```
/*****************************
 Scooterputer
 Logic / Sensor Module - Arduino

 Created April 23, 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)
: Accumulate trip meter & ODO values every 1/25th of mile

*****************************/
#include <TimedAction.h>
#include <NewSoftSerial.h>
#include <TinyGPS.h>
#include<DallasTemperature.h>
#include <OneWire.h>
#include <Wire.h>
#include <string.h>

// Uncomment the defs to send values to Serial for debug
// GPS is using the serial port and will be disabled while debuggin
//#define DEBUG_ACCEL
//#define DEBUG_TEMP
//#define DEBUG_BATTMON
//#define DEBUG_RTC
//#define DEBUG_CELL
//#define DEBUG_BUZZER
//#define DEBUG_RESETS

// Digital pin assignments
#define   GPS_RX_PIN            0// GPS serial receive data
```

```
#define   GPS_TX_PIN            1// GPS serial transmit data
#define    CELL_RX_PIN          2// CELL serial receive data
#define    CELL_TX_PIN          3// CELL serial transmit data
#define   TS_TX_PIN             4// TouchShield serial transmit d
#define   TS_RX_PIN             5// TouchShield serial receive da
#define   TEMP_PIN              6// Temperature sensor DS18B20
#define   BUZZ_PIN              7// Buzzer 2.048Khz

// Analog pin assignments
#define    ACCEL_X_PIN          0// Accelerometer ADXL-320 X axis
#define    ACCEL_Y_PIN          1// Accelerometer ADXL-320 Y axis
#define   BATT_PIN              2// Battery Monitor level
#define    RTC_SDA_PIN          4// Real time clock DS1307 data
#define    RTC_SCL_PIN          5// Real time clock DS1307 clock

// RTC non-volatile storage
#define    RTC_I2C_ADDR       0x68// Real time clock address on th
#define   RAM_ADDR               8// RTC RAM registers start at ad
#define  TRIPA_ADDR       RAM_ADDR// Trips, ODO, lat and long a
#define  TRIPB_ADDR      TRIPA_ADDR + 4
#define ODO_ADDR     TRIPB_ADDR + 4
#define LAT_ADDR     ODO_ADDR + 4
#define LON_ADDR     LAT_ADDR + 4

#define MILES_PER_METER     0.00062137f
#define EARTH_RADIUS_METERS 6372795.0f

// Simple serial protocol
#define  STX            0x02// Start of text
#define  ETX            0x03// End of text
#define  CR             0x0D// Carriage return
#define   CTRL_Z        0x1A// Control-z

// Tasks - TimedAction( msec, funcName )
TimedAction RTC_UpdateTask = TimedAction( 1000, RTC_Update );
TimedAction TEMP_UpdateTask = TimedAction( 10000, TEMP_Update );
TimedAction TS_SendDataPacketTask = TimedAction( 250, TS_SendDataPa
TimedAction TS_ReceiveDataPacketTask = TimedAction( 1000, TS_Recei
```

```
TimedAction CELL_NetConnectTimeoutTask = TimedAction( 20000, CELL_N
TimedAction CELL_ReplyTimeoutTask = TimedAction( 5000, CELL_ReplyTi

// Interfaces
NewSoftSerial TS_Serial  =NewSoftSerial( TS_RX_PIN, TS_TX_PIN );
NewSoftSerial CELL_Serial  =NewSoftSerial( CELL_RX_PIN, CELL_TX_PIN
OneWire oneWire( TEMP_PIN );
DallasTemperature TEMP_Sensor( &oneWire );
TinyGPS gps;

// Cell protocol Command - Reply
enum CellCmnd
{
  Cmnd_None,
  Cmnd_Connect,
  Cmnd_SMS_Type,
  Cmnd_SMS_Indication,
  Cmnd_CallForwarding,
  Cmnd_Location,
  Cmnd_SendGPSMessage,
  Cmnd_DeleteMsgs,
  Cmnd_OK
} nextCellCmnd = Cmnd_Connect;

enum CellReply
{
  Reply_None,
  Reply_Connect,
  Reply_SMS_Type,
  Reply_SMS_Indication,
  Reply_CallForwarding,
  Reply_Location,
  Reply_SendGPSMessage,
  Reply_MessageText,
  Reply_DeleteMsgs,
  Reply_OK
} nextCellReply = Reply_Connect;
```

```
// Cell Module
#define BUFFSIZE    100

// Private cell phone number for fowarding calls and sending locati
// Put your cell phone number here...
char *privCellPhone ="+15555551212";

char cellMsgBuff[ BUFFSIZE ] ="";
char cellCmndBuff[ BUFFSIZE ] ="";
char cellPhoneNumber[ 20 ] ="";
int cellMsgBuffIndex = 0;
char incomingChar  =  0;
int connectRespCnt = 0;
bool cellConnected =false;
bool cellNetConnectTimeout =false;
bool cellReplyTimeout =false;

// GPS
float latitude = 0.0;
float longitude = 0.0;
float f_TripA = 0.0;
float f_TripB = 0.0;
float f_ODO = 0.0;

// Sensor values
int  valMPH                  =   0;// Miles per hour, GPS: 0 to 10
long  valODO                 =  0L;// Odometer in miles, internal:
int  valTripA                =   0;// Trip A miles, internal: 0 to
int  valTripB                =   0;// Trip B miles, internal: 0 to
int   valGPSPulse            =   0;// Indicator of GPS acquisitior
int   valSpareInt            =   0;// Spare integer value
int   valTimeHH              =   0;// Current time hours, RTC: 1 t
int   valTimeMM              =   0;// Current time minutes, RTC: 0
int    valDateMonth          =   0;// Current date month, RTC: 1 t
int   valDateDay             =   0;// Current date day, RTC: 1 to
int   valDateYear            =   0;// Current date year, RTC: 0 to
int  valTemp                 =   0;// Temperature, TEMP: -255 to 2
int  valBatt                 =   0;// Battery level, BATT: 0 to 10
```

```
int  valLean                =   0;// Lean amount, ACCEL: -90 to 9
long   valHeading           =    0L;// Heading, GPS: 0 to 36000 (de
int   valLatDegrs           =    0;// Latitude degrees, GPS:
long   valLatFrac           =    0L;// Latitude fraction, GPS:
int  valLatNS               =   0;// Latitude NS, GPS: N = 0, S =
int   valLonDegrs           =    0;// Longitude degrees, GPS:
long   valLonFrac           =    0L;// Longitude fraction, GPS:
int  valLonEW               =   2;// Longitude EW, GPS: E = 2, W

void setup( )
{
  pinMode(  BUZZ_PIN,OUTPUT );

  Serial.begin(          9600          );// Start GPS interface
  CELL_Serial.begin(       4800        );// Start CELL Module inte
  TS_Serial.begin(        19200        );// Start TouchShield inte
  TEMP_Sensor.begin(                   );// Start Temperature inte
  Wire.begin();                          // Start RTC interface

                TEMP_Update(           );// Get initial temperatur
                                         // (modified DallasTemper

          RTC_UpdateTask.enable(       );// Enable task for readin
         TEMP_UpdateTask.enable(       );// Enable task for readin
      TS_SendDataPacketTask.enable(    );// Enable task for sendin
    TS_ReceiveDataPacketTask.enable(   );// Enable task for receiv
   CELL_NetConnectTimeoutTask.enable(  );// Enable task for Cell N
    CELL_NetConnectTimeoutTask.reset(  );// Wait a full time perio
      CELL_ReplyTimeoutTask.disable(   );// Disable the Cell reply

  //  Read trip, ODO, and location data from RTC storage to init va
  f_TripA = RTC_ReadFloat( TRIPA_ADDR );
  f_TripB = RTC_ReadFloat( TRIPB_ADDR );
  f_ODO   = RTC_ReadFloat( ODO_ADDR );

  latitude = RTC_ReadFloat( LAT_ADDR );
  longitude = RTC_ReadFloat( LON_ADDR );
```

```
  valTripA = f_TripA;
  valTripB = f_TripB;
  valODO   = f_ODO;

//  KEEP THIS - Needed to set the odometer
//  f_ODO = 5577.0f;
//  valODO = f_ODO;
//  RTC_WriteFloat( ODO_ADDR, f_ODO );

//  KEEP THIS - Needed to set the time and date in the DS1307 RTC
//  second, minute, hour, dayOfWeek, dayOfMonth, month, year
//  dayOfWeek: 1 thru 7, Sunday thru Saturday
//  RTC_SetTimeDate( 0, 25, 10, 1, 11, 7, 10 );

//  KEEP THIS - Needed to re-init LS23060 GPS if reset back to defa
//  Serial.println( "$PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29
//  Serial.println( "$PMTK251,19200*22" );  // Data rate is 19200 b
//  Serial.println( "$PMTK251,9600*17" );   // Data rate is 9600 ba
//  Serial.println( "$PMTK251,4800*14" );   // Data rate is 4800 ba
//  Serial.println( "$PMTK220,500*2B" );    // Send sentences every

//  digitalWrite( GPS_TX_PIN, HIGH );     // Keep the TX pin HIGH

       BUZZ_Play(    );// Signal setup complete

  // Serial comm to PC if any debug defs exist
  // Be sure to disable all serial comm when uploading to the Touch
  #if defined DEBUG_ACCEL || defined DEBUG_TEMP || defined DEBUG_BA
  Serial.println( "Debug enabled..." );
  #define DEBUG
  #endif
}

//************************
// Ready to rock...
//
void loop( )
{
```

```
  // Get updates from the sensors
       #ifndef    DEBUG// Can't use GPS if serial port is being use
  GPS_Update( );
  #endif

  ACCEL_Update( );
  BATTMON_Update( );
  RTC_UpdateTask.check( );
  TEMP_UpdateTask.check( );

  CELL_Update( );
  CELL_Actions( );
 CELL_NetConnectTimeoutTask.check( );
  CELL_ReplyTimeoutTask.check( );

  if( cellNetConnectTimeout || ( nextCellCmnd == Cmnd_None ) && ( n
  {
    // Send updates to the display and check for button presses
   TS_SendDataPacketTask.check( );
   TS_ReceiveDataPacketTask.check( );
  }
}

//************************
// Accelerometer AXDL-320
//
void ACCEL_Update( )
{
  int accelX  =analogRead( ACCEL_X_PIN );
  int accelY  =analogRead( ACCEL_Y_PIN );

  // Full range is 453 to 580 (127)
   valLean =map( accelY, 473, 560, 90, -90 );

  #ifdef DEBUG_ACCEL
  Serial.print( "ACCEL X, Y = ");
  Serial.print(  accelX,DEC );
  Serial.print( ", " );
```

```
  Serial.print(  accelY,DEC );
  Serial.print( " " );
  Serial.print( "valLean = " );
  Serial.println(  valLean,DEC );
  #endif
}

//************************
// Temperature DS18B20
// Timed action every 32 secs
//
void TEMP_Update( )
{
 TEMP_Sensor.requestTemperatures( );
  valTemp = (int )TEMP_Sensor.getTempFByIndex( 0 );

  #ifdef DEBUG_TEMP
  Serial.print( "valTemp = " );
  Serial.println(  valTemp,DEC );
  #endif
}

//************************
// Battery Monitor
//
void BATTMON_Update( )
{
  int battLevel =analogRead( BATT_PIN );

  valBatt =map( battLevel, 604, 870, 0, 100 );

  #ifdef DEBUG_BATTMON
  Serial.print( "BATT = ");
  Serial.print(  battLevel,DEC );
  Serial.print( " " );
  Serial.print( "valBatt = " );
  Serial.println(  valBatt,DEC );
  #endif
```

```
}

//************************
// Realtime Clock DS1307
// Timed action every second
//
void RTC_Update( )
{
  // Reset the register pointer
  Wire.beginTransmission( RTC_I2C_ADDR );
  Wire.send( 0 );
  Wire.endTransmission( );

  Wire.requestFrom( RTC_I2C_ADDR, 7 );

  // Need to mask off the control bits on a couple of these
  byte second     = bcdToDec(Wire.receive( ) & 0x7f );
  byte minute     = bcdToDec(Wire.receive( ) );
  byte hour       = bcdToDec(Wire.receive( )  &  0x1f  );// 0x3f fo
  byte dayOfWeek  = bcdToDec(Wire.receive( ) );
  byte dayOfMonth = bcdToDec(Wire.receive( ) );
  byte month      = bcdToDec(Wire.receive( ) );
  byte year       = bcdToDec(Wire.receive( ) );

  valTimeHH       = (int )hour;
  valTimeMM       = (int )minute;
  valDateDay      = (int )dayOfMonth;
  valDateMonth    = (int )month;
  valDateYear     = (int )year;

 #ifdef DEBUG_RTC
  Serial.print(  hour,DEC );
  Serial.print( ":" );
  Serial.print(  minute,DEC );
  Serial.print( ":" );
  Serial.print(  second,DEC );
  Serial.print( "  " );
  Serial.print(  month,DEC );
```

```
  Serial.print( "/" );
  Serial.print(  dayOfMonth,DEC );
  Serial.print( "/" );
  Serial.print(  year,DEC );
  Serial.print( "  Day_of_week:" );
  Serial.println(  dayOfWeek,DEC );

  Serial.print(  valTimeHH,DEC );
  Serial.print( ":" );
  Serial.print(  valTimeMM,DEC );
  Serial.print( "  " );
  Serial.print(  valDateMonth,DEC );
  Serial.print( "/" );
  Serial.print(  valDateDay,DEC );
  Serial.print( "/" );
  Serial.println(  valDateYear,DEC );
  #endif
}

//*************************
// GPS LS23060
//

  long lat, lon;
  float f_lat, f_lon;
  unsigned long age = 0;
  unsigned long heading = 0;
  float mph = 0.0;
  float distance = 0.0;
  int incomingByte;

/*
  unsigned long date, time, chars;
  int year;
  byte month, day, hour, minute, second, hundredths;
  unsigned short sentences, failed;

  float fHeading = gps.f_course( );
```

```
  alt = gps.altitude( );
  gps.get_datetime( &date, &time, &age );
  gps.crack_datetime(&year, &month, &day, &hour, &minute, &second,
  gps.stats(&chars, &sentences, &failed);
*/

void GPS_Update( )
{
  while( Serial.available( ) )
  {
    incomingByte =Serial.read( );

    if(  gps.encode(  incomingByte  )  )// Returns true if valid
    {
     gps.get_position( &lat, &lon, &age );
     gps.f_get_position( &f_lat, &f_lon, &age );
     heading = gps.course( );
     mph = gps.f_speed_mph( );

     // If we're moving, calculate and save distance traveled
     if( mph > 2.0f )
     {
       if( latitude != 0.0f )
       {
         float distance = DistanceBetween2Points( latitude, longit

         if(  distance  >=  0.04f  )// Accumulate every 25th of
         {
           f_TripA += distance;
           f_TripB += distance;
           f_ODO += distance;

           if(  f_TripA  >  9999.9f  )// Reset trip meters after 9
             f_TripA = 0.0f;

           if( f_TripB > 9999.9f )
             f_TripB = 0.0f;
```

```
            valTripA  =  f_TripA;// Update display values
        valTripB = f_TripB;
        valODO = f_ODO;

        latitude = f_lat;
        longitude = f_lon;

        // Save trips and ODO floats to RTC storage
       RTC_WriteFloat( TRIPA_ADDR, f_TripA );
       RTC_WriteFloat( TRIPB_ADDR, f_TripB );
       RTC_WriteFloat( ODO_ADDR, f_ODO );
      }
    }
    else
    {
      latitude = f_lat;
      longitude = f_lon;
    }
    // Save latitude and longitude to RTC storage
   RTC_WriteFloat( LAT_ADDR, latitude );
   RTC_WriteFloat( LON_ADDR, longitude );
}

// N=0 S=1 E=2 W=3
  valLatNS = ( lat < 0 ) ? 1 : 0;// +Lat is North, -Lat i
 lat = ( lat < 0 ) ? lat * -1 : lat;// Now make latitude abs

  valLonEW = ( lon < 0 ) ? 3 : 2;// +Lon is East, - Lon i
 lon = ( lon < 0 ) ? lon * -1 : lon;// Now make longitude ab

valLatDegrs = lat / 100000L;
valLatFrac  = lat % 100000L;
valLonDegrs = lon / 100000L;
valLonFrac  = lon % 100000L;

                valMPH    =    mph;// Update MPH display

// We won't need every value change in heading so check range
```

```
      if( ( heading > valHeading + 1125L ) || ( heading < valHeadin
        valHeading = heading;

      // Toggle the GPS pulse to indicate ongoing acquisition
      valGPSPulse = ( valGPSPulse == 0 ) ? 1 : 0;
    }
  }
}

//************************
// Great Circle distance calculation
// Returns the distance between two lat/lon points on this great Ea
// (Note: Assumes sea level, does not take into account altitude. C
//
float DistanceBetween2Points(float  Lat1,float  Lon1,float  Lat2,f
{
  float dLat  =radians( Lat2 - Lat1 );
  float dLon  =radians( Lon2 - Lon1 );

  float a  =sin( dLat / 2.0f ) *sin( dLat / 2.0f ) +
            cos( radians( Lat1 ) ) *cos( radians( Lat2 ) ) *
            sin( dLon / 2.0f ) *sin( dLon / 2.0f );

  float d = 2.0f *atan2( sqrt(  a  ),sqrt( 1.0f - a ) );

  return d * EARTH_RADIUS_METERS * unit_conversion;
}


//************************
// Cell Module GSM/GPRS
// CELL_Update( )
// Assemble messages from the Cell Module
//
void CELL_Update( )
{
  while( CELL_Serial.available( ) )
  {
```

```
incomingChar = CELL_Serial.read(           );// Get the charact

#ifdef DEBUG_CELL
Serial.print( incomingChar );
#endif

// First check to see if we're waiting to be prompted by the ce
// (greater than sign) to enter a text message to be sent, or h
if( nextCellCmnd == Cmnd_SendGPSMessage )
{
  if( incomingChar ==>' || cellReplyTimeout )
  {
                          SendGPSMessage(    );// Send text mes
    cellMsgBuffIndex = 0;
     cellMsgBuff[cellMsgBuffIndex] ='\0';

    CELL_ResetReplyTimeout( );
    nextCellCmnd = Cmnd_None;
    nextCellReply = Reply_SendGPSMessage;
  }
}
// Look for a CR which terminates incoming messages and replies
else if( incomingChar ==\r' )
{
  if(     cellMsgBuffIndex     >     0     )// Just ignore com
  {
    // We have a message...
    ProcessCellReply( );
  }

  // Reset index to beginning for next message
  cellMsgBuffIndex = 0;
   cellMsgBuff[cellMsgBuffIndex] ='\0';
}
// Store incoming characters until a CR is received, which indi
// a message is ready to be processed
else if( incomingChar > 0x1F && incomingChar < 0x7F )
{
```

```
      cellMsgBuff[cellMsgBuffIndex] = incomingChar;
       if( ++cellMsgBuffIndex >= BUFFSIZE )
         cellMsgBuffIndex = 0;
       cellMsgBuff[cellMsgBuffIndex] ='\0';
     }
   }

   // Reset message state if timed out waiting for a reply
   // Send AT command in attempt to resync
   if( cellReplyTimeout )
           {
     cellMsgBuffIndex = 0;
      cellMsgBuff[cellMsgBuffIndex] ='\0';
     CELL_ResetReplyTimeout( );
     nextCellCmnd = Cmnd_OK;
     nextCellReply = Reply_None;
   }
}

//************************
// Cell Module GSM/GPRS
// ProcessCellReply( )
// Process replies from the Cell Module and determine the next Cmnd
// Refer to SM5100B AT Command Set guide
//
void ProcessCellReply( )
{
   // Reset message state if cell decides to start over...
   if( strncmp( cellMsgBuff,"+SIND: 1", 10 ) == 0)
   {
     nextCellReply = Reply_Connect;
     nextCellCmnd = Cmnd_Connect;
     connectRespCnt = 0;
      cellConnected =false;
     CELL_NetConnectTimeoutTask.enable( );
     CELL_NetConnectTimeoutTask.reset( );
      return;
   }
```

```
// Check to see if we've received an unsolicited message...
else if( cellConnected && ( strncmp( cellMsgBuff,"+CMT:", 5 ) ==
{
  // We're receiving a message...
  // Get source phone number, then set up to get the text on the
 strncpy( cellPhoneNumber, &cellMsgBuff[7], 12 );
  cellPhoneNumber[13] ='\0';
 nextCellReply = Reply_MessageText;
 nextCellCmnd = Cmnd_None;
  return;
   }

/*
 The following automates connecting, configuring call forwarding
 Stream from cell at startup should look something like the foll
  +SIND: 1
+SIND: 10,"SM",1,"FD",1,"LD",1,"MC",1,"RC",1,"ME",1
  +SIND: 3
  +SIND: 11
  +SIND: 4
*/

switch( nextCellReply )
{
  case Reply_Connect:
    if( strncmp( cellMsgBuff,"+SIND: 4", 10 ) == 0 )
      connectRespCnt += 1;
    else if( strncmp( cellMsgBuff,"+SIND: 11", 10 ) == 0 )
      connectRespCnt += 2;

    // If we've gotten a 4 and an 11, assume we're good to
    if( connectRespCnt > 2 && !cellConnected )
    {
       cellConnected =true;
     CELL_NetConnectTimeoutTask.disable( );
      nextCellCmnd = Cmnd_CallForwarding;
      nextCellReply = Reply_None;
    }
```

```
  // Reset the timeout interval with each connect response whil
 CELL_NetConnectTimeoutTask.reset( );
  break;

case Reply_CallForwarding:
  if( ( strncmp( cellMsgBuff,"OK", 2 ) == 0 ) || ( strncmp( c
  {
   CELL_ResetReplyTimeout( );
   nextCellCmnd = Cmnd_SMS_Type;
   nextCellReply = Reply_None;
  }
  break;

case Reply_SMS_Type:
  if( ( strncmp( cellMsgBuff,"OK", 2 ) == 0 ) || ( strncmp( c
  {
   CELL_ResetReplyTimeout( );
   nextCellCmnd = Cmnd_SMS_Indication;
   nextCellReply = Reply_None;
  }
  break;

case Reply_SMS_Indication:
  if( ( strncmp( cellMsgBuff,"OK", 2 ) == 0 ) || ( strncmp( c
  {
   CELL_ResetReplyTimeout( );
   nextCellCmnd = Cmnd_DeleteMsgs;
   nextCellReply = Reply_None;
  }
  break;

case Reply_Location:
  if( ( strncmp( cellMsgBuff,"OK", 2 ) == 0 ) || ( strncmp( c
  {
   CELL_ResetReplyTimeout( );
   nextCellCmnd = Cmnd_None;
   nextCellReply = Reply_None;
  }
```

```
  break;

case Reply_SendGPSMessage:
  if( ( strncmp( cellMsgBuff,"OK", 2 ) == 0 ) || ( strncmp( c
  {
   CELL_ResetReplyTimeout( );
   nextCellCmnd = Cmnd_None;
   nextCellReply = Reply_None;
  }
  break;

case Reply_MessageText:
  // We received a text message and captured the sending phone
  // Parse message and reply with GPS location message if reque
  // Cell Module for when to send the message data with a '>'.
  nextCellCmnd = ( strncmp( cellMsgBuff,"Loc?", 4 ) == 0 ) ? Cr
 nextCellReply = Reply_None;
  break;

case Reply_DeleteMsgs:
  if( ( strncmp( cellMsgBuff,"OK", 2 ) == 0 ) || ( strncmp( c
  {
   CELL_ResetReplyTimeout( );
   strcpy( cellPhoneNumber, privCellPhone );
      nextCellCmnd = Cmnd_Location;// Cmnd_Location if send l
   nextCellReply = Reply_None;
  }
  break;

case Reply_OK:
 CELL_ResetReplyTimeout( );
 nextCellCmnd = Cmnd_None;
 nextCellReply = Reply_None;
  break;

default:
 nextCellCmnd = Cmnd_None;
 nextCellReply = Reply_None;
```

```
      break;
  }
}

//************************
// Cell Module GSM/GPRS
// CELL_Actions( )
// Execute Cell Module Commands
// Refer to SM5100B AT Command Set guide
//
void CELL_Actions( )
{
  switch( nextCellCmnd )
  {
    case Cmnd_Connect:
      break;

    case Cmnd_CallForwarding:
       strcpy( cellCmndBuff,"AT+CCFC=0,3,\"" );
      strcat( cellCmndBuff, privCellPhone );
       strcat( cellCmndBuff,"\",129" );
      SendATCommand( cellCmndBuff );
      nextCellReply = Reply_CallForwarding;
      nextCellCmnd = Cmnd_None;
       break;

    case Cmnd_SMS_Type:
       SendATCommand("AT+CMGF=1" );
      nextCellReply = Reply_SMS_Type;
      nextCellCmnd = Cmnd_None;
       break;

    case Cmnd_SMS_Indication:
       SendATCommand("AT+CNMI=3,3" );
      nextCellReply = Reply_SMS_Indication;
      nextCellCmnd = Cmnd_None;
       break;
```

```
  case Cmnd_Location:
     strcpy( cellCmndBuff,"AT+CMGS=\"" );
    strcat( cellCmndBuff, cellPhoneNumber );
     strcat( cellCmndBuff,"\"" );
    SendATCommand( cellCmndBuff );
    nextCellReply = Reply_None;
    nextCellCmnd = Cmnd_SendGPSMessage;
     break;

  case Cmnd_DeleteMsgs:
     SendATCommand("AT+CMGD=1,4" );
    nextCellReply = Reply_DeleteMsgs;
    nextCellCmnd = Cmnd_None;
     break;

  case Cmnd_OK:
     SendATCommand("AT" );
    nextCellReply = Reply_OK;
    nextCellCmnd = Cmnd_None;
     break;

  default:
    break;
 }

#ifdef DEBUG_CELL
 while( Serial.available( ) )
 {
  char incoming_char =Serial.read(      );// Get the character f

  if( incoming_char =='~'                )// If it's a tilde...
                  incoming_char    =    CR;// ...convert to a car
  else if( incoming_char =='^'           )// If it's an up caret
             incoming_char   =     CTRL_Z;// ...convert to ctrl-

  CELL_Serial.print(    incoming_char    );// Send the character
  Serial.print(      incoming_char      );// Echo it back to the
```

```
    if( incoming_char == CR || incoming_char == CTRL_Z )
    {
      Serial.println( );
      break;
    }
  }
    #endif
}

void  SendATCommand(char *cmnd )
{
  delay( 150 );
  CELL_Serial.println( cmnd );

  CELL_StartReplyTimeout( );

  #ifdef DEBUG_CELL
  Serial.print( "\n>> " );
  Serial.println( cmnd );
  #endif
}

void SendGPSMessage( )
{
  // Send text message with GPS location
  delay( 150 );
  CELL_Serial.print( "Loc: "   );
  CELL_Serial.print(  valLatDegrs,DEC );
  CELL_Serial.print( "." );
  CELL_Serial.print(  valLatFrac,DEC );
  CELL_Serial.print( ", " );
  CELL_Serial.print(  valLonDegrs,DEC );
  CELL_Serial.print( "." );
  CELL_Serial.print(  valLonFrac,DEC );
  CELL_Serial.print( " Heading: " );
  CELL_Serial.print( heading / 1000,DEC );
  CELL_Serial.print( " MPH: " );
  CELL_Serial.print(  valMPH,DEC );
```

```
  CELL_Serial.print( " Time: " );
  CELL_Serial.print(  valTimeHH,DEC );
  CELL_Serial.print( ":" );
  CELL_Serial.print(  valTimeMM,DEC );
  CELL_Serial.print( CTRL_Z, BYTE );

  CELL_StartReplyTimeout( );

  #ifdef DEBUG_CELL
   Serial.println( "Sent location..." );
  #endif
}


//*****************************************************
// Cell Module Message Repy Timeout
// Timed Action
// Manages the timer for cell message replies
//
void CELL_ReplyTimeout( )
{
   cellReplyTimeout =true;
  CELL_ReplyTimeoutTask.disable( );

  #ifdef DEBUG_CELL
   Serial.println( "ReplyTimeout" );
  #endif
}

void CELL_StartReplyTimeout( )
{
  // Start reply timeout task...
   cellReplyTimeout =false;
  CELL_ReplyTimeoutTask.enable( );
  CELL_ReplyTimeoutTask.reset( );
}

void CELL_ResetReplyTimeout( )
```

```
{
  cellReplyTimeout =false;
 CELL_ReplyTimeoutTask.disable( );
}

//*****************************************************
// Timeout for Cell Module Network Connection
// Timed Action
// Re-enables serial comm with TouchShield after timeout
//
void CELL_NetConnectTimeout( )
{
  cellNetConnectTimeout =true;
 CELL_NetConnectTimeoutTask.disable( );

 #ifdef DEBUG_CELL
 Serial.println( "NetConnectTimeout" );
 #endif
}

//************************
// BUZZER 2.048Khz
//
void BUZZ_Play( )
{
  for  (int i = 0; i < 1048; i++ )
  {
    // 1 / 2048Hz = 488uS, or 244uS high and 244uS low to create 50
    digitalWrite(  BUZZ_PIN,HIGH );
    delayMicroseconds( 244 );
    digitalWrite(  BUZZ_PIN,LOW );
    delayMicroseconds( 244 );
   }
}


//*****************************************************
// Serial data transfer to the TouchShield display
```

```
// Timed Action
//
void TS_SendDataPacket( )
{
  TS_Serial.print( STX, BYTE );

  TS_WriteInt( valMPH );

  TS_WriteLong( valODO );

  TS_WriteInt( valTripA );
  TS_WriteInt( valTripB );

  TS_WriteInt( valGPSPulse );
  TS_WriteInt( valSpareInt );

  TS_WriteInt( valTimeHH );
  TS_WriteInt( valTimeMM );

  TS_WriteInt( valDateMonth );
  TS_WriteInt( valDateDay );
  TS_WriteInt( valDateYear );

  TS_WriteInt( valTemp );
  TS_WriteInt( valBatt );
  TS_WriteInt( valLean );

  TS_WriteLong( valHeading );
  TS_WriteInt( valLatDegrs );
  TS_WriteLong( valLatFrac );
  TS_WriteInt( valLatNS );

  TS_WriteInt( valLonDegrs );
  TS_WriteLong( valLonFrac );
  TS_WriteInt( valLonEW );

  TS_Serial.print( ETX, BYTE );
}
```

```
void  TS_WriteInt(int value )
{
  // Write high byte, low byte
  TS_Serial.print( ( unsigned char )( value >> 8 ) );
  TS_Serial.print( ( unsigned char  )value  );
}

void  TS_WriteLong(long value )
{
  // Write high word, low word
   TS_WriteInt( (unsigned int )( value >> 16 ) );
   TS_WriteInt( (unsigned int  )value  );
}

// Handy for debugging communications with TouchShield
int whirlyGig = 0;
void TS_WriteWhirlyGig( )
{
  char* whirlyGigs[] = {"|", "/", "-", "\\", "|", "/", "-", "\\" }

  TS_Serial.print( whirlyGigs[ whirlyGig ] );
  whirlyGig = ( ++whirlyGig > 7 ) ? 0 : whirlyGig;
}



//*****************************************************
// Data (button presses) coming from the TouchShield
//
void TS_ReceiveDataPacket( )
{
  while( TS_Serial.available( ) > 0 )
  {
    switch( TS_Serial.read( ) )
    {
      case 'A':             // Reset TripA meter
        f_TripA = 0.0f;
        valTripA = f_TripA;
```

```
        RTC_WriteFloat( TRIPA_ADDR, f_TripA );

        #ifdef DEBUG_RESETS
        Serial.println( "Trip A pressed" );
        #endif
        break;

      case 'B':            // Reset TripB meter
        f_TripB = 0.0f;
        valTripB = f_TripB;
        RTC_WriteFloat( TRIPB_ADDR, f_TripB );

        #ifdef DEBUG_RESETS
        Serial.println( "Trip B pressed" );
        #endif
        break;
    }

//    TS_Serial.flush( );
  }
}

//**************************
// RTC DS1307
// Utility functions

// Convert decimal to BCD
byte  decToBcd(byte val )
{
  return ( ( val / 10 * 16) + ( val % 10 ) );
}

// Convert BCD to decimal
byte  bcdToDec(byte val )
{
  return ( ( val / 16 * 10) + ( val % 16 ) );
}
```

```
// Set the time and date
void  RTC_SetTimeDate(byte         second,// 0-59
                      byte         minute,// 0-59
                      byte           hour,// 1-23
                      byte      dayOfWeek,// 1-7
                      byte     dayOfMonth,// 1-28/29/30/31
                      byte          month,// 1-12
                      byte     year       )// 0-99
{
  Wire.beginTransmission( RTC_I2C_ADDR );
  Wire.send( 0 );
  Wire.send(   decToBcd(   second   )   );// 0 to bit 7 starts the
  Wire.send( decToBcd( minute ) );
  Wire.send(  decToBcd(  hour  ) | 0x40  );// If you want 12 hour am
                                          // bit 6 (also need to ch
  Wire.send( decToBcd( dayOfWeek ) );
  Wire.send( decToBcd( dayOfMonth ) );
  Wire.send( decToBcd( month ) );
  Wire.send( decToBcd( year ) );
  Wire.endTransmission( );
}

//**************************
// RTC DS1307
// RAM register access functions
//

// Reads a 2-byte integer value from the RTC RAM registers
int  RTC_ReadInteger(int valAddr )
{
  // Set the register pointer
  Wire.beginTransmission( RTC_I2C_ADDR );
  Wire.send( valAddr );
  Wire.endTransmission( );

  // Read 2 bytes into int value
  Wire.requestFrom( RTC_I2C_ADDR, 2 );
  int value  =Wire.receive( );
```

```
  value = ( value << 8 ) +Wire.receive( );

  return value;
}

// Reads a 4-byte float value from the RTC RAM registers
float  RTC_ReadFloat(int valAddr )
{
  float value;
  byte  *byteArray = (byte *) &value;

  // Set the register pointer
  Wire.beginTransmission( RTC_I2C_ADDR );
  Wire.send( valAddr );
  Wire.endTransmission( );

  // Read 4 bytes can convert to float value
  Wire.requestFrom( RTC_I2C_ADDR, 4 );
   byteArray[3] =Wire.receive( );
   byteArray[2] =Wire.receive( );
   byteArray[1] =Wire.receive( );
   byteArray[0] =Wire.receive( );

  return value;
}

// Writes a 2-byte integer value to the RTC RAM registers
void  RTC_WriteInteger(int  valAddr,int value )
{
  if( valAddr > 7 && valAddr  < 63 )// Don't let writes go to the
  {
    Wire.beginTransmission( RTC_I2C_ADDR );
    Wire.send( valAddr );

    // Write high byte, low byte
    Wire.send( ( unsigned char )( value >> 8 ) );
    Wire.send( ( unsigned char )value  );
```

```
    Wire.endTransmission( );
    }
}

// Writes a 4-byte float value to the RTC RAM registers
void  RTC_WriteFloat(int  valAddr,float value )
{
  if( valAddr > 7 && valAddr  < 61 )// Don't let writes go to the
  {
    Wire.beginTransmission( RTC_I2C_ADDR );
    Wire.send( valAddr );

    // Write high word (high byte/low byte), low word (high byte/lo
    byte  *byteArray;
    byteArray = (byte *) &value;
    Wire.send( byteArray[3] );
    Wire.send( byteArray[2] );
    Wire.send( byteArray[1] );
    Wire.send( byteArray[0] );

    Wire.endTransmission( );
   }
}
```