
  HIGH Attn                                     ' get FP module attention
  DO : LOOP UNTIL (Ready = IsHigh)              ' wait for FP module

  ' send temp to FlexiPanel module
  I2COUT SDA, WrFlxPnl, CTmp, [tempF.BYTE0, tempF.BYTE1]
  INPUT SCL
  DO : LOOP UNTIL (SCL = IsHigh)

  ' send status to irGUI
  SELECT status
    CASE StatOff
      I2COUT SDA, WrFlxPnl, Stat, ["OFF    ", 0]

    CASE StatIdle
      I2COUT SDA, WrFlxPnl, Stat, ["Idle   ", 0]

    CASE StatCool
      I2COUT SDA, WrFlxPnl, Stat, ["Cooling", 0]

    CASE StatHeat
      I2COUT SDA, WrFlxPnl, Stat, ["Heating", 0]

    CASE StatFan
      I2COUT SDA, WrFlxPnl, Stat, ["Fan    ", 0]
  ENDSELECT
  INPUT SCL
  DO : LOOP UNTIL (SCL = IsHigh)

  ' get temperature setpoint
  I2CIN SDA, RdFlxPnl, StPt, [setPoint]

  ' get system mode
  I2CIN SDA, RdFlxPnl, Mode, [sysMode]

  ' get fan control mode
  I2CIN SDA, RdFlxPnl, FCtl, [fanCtrl]

  INPUT Attn                                    ' release FP module
  RETURN



' Process current temperature with control settings from
' FlexiPanel module

Process_Temp:
  BRANCH sysMode, [System_Off, Cool_On, Heat_On]


System_Off:
  IF (fanCtrl = FanAuto) THEN
```

```
    status = StatOff
    Fan = IsOff
  ELSE
    status = StatFan
    Fan = IsOn
  ENDIF
  Cool = IsOff
  Heat = IsOff
  RETURN


Cool_On:
  IF (tempF > setPoint) THEN
    status = StatCool
    Fan = IsOn
    Cool = IsOn
    Heat = IsOff
  ELSE
    GOTO Manual_Fan
  ENDIF
  RETURN


Heat_On:
  IF (tempF < setPoint) THEN
    status = StatHeat
    Fan = IsOn
    Cool = IsOff
    Heat = IsOn
  ELSE
    GOTO Manual_Fan
  ENDIF
  RETURN


Manual_Fan:
  Cool = IsOff
  Heat = IsOff
  IF (fanCtrl = FanOn) THEN
    status = StatFan
    Fan = IsOn
  ELSE
    status = StatIdle                          ' not currently active
    Fan = IsOff
  ENDIF
  RETURN
```