```
/*****************************
 Scooterputer
GUI Module - TouchShield Slide

Created Feb 16, 2010
Rev 1.0
Kurt Schulz

This software is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Update History:
Rev 1.1  07/11/2010
: Fixed heading value scaling 0 - 36000 (not 360000)

*****************************/
#include<TimedAction.h>
#include<SubPGraphics.h>

#define  BUTTON_W         52// Button BMP image width
#define  BUTTON_H         45// Button BMP image height
#define   BUTTON_COUNT     1// Black button only

enum Colors
{
          Black,// Button color
  DarkBlue,
  DarkGreen,
  DarkCyan,
  DarkRed,
  DarkViolet,
  DarkYellow,
  DarkGray,

  Gray,
  Blue,
  Green,
```

```
  Cyan,
  Red,
  Violet,
  Yellow,
  Steel,

  LightGray,
  LightBlue,
  LightGreen,
  LightCyan,
  LightRed,
  LightViolet,
  LightYellow,
  White,
};

byte rgb[][3] =
{
    0x00,  0x00,  0x00,// Black
    0x00,  0x00,  0x40,// DarkBlue
    0x00,  0x40,  0x00,// DarkGreen
    0x00,  0x40,  0x40,// DarkCyan
    0x40,  0x00,  0x00,// DarkRed
    0x40,  0x00,  0x40,// DarkViolet
    0x40,  0x40,  0x00,// DarkYellow
    0x40,  0x40,  0x40,// DarkGray

    0x80,  0x80,  0x80,// Gray
    0x00,  0x00,  0x80,// Blue
    0x00,  0x80,  0x00,// Green
    0x00,  0x80,  0x80,// Cyan
    0x80,  0x00,  0x00,// Red
    0x80,  0x00,  0x80,// Violet
    0x80,  0x80,  0x00,// Yellow
    0xA0,  0xA0,  0xA0,// Steel

    0xC0,  0xC0,  0xC0,// LightGray
    0x00,  0x00,  0xFF,// LightBlue
```

```
    0x00,  0xFF,  0x00,// LightGreen
    0x00,  0xFF,  0xFF,// LightCyan
    0xFF,  0x00,  0x00,// LightRed
    0xFF,  0x00,  0xFF,// LightViolet
    0xFF,  0xFF,  0x00,// LightYellow
    0xFF,  0xFF,  0xFF,// White
};

// Global vars
int savMouseX, savMouseY = 0;
boolean batteryAlertFlash =false;
unsigned int screenColor = Black;
boolean commError =false;
char* headings[] = {"N", "NE", "E", "SE", "S", "SW", "W", "NW", "N
int headingIndex = 9;
char* gpsNSEW[] = {"N", "S", "E", "W" };

// Scooterputer Tasks - TimedAction( msec, funcName )
TimedAction updateDisplayTask = TimedAction( 20, UpdateDisplayTask
TimedAction batteryAlertTask = TimedAction( 500, BatteryAlertTask )

// Screen object definitions
struct ButtonObj
{
  int X;
  int Y;
  Colors Color;
  char *Label;
  void ( *MouseHandler )( );
};

struct LEDDisplayObj
{
  int X;
  int Y;
  Colors Color;
  char SegLen;
  int Digits;
```

```
};

struct TextLabelObj
{
  int X;
  int Y;
  int Height;
  Colors Color;
  boolean Bold;
};

struct BatteryMonitorObj
{
  int X;
  int Y;
  int Width;
  int Height;
  char *MinVolts;
  char *MaxVolts;
  int Percent;
  int ZoneWidth;
  int ZoneHeight;
  int PosX;
  int PosY;
  int RedZoneY;
  boolean AlertActive;
};

struct LeanGaugeObj
{
  int X;
  int Y;
  int Width;
  int Height;
  int Percent;
  int LeanX;
  int StartX;
  int StartY;
```

```
};

struct ImageObj
{
  int X;
  int Y;
  char *bmpFile;
};

//****************************************
// Screen Objects
//

// Buttons
ButtonObj btnSpare1  = { 2, 193, Black," ", Spare1hMouseHandler };
ButtonObj btnLean  = { 55, 193, Black,"Lean", LeanMouseHandler };
ButtonObj btnMaxS  = { 108, 193, Black,"Max S", MaxSMouseHandler }
ButtonObj btnTripA = { 161, 193, Black,"TripA", TripAMouseHandler
ButtonObj btnTripB = { 214, 193, Black,"TripB", TripBMouseHandler
ButtonObj btnSpare2  = { 267, 193, Black," ", Spare2hMouseHandler

// MPH and Max Speed
LEDDisplayObj dispMPH = { 105, 50, LightGreen, 28, 2 };
LEDDisplayObj dispMaxS = { 192, 70, Green, 8, 2 };
TextLabelObj labelMPH = { 192, 103, 10, Green,false };

// Odometer
LEDDisplayObj dispOdometer = { 119, 116, Green, 8, 5 };
TextLabelObj labelODO = { 192, 128, 10, Green,false };

// Ride Time
LEDDisplayObj dispRideHH = { 244, 64, DarkCyan, 8, 2 };
TextLabelObj labelColonR = { 268, 71, 24, DarkCyan,true };
LEDDisplayObj dispRideMM = { 274, 64, DarkCyan, 8, 2 };

// Trip Meters
LEDDisplayObj dispTripA = { 250, 90, Green, 8, 4 };
LEDDisplayObj dispTripB = { 250, 116, Green, 8, 4 };
```

```
TextLabelObj labelA = { 305, 103, 8, Gray,false };
TextLabelObj labelB = { 305, 129, 8, Gray,false };

// Time Of Day
LEDDisplayObj dispTimeHH = { 255, 5, Cyan, 10, 2 };
TextLabelObj labelColonT = { 286, 15, 24, Cyan,true };
LEDDisplayObj dispTimeMM = { 292, 5, Cyan, 10, 2 };

// Date
LEDDisplayObj dispDateMonth = { 255, 35, Gray, 5, 2 };
TextLabelObj labelSlashD = { 273, 41, 6, Gray,true };
LEDDisplayObj dispDateDay = { 280, 35, Gray, 5, 2 };
TextLabelObj labelSlashY = { 298, 41, 6, Gray,true };
LEDDisplayObj dispDateYear = { 305, 35, Gray, 5, 2 };

// Temperature
LEDDisplayObj dispTemp = { 5, 5, Cyan, 10, 3 };
TextLabelObj labelFahr = { 60, 22, 10, Cyan,false };

// Battery Monitor & Lean Gauge
BatteryMonitorObj dispBattMon = { 5, 53, 28, 110,"10.0v", "14.0v",
LeanGaugeObj gaugeLean = { 80, 5, 160, 15, 0, 0, 0, 0 };

// Compass
ImageObj Scooter = { 66, 145,"Scooter.bmp" };
TextLabelObj labelGPSHeading = { 140, 170, 12, Yellow,true };

// GPS Location
TextLabelObj labelGPSLat = { 192, 158, 10, Cyan,false };
LEDDisplayObj dispGPSLatDegrs = { 226, 150, Cyan, 6, 2 };
TextLabelObj labelGPSDecLat = { 246, 158, 18, Cyan,true };
LEDDisplayObj dispGPSLatFrac = { 251, 150, Cyan, 6, 5 };
TextLabelObj labelGPS_NS = { 305, 158, 12, Cyan,true };

TextLabelObj labelGPSLon = { 192, 178, 10, Cyan,false };
LEDDisplayObj dispGPSLonDegrs = { 226, 170, Cyan, 6, 2 };
TextLabelObj labelGPSDecLon = { 246, 178, 18, Cyan,true };
LEDDisplayObj dispGPSLonFrac = { 251, 170, Cyan, 6, 5 };
```

```
TextLabelObj labelGPS_EW = { 305, 178, 12, Cyan,true };

TextLabelObj labelGPSPulse = { 172, 169, 6, LightCyan,true };

// Communication Error Indicator
TextLabelObj labelCommError = { 192, 55, 6, LightRed,false  };

// Screen Object drawing and update methods

//*****************************************
// DrawButton Function
// Object override
//
void DrawButton( ButtonObj &button )
{
  DrawButton( button.X, button.Y, button.Color, button.Label );
}

//*****************************************
// DrawButton Function
// button.X, button.Y = left, top;
//
void  DrawButton(int  x,int  y,unsigned int  color,char *label )
{
  char* bImages[] = {"BlackGloss52.bmp"};

  color = ( color > Gray ) ? ( color % BUTTON_COUNT ) + 1 : color;
  int imgIndex = color;

  // Load the button graphic image
   image( loadImage( bImages[imgIndex] ), x, y );

  // Calculate the location of the button's label
  // TODO: fix it
  int labelLocX = ( x + (BUTTON_W / 2) ) - ( ( (sizeof( label ) / 2
  labelLocX = ( labelLocX > x ) ? labelLocX : x + 1;
  int labelLocY = y + BUTTON_H / 2;
```

```
  // Paint the label text based
  stroke( 0x10 );
  text( label, labelLocX + 1, labelLocY + 1, 8 );
  stroke( 0xE0 );
  text( label, labelLocX, labelLocY, 8 );
}

//*****************************
// Draw7SegmentDisplay_UInt Function
// Object override
//
void DrawLEDDisplay_Integer ( LEDDisplayObj &ledDisplay,long value
{
 DrawLEDDisplay_Integer( ledDisplay.X, ledDisplay.Y, ledDisplay.Co
}

//*****************************
// Draw7SegmentDisplay_UInt Function
// x, y = left, top
//
void DrawLEDDisplay_Integer(int  x,int  y,unsigned int  color,long
{
  stroke( rgb[color][0], rgb[color][1], rgb[color][2] );
  fill( rgb[screenColor][0], rgb[screenColor][1], rgb[screenColor][

 // draw each digit, right to left
  for  (int i = digits - 1; i >= 0; i-- )
  {
    Display7SegmentDigit( x + ( i * segLen * 1.5 ), y, value % 10L
    value = value / 10L;
  }
}

//*****************************
// DrawTextLabel Function
// Object override
//
void DrawTextLabel ( TextLabelObj &textLabel,char  *oldLabel,char
```

```
{
 DrawTextLabel( textLabel.X, textLabel.Y, textLabel.Height, textLa
}

//***************************
// DrawTextLabel Function
// x, y = left, top
//
void DrawTextLabel(int  x,int  y,int  height,unsigned int  color,b
{
 fill( rgb[screenColor][0], rgb[screenColor][1], rgb[screenColor][

  if( oldLabel !="" )
  {
   stroke( rgb[screenColor][0], rgb[screenColor][1], rgb[screenCol
    text( oldLabel, x, y, height );

    if( bold )
      text( oldLabel, x + 1, y, height );
  }

  stroke( rgb[color][0], rgb[color][1], rgb[color][2] );
  text( newLabel, x, y, height );

  if( bold )
    text( newLabel, x + 1, y, height );
}

//***************************
// DrawBattMon Function
// Object override
//
void DrawBattMon ( BatteryMonitorObj &battMon )
{
  // Pre-calculate floating point math
  battMon.ZoneWidth = battMon.Width * 0.66;
  battMon.ZoneHeight = battMon.Height * 0.32;
  battMon.PosX = battMon.X + ( battMon.Width * 0.80 + 0.5 );
```

```
  battMon.RedZoneY = battMon.Y + 5 + ( battMon.ZoneHeight * 2 );

 DrawBattMon( battMon.X, battMon.Y, battMon.Width, battMon.Height,
}

//***************************
// DrawBattMon Function
// x, y = left, top
//
void  DrawBattMon(int  x,int  y,int  width,int  height,char  *minVo
{
  noStroke( );
  fill( 0xA0, 0x40, 0x00 );
  rect( x, y, zoneWidth, zoneHeight );
   fill( 0x00, 0x40, 0x00 );
  rect( x, y + zoneHeight + 3, zoneWidth, zoneHeight );
   fill( 0x40, 0x00, 0x00 );
  rect( x, y + zoneHeight + zoneHeight + 5, zoneWidth, zoneHeight )

  stroke( 0xA0, 0xA0, 0xA0 );
 fill( rgb[screenColor][0], rgb[screenColor][1], rgb[screenColor][
  text( maxVolts, x, y - 12 );
  text( minVolts, x, y + height + 7 );
}

//***************************
// UpdateBattMon Function
// Updates the indicator; percent 0 to 100
//
void UpdateBattMon( BatteryMonitorObj &battMon,unsigned int  color
{
  int bottomY = battMon.Y + battMon.Height;
  int posY;

  // Constrain percent range: 0 to 100
  percent = ( percent > 100 ) ? 100 : ( percent < 0 ) ? 0 : percent

  noStroke( );
```

```
  // Erase the old indication
  posY = bottomY - ( ( battMon.Percent * battMon.Height ) / 100 );
 fill( rgb[screenColor][0], rgb[screenColor][1], rgb[screenColor][
  triangle( battMon.PosX, posY, battMon.PosX + 10, posY - 10, battM

  // Draw the new indication
  posY = bottomY - ( ( percent * battMon.Height ) / 100 );
 fill( rgb[color][0], rgb[color][1], rgb[color][2] );
  triangle( battMon.PosX, posY, battMon.PosX + 10, posY - 10, battM

  // Flash red zone if battery in trouble
  if( posY > battMon.RedZoneY )
  {
    if( !battMon.AlertActive )
    {
     batteryAlertTask.enable( );
      fill( 0xFF, 0x00, 0x00 );
     rect( battMon.X, battMon.RedZoneY, battMon.ZoneWidth, battMon
      battMon.AlertActive =true;
    }
  }
  else
  {
    if( battMon.AlertActive )
    {
     batteryAlertTask.disable( );
      fill( 0x40, 0x00, 0x00 );
     rect( battMon.X, battMon.RedZoneY, battMon.ZoneWidth, battMon
      battMon.AlertActive =false;
    }
  }

  battMon.Percent = percent;
}

//***************************
// DrawLeanGauge Function
```

```
// Object override
//
void DrawLeanGauge( LeanGaugeObj &leanGauge )
{
  leanGauge.StartX = leanGauge.X + leanGauge.Width / 2;
  leanGauge.StartY = leanGauge.Y + 1;
  leanGauge.LeanX = leanGauge.StartX;
  leanGauge.Percent = 0;

 DrawLeanGauge( leanGauge.X, leanGauge.Y, leanGauge.Width, leanGau
}

//***************************
// DrawLeanGauge Function
// x, y = left, top
//
void  DrawLeanGauge(int  x,int  y,int  width,int  height,int  start
{
  stroke( rgb[Gray][0], rgb[Gray][1], rgb[Gray][2] );
  fill( rgb[Black][0], rgb[Black][1], rgb[Black][2] );
  rect( x, y, width, height );

  line( startX, y, startX, y - 3 );
  line( startX, y + height, startX, y + height + 3 );

   text("LEAN", startX - 14, y + height + 10, 8 );
   text("L", x, y + height + 8, 8 );
   text("R", x + width, y + height + 8, 8 );
}

//***************************
// UpdateLeanGauge Function
// percent -100 to 100
//
void UpdateLeanGauge( LeanGaugeObj &leanGauge,unsigned int  color,
{
  int barX = 0;
  int width = 0;
```

```
 int height = leanGauge.Height - 2;
 int rangeColor = DarkGreen;

 // Constrain percent range: -100 to 100
 percent = ( percent > 100 ) ? 100 : ( percent < -100 ) ? -100 : p

 noStroke( );

// First handle cases where values are zero or where the new readi
 if( leanGauge.Percent == 0 && percent == 0 )
 {
  fill( rgb[color][0], rgb[color][1], rgb[color][2] );
  rect( leanGauge.StartX, leanGauge.StartY, 1, height );
  leanGauge.LeanX = barX = leanGauge.StartX;
 }
 else if( leanGauge.Percent > 0 && percent < 0 )
 {
  fill( rgb[rangeColor][0], rgb[rangeColor][1], rgb[rangeColor][2
  rect( leanGauge.StartX, leanGauge.StartY, leanGauge.LeanX - lea
  leanGauge.LeanX = barX = leanGauge.StartX;
 }
 else if( leanGauge.Percent < 0 && percent > 0 )
 {
  fill( rgb[rangeColor][0], rgb[rangeColor][1], rgb[rangeColor][2
  rect( leanGauge.LeanX, leanGauge.StartY, leanGauge.StartX - lea
  leanGauge.LeanX = barX = leanGauge.StartX;
 }

 // Now draw or erase lean bar based on the new percent's value
 // in relation to the old percent's (leanGauge.Percent) value
 if( percent > leanGauge.Percent )
 {
   if( percent >= 0 )
   {
     barX = leanGauge.StartX + ( ( percent * ( ( leanGauge.Width -
    fill( rgb[color][0], rgb[color][1], rgb[color][2] );
   }
   else
```

```
    {
      barX = leanGauge.StartX - ( ( percent * -1 * ( ( leanGauge.Wi
     fill( rgb[rangeColor][0], rgb[rangeColor][1], rgb[rangeColor]
    }

    width = barX - leanGauge.LeanX;
    rect( leanGauge.LeanX, leanGauge.StartY, width, height);
  }
  else if( percent < leanGauge.Percent )
  {
    if( percent >= 0 )
    {
      barX =  leanGauge.StartX + ( ( percent * ( ( leanGauge.Width
     fill( rgb[rangeColor][0], rgb[rangeColor][1], rgb[rangeColor]
    }
    else
    {
      barX = leanGauge.StartX - ( ( percent * -1 * ( ( leanGauge.Wi
     fill( rgb[color][0], rgb[color][1], rgb[color][2] );
    }

    width = leanGauge.LeanX - barX;
    rect( barX, leanGauge.StartY, width, height);
  }

  // Save off the new lean X and percent values
  leanGauge.LeanX = barX;
  leanGauge.Percent = percent;
}

//***************************
// ResetLeanGauge Function
// Erases the min/max lean indications (dark green)
//
void ResetLeanGauge( LeanGaugeObj &leanGauge,unsigned int color )
{
  int height = leanGauge.Height - 2;
```

```
  noStroke( );
  fill( rgb[Black][0], rgb[Black][1], rgb[Black][2] );
  rect( leanGauge.X + 1, leanGauge.Y + 1, leanGauge.Width - 2, heig

  fill( rgb[color][0], rgb[color][1], rgb[color][2] );

  if( leanGauge.Percent >= 0.0 )
    rect( leanGauge.StartX, leanGauge.StartY, leanGauge.LeanX - lea
  else
    rect( leanGauge.LeanX, leanGauge.StartY, leanGauge.StartX - lea
}

//***************************
// UpdateHeading Function
// 0L <= headingValue <= 36000L
//
void  UpdateHeading(long headingValue )
{
    int newHeadingIndex;

    // Check value range and set heading string array index
    if( ( headingValue >= 0L ) && ( headingValue <= 36000L ) )
       newHeadingIndex = (int )( ( headingValue / 4500.0 ) + 0.5 );
    else
      newHeadingIndex = 9;

   DrawTextLabel( labelGPSHeading, headings[headingIndex] , headin
    headingIndex = newHeadingIndex;
}

//***************************************
// DrawImage Function
//  Object override
//
void DrawImage( ImageObj &bmpImage )
{
  DrawImage( bmpImage.X, bmpImage.Y, bmpImage.bmpFile );
}
```

```
//****************************************
// DrawImage Function
// x, y = left, top;
//
void  DrawImage(int  x,int  y,char *bmpImage )
{
   image( loadImage( bmpImage ), x, y );
}

// Simple Serial Protocol
#define  STX       2// Start of text
#define  ETX       3// End of text
#define  EOT       4// End of transmission
#define  ENQ       5// Enquiry
#define  TAB       9// Tab delimiter

// Sensor Values
unsigned int valMPH, savMPH = 0;
unsigned int valMaxS, savMaxS = 0;
unsigned long valODO, savODO = 0L;
unsigned int valTripA, savTripA = 0;
unsigned int valTripB, savTripB = 0;
unsigned int valGPSPulse, savGPSPulse = 0;
unsigned int valSpareInt, savSpareInt = 0;
unsigned int valTimeHH, savTimeHH = 0;
unsigned int valTimeMM, savTimeMM = 0;
unsigned int valDateMonth, savDateMonth = 0;
unsigned int valDateDay, savDateDay = 0;
unsigned int valDateYear, savDateYear = 0;
int valTemp, savTemp = 0;
unsigned int valBatt, savBatt = 0;
int valLean, savLean = 0;
unsigned long valHeading, savHeading = 0L;
unsigned int valLatDegrs, savLatDegrs = 0;
unsigned long valLatFrac, savLatFrac = 0L;
unsigned int valLatNS, savLatNS = 0;
unsigned int valLonDegrs, savLonDegrs = 0;
```

```
unsigned long valLonFrac, savLonFrac = 0L;
unsigned int valLonEW, savLonEW = 2;


//***************************
// ButtonDown Function
// Returns true if touch is within the bounds of the passed button
//
boolean ButtonDown( ButtonObj &button )
{
  if( mouseX >= button.X && mouseX <= button.X + BUTTON_W &&
      mouseY >= button.Y && mouseY <= button.Y + BUTTON_H )
  {
    button.MouseHandler( );
    return true;
  }

  return false;
}

//***************************
// ButtonUp Function
// Reposition mouseX, mouseY (wish there was a better way!)
//
void ButtonUp(  )
{
  mouseX = savMouseX;
  mouseY = savMouseY;
}

//***************************
// Comm Button Mouse Handler
//
void Spare1hMouseHandler( )
{
}

//***************************
```

```
// Lean Button Mouse Handler
// Reset Lean Gauge min and max indications
//
void LeanMouseHandler( )
{
  ResetLeanGauge( gaugeLean, LightGreen );
}

//***************************
// Max Speed Button Mouse Handler
// Reset MaxSpeed
//
void MaxSMouseHandler( )
{
  valMaxS = savMaxS = 0;
  DrawLEDDisplay_Integer( dispMaxS, valMaxS );
}

//***************************
// Trip A Button Mouse Handler
// Reset TripA Meter
//
void TripAMouseHandler( )
{
  Serial.println( "-A" );
}

//***************************
// Trip B Button Mouse Handler
// Reset TripB Meter
//
void TripBMouseHandler( )
{
  Serial.println( "-B" );
}

//***************************
// Ride Button Mouse Handler
```

```
//
void Spare2hMouseHandler( )
{
}

//*********************************
// CheckButtonPresses Function
//
void CheckButtonPresses( )
{
  savMouseX = mouseX;
  savMouseY = mouseY;

  gettouch( );

  if( ButtonDown( btnSpare1 ) )
    ButtonUp( );
  else if( ButtonDown( btnLean ) )
    ButtonUp( );
  else if( ButtonDown( btnMaxS ) )
    ButtonUp( );
  else if( ButtonDown( btnTripA ) )
    ButtonUp( );
  else if( ButtonDown( btnTripB ) )
    ButtonUp( );
  else if( ButtonDown( btnSpare2 ) )
    ButtonUp( );
//  else
//    updateDisplayTask.disable( );
}

//*********************************
// Setup - one time initialization
//
void setup()
{
  // Uncomment the following line to upload images to the TouchShie
  // Press the Slide prog button, download, wait for prompt, then s
```

```
  // Comment this out after images have been uploaded so that it do
//  open(FlashTransfer);

  DrawScreen( );

  // Init timed tasks
  updateDisplayTask.enable( );
  batteryAlertTask.disable( );

  Serial.begin( 19200 );
  Serial.flush( );
}

//**********************************
// DrawScreen Function
//
void DrawScreen( )
{
  // Black background
  background( rgb[screenColor][0], rgb[screenColor][1], rgb[screenC

  // Buttons
  DrawButton( btnSpare1 );
  DrawButton( btnLean );
  DrawButton( btnMaxS );
  DrawButton( btnTripA );
  DrawButton( btnTripB );
  DrawButton( btnSpare2 );

  // MPH and Max Speed
  DrawLEDDisplay_Integer( dispMPH, valMPH );
  DrawLEDDisplay_Integer( dispMaxS, valMaxS );
   DrawTextLabel( labelMPH,"", "MPH" );

  // Odometer
  DrawLEDDisplay_Integer( dispOdometer, valODO );
   DrawTextLabel( labelODO,"", "ODO" );
```

```
// Trip Meters
DrawLEDDisplay_Integer( dispTripA, valTripA );
DrawLEDDisplay_Integer( dispTripB, valTripB );
 DrawTextLabel( labelA,"", "A" );
 DrawTextLabel( labelB,"", "B" );

// Battery Monitor
DrawBattMon( dispBattMon );
UpdateBattMon( dispBattMon, LightYellow, 50 );

// Lean Gauge
DrawLeanGauge( gaugeLean );
UpdateLeanGauge( gaugeLean, LightGreen, 0 );

// Time
DrawLEDDisplay_Integer( dispTimeHH, valTimeHH );
 DrawTextLabel( labelColonT,"", ":" );
DrawLEDDisplay_Integer( dispTimeMM, valTimeMM );

// Date
DrawLEDDisplay_Integer( dispDateMonth, valDateMonth );
 DrawTextLabel( labelSlashD,"", "/" );
DrawLEDDisplay_Integer( dispDateDay, valDateDay );
 DrawTextLabel( labelSlashY,"", "/" );
DrawLEDDisplay_Integer( dispDateYear, valDateYear );

// Temperature
DrawLEDDisplay_Integer( dispTemp, valTemp );
 DrawTextLabel( labelFahr,"", "F" );

// Compass
DrawImage( Scooter );
 DrawTextLabel( labelGPSHeading,"", headings[9] );

// GPS
 DrawTextLabel( labelGPSLat,"", "LAT" );
DrawLEDDisplay_Integer( dispGPSLatDegrs, valLatDegrs );
 DrawTextLabel( labelGPSDecLat,"", "." );
```

```
  DrawLEDDisplay_Integer( dispGPSLatFrac, valLatFrac );
   DrawTextLabel( labelGPS_NS,"", gpsNSEW[0] );

   DrawTextLabel( labelGPSLon,"", "LON" );
  DrawLEDDisplay_Integer( dispGPSLonDegrs, valLonDegrs );
   DrawTextLabel( labelGPSDecLon,"", "." );
  DrawLEDDisplay_Integer( dispGPSLonFrac, valLonFrac );
   DrawTextLabel( labelGPS_EW,"", gpsNSEW[2] );

   DrawTextLabel( labelGPSPulse,"", "o" );
}

//**********************************
// Loop - the main gear
//
void loop( )
{
  // Check to see if any of the buttons have been pressed
  CheckButtonPresses( );

  // Check for data from Arduino
  AR_CheckSerial( );

  // Update the dsplay values if time is up
  updateDisplayTask.check( );
  batteryAlertTask.check( );
}

//**********************************
// CheckSerial Function
// Reads serial data from Arduino
//
boolean AR_CheckSerial( )
{
  if( Serial.available( ) >= 53 )
  {
    if( Serial.read( ) == STX )
    {
```

```
valMPH = AR_ReadInt( );

valODO = AR_ReadLong( );

valTripA = AR_ReadInt( );
valTripB = AR_ReadInt( );

valGPSPulse = AR_ReadInt( );
valSpareInt = AR_ReadInt( );

valTimeHH = AR_ReadInt( );
valTimeMM = AR_ReadInt( );

valDateMonth = AR_ReadInt( );
valDateDay = AR_ReadInt( );
valDateYear = AR_ReadInt( );

valTemp = AR_ReadInt( );
valBatt = AR_ReadInt( );
valLean = AR_ReadInt( );

valHeading = AR_ReadLong( );
valLatDegrs = AR_ReadInt( );
valLatFrac = AR_ReadLong( );
valLatNS = AR_ReadInt( );

valLonDegrs = AR_ReadInt( );
valLonFrac = AR_ReadLong( );
valLonEW = AR_ReadInt( );

// Last char in data packet should be ETX
if( Serial.read( ) == ETX )
{
  if( commError )
  {
    // Erase COMM ERROR indication
    DrawTextLabel( labelCommError,"COMM", "" );
     commError =false;
```

```
        }

        return true;    // Return true if packet received
      }
    }
    // No STX/ETX is no good - clear the pipes and try again
    Serial.flush( );
    DrawTextLabel( labelCommError,"", "COMM" );
     commError =true;
   }

  return false;    // Return false if packet not received
}

//**********************************
// AR_ReadInt Function
// Reads an integer value from Arduino
//
int AR_ReadInt( )
{
  int value  =Serial.read( );
  value = ( value << 8 ) +Serial.read( );

  return value;
}

//**********************************
// AR_ReadLong Function
// Reads a long value from Arduino
//
long AR_ReadLong( )
{
  long value  =Serial.read( );
  value = ( value << 8 ) +Serial.read( );
  value = ( value << 8 ) +Serial.read( );
  value = ( value << 8 ) +Serial.read( );

  return value;
```

```
}

//***************************
// Battery Alert task - invoked by loop( ) every 'n' msecs
// Flashes the BattMon red zone if battery is tanking
// Task is enabled/disabled in UpdateBattMon( )
//
void BatteryAlertTask( )
{
  noStroke( );

  if( batteryAlertFlash )
  {
    fill( 0x40, 0x00, 0x00 );
   rect( dispBattMon.X, dispBattMon.RedZoneY, dispBattMon.ZoneWidt
  }
  else
  {
    fill( 0xFF, 0x00, 0x00 );
   rect( dispBattMon.X, dispBattMon.RedZoneY, dispBattMon.ZoneWidt
  }

  batteryAlertFlash = ( batteryAlertFlash ) ?false  :true;
}

//***************************
// Update Display task - invoked by loop( ) every 'n' msecs
//
void UpdateDisplayTask( )
{
  // Miles Per Hour
  if( valMPH != savMPH )
  {
   DrawLEDDisplay_Integer( dispMPH, valMPH );
    savMPH = valMPH;

    if( valMaxS < valMPH )
    {
```

```
    valMaxS = valMPH;
    DrawLEDDisplay_Integer( dispMaxS, valMaxS );
  }
}
// Odometer
if( valODO != savODO )
{
 DrawLEDDisplay_Integer( dispOdometer, valODO );
  savODO = valODO;
}

// Another data packet from Arduino?
AR_CheckSerial( );

// Trip Meter A
if( valTripA != savTripA )
{
 DrawLEDDisplay_Integer( dispTripA, valTripA );
  savTripA = valTripA;
}
// Trip Meter B
if( valTripB != savTripB )
{
 DrawLEDDisplay_Integer( dispTripB, valTripB );
  savTripB = valTripB;
}
// GPS acquisition indicator
if( valGPSPulse != savGPSPulse )
{
  if( valGPSPulse > 0 )
    DrawTextLabel( labelGPSPulse,"o", "O" );
  else
    DrawTextLabel( labelGPSPulse,"O", "o" );

  savGPSPulse = valGPSPulse;
}

// Any more data from Arduino?
```

```
AR_CheckSerial( );

// Time Of Day hours
if( valTimeHH != savTimeHH )
{
 DrawLEDDisplay_Integer( dispTimeHH, valTimeHH );
  savTimeHH = valTimeHH;
}
// Time Of Day minutes
if( valTimeMM != savTimeMM )
{
 DrawLEDDisplay_Integer( dispTimeMM, valTimeMM );
  savTimeMM = valTimeMM;
}
// Date
if( ( valDateDay != savDateDay ) || ( valDateMonth != savDateMont
{
 DrawLEDDisplay_Integer( dispDateMonth, valDateMonth );
 DrawLEDDisplay_Integer( dispDateDay, valDateDay );
 DrawLEDDisplay_Integer( dispDateYear, valDateYear );
  savDateMonth = valDateMonth;
  savDateDay = valDateDay;
  savDateYear = valDateYear;
}

// Keep the data from Arduino coming in...
AR_CheckSerial( );

// Temperature
if( valTemp != savTemp )
{
 DrawLEDDisplay_Integer( dispTemp, valTemp );
  savTemp = valTemp;
}
// Battery Monitor
if( valBatt != savBatt )
{
 UpdateBattMon( dispBattMon, LightYellow, valBatt );
```

```
  savBatt = valBatt;
}
// Lean Gauge
int leanDiff = valLean - savLean;
if( leanDiff < 0 )
  leanDiff *= -1;

if( ( leanDiff > 4 ) && ( valLean > -100 ) && ( valLean < 100 ) )
{
 UpdateLeanGauge( gaugeLean, LightGreen, valLean );
  savLean = valLean;
}
// Compass Heading
if( valHeading != savHeading )
{
 UpdateHeading( valHeading );
 savHeading = valHeading;
}

// Can't ignore Mr. Arduino...
AR_CheckSerial( );

// Latitude degrees
if( valLatDegrs != savLatDegrs )
{
 DrawLEDDisplay_Integer( dispGPSLatDegrs, valLatDegrs );
 savLatDegrs = valLatDegrs;
  }
// Latitude fractional
if( valLatFrac != savLatFrac )
{
 DrawLEDDisplay_Integer( dispGPSLatFrac, valLatFrac );
 savLatFrac = valLatFrac;
  }
// Latitude NS
if( valLatNS != savLatNS )
{
 DrawTextLabel( labelGPS_NS, gpsNSEW[savLatNS], gpsNSEW[valLatNS
```

```
    savLatNS = valLatNS;
  }

  // Arduinny can be demanding...
  AR_CheckSerial( );

  // Longitude degrees
  if( valLonDegrs != savLonDegrs )
  {
   DrawLEDDisplay_Integer( dispGPSLonDegrs, valLonDegrs );
    savLonDegrs = valLonDegrs;
  }
  // Longitude fractional part
  if( valLonFrac != savLonFrac )
  {
   DrawLEDDisplay_Integer( dispGPSLonFrac, valLonFrac );
    savLonFrac = valLonFrac;
  }
  // Longitude EW
  if( valLonEW != savLonEW )
  {
   DrawTextLabel( labelGPS_EW, gpsNSEW[savLonEW], gpsNSEW[valLonEW
    savLonEW = valLonEW;
  }
}
```