```
//Arduino Mega 2560 Marine AutoPilot
//
//Written by Dave Lawson
//Started December 2015 – Work in Progress- Completed??

#include <Wire.h>
#include <math.h>
#include <EEPROM.h>
#include <genieArduino.h>
Genie genie;
#define RESETLINE 4  //Reset for 4D display

//Variable Declarations
int LeftRudder = 53;            // Digital Pin for Left Rudder
int RightRudder = 51;           // Digital Pin for Right Rudder
int RudderRef = 1;              // Analog Pin for Rudder Position Sensor
int NumNMEA = 1;                // number of NMEA sentences to read per cycle.
int NumHeading = 3;             // number of Heading Sensor reads per cycle.
int RR = 7;                     // Rudder Position indicator for Display
int Sel = 10;                   // Pointer for advanced settings Select Button.
double HeadingTrue = 0;         // True Heading (Polar North) -- Degrees (Deviation is applied)
double HeadingMag = 0;          // Magnetic Heading from Heading Sensor. -- Degrees (Deviation is applied)
double HeadingVar = 0;          // Heading Variation (difference between Magnetic and True also known as declination) -- Signed Degrees
double HeadingDev = 0;          // Deviation, Use to trim compass if needed. -- Degrees
double NavBearing = -1;         // Bearing used for Heading Navigation. (Set to -1 when HeadingNav is Suspended)
double StartLat = 0;            // Starting point Latitude for a route. (Radians).
double StartLon = 0;            // Starting point Longitude for a route. (Radians).
double EndLat = 0;              // End Point Latitude for a route. (Radians).
double EndLon = 0;              // End Point Longitude for a route. (Radians).
double ActiveStartLat = 0;      // Next six vars track the lat and lon while a route is active. (Radians)
double ActiveStartLon = 0;
double ActiveEndLat = 0;
double ActiveEndLon = 0;
double CurLat = 0;
double CurLon = 0;
double ER = 6371008.0;          //Earths Radius in meters.
double IB = 0;                  //Initial Bearing from Start Point. (Radians)
double SB = 0;                  //Start Bearing for continuous path calcs.  This changes as we travel along the great circle, this is also known as the Forward Azimuth (Radians)
double NB = 0;                  //New Bearing, Bearing from current position to end point. (Radians)
double CH = 0;                  //Current Heading in Radians (Note: can be true or mag, Set by the Current Active Nav Routine.)
double SDTDC = 0;               //Degrees to turn compensated for distance to course line. (Radians)
double ATD = 0;                 //Distance Traveled along track in meters (calculated in cross track error function)
double QRV = 1.5;               //Quiescent Rudder Value - This is used to calculate a +- range centered around zero degrees rudder angle where Autopilot Steering remains inactive.
double RG = 1;                  //Rudder Gain - value between 0 and 3.  3 being full gain.  This value determines the rate of steering correction.  Basically, degrees of correction x rudder gain = degrees of rudder
applied.
double RM = 35;                 //Rudder Maximum Turning Degrees measured in one direction from zero.  Applied to both directions.  The max allowed is 45 degrees (hard coded).
double RT = 0;                  //Rudder Trim.  +- value equal to the number of degrees to trim the rudder to center.  Use for minor correction, otherwise adjust rudder position sensor linkage.
double RVmax = 1.94;            //voltage at starboard max.  Motorolla=4.45 ComNav = 2.95
double RVmin = 1.16;            //voltage at port max.       Motorolla=.5   ComNav = 1.72
double NMPH = 0;                //Nautical Miles Per Hour.
long NavMillis = 0;             //millis Clock value for elapsed time calculations.
double Radius = .1;             //Radius in nautical miles for Nav Functions 180, 360.
int NavFunction = 0;            //Nav Function Selection.  0 = none. 1=p180, 2=s180, 3=p360, 4 =s360.
double NavFBearing = -1;        //Nav Function Bearing.
double NavDT = 0;               //Tracks Nav Function Degrees Turned.  Used to end a 180.

String const Off = "Off";
String const Port = "Port";
```

```
String const Starb = "Starboard";
String NMEA_Sentence[10];      //Array that holds the NMEA data read from the NMEA RS232 Serial Port.
String NMEA_Heading[10];       //Array that holds the NMEA data read from the NMEA RS422 Heading Sensor.
char inChar = ' ';             //single character used when reading serial data.
String TempString;



boolean MorT = false;                //HeadingNav Compass Data: False for Magnetic, True for True.
boolean HeadingSet = false;          //Heading Nav Set button flag
boolean RouteAuto = false;           //Route Nav Auto button flag
boolean CourseActive = false;        //Flag set True when autopilot is active.
boolean GPRMBflag = false;           //GPRMB detected flag.
boolean GPRMCflag = false;           //Set to true the first occurrence after GPRMB is detected.
boolean GPGLLflag = false;           //GPGLL detected flag.  GPGLL provides our current position so we monitor to make sure we are getting updates.
boolean DVflag = false;              //Declination detected flag.  Extracted from GPHDG.
int NoGPRMBCounter = 0;              //Counts how many reads are performed without receiving a GPRMB.
int NoGPRMBLimit = 8;                //Number of reads with no GPRMB that sets CourseActive to false.
int NoGPGLLCounter = 0;              //Counts how many reads are performed without receiving a GPGLL.
int NoGPGLLLimit = 5;                //Number of reads with no GPGll that sets CourseActive to false.
long RefDir[9] = {0,0,0,0,0,0,0,0,0};     //Used to control display refresh of NSEW coordinate descriptions to control screen flicker.
int EE = 0;                          //Var used to read and write to EEPROM.


void setup()
{
  if(EEPROM.get(0x00,EE) == 1)                    //EEPROM location 0x00 = 1 then saved settings exist so load them to variables.
   {
        EEPROM.get(0x05,NumNMEA);                  //Number of NMEA reads per cycle.
        EEPROM.get(0x0A,NumHeading);               //Number of Heading reads per cycle.
        EEPROM.get(0x0F,NoGPRMBLimit);             //Fault limit for cycles with no GPRMB.
        EEPROM.get(0x14,NoGPGLLLimit);             //Fault limit for cycles with no GPGLL.
        EEPROM.get(0x19,RVmin);                    //Rudder Voltage Minimum.
        EEPROM.get(0x23,RVmax);                    //Rudder Voltage Maximum.
        EEPROM.get(0x2D,RM);                       //Rudder Maximum Degrees Deflection.
        EEPROM.get(0x37,RT);                       //Rudder Trim.
        EEPROM.get(0x41,QRV);                      //Quiescent Rudder Value.
        EEPROM.get(0x4B,RG);                       //Rudder Gain.
        EEPROM.get(0x55,Radius);                   //Radius for Nav Functions.
   }


  analogReference(EXTERNAL);                      // Use the external reference for more stability on the rudder position indicator.  Using 3.3 voltage input.
  Serial3.begin(200000);                          // Used for Display Communication
  genie.Begin(Serial3);                           // Use Serial3 for talking to the Genie Library, and to the 4D Systems display
  genie.AttachEventHandler(myGenieEventHandler);  // Attach the user function Event Handler for processing events

  pinMode(RESETLINE, OUTPUT);                     // Set D4 on Arduino to Output (4D Arduino Adaptor V2 - Display Reset)
  digitalWrite(RESETLINE, 1);                     // Reset the Display via D4
  delay(100);
  digitalWrite(RESETLINE, 0);                     // unReset the Display via D4

  delay (3500);                                   // let the display start up after the reset (This is important)
  genie.WriteContrast(15);                        // Initialize the brightness 0 - 15
  genie.WriteObject(GENIE_OBJ_TRACKBAR, 0x00, 15);      // initialize contrast trackbar to 15
  genie.WriteObject(GENIE_OBJ_TRACKBAR, 0x01, 10);      // initialize gain trackbar to 10 (this is RG * 10)
  genie.WriteObject(GENIE_OBJ_TRACKBAR, 0x02, RM);      // initialize display Rudder Max Slider
  genie.WriteObject(GENIE_OBJ_TRACKBAR, 0x03, RT+5);    // initialize display Rudder Trim Slider
```

```
  genie.WriteObject(GENIE_OBJ_TRACKBAR, 0x04, QRV * 10);               // initialize display QRV Slider
  genie.WriteObject(GENIE_OBJ_TRACKBAR, 0x05, Radius * 10.0);            // initialize display Radius Slider
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x00, RG * 10);           // initialize Rudder Max Custom Digits
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x01, RM);                // initialize Rudder Max Custom Digits
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x02, RT);                // initialize Rudder Trim Custom Digits
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x03, QRV * 10);          // initialize QRV Custom Digits
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x04, NumNMEA);           // initialize NumNMEA Custom Digits
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x05, NumHeading);        // initialize NumHeading Custom Digits
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x06, NoGPRMBLimit);      // initialize NoGPRMBLimit Custom Digits
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x07, NoGPGLLLimit);      // initialize NoGPGLLLimit Custom Digits
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x06, Radius * 10.0);        // initialize Radius Custom Digits * 10.
  SendRudderMaxMintoDisplay();                                 // initialize Rudder Max Min voltage Custom Digits
  genie.WriteObject(GENIE_OBJ_WINBUTTON,0x01,1);                       // initialize suspend button for route nav.
  genie.WriteObject(GENIE_OBJ_WINBUTTON,0x09,1);                       // initialize suspend button for heading nav.
  genie.WriteObject(GENIE_OBJ_WINBUTTON,0x1B,1);                       // initialize suspend button for Nav Functions.
  genie.WriteObject(GENIE_OBJ_WINBUTTON,0x0D,1);                       // initialize HeadingNav Mag/True buttons to Mag.


  pinMode(LeftRudder, OUTPUT);                                 //Set up the Digital Pinouts (Port Rudder)
  pinMode(RightRudder, OUTPUT);                                //Set up the Digital Pinouts (Starboard Rudder)

  Serial1.begin(4800);                                  // initialize serial for Furuno NMEA, RS232
  Serial2.begin(4800);                                  // initialize serial for Heading Sensor RS422
  Serial.begin(115200);                                 // initialize serial for USB
}


void loop()
{

//digitalWrite(LeftRudder,HIGH);                                 // Debugging
//digitalWrite(RightRudder,HIGH);                                // Debugging

  genie.DoEvents();                                            // This calls the library each loop to process the queued responses from the display
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x03, HeadingTrue);                    // Send Heading to RouteNav display
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x02, RtoD360(SB));                    // Send StartBearing to RouteNav Display
  if(MorT == false){                                          // We have two options for HeadingNav, Display in Magnetic or True. False = Magnetic else True.
   genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x01, HeadingMag);                    // Send Mag Heading to HeadingNav Display
   if(NavBearing == -1){                                       // if NavBearing is -1 then HeadingNav is suspended so just show the current heading as Bearing.
   genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x00, HeadingMag);                    // otherwise the Bearing will be set by NavBearing as a constant.
   }
  }
  else
  {
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x01, HeadingTrue);                    // Send True Heading to HeadingNav Display
   if(NavBearing == -1){                                       // if NavBearing is -1 then HeadingNav is suspended so just show the current heading as Bearing.
   genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x00, HeadingTrue);                   // otherwise the Bearing will be set by NavBearing as a constant.
   }
  }
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x05, HeadingMag);                     // Send Mag Heading to Nav Functions Display
  if(NavFunction == 0){
   genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x04, 0);
  }
  else
  {
   genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x04, NavFBearing);
  }
```

```
//TestData();                        //Enable for debugging when not connected to NMEA and Heading Sensors.

//SendNMEA_DataArray();                //Sends incoming NMEA to Serial0 for monitoring on laptop, used for debugging.
//SendHeadingSensor_DataArray();        //Sends incoming HeadingData to Serial0 for monitoring on laptop, used for debugging.

ClearNMEA_DataArray();               //Clears NMEA array, we start fresh every cycle.
GetNMEA();                           //Get incoming NMEA data.
ParseNMEA();                         //Parse out the pieces we need from NMEA.
DisplayRudderAngle();                //Send the rudder angle to the display.
ClearHeadingSensor_DataArray();      //Clears HeadingSensor array, we start fresh every cycle.
GetHeadingSensor();                  //Get incoming Heading data.
ParseHeadingSensor();                //Parse out the pieces we need from the Heading Data.
DisplayRudderAngle();                //Send the rudder angle to the display.
NavController();                     //Call the Main Navagation Controller.
DisplayRudderAngle();                //Send the rudder angle to the display.


//CourseNavigation();                // Enable these for debugging only.
//Counters();
//ShowHeadingData();
//ShowRadianData();
//ShowCalculationResults(StartLat,StartLon,EndLat,EndLon,CurLat,CurLon);
//TestDistance();
//TestCourseAdjust();
//SteeringRoutine();

}

//****************** SUBS and FUNCTIONS below this line *********************************

void ClearNMEA_DataArray(){                          //***Clears the NMEA Data Array
  for(int x=0;x < NumNMEA;x++){
    NMEA_Sentence[x] = "";
  }
}

void ClearHeadingSensor_DataArray(){                 //***Clears the Heading Sensor Array
  for(int x=0;x < NumHeading;x++){
    NMEA_Heading[x] = "";
  }
}

void SendNMEA_DataArray(){                           //***Sends Contents of NMEA data Array to Serial Monitor - Used for Debugging
  for(int x=0;x < NumNMEA;x++){


    if(DataCheck(NMEA_Sentence[x])== 1){
     Serial.print("GoodNMEA - ");
    }
    else
    {
     Serial.print("BadNMEA  - ");
    }
     Serial.println(NMEA_Sentence[x]);
  }


}
```

```
void SendHeadingSensor_DataArray(){                 //***Sends Contents of Heading data Array to Serial Monitor - Used for Debugging
  for(int x=0;x < NumHeading;x++){
    //Serial.println("test");

    if(DataCheck(NMEA_Heading[x])== 1){
     Serial.print("GoodHead - ");
    }
    else
    {
     Serial.print("BadHead  - ");
    }
     Serial.println(NMEA_Heading[x]);
  }


}


boolean DataCheck(String NMEAstring){                        //*****************CHECKSUM TEST**************

  int checksum = 0;                                          //Initialize the test var for our calculated checksum
  int L = NMEAstring.length();                               //Get the length of the NMEA sentence to test.
  for(int i = 1; i < L - 3; i++){                            //Loop through the characters in the string ignoring the first ($) and last three characters.
  checksum = checksum ^ int(NMEAstring.charAt(i));           //XOR the values to the checksum var.
  }                                                          //We finished the loop so now checksum holds our checksum value in HEX.

  String NMEAchecksum_sent = NMEAstring.substring(L-2,L);    //Extract the checksum sent with the Sentence.  It's the last two characters.

  NMEAstring = String(checksum, HEX);                        //Since we have all the data we need now, re-use the NMEAstring to hold our calculated checksum as a string.
  NMEAstring.toUpperCase();                                  //When we do our comparison it will be case sensitive so force Hex A-F to upper case.
  if(NMEAstring.length() ==1){                               //The value sent with the sentence is always two digits with a leading zero for single digit values.
    NMEAstring = "0" + NMEAstring;                           //Our Caculated value may only be a single Hex digit 0-F so we need to test and add a leading zero for single digit values.
  }
  if(NMEAstring == NMEAchecksum_sent){                       //Now we make the comparison
    return true;                                             //If our value matches the value sent with the string we return TRUE otherwise FALSE.
  }
  else
  {
    return false;
  }

 }
void ParseHeadingSensor(){                                   //***This routine parses the Heading data from the Heading Sensor NMEA sentences.
  String DisplayString;                                      // Temp Var used to build display for incoming heading data.
  for(int x=0;x < NumHeading;x++){                           // Next 4 lines send Heading Sentence Data to Display (Adv Settings)
    if(x < 3){
      DisplayString = DisplayString + NMEA_Heading[x] + "\n";
    }
    if(NMEA_Heading[x].substring(1,6)== "HCHDG" && DataCheck(NMEA_Heading[x])== true){        //Detect Magnetic Heading Data from $HCHDG.
      TempString = "";
      for(int y=7;NMEA_Heading[x].substring(y,y + 1)!= "," && NMEA_Heading[x].substring(y,y + 1)!="";y++){    //Start at first comma and loop until the next comma.
        TempString += NMEA_Heading[x].substring(y,y + 1);    //Build a string of characters between the two commas, this is our Magnetic Heading.
      }

      HeadingMag = D360(TempString.toFloat()+HeadingDev);    //Convert from String to Numeric, apply Heading Dev and store in global variable HeadingMag for use in
other routines.
```

```
      HeadingTrue = D360(HeadingMag+HeadingVar);                                        //Calculate the True Heading by applying the variation.  Note:  D360 is function to rationalize to 360 compass
degrees.
    }
  }
  genie.WriteStr(0x00,DisplayString);
}

void ParseNMEA(){                                                                   // ***This routine parses the data from the NMEA sentences into assigned variables.

  Serial.print("GPMRB=");                                                           // Next 4 lines to Serial Monitor for Debugging.
  Serial.println(NoGPRMBCounter);
  Serial.print("GLL=");
  Serial.println(NoGPGLLCounter);
  NoGPRMBCounter++;                                                                 // counts read cycles, resets when GPRMB sentence is received.
  if(NoGPRMBCounter >= NoGPRMBLimit){                                               // If the NoGPRMB count exceeds the limit value then we set CourseActive to false.
    NoGPRMBCounter = NoGPRMBLimit;                                                  // We don't want to overflow so if we are at the limit then set the counter to the limit.
    GPRMBflag = false;
    GPRMCflag = false;
  }
  NoGPGLLCounter++;                                                                 // Same as above but now we checking for GPGLL data used to track current position.
  if(NoGPGLLCounter >= NoGPGLLLimit){
    NoGPGLLCounter = NoGPGLLLimit;
    GPGLLflag = false;
  }

  for(int x=0;x < NumNMEA;x++){                                                     // Here we start the main loop to parse the NMEA data array.

    if(NMEA_Sentence[x].substring(1,6)== "GPHDT" && DataCheck(NMEA_Sentence[x])== true){          // *** Detect True Heading Data from $GPHDT.
      TempString = "";

      for(int y=7;NMEA_Sentence[x].substring(y,y + 1)!= "," && NMEA_Sentence[x].substring(y,y + 1)!="";y++){
        TempString += NMEA_Sentence[x].substring(y,y + 1);
      }
      //HeadingTrue = TempString.toFloat();  // use this if not using heading sensor

    }
    if(NMEA_Sentence[x].substring(1,6)== "GPHDG" && DataCheck(NMEA_Sentence[x])== true){          // *** Store the Furuno Declination (Heading Variation) $GPHDG.
      TempString = "";                                                              // Initialize the TempString.
      int c = 0;                                                                    // comma counter initialize to 0
      int p = 0;                                                                    // p keeps track of the current character position within the NMEA sentence.
      for(int z=6;NMEA_Sentence[x].substring(z,z + 1)!="" && c < 4;z++){            // Find the 4th comma.  We know our variation data starts here.
        if(NMEA_Sentence[x].substring(z,z + 1)== ","){                              // test for comma.
          c++;                                                                      // found comma so increment the comma counter. Once we find the position of the sixth comma we no longer need the c counter.
          p = z;                                                                    // set P to the character position.
        }                                                                           // if not found we keep looping until we find the sixth one.
      }                                                                             // Found it so loop complete and the comma is located at character position p.
      p++;                                                                          // increment p so we skip over comma. we want the data immediately after the comma.
      for(int y=p;NMEA_Sentence[x].substring(y,y + 1)!= "," && NMEA_Sentence[x].substring(y,y + 1)!="";y++){   // Assemble the variation data located at p and continues until we run up to the next comma.
        TempString += NMEA_Sentence[x].substring(y,y + 1);                          // keep adding characters until the next comma.
        p++;                                                                        // increment p to keep track of the character position.
      }                                                                             // loop completed so next comma was found. Our TempString should now hold the variation data.
      p++;                                                                          // increment p to skip over comma. Our string is complete but we need to skip the next comma because our direction indicator
follows it for one character.
      HeadingVar = TempString.toFloat();                                            // Store the Declination
      if(NMEA_Sentence[x].substring(p,p + 1)== "W"){
        HeadingVar *= -1.0;                                                         // Check for Easterly or Westerly Declination, Since west is subtracted we sign as negative.
      }
      DVflag = true;
```

```
    }
    if(NMEA_Sentence[x].substring(1,6)== "GPRMB" && DataCheck(NMEA_Sentence[x])== true){                    // *** Store the Furuno End Position Coordinates from $GPRMB.
      TempString = "";                                                                          // Initialize the TempString, this will collect the Furuno coordinate.


      int c = 0;                                                                               // comma counter initialize to 0
      int p = 0;                                                                               // p keeps track of the current character position within the NMEA sentence.
      for(int z=6;NMEA_Sentence[x].substring(z,z + 1)!="" && c < 6;z++){                                  // Find the 6th comma.  We know our Lat info starts here.
        if(NMEA_Sentence[x].substring(z,z + 1)== ","){                                                 // test for comma.
          c++;                                                                                 // found comma so increment the comma counter. Once we find the position of the sixth comma we no longer need the c counter.
          p = z;                                                                               // set P to the character position.
        }                                                                                      // keep looping until we find the sixth one.
      }                                                                                        // Found it so loop complete and the comma is located at character position p.
      p++;                                                                                     // increment p so we skip over comma. we want the data immediately after the comma.
      for(int y=p;NMEA_Sentence[x].substring(y,y + 1)!= "," && NMEA_Sentence[x].substring(y,y + 1)!="";y++){   // Assemble the End latitude located at p and continues until we run up to the next comma.
        TempString += NMEA_Sentence[x].substring(y,y + 1);                                             // keep adding characters until the next comma.
        p++;                                                                                   // increment p to keep track of the character position.
      }                                                                                        // loop completed so next comma was found. Our TempString should now hold the furuno coordinate.
      p++;                                                                                     // increment p to skip over comma. Our string is complete but we need to skip the next comma because our direction indicator
follows it for one character.
      EndLat = FurunoLatLonToRadian(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1));                // Store the End Latitude in Radians by calling the coversion function.  Need to supply the
furuno string
      //genie.WriteStr(0x03, FurunoLatLonToDisplayFormat(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1)));  //Send to Display
      FurunoLatLonToDisplayFormatDP(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1),0x03);
      TempString = "";                                                                          // and the next character following our last comma -- This holds the direction N S E or W.
      p++;
      p++; //Increment to skip over comma.                                                      // Setup for longitude.  Initialize Tempstring, increment p to next comma and again to skip over it.

      for(int y=p;NMEA_Sentence[x].substring(y,y + 1)!= "," && NMEA_Sentence[x].substring(y,y + 1)!="";y++){   // Assemble the Furuno End longitude. Same basic process as above.
      TempString += NMEA_Sentence[x].substring(y,y + 1);
      p++;
      }
      p++;  //increment to skip over comma.


      EndLon = FurunoLatLonToRadian(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1));                // Store the End Lon in Radians.
      //genie.WriteStr(0x04, FurunoLatLonToDisplayFormat(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1)));  //Send to Display
      FurunoLatLonToDisplayFormatDP(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1),0x04);
      NoGPRMBCounter = 0;                                                                       // GPRMB parsed so reset NoGPRMB counter to zero.
      GPRMBflag = true;                                                                         // We received a GPRMB so set the flag to True.

      if(EndLat != ActiveEndLat || EndLon != ActiveEndLon){                                              // Check to see if the destination has changed.
        ActiveEndLat = EndLat;                                                                  // If it has update the active EndLat
        ActiveEndLon = EndLon;                                                                  // also update the active EndLon
        GPRMCflag = false;                                                                      // Set the GPRMCflag to false so the start point gets updated.
      }
    }
    if(NMEA_Sentence[x].substring(1,6)== "GPRMC" && DataCheck(NMEA_Sentence[x])== true && GPRMBflag == true && GPRMCflag == false){                    // *** Store the Furuno Start Position
Coordinates from $GPRMC.
      TempString = "";


      int c = 0;                                                                               //comma counter initialize to 0
      int p = 0;
      for(int z=6;NMEA_Sentence[x].substring(z,z + 1)!="" && c < 3;z++){                                  //Find the 3rd comma.
        if(NMEA_Sentence[x].substring(z,z + 1)== ","){
          c++;
          p = z;
```

```
    }
  }
  p++; //increment to skip over comma.
  for(int y=p;NMEA_Sentence[x].substring(y,y + 1)!= "," && NMEA_Sentence[x].substring(y,y + 1)!="";y++){  // Assemble the Start latitude.
    TempString += NMEA_Sentence[x].substring(y,y + 1);
    p++;
  }
  p++; //increment to skip over comma.
  StartLat = FurunoLatLonToRadian(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1)); // Store the Start Latitude in Radians
  //genie.WriteStr(0x01, FurunoLatLonToDisplayFormat(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1)));  //Send to Display
  FurunoLatLonToDisplayFormatDP(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1),0x01);
  TempString = "";                                                                // re-initialize var for Start lon build.
  p++;
  p++; //increment to skip over comma.
  for(int y=p;NMEA_Sentence[x].substring(y,y + 1)!= "," && NMEA_Sentence[x].substring(y,y + 1)!="";y++){  // Assemble the Furuno Start longitude.
  TempString += NMEA_Sentence[x].substring(y,y + 1);
  p++;
  }
  p++; //increment to skip over comma.
  StartLon = FurunoLatLonToRadian(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1));            // Store the Start Longitude in Radians.
  //genie.WriteStr(0x02, FurunoLatLonToDisplayFormat(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1)));  //Send to Display
  FurunoLatLonToDisplayFormatDP(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1),0x02);

  GPRMCflag = true;
  ActiveStartLat = StartLat;
  ActiveStartLon = StartLon;
}



if(NMEA_Sentence[x].substring(1,6)== "GPGLL" && DataCheck(NMEA_Sentence[x])== true){                    //Store the Furuno Current Position Coordinates from $GPGLL.
  TempString = "";


  int c = 0;                                                                 //comma counter initialize to 0
  int p = 0;
  for(int z=6;NMEA_Sentence[x].substring(z,z + 1)!="" && c < 1;z++){                                   //Find the 1st comma.
    if(NMEA_Sentence[x].substring(z,z + 1)== ","){
      c++;
      p = z;
    }
  }
  p++;  //increment to skip over comma.
  for(int y=p;NMEA_Sentence[x].substring(y,y + 1)!= "," && NMEA_Sentence[x].substring(y,y + 1)!="";y++){  // Assemble the Cur latitude.
    TempString += NMEA_Sentence[x].substring(y,y + 1);
    p++;
  }
  p++; //increment to skip over comma.
  CurLat = FurunoLatLonToRadian(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1));              // Store the Current Latitude in Radians.
  //genie.WriteStr(0x05, FurunoLatLonToDisplayFormat(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1)));  //Send to Display
  //genie.WriteStr(0x07, FurunoLatLonToDisplayFormat(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1)));  //Send to Display
  FurunoLatLonToDisplayFormatDP(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1),0x05);
  FurunoLatLonToDisplayFormatDP(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1),0x07);
  TempString = "";                                                               // re-initialize var for Cur lon build.
  p++;
  p++; //increment to skip over comma.
  for(int y=p;NMEA_Sentence[x].substring(y,y + 1)!= "," && NMEA_Sentence[x].substring(y,y + 1)!="";y++){  // Assemble the Furuno Cur longitude.
    TempString += NMEA_Sentence[x].substring(y,y + 1);
```

```
        p++;
      }
      p++; //increment to skip over comma.
      CurLon = FurunoLatLonToRadian(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1));        // Store the Current Longitude in Radians.
      //genie.WriteStr(0x06, FurunoLatLonToDisplayFormat(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1)));  //Send to Display
      //genie.WriteStr(0x08, FurunoLatLonToDisplayFormat(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1)));  //Send to Display
      FurunoLatLonToDisplayFormatDP(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1),0x06);
      FurunoLatLonToDisplayFormatDP(TempString.toFloat(),NMEA_Sentence[x].substring(p,p + 1),0x08);
      GPGLLflag = true;                                                      // Success so set flag.
      NoGPGLLCounter = 0;
    }

    if(NMEA_Sentence[x].substring(1,6)== "GPVTG" && DataCheck(NMEA_Sentence[x])== true){            // *** Store the Current Velocity in Knots from $GPVTG.
      TempString = "";                                                       // Initialize the TempString.
      int c = 0;                                                             // comma counter initialize to 0
      int p = 0;                                                             // p keeps track of the current character position within the NMEA sentence.
      for(int z=6;NMEA_Sentence[x].substring(z,z + 1)!="" && c < 5;z++){                     // Find the 5th comma.  We know our velocity data starts here.
        if(NMEA_Sentence[x].substring(z,z + 1)== ","){                       // test for comma.
          c++;                                                               // found comma so increment the comma counter. Once we find the position of the sixth comma we no longer need the c counter.
          p = z;                                                             // set P to the character position.
        }                                                                    // if not found we keep looping until we find the sixth one.
      }                                                                      // Found it so loop complete and the comma is located at character position p.
      p++;                                                                   // increment p so we skip over comma. we want the data immediately after the comma.
      for(int y=p;NMEA_Sentence[x].substring(y,y + 1)!= "," && NMEA_Sentence[x].substring(y,y + 1)!="";y++){   // Assemble the velocity data located at p and continues until we run up to the next comma.
        TempString += NMEA_Sentence[x].substring(y,y + 1);                   // keep adding characters until the next comma.
        p++;                                                                 // increment p to keep track of the character position.
      }                                                                      // loop completed so next comma was found. Our TempString should now hold the velocity data.

      NMPH = TempString.toFloat();                                           // Store the Nautical Miles per Hour as NMPH.
      Serial.print("NMPH = ");
      Serial.println(NMPH,2);
    }                                                                        // All Done with GPVTG.

  }                                                                          // Get Next NMEA Sentence to process if available.

}                                                                            // All Done with NMEA, Exit Function.

int readline(int readch, char *buffer, int len) // This routine reads the serial buffer.
{
  static int pos = 0;
  int rpos;

  if (readch > 0) {                    //Check status of the Character just read.
    switch (readch) {
      case '\n': // Ignore new-lines       //End of line we ignore.  We'll wait for the return instead.
        break;
      case '\r': // Return on CR           //
        rpos = pos;
        pos = 0;  // Reset position index ready for next time
        return rpos;
      default:
        if (pos < len-1) {             //Check to see if we've not reached the end of the buffer.
          buffer[pos++] = readch;          //yes--So store the character at the current position then increment the position counter.
          buffer[pos] = 0;             //The position was incremented so store zero at the location to null terminate for good practice.
        }
    }
  }
  // No end of line has been found, so return -1.
```

```
  return -1;
}

void GetNMEA() // This routine calls the serial buffer to retrieve the Chart Plotter NMEA Sentences.
{
  if(Serial1.available()>40)
  {
    int x = 0;
    static char buffer[80];
    while(x < NumNMEA){                                   //This determines how many Sentences to get.
      if (readline(Serial1.read(), buffer, 80) > 0) {   //Pole the buffer to see if there is a complete sentence available
      delay(2);
      NMEA_Sentence[x] = buffer;                        //If available Store the Sentance as a string and increment to wait for next sentence.
      if(DataCheck(NMEA_Sentence[x])== 1){
        Serial.print("GoodNMEA - ");
      }
      else
      {
        Serial.print("BadNMEA  - ");
      }
      Serial.println(NMEA_Sentence[x]);
      x++;
      }
    }
  }
}

void GetHeadingSensor()                                     // This routine calls the serial buffer to retrieve the Heading Sensor NMEA Sentances.
{
 if(Serial2.available()>40)
 {
  int x = 0;
  static char buffer[80];
    while(x < NumHeading){                                //This determines how many Sentences to get.
      if (readline(Serial2.read(), buffer, 80) > 0) {      //Pole the buffer to see if there is a complete sentence available
      delay(2);
      NMEA_Heading[x] = buffer;                             //If available Store the Sentance as a string and increment to wait for next sentence.
      if(DataCheck(NMEA_Heading[x])== 1){
        Serial.print("GoodHead - ");
      }
      else
      {
         Serial.print("BadHead  - ");
      }
      Serial.println(NMEA_Heading[x]);
      x++;
      }
    }
  }
}

double RtoD(double Radians){                         //Function convert Radians to Degrees.
  return Radians * 180.0 / PI;
}
double RtoD360(double Radians){                     //Function convert Radians to Degrees rationalized to compass 360.
  return fmod(Radians * 180.0 / PI + 360.0,360.0);
}
```

```
double DtoR(double Degrees){                        //Function convert Degrees to Radians.
  return Degrees * PI / 180.0;
}

double D360(double Degrees){                    //Function to rationalize to compass 360.
  return fmod(Degrees + 360.0,360.0);
}
double R2PI(double Radians){                    //Function to rationalize signed radians to 0-2PI.
  return fmod(Radians + 2.0*PI,PI);
}
String FurunoLatLonToDisplayFormat(double LatLon, String NSEW)            //***This was used by the parsing routine to send Latitude and Longitude coords to the Display as a Text String.
{                                                                          //LatLon is unsigned and NSEW is a single character N,S,E or W.
  double z = LatLon, fractional, integer;                                 //variable declaration
  z = LatLon / 100.0;                                                      //divide by 100 to move the decimal left between the Furuno Deg and Min.
  fractional = modf(z, &integer);                                          //Split the number at the decimal point into two vars holding Deg and Min.

  return NSEW + " " + String(integer,0) + char(176)+ " " + String(fractional * 100,4) + "'";  //here's our string.
}                                                                          //end function.

void FurunoLatLonToDisplayFormatDP(double LatLon, String NSEW, byte DP)  //***This is used by the parsing routine to send Latitude and Longitude coords to the Display as Integer values d,m,f.
{                                                                          //LatLon is unsigned and NSEW is a single character N,S,E or W.  DP is the NSEW String Object Hex Value.
  double z = LatLon, fractional, integer, d, m, f;                         //variable declaration
  z = LatLon / 100.0;                                                      //divide by 100 to move the decimal left between the Furuno Deg and Min.
  fractional = modf(z, &integer);                                          //Split the number at the decimal point into two vars holding Deg and Min.
  d = integer;                                                             //Store the degrees.
  z = fractional * 100.0;                                                  //Minutes * 100 to get the decimal place back where it belongs.
  fractional = modf(z, &integer);                                          //Split the minutes at the decimal.
  m = integer;                                                             //Store the whole minutes.
  f = fractional * 10000.0;                                                //Store the fractional minutes but multiply * 10000 remove the decimal for display parsing.


  if(millis() > RefDir[DP] + 2000){                                        //To control screen flicker we only update NSEW every two seconds so check to see if it's time for a refresh.
    RefDir[DP] = millis();                                                 //It's time so update the Refresh pointer with the current millis.
    genie.WriteStr(DP,NSEW);                                               //Write the NSEW to the display for the Object given by the DP Hex Value.  These are hard coded in the function call in the parsing routine.
  }
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 2*DP + DP + 0x0D, d);       //Now update the display degrees, minutes and fractions for the Object given by the DP Hex Value.
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 2*DP + DP + 0x0E, m);       //There is an order to the Object Hex Values that can be mathmatically addressed by relation to the DP value.
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 2*DP + DP + 0x0F, f);
}                                                                          //end function.

void NSEWtoDisplay()                                                       //This routine will update only the Start Lat and Lon Letter Coordinate for RouteNavigation.
{
  String NSEW;
  if(millis() > RefDir[1] + 2000){                                         //To control screen flicker we only update NSEW every two seconds so check to see if it's time for a refresh.
    RefDir[1] = millis();                                                  //It's time so update the Refresh pointer with the current millis.
    RefDir[2] = millis();                                                  //Write the NSEW to the display for the Object given by the DP Hex Value.  These are hard coded in the function call in the parsing routine.
  NSEW = NSEWfromSignedRadian(StartLat,"Lat");
  genie.WriteStr(0x01,NSEW);
  NSEW = NSEWfromSignedRadian(StartLon,"Lon");
  genie.WriteStr(0x02,NSEW);
  }

}


String NSEWfromSignedRadian(double SignedLatLon,String LATorLON)        //This function will return the letter coordinate for a signed radian Lat or Lon.
{
  String NSEW;
```

```
  if(LATorLON == "Lat")                                   //Check to see if we have a Lat coordinate
  {
    if(SignedLatLon < 0)                                  //If Lat then check for sign, Value less than zero is negative so it's South.
    {
      NSEW = "S";
    }
    else
    {
      NSEW = "N";                                         //If Lat and it's not S we default to N.
    }
  }
  else                                                    //If it's not Lat then we default to Lon and check sign to determine E or W.
  {
    if(SignedLatLon < 0)
    {
      NSEW = "W";                                         //Lon Less than zero is negative so it's W.
    }
    else
    {
      NSEW = "E";                                         //If Lon is not W we default to E.
    }
  }                                                       //All done, we have a NSEW now.

  return NSEW;                                            //All Done here is our coordinate formatted for the display.  Should look something like N 117°34.1234'

}

String FurunoLatLonToDisplayFormatSigned(double SignedLatLon, String LATorLON)  //This is used to send signed lat or lon to the display.  This operates outside of the parsing routine.
{                                                         //Signed LatLon is the signed radian Lat or Lon and LATorLON is a three character string indicating what
                                                          //coordinate value is being sent, either Lat or Lon.  We need this to determine N or S, E or W from the sign.

  String NSEW;
  double LatLon;                                          //variable declaration
  double z = LatLon, fractional, integer;                 //variable declaration

  if(LATorLON == "Lat")                                   //Check to see if we have a Lat coordinate
  {
    if(SignedLatLon < 0)                                  //If Lat then check for sign, Value less than zero is negative so it's South.
    {
      NSEW = "S";
    }
    else
    {
      NSEW = "N";                                         //If Lat and it's not S we default to N.
    }
  }
  else                                                    //If it's not Lat then we default to Lon and check sign to determine E or W.
  {
    if(SignedLatLon < 0)
    {
      NSEW = "W";                                         //Lon Less than zero is negative so it's W.
    }
    else
    {
      NSEW = "E";                                         //If Lon is not W we default to E.
    }
  }                                                       //All done, we have a NSEW now.
```

```
  LatLon = abs(SignedLatLon);                              //We need an unsigned value for the display so take the absolute value of the signed radian value.

  z = RtoD(LatLon);                                        //Convert Radians to Degrees.
  fractional = modf(z, &integer);                          //Split the number at the decimal point into two vars holding Deg and Decimal Deg.
  fractional = fractional * 60;                            //Convert the fractional part to minutes so multipy by 60.

  return NSEW + " " + String(integer,0) + char(176)+ " " + String(fractional,4) + "'";   //All Done here is our coordinate formatted for the display.  Should look something like N 117°34.1234'

}                                                          //end Function.



double FurunoLatLonToRadian(double LatLon, String NSEW)       // This routine takes the Furuno GPS Lat or Lon format
{                                                             // sent by NMEA and returns the radian measure.
                                                              // Example: 3312.2567 W  Returns -.57952398...

  double z = LatLon, fractional, integer;                    //variable declaration
  z = LatLon / 100.0;                                        //divide by 100 to move the decimal left between the Furuno Deg and Min.
  fractional = modf(z, &integer);                            //Split the number at the decimal point into two vars holding Deg and Min.
  double z1 = integer + fractional * 100.0 / 60.0;           //Build the decimal coordinate. Note on the minutes to shift the decimal
                                                              //back two places and convert to degrees by /60.
  if(NSEW == "W" || NSEW == "S"){                            //West and South are signed as negative so need to check and adjust.
    z1 = z1 * -1;
  }
  z1 = z1 * PI/180.0;                                        //Convert the Decimal coordinate to radian

  return z1;

}


double FurunoLatLonToDecimal(double LatLon, String NSEW)  // This routine takes the Furuno GPS Lat or Lon format
{                                                          // sent by NMEA and returns the Decimal measure.
                                                           // Example: 3312.2567 W  Returns -33.20427833...

  double z = LatLon, fractional, integer;                  //variable declaration
  z = LatLon / 100.0;                                      //divide by 100 to move the decimal left between the Furuno Deg and Min.
  fractional = modf(z, &integer);                          //Split the number at the decimal point into two vars holding Deg and Min.
  double z1 = integer + fractional * 100.0 / 60.0;         //Build the decimal coordinate. Note on the minutes to shift the decimal back two places and convert to degrees by /60.

  if(NSEW == "W" || NSEW == "S"){                          //West and South are signed as negative so need to check and adjust.
    z1 = z1 * -1;
  }
  return z1;

}


double DistanceFormula(double Lat1, double Lat2, double Lon1, double Lon2)     //This function returns the distance between two GPS coords using the Haversine formula.
{                                                                             //Requires Radian coordinates , Returns Meters
  double R = ER;                                                              //Earth Radius in kilometers
  double DeltaLat = Lat2 - Lat1;
  double DeltaLon = Lon2 - Lon1;
  double A = pow(sin(DeltaLat/2.0),2)+cos(Lat1)*cos(Lat2)*pow(sin(DeltaLon/2.0),2);
  double C = 2.0 * atan2(sqrt(A),sqrt(1.0-A));
  double D = R * C;
  return D;
```

```
}

double BearingFormula(double Lat1, double Lat2, double Lon1, double Lon2)          //This function returns the intial bearing between two GPS coords.
{                                                                      //Requires Radian coordinates in +- Pi radians.
  double y = sin(Lon2-Lon1) * cos(Lat2);
  double x = cos(Lat1) * sin(Lat2) - sin(Lat1) * cos(Lat2) * cos(Lon2 - Lon1);
  return atan2(y, x);
}

double ForwardAzimuth(){                                                   //This function calulates the forward azimuth for a great circle at a given distance along the course line.
  double R = ER;   //Earth Radius in meters
  double atLat = asin(sin(ActiveStartLat)*cos(ATD/R)+cos(ActiveStartLat)*sin(ATD/R)*cos(IB));                //Latitude at distance given by ATD on a course line.
  double atLon = ActiveStartLon + atan2(sin(IB)*sin(ATD/R)*cos(ActiveStartLat),cos(ATD/R)-sin(ActiveStartLat)*sin(atLat));  //Longitude at distance given by ATD on a course line.

  Serial.println("Next Four Lines are atLat,CurLat,atLon,CurLon sent by ForwardAzimuth Function");
  Serial.print("atLat  = ");
  Serial.println(atLat,5);
  Serial.print("Curlat = ");
  Serial.println(CurLat,5);
  Serial.print("atLon  = ");
  Serial.println(atLon,5);
  Serial.print("CurLon = ");
  Serial.println(CurLon,5);

  return BearingFormula(atLat,ActiveEndLat,atLon,ActiveEndLon);                                  //result is signed +-Pi Radians.

}


double XtrackError(double Lat1, double Lat2, double Lat3, double Lon1, double Lon2, double Lon3)    //***This function returns the cross track error in meters and ATD in meters.
{

  double R = ER;                                                    //Assign R as the Earths Radius in km.
  double d1 = DistanceFormula(Lat1,Lat3,Lon1,Lon3)/ R;                        //Requires Start, End and Current GPS coord in Radians.
  double b1 = BearingFormula(Lat1,Lat3,Lon1,Lon3);
  double b2 = BearingFormula(Lat1,Lat2,Lon1,Lon2);
  double dxt = asin(sin(d1) * sin(b1-b2))* R;
  if(cos(dxt/R) != 0.0){                                            //Check to insure we don't divide by zero in the next calculation.
    double ValCheck = cos(d1)/cos(dxt/R);                                  //We need to check the next value as it will be used to calculate ATD by taking the arc cosine
    if(ValCheck > 1.0){                                             //acos can only accept values between -1 and 1.  Sometimes floating point math produces microfractions that can produce a nan.
      ValCheck = 1.0;                                               //We test for a value that is outside of the limit and if so set the value at the limit.
    }
    if(ValCheck < -1.0){
      ValCheck = -1.0;
    }
    ATD = acos(ValCheck)*R;                                             //This line calculates the distance traveled along the course line
  }                                                                 //also known as along track distance (ATD) and assigns to global in kilometers.
  Serial.print("dxt crosstrack in meters= ");
  Serial.println(dxt,5);
  Serial.print("dl = ");
  Serial.println(d1,5);
  Serial.print("b1 = ");
  Serial.println(b1,5);
  Serial.print("b1 deg = ");
  Serial.println(RtoD(b1),5);
  Serial.print("b2 = ");
  Serial.println(b2,5);
  Serial.print("b2 deg = ");
```

```
  Serial.println(RtoD(b2),5);
  Serial.print("ATD in km = ");
  Serial.println(ATD,5);
  Serial.print("Dist Point 1 to Point 3 in meters= ");
  Serial.println(d1 * R,5);
  Serial.print("IB = ");
  Serial.println(RtoD(IB),5);
  Serial.print("SB = ");
  Serial.println(RtoD(SB),5);
  Serial.println(Lat3,6);
  Serial.println(Lon3,6);
  return dxt;                                              //Return xtrack error in meters so need to divide km by 1000.
}


double DegreesOffCourse(double Lat1, double Lat2, double Lat3, double Lon1, double Lon2, double Lon3)  //*** A Rountine To Calculate Degrees off Course.
{
  double Mybear1 = BearingFormula(Lat1,Lat3,Lon1,Lon3);                              // This function calculates the degrees off-course to the End Point
  double Mybear2 = BearingFormula(Lat3,Lat2,Lon3,Lon2);                              // from the current Position.
  double DoffC = fmod(Mybear1 * 180.0 / PI + 360.0,360.0) - fmod(Mybear2 * 180.0 / PI + 360.0,360.0);          // Requires Start, End and Current GPS coords in Radians.
  return DoffC;                                                             // Returns Degrees off-course (-)=Left , (+)=Right from the corrected Bearing.
}


void TestDistance()                                                         //*** This is just a test routine.
{
double x1 = FurunoLatLonToRadian(3711.3, "N");
double y1 = FurunoLatLonToRadian(12219.738, "W");
double x2 = FurunoLatLonToRadian(3711.038, "N");
double y2 = FurunoLatLonToRadian(12202.681, "W");
double x3 = FurunoLatLonToRadian(3711.235, "N");
double y3 = FurunoLatLonToRadian(12204.82, "W");
double Mybear1 = BearingFormula(x1,x3,y1,y3);                               //
double Mybear2 = BearingFormula(x3,x2,y3,y2);                               // These three lines produce the degrees off course.
double DofC = fmod(Mybear1 * 180.0 / PI + 360.0,360.0) - fmod(Mybear2 * 180.0 / PI + 360.0,360.0);  // (-)=Left , (+)=Right
Serial.print("A");
Serial.println(fmod(Mybear1 * 180.0 / PI + 360.0,360.0));
Serial.print("B");
Serial.println(fmod(Mybear2 * 180 / PI + 360.0,360.0));


//double MyDist = XtrackError(x1,x2,x3,y1,y2,y3);
String mstring = String(DofC,2);
}

void ShowRadianData(){         //*** Test Routine to Display Radian Data.
Serial.println(StartLat,4);
Serial.println(StartLon,4);
Serial.println(EndLat,4);
Serial.println(EndLon,4);
Serial.println(CurLat,4);
Serial.println(CurLon,4);
}

void ShowCalculationResults(double Lat1, double Lon1,double Lat2,double Lon2,double Lat3,double Lon3){  //*** Test Routine to give calculation data.
Serial.println();
Serial.print("Initial Bearing: ");
Serial.println(fmod(BearingFormula(Lat1,Lat2,Lon1,Lon2) * 180.0 / PI + 360.0,360.0),1);
```

```
Serial.print("Distance (NM): ");
Serial.println(DistanceFormula(Lat1,Lat2,Lon1,Lon2)*0.000539957,2);  //converted to NM.
}

double GetRudderVoltage(){                    //Returns the Rudder Reference Voltage.
  int RudderVoltage = analogRead(RudderRef);       //Read the sensor on analog port defined by RudderRef in variable declarations.
  float voltage = RudderVoltage * (3.3/ 1024.0);     //Scale the value to correspond to a voltage 0 - 5V.
  return voltage;
}


double GetRudderAngle(){                      //*** This Routine returns the rudder angle from the rudder angle sensor.
  int RudderVoltage = analogRead(RudderRef);       //Read the sensor on analog port defined by RudderRef in variable declarations.
  float voltage = RudderVoltage * (3.3/ 1024.0);     //Scale the value to correspond to a voltage 0 - 5V.

  return round((voltage - RVmin)* 90.0 / (RVmax - RVmin)-45.0);         //Convert the voltage to Rudder Angle and scale -45 to +45,  Center is at 0.
}

void DisplayRudderAngle(){                    //*** Routine to Display Rudder Angle
  int RudderVoltage = analogRead(RudderRef);       //Read the sensor on analog port defined by RudderRef in variable declarations.
  float voltage = RudderVoltage * (3.3/ 1024.0);    //Scale the value to correspond to a voltage 0 - 5V.
  Serial.print("Rudder Voltage = ");
  Serial.println(voltage);

  double fractional, integer;                                       //Variables to display voltage using custom digits.
  fractional = modf(voltage, &integer);                             //Split the number at the decimal point into two vars holding volts and fractional volts.
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x08, integer);               //Write the volts to the display
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x09, fractional * 100.0);          //Write the fractional volts to the display


  //int RA = round(((voltage -.5)*22.784)-45);       //Convert the voltage to Rudder Angle and scale -45 to +45,  Center is at 0.
  //int RA = round((voltage - RVmin)* 90 / (RVmax - RVmin)-45);       //Convert the voltage to Rudder Angle and scale -45 to +45,  Center is at 0.
  int RA = (voltage - RVmin)* 90.0 / (RVmax - RVmin)-45.0;
  if(RA <= -2){
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x00, abs(RA));
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x01, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x02, abs(RA));
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x03, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x04, abs(RA));
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x05, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x06, abs(RA));
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x07, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x08, abs(RA));
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x09, 1);
  }
  if(RA > -2 && RA < 2){
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x00, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x01, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x02, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x03, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x04, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x05, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x06, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x07, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x08, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x09, 1);
  }
  if(RA >= 2){
```

```
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x00, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x01, RA);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x02, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x03, RA);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x04, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x05, RA);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x06, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x07, RA);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x08, 1);
    genie.WriteObject(GENIE_OBJ_GAUGE, 0x09, RA);
  }
  //genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x03, abs(RA));   This line can be used to send the rudder angle to the display.
}

void CourseAdjustCalc(){            //*** This routine calculates the course correction needed to return to the course line in signed radians.
                                    //The goal of this calc is to provide steering information to take the shortest path back to the course line,
                                    //with minimal overrun and oscillation.

  //We need the next three items for calculations.
  IB = BearingFormula(ActiveStartLat,ActiveEndLat,ActiveStartLon,ActiveEndLon);          //This calculates the initial bearing for the current course.
  NB = BearingFormula(CurLat,ActiveEndLat,CurLon,ActiveEndLon);                          //This calcualtes the bearing from current position to end point. Used for intermediate calcs.
  CH = DtoR(HeadingTrue);                                                                //This is our current heading (True). It is supplied in degrees so we convert to radians.


  double XTEM = XtrackError(ActiveStartLat,ActiveEndLat,CurLat,ActiveStartLon,ActiveEndLon,CurLon);  // Call the cross track error calculation, we need this signed value for the next calc.
  SB = IB;
  //SB = ForwardAzimuth();          //Since the bearing changes along the great circle we continuously update by calling the Forward Azimuth function.
                                    //The FA fuction depends on data updated by XtrackError function so XTE should always be called ahead of FA.
  double OE = fmod(3.0*PI+SB, 2.0*PI);     //This line calculates a value 180 degrees of the initial bearing.  Used for calculations
  double SDTD = 0;
  double TD = 0;
  if(CH >= OE){                     //This logic determines which side of the course line we are on and provides a signed value.
    TD = 2.0*PI-CH+NB;              //The value represents the number of radians needed to turn to achieve the New Heading Value.
  }                                 //The New Heading is given as the heading from the current position to the course end point.
  else                              //This is the first step to insure we orient in the direction toward the end point.
  {                                 //We need this to determine the shortest direction to turn in the case of extreme heading error.
    TD = NB-CH;
  }
  if(TD >= PI){                     //This code insures we turn the shortest direction of travel.  We don't turn 359°when we could turn -1°
    TD = TD-2.0*PI;
  }
  if(NB >= SB){                     //NotUsed*******This code modifies the degrees needed to turn to achieve a heading perpendicular to the course line.
    SDTD = TD+SB-NB+(PI/2.0);       //**************Perpendicular to the course line is the shortest return path to the course line.
  }                                 //**************Again we need to account for what side of the line we are on.
  else
  {
    SDTD = TD+SB-NB-(PI/2.0);       //Note**********SDTD not being used, this is now calculated by the SDTDC.
  }
                                    //  Bug Fix, changed XTEM <= to 90 instead of pi/2. Causing Sinusoidal path over course line. 5/31/2016.
  if(abs(XTEM)<= 90){               //  This code uses the crosstrack error to modify the final course correction based upon
    SDTDC = TD+SB-NB-(XTEM*PI/180.0);          // proximity to the course line.  If we are close then we need to be oriented more in the
  }                                 // direction of the end point.  At greater than 90 meters we orient perpendicular to take
  else                              // the shortest path back toward the course line. As we close in from 90 meters we make
  {                                 // adjustments to the heading so when we arrive at the course line the heading and forward Azimuth
    SDTDC = TD+SB-NB-(PI/2.0*abs(XTEM)/XTEM);  // are in agreement.
  }
                                    // SDTDC is the signed value in directional turning radians that can be passed to the steering routine.
```

```
Serial.print("Turning Degrees Needed = ");
Serial.println(SDTDC/PI*180.0,1);
Serial.print("Forward Azimuth = ");
Serial.println(fmod(SB/PI*180.0+360.0,360.0),1);
Serial.print("Initial Bearing = ");
Serial.println(fmod(IB/PI*180.0+360.0,360.0),1);
Serial.print("ATD= ");
Serial.println(ATD,1);


}

void HeadingNavigationAdjustCalc(){          //This Routine determines the heading correction needed to navigate to a preset bearing.
                                             //Used by the Heading Nav option.
//We need the next two items for calculations.

  NB = DtoR(NavBearing);                     //This is our Bearing Set by the HeadingNav Display.  Magnetic degrees converted to radians.
  CH = DtoR(HeadingMag);                     //This is our Current Heading with Deviation applied. Magnetic degrees converted to radians.

  double OE = fmod(3.0*PI+NB, 2.0*PI);       //This line calculates a value 180 degrees of the initial bearing.  Used for calculations
  double SDTD = 0;
  double TD = 0;
  if(CH >= OE){                              //This logic determines the current orientation to the bearing and provides a signed value.
   TD = 2.0*PI-CH+NB;                        //The value represents the number of radians needed to turn to achieve the Bearing Value.
  }
  else                                       //This is the first step to insure we orient toward the bearing.
  {                                          //We need this to determine the shortest direction to turn in the case of extreme heading error.
   TD = NB-CH;
  }
  if(TD >= PI){                              //This code insures we turn the shortest direction of travel.  We don't turn 359° when we could turn - 1°
   TD = TD-2.0*PI;
  }
  SDTDC = TD;                                //This is the Degrees needed to turn for course correction in Signed Radians.
}

void NavFunctionAdjustCalc(){                //***This Routine determines the heading correction needed to execute a Nav Function of 180 or 360.
                                             //Used by the Nav Function option.

  long ElapsedMillis = millis() - NavMillis;                       //Calculate how much time elapsed between Adjustment Calls.

  double AD = NMPH * ElapsedMillis / PI / Radius / 20000.0;        //Calculate Number of Degrees to adjust by applying rate time and distance for a given radius.
  NavDT = NavDT + AD;                                              //Counts Number of Adjusted Degrees applied to bearing

  if(NavFunction == 1 || NavFunction == 3){                        //If nav function 1 or 3 we are turning port.
   NavFBearing = D360(NavFBearing - AD);                           //Modify bearing by subtacting correction.
  }
  else                                                             //If nav function not 1 or 3 then by default it's 2 or 4 and we are turning Starb.
  {
   NavFBearing = D360(NavFBearing + AD);                           //Modify bearing by adding correction.
  }
  NavMillis = millis();                                            //We have our correction so store the current clock millis for next cycle.

                                                                   //Note: When in 180 mode, This code segment insures we don't turn beyond 180 and holds at 180 to the initial bearing until suspended.
  if(NavFunction == 1 || NavFunction == 2){                        //Check to see if we are using NavFunction 180.
   if(abs(NavDT) >= 180){                                          //180 active so check to see if we have completed 180 degrees of correction.
    NavFBearing = D360(IB + 180.0);                                //We've turned 180 so override corrections and hold course 180 to the Initial Bearing.
   }
```

```
  }

  NB = DtoR(NavFBearing);              //This is our new Bearing. It will keep adjusting based upon a fixed radius adjusted for time and speed.  Magnetic degrees converted to radians.
  CH = DtoR(HeadingMag);               //This is our Current Heading with Deviation applied. Magnetic degrees converted to radians.

  double OE = fmod(3.0*PI+NB, 2.0*PI);      //This line calculates a value 180 degrees of the initial bearing.  Used for calculations
  double TD = 0;
  if(CH >= OE){                        //This logic determines the current orientation to the bearing and provides a signed value.
    TD = 2.0*PI-CH+NB;                 //The value represents the number of radians needed to turn to achieve the Bearing Value.
  }
  else                                 //This is the first step to insure we orient toward the bearing.
  {                                    //We need this to determine the shortest direction to turn in the case of extreme heading error.
    TD = NB-CH;
  }
  if(TD >= PI){                        //This code insures we turn the shortest direction of travel.  We don't turn 359° when we could turn -1°
    TD = TD-2.0*PI;
  }
  SDTDC = TD;                          //This is the Degrees needed to turn for course correction in Signed Radians.
}



void SteeringRoutine(){                                       // *** THIS IS THE STEERING ROUTINE

  double RA = GetRudderAngle();                               //Get the Rudder Angle from the rudder angle position sensor.
  double SDTDCd = RtoD(SDTDC);                                //SDTDC is radians needed to turn calculated by CourseAdjustment routine or HeadingNavigationAdjustment routine or NavFunction
Routine.
  double SDTDCi = SDTDCd;                                     //SDTDCd is converted to degrees and SDTDCi is an intermediate that is adjusted by the following calc.
  if(SDTDCd < RM * -1.0){                                     //Here we check to see degrees needed exceeds the Rudders Maximum Range constant given by RM.
    SDTDCi = RM * -1.0;                                       //If we exceed the maximum port we reduce SDTDCi to RM.  RM is unsigned so we need to sign - for port.
  }
  if(SDTDCd > RM){                                            //Check for exceeding Rudder Maximum again only this time it's in the case of Starboard correction.
    SDTDCi = RM;                                              //If we exceed maximum starboard we reduce SDTDCi to RM.
  }
  double TD = SDTDCi * RG - RA + RT;                          //Calculate how many degrees to turn the rudder factoring in current Rudder Angle, Rudder Gain, Rudder Trim and SDTDC.
  Serial.print("RudderTurningDegrees = ");
  Serial.println(TD,2);
  if(TD >= -1 * QRV && TD <= QRV){                            //Check if we are in the Quiescent Range centered around zero TD.
    digitalWrite(LeftRudder,LOW);                             //If we are then set rudder outputs low to insure we are at idle helm pump.
    digitalWrite(RightRudder,LOW);
    LED_Direction(Off);
  }
  if(TD < -1 * QRV && RA > -1 * RM){                          //If our turning degrees needed is negative and we aren't at maximum rudder then we insure the right helm command is idle
    digitalWrite(RightRudder, LOW);                           //and issue the left helm command by setting the port high.
    digitalWrite(LeftRudder,HIGH);
    LED_Direction(Port);
  }
  if(TD > QRV && RA < RM){                                    //If our turning degrees needed is positive and we aren't at maximum rudder then we insure the left helm command is idle
    digitalWrite(LeftRudder,LOW);                             //and issue the right helm command by setting the starboard high.
    digitalWrite(RightRudder,HIGH);
    LED_Direction(Starb);
  }
}


void TestData(){

NMEA_Heading[0] = "$HCHDG,222.3,,,,E*06";
```

```
NMEA_Heading[1] = "$HCHDG,222.3,,,,E*06";
NMEA_Heading[2] = "$HCHDG,222.3,,,,E*06";
NMEA_Sentence[0] = "$GPHDG,28.2,,E,13.4,E*7E";
NMEA_Sentence[1] = "$GPRMB,A,0.00,L,0000,1001,3310.3463,N,11734.6109,W,9.5,258.4,0.0,V*2E";
NMEA_Sentence[2] = "$GPRMC,213102,A,3312.2571,N,11723.4573,W,0.0,309.4,131215,13.4,E*5B";
NMEA_Sentence[3] = "$GPGLL,3312.2571,N,11723.4573,W,213102,A*3A";


}

void ShowHeadingData(){          //Test Routine to Display Heading and Declination Data.
Serial.println(HeadingMag,1);
Serial.println(HeadingVar,1);
Serial.println(StartLat,4);
Serial.println(StartLon,4);
Serial.println(EndLat,4);
Serial.println(EndLon,4);
Serial.println(CurLat,4);
Serial.println(CurLon,4);

}


void TestCourseAdjust(){
  ActiveStartLat = StartLat;
  ActiveStartLon = StartLon;
  ActiveEndLat = EndLat;
  ActiveEndLon = EndLon;
  CourseAdjustCalc();
  Serial.print("Turn ");
  Serial.println(SDTDC/PI*180.0,1);
  Serial.print("FA ");
  Serial.println(fmod(SB/PI*180.0+360.0,360.0),1);
  Serial.print("IB ");
  Serial.println(fmod(IB/PI*180.0+360.0,360.0),1);
  Serial.print("ATD ");
  Serial.println(ATD,1);
}

void NavController(){                                   //*** This is the Main Navigation Controller, it determines which type of Navigation to use.
  if(RouteAuto == true){                                 // Check to see if Route Navigation is selected.
    CourseNavigation();                                  // if yes, call the Course(route)Navigation controller.
    Serial.println("CourseNavSelected");
  }
  if(HeadingSet == true){                                // Check to see if Heading Navigation is selected.
    HeadingNavigation();                                 // if yes, call the Heading Navigation controller.
    Serial.println("HeadingNavSelected");
  }
  if(NavFunction != 0){                                 // Check to see if NavFunction Navigation is selected.
    NavFunctionNavigation();                              // if yes, call the NavFunctionNavigation controller.
    Serial.println("NavFunctionSelected = ");
    Serial.print(NavFunction);
  }
  if(RouteAuto == false && HeadingSet == false && NavFunction == 0){          // if all options are off.
    SetRuddersInactive();                                // insure rudders are inactive.
    Serial.println("NoAutoNavSelected");
  }
}
```

```
void CourseNavigation(){                                        //*** This is the course navigation controller routine
  if(GPRMBflag==false || GPRMCflag==false || GPGLLflag==false || DVflag==false){    // check to see if we have all the information needed to navigate.
    CourseActive = false;                                       // if not set the CourseActive flag to false and don't auto navigate.
    digitalWrite(LeftRudder,LOW);                               // Insure Rudders are inactive
    digitalWrite(RightRudder,LOW);
    LED_Direction(Off);                                         // Turn off Direction indicators.
    Serial.println("CourseActive = False");
  }
  else                                                          // if we have all the necessary data to auto navigate
  {
    CourseActive = true;                                        // Set the Course Active flag to True.
    Serial.println("CourseActive = True");
    CourseAdjustCalc();                                         // Call the course adjustment calculation routine.
    SteeringRoutine();                                          // Call the Steering Routine.  Now we are auto navigating.
  }
}
void HeadingNavigation(){                                       //*** This is the heading navigation controller routine
  if(NavBearing == -1 || GPGLLflag==false || DVflag==false){    // check to see if we have all the information needed to navigate.
    CourseActive = false;                                       // if not set the CourseActive flag to false and don't auto navigate.
    digitalWrite(LeftRudder,LOW);                               // Insure Rudders are inactive
    digitalWrite(RightRudder,LOW);
    LED_Direction(Off);                                         // Turn off Direction indicators.
    Serial.println("HeadingNavActive = False");
  }
  else                                                          // if we have all the necessary data to auto navigate
  {
    CourseActive = true;                                        // Set the Course Active flag to True.
    Serial.println("HeadingNavActive = True");
    HeadingNavigationAdjustCalc();                              // Call the course adjustment calculation routine.
    SteeringRoutine();                                          // Call the Steering Routine.  Now we are auto navigating.
  }
}

void NavFunctionNavigation(){                                   //*** This is the NavFunction (180, 360) navigation controller routine.
  if(NavFBearing == -1 || NavFunction==0 || DVflag==false){     // check to see if we have all the information needed to navigate.
    CourseActive = false;                                       // if not set the CourseActive flag to false and don't auto navigate.
    digitalWrite(LeftRudder,LOW);                               // Insure Rudders are inactive
    digitalWrite(RightRudder,LOW);
    LED_Direction(Off);                                         // Turn off Direction indicators.
    Serial.println("NavFunctionActive = False");
  }
  else                                                          // if we have all the necessary data to auto navigate
  {
    CourseActive = true;                                        // Set the Course Active flag to True.
    Serial.println("NavFunctionActive = True");
    NavFunctionAdjustCalc();                                    // Call the NavFunction adjustment calculation routine.
    SteeringRoutine();                                          // Call the Steering Routine.  Now we are auto navigating.
  }
}

void SetRuddersInactive(){                                      //*** This routine is called to deactivate rudder control by setting the rudder outputs to low.
  CourseActive = false;                                         // Reset CourseActive flag.
  digitalWrite(LeftRudder,LOW);                                 // Insure Rudders are inactive
  digitalWrite(RightRudder,LOW);                                // by writing low to both rudder control pins.
  LED_Direction(Off);                                           // Set display rudder indicators led's to off.
}
```

```
void Counters(){

Serial.print("RMB Count =");
Serial.println(NoGPRMBCounter);
Serial.print("GLL Count =");
Serial.println(NoGPGLLCounter);
Serial.print("IB,SB");
Serial.println(RtoD(IB));
Serial.println(RtoD(SB));
}

void LED_Direction(String Direction){            // This routine is used to set the LED Status Indicators on the Display Panel
  if(Direction == "Port"){
    genie.WriteObject(GENIE_OBJ_LED, 0x01, 0);
    genie.WriteObject(GENIE_OBJ_LED, 0x03, 0);
    genie.WriteObject(GENIE_OBJ_LED, 0x0A, 0);
    genie.WriteObject(GENIE_OBJ_LED, 0x00, 1);
    genie.WriteObject(GENIE_OBJ_LED, 0x02, 1);
    genie.WriteObject(GENIE_OBJ_LED, 0x0B, 1);
  }
  if(Direction == "Starboard"){
    genie.WriteObject(GENIE_OBJ_LED, 0x00, 0);
    genie.WriteObject(GENIE_OBJ_LED, 0x02, 0);
    genie.WriteObject(GENIE_OBJ_LED, 0x0B, 0);
    genie.WriteObject(GENIE_OBJ_LED, 0x01, 1);
    genie.WriteObject(GENIE_OBJ_LED, 0x03, 1);
    genie.WriteObject(GENIE_OBJ_LED, 0x0A, 1);
  }
  if(Direction == "Off"){
    genie.WriteObject(GENIE_OBJ_LED, 0x00, 0);
    genie.WriteObject(GENIE_OBJ_LED, 0x02, 0);
    genie.WriteObject(GENIE_OBJ_LED, 0x01, 0);
    genie.WriteObject(GENIE_OBJ_LED, 0x03, 0);
    genie.WriteObject(GENIE_OBJ_LED, 0x0A, 0);
    genie.WriteObject(GENIE_OBJ_LED, 0x0B, 0);
  }

}

void SendRudderMaxMintoDisplay()                                        //***This Routine Writes the Rudder Max and Min values to the Display in Advanced Settings.
{
  double fractional, integer;                                           //Variables to display voltage using custom digits.
  fractional = modf(RVmin, &integer);                                   //Split the Voltage at the decimal point into two vars holding volts and fractional volts.
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x0A, integer);            //Write the volts to the display
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x0B, fractional * 100);   //Write the fractional volts to the display
  fractional = modf(RVmax, &integer);                                   //Split the Voltage at the decimal point into two vars holding volts and fractional volts.
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x0C, integer);            //Write the volts to the display
  genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x0D, fractional * 100);   //Write the fractional volts to the display
}


void myGenieEventHandler(void)
{
  Serial.println("CheckingEvents");
  genieFrame Event;
  genie.DequeueEvent(&Event); // Remove the next queued event from the buffer, and process it below
```

```
//If the cmd received is from a Reported Event (Events triggered from the Events tab of Workshop4 objects)
if (Event.reportObject.cmd == GENIE_REPORT_EVENT)
{
  if (Event.reportObject.object == GENIE_OBJ_FORM)                    // Check for Form Activation
  {
    if (Event.reportObject.index == 1)                               // If we have a message from Trackbar 0 (display contrast adjust)
    {
     NSEWtoDisplay();                                                // Update the NSEW for the Start Lat and Lon.
    }
  }


  if (Event.reportObject.object == GENIE_OBJ_TRACKBAR)                // Check for Trackbar messages
  {
    if (Event.reportObject.index == 0)                               // If we have a message from Trackbar 0 (display contrast adjust)
    {
      int contrast_val = genie.GetEventData(&Event);                 // Receive the event data from Trackbar 0
      genie.WriteContrast(contrast_val);                             // Set the contrast to the new value.
    }
    if (Event.reportObject.index == 1)                               // If we have a message from Trackbar 1 (Rudder Gain adjust)
    {
      RG = genie.GetEventData(&Event) / 10.0;                        // Receive the event data divide by 10 and set Rudder Gain.

    }
    if (Event.reportObject.index == 2)                               // If we have a message from Trackbar 2 (Rudder Trim adjust)
    {
      RM = genie.GetEventData(&Event);                               //  Receive the event data and set Rudder Max.

    }
    if (Event.reportObject.index == 3)                               // If we have a message from Trackbar 3 (Rudder Trim adjust)
    {
      RT = genie.GetEventData(&Event)- 5;                            // Receive the event and subtract 5 to scale plus or minus 5 for Rudder Trim.
      genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x02, RT);          // Write the new value to the TrackBar Numeric Display.
    }
    if (Event.reportObject.index == 4)                               // If we have a message from Trackbar 4 (QRV adjust)
    {
      QRV = genie.GetEventData(&Event) / 10.0;                       // Receive the event data divide by 10 and set QRV (sensitivity).

    }
    if (Event.reportObject.index == 5)                               // If we have a message from Trackbar 5 (Radius adjust)
    {
      Radius = genie.GetEventData(&Event) / 10.0;                    // Receive the event data divide by 10 and set Radius.
      EEPROM.put(0x55,Radius);                                       // Save changes to EEPROM.
      genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x06, Radius * 10.0);  // Update Radius Numeric Display. Can only send ints so multiply by 10, LedDigits will insert the decimal point.

    }




  }
  if (Event.reportObject.object == GENIE_OBJ_WINBUTTON)              // Check for WinButton messages
  {
    if (Event.reportObject.index == 0)                               // If we have a message from WinButton 0 (Route Nav Auto Button)
    {
      int ButtonVal = genie.GetEventData(&Event);                    // Receive the event data from WinButton 0
      if(ButtonVal == 1)                                             // 1 = Button On State
```

```
  {
    RouteAuto = true;                                   // Set flags
    genie.WriteObject(GENIE_OBJ_WINBUTTON,0x09,1);                 // Set HeadingSuspend to Off
    genie.WriteObject(GENIE_OBJ_WINBUTTON,0x1B,1);                 // Set NavFunctionSuspend to Off
    HeadingSet = false;                                 // Insure Heading Nav is Suspended.
    NavFunction = 0;                                    // Insure NavFunction option set to 0 for suspend.
    NavBearing = -1;                                    // Set NavBearing value to indicate suspend (used by heading nav).
    NavFBearing = -1;                                   // Set NavFBearing value to indicate suspend (used by NavFunction nav).

    SetRuddersInactive();                               // Rudders Inactive (anytime suspend is called we deactivate and let the new controller re-activate.)
  }
}
if (Event.reportObject.index == 1)                      // If we have a message from WinButton 1 (Route Nav Suspend Button)
{
  int ButtonVal = genie.GetEventData(&Event);                  // Receive the event data from WinButton 1
  if(ButtonVal == 1)                                    // 1 = Button On State
  {
    RouteAuto = false;                                  // Set flags Route Nav off
    SetRuddersInactive();                               // Rudders Inactive (anytime suspend is called we deactivate and let the new controller re-activate.)
    SB = 0;
  }
}
if (Event.reportObject.index == 0x1B)                   // If we have a message from WinButton 27 (Nav Function Suspend Button)
{
  int ButtonVal = genie.GetEventData(&Event);                  // Receive the event data from WinButton 27
  if(ButtonVal == 1)                                    // 1 = Button On State
  {
    NavFunction = 0;                                    // Set NavFunction option to 0 (Suspend)
    SetRuddersInactive();                               // Rudders Inactive (anytime suspend is called we deactivate and let the new controller re-activate.)
    NavFBearing = -1;                                   // Set NavFBearing value to indicate suspend (-1).
  }
}

if (Event.reportObject.index == 0x17)                   // If we have a message from WinButton 23 (Nav Function Port180)
{
  int ButtonVal = genie.GetEventData(&Event);                  // Receive the event data from WinButton 23
  if(ButtonVal == 1)                                    // 1 = Button On State
  {
    NavFunction = 1;                                    // Set NavFunction Option to indicate Port180 (option 1)
    NavFunctionOption();                                // Call The NavFunctionOption Initiation.
  }
}
if (Event.reportObject.index == 0x19)                   // If we have a message from WinButton 25 (Nav Function Starb180)
{
  int ButtonVal = genie.GetEventData(&Event);                  // Receive the event data from WinButton 25
  if(ButtonVal == 1)                                    // 1 = Button On State
  {
    NavFunction = 2;                                    // Set NavFunction Option to indicate Starb180 (option 2)
    NavFunctionOption();                                // Call The NavFunctionOption Initiation.
  }
}
if (Event.reportObject.index == 0x18)                   // If we have a message from WinButton 24 (Nav Function Port360)
{
  int ButtonVal = genie.GetEventData(&Event);                  // Receive the event data from WinButton 24
  if(ButtonVal == 1)                                    // 1 = Button On State
  {
    NavFunction = 3;                                    // Set NavFunction Option to indicate Port360 (option 3)
    NavFunctionOption();                                // Call The NavFunctionOption Initiation.
```

```
  }
}
if (Event.reportObject.index == 0x1A)                               // If we have a message from WinButton 26 (Nav Function Starb360)
{
  int ButtonVal = genie.GetEventData(&Event);                      // Receive the event data from WinButton 26
  if(ButtonVal == 1)                                       // 1 = Button On State
  {
    NavFunction = 4;                                       // Set NavFunction Option to indicate Starb360 (option 4)
    NavFunctionOption();                                     // Call The NavFunctionOption Initiation.
  }
}

if (Event.reportObject.index == 0x14)                               // If we have a message from WinButton 20 (Settings ADV Save button)
{
  int ButtonVal = genie.GetEventData(&Event);                      // Receive the event data from WinButton 20
  if(ButtonVal == 0)                                       // Momentary buttons send 0 as the activated message.
  {
    //put the EEPROM save instructions here.
    EE = 1;
    EEPROM.put(0x00,EE);                                      //We received the save command so set location 0x00 to 1 to indicate we have data saved.
    EEPROM.put(0x05,NumNMEA);                                      //Number of NMEA reads per cycle.
    EEPROM.put(0x0A,NumHeading);                                   //Number of Heading reads per cycle.
    EEPROM.put(0x0F,NoGPRMBLimit);                                   //Fault limit for cycles with no GPRMB.
    EEPROM.put(0x14,NoGPGLLLimit);                                  //Fault limit for cycles with no GPGLL.
    EEPROM.put(0x19,RVmin);                                      //Rudder Voltage Minimum.
    EEPROM.put(0x23,RVmax);                                      //Rudder Voltage Maximum.
    EEPROM.put(0x2D,RM);                                      //Rudder Maximum Degrees Deflection.
    EEPROM.put(0x37,RT);                                      //Rudder Trim.
    EEPROM.put(0x41,QRV);                                      //Quiscent Rudder Value.
    EEPROM.put(0x4B,RG);                                      //Rudder Gain.
    Serial.println("EEPROM UPDATED");
    //all done

  }
}



if (Event.reportObject.index == 2)                               // If we have a message from WinButton 2 (Route Nav ReRoute Button)
{
  int ButtonVal = genie.GetEventData(&Event);                      // Receive the event data from WinButton 2
  if(ButtonVal == 0)                                       // Momentary buttons send 0 as the value
  {
    NoGPRMBCounter = NoGPRMBLimit + 1;                              // Set GPMRB counter past limit to trigger a re-route from current location.

  }
}
if (Event.reportObject.index == 8)                               // If we have a message from WinButton 8 (Heading Nav Set Button)
{
  int ButtonVal = genie.GetEventData(&Event);                      // Receive the event data from WinButton 8
  if(ButtonVal == 1)                                       // 1 = Button On State
  {
    if(HeadingSet == false){
      RouteAuto = false;                                     // Set flags Route Nav off
      SB = 0;
      NavFunction = 0;                                     // Insure NavFunction option is suspended - Option(0)
      NavFBearing = -1;                                     // Set NavFBearing to -1 for NavFunction suspend mode.
      SetRuddersInactive();                                     // Rudders Inactive (anytime suspend is called we deactivate and let the new controller re-activate.)
```

```
      genie.WriteObject(GENIE_OBJ_WINBUTTON,0x01,1);              // Insure Route Nav button is suspended
      genie.WriteObject(GENIE_OBJ_WINBUTTON,0x1B,1);              // Insure NavFunction button is suspended
      HeadingSet = true;                                          // HeadingSet = true;
      NavBearing = HeadingMag;                                    // Set our NavBearing to the current Magnetic heading with Deviation Trim added.
      if(MorT == false){
        genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x00, NavBearing);    // Send the NavBearing to the display.
      }
      else
      {
        genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x00, D360(NavBearing + HeadingVar));  //convert to True for display only.
      }
    }
  }
}
if (Event.reportObject.index == 9)                                // If we have a message from WinButton 9 (Heading Nav Suspend Button)
{
  int ButtonVal = genie.GetEventData(&Event);                     // Receive the event data from WinButton 9
  if(ButtonVal == 1)                                              // 1 = Button On State
  {
    HeadingSet = false;                                           // Set Heading Nav flag to off
    SetRuddersInactive();                                         // Rudders Inactive (anytime suspend is called we deactivate and let the new controller re-activate.)
    NavBearing = -1;                                              // Set NavBearing value to indicate suspend.
  }
}
if (Event.reportObject.index == 0x0A)                             // If we have a message from WinButton 10 (Decrement Button)
{
  int ButtonVal = genie.GetEventData(&Event);                     // Receive the event data from WinButton 10
  if(ButtonVal == 0)                                              // Momentary buttons send 0 as the value
  {
    if(HeadingSet == true){                                       // If we are Heading Navigating then:
      NavBearing = D360(NavBearing - 1);                          // Decrement the current Nav Bearing.
      if(MorT == false){
        genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x00, NavBearing);    // Send the NavBearing to the display.
      }
      else
      {
        genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x00, D360(NavBearing + HeadingVar));  //convert to True for display only.
      }
    }
  }
}
if (Event.reportObject.index == 0x0B)                             // If we have a message from WinButton 11 (Increment Button)
{
  int ButtonVal = genie.GetEventData(&Event);                     // Receive the event data from WinButton 11
  if(ButtonVal == 0)                                              // Momentary buttons send 0 as the value
  {
    if(HeadingSet == true){                                       // If we are Heading Navigating then:
      NavBearing = D360(NavBearing + 1);                          // Increment the current Nav Bearing.
      if(MorT == false){
        genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x00, NavBearing);    // Send the NavBearing to the display.
      }
      else
      {
        genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x00, D360(NavBearing + HeadingVar));  //convert to True for display only.
      }
    }
  }
}
```

```
if (Event.reportObject.index == 0x0D)                    // If we have a message from WinButton 13 (Mag Heading Button)
{
  int ButtonVal = genie.GetEventData(&Event);            // Receive the event data from WinButton 13
  if(ButtonVal == 1)
  {
    MorT = false;
    if(HeadingSet == true){                               // If we are Heading Navigating then:
      genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x00, NavBearing);    // Send the NavBearing to the display.
    }
  }
}
if (Event.reportObject.index == 0x0E)                    // If we have a message from WinButton 14 (True Heading Button)
{
  int ButtonVal = genie.GetEventData(&Event);            // Receive the event data from WinButton 14
  if(ButtonVal == 1)
  {
    MorT = true;
    if(HeadingSet == true){                               // If we are Heading Navigating then:
      genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x00, D360(NavBearing + HeadingVar));    // Send the NavBearing to the display.
    }
  }
}
if (Event.reportObject.index == 0x15)                    // If we have a message from WinButton 21 (Advanced Settings SEL Button)
{
  int ButtonVal = genie.GetEventData(&Event);            // Receive the event data from WinButton 21
  if(ButtonVal == 0)                                     // Momentary buttons send 0 as the value
  {
    if(Sel == 10)                                        //10 is the inactive selection at the end of the select cycle.
    {
      Sel = 4;                                           //Sel = 10 so go to beginning of selection cycle starting at 4.
      genie.WriteObject(GENIE_OBJ_LED, Sel, 1);          //Turn on LED 4.
    }
    else                                                 //If we aren't at the end of the selection cycle.
    {
      Sel++;                                             //Advance cycle by 1.
      genie.WriteObject(GENIE_OBJ_LED, Sel-1, 0);        //Turn off the previous select cycle LED
      genie.WriteObject(GENIE_OBJ_LED, Sel, 1);          //Turn on the current selection LED.
    }
   }
}
if (Event.reportObject.index == 0x11)                    // If we have a message from WinButton 17 (Advanced Settings DN Button)
{
  int ButtonVal = genie.GetEventData(&Event);            // Receive the event data from WinButton 17
  if(ButtonVal == 0)                                     // Momentary buttons send 0 as the value
  {
    switch (Sel) {
      case 4:
       NumNMEA--;
       if(NumNMEA < 1){
        NumNMEA = 1;
       }
       genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x04, NumNMEA);
       break;
      case 5:
       NumHeading--;
       if(NumNMEA < 1){
        NumNMEA = 1;
       }
```

```
      genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x05, NumHeading);
      break;
     case 6:
      NoGPRMBLimit--;
      if(NoGPRMBLimit < 1){
       NoGPRMBLimit = 1;
      }
      genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x06, NoGPRMBLimit);
      break;
     case 7:
      NoGPGLLLimit--;
      if(NoGPGLLLimit < 1){
       NoGPGLLLimit = 1;
      }
      genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x07, NoGPGLLLimit);
      break;
     case 8:
      RVmin = GetRudderVoltage();
      SendRudderMaxMintoDisplay();
      break;
     case 9:
      RVmax = GetRudderVoltage();
      SendRudderMaxMintoDisplay();
      break;
     case 10:
      break;

   }
  }
 }
 if (Event.reportObject.index == 0x12)                              // If we have a message from WinButton 18 (Advanced Settings DN Button)
{
  int ButtonVal = genie.GetEventData(&Event);                      // Receive the event data from WinButton 18
  if(ButtonVal == 0)                                               // Momentary buttons send 0 as the value
  {
   switch (Sel) {
     case 4:
      NumNMEA++;
      if(NumNMEA > 10){
       NumNMEA = 10;
      }
      genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x04, NumNMEA);
      break;
     case 5:
      NumHeading++;
      if(NumNMEA > 10){
       NumNMEA = 10;
      }
      genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x05, NumHeading);
      break;
     case 6:
      NoGPRMBLimit++;
      if(NoGPRMBLimit > 1000){
       NoGPRMBLimit = 1000;
      }
      genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x06, NoGPRMBLimit);
      break;
     case 7:
```

```
            NoGPGLLLimit++;
            if(NoGPGLLLimit > 1000){
             NoGPGLLLimit = 1000;
            }
            genie.WriteObject(GENIE_OBJ_CUSTOM_DIGITS, 0x07, NoGPGLLLimit);
            break;
           case 8:
            RVmin = GetRudderVoltage();
            SendRudderMaxMintoDisplay();
            break;
           case 9:
            RVmax = GetRudderVoltage();
            SendRudderMaxMintoDisplay();
            break;
           case 10:
            break;

         }
       }
      }



    }
   }

  //If the cmd received is from a Reported Object, which occurs if a Read Object (genie.ReadOject) is requested in the main code, reply processed here.
  if (Event.reportObject.cmd == GENIE_REPORT_OBJ)
  {
    if (Event.reportObject.object == GENIE_OBJ_USER_LED)              // If the Reported Message was from a User LED
    {
      if (Event.reportObject.index == 0)                              // If UserLed0 (Index = 0)
      {
        bool UserLed0_val = genie.GetEventData(&Event);               // Receive the event data from the UserLed0
        UserLed0_val = !UserLed0_val;                                 // Toggle the state of the User LED Variable
        genie.WriteObject(GENIE_OBJ_USER_LED, 0, UserLed0_val);       // Write UserLed0_val value back to to UserLed0
      }
    }
  }
}


void NavFunctionOption(){                  //NavFunction Initiation.  The NavFunction Buttons call this when pressed on.

  //Suspend Route and Heading functions

  RouteAuto = false;                                            // Set flags Route Nav off
  SB = 0;                                                       // Reset Start Bearing Var
  HeadingSet = false;                                           // Insure Heading Nav is Suspended.
  NavBearing = -1;                                              // Set NavBearing value to indicate suspend (used by heading nav).
  genie.WriteObject(GENIE_OBJ_WINBUTTON,0x01,1);                // Insure Route Nav button is suspended
  genie.WriteObject(GENIE_OBJ_WINBUTTON,0x09,1);                // Insure HeadingSuspend Button is suspended
  SetRuddersInactive();                                         // Rudders Inactive (anytime suspend is called we deactivate and let the new controller re-activate.)

  //Initiate the NavFunction

  IB = HeadingMag;                         //Set Initial Bearing to the Current Magnetic Heading
  NavFBearing = HeadingMag;                //Initialize NavFBearing to Current Magnetic Heading.  This is the bearing we follow during our turn, as such it adjusts constantly during execution.
```

```
  NavDT = 0;                          //Initialize Degrees Turned to zero.  This is used to track our progress.
  NavMillis = millis();               //Initialize to current clock value.  Used to calculate elapsed time between adjustments for rate,time and distance calcs applied to bearing adjustment.

}
```