EMBEDDED SYSTEM FOR QUADRICOPTER

Technical Report #3

GROUND STATION
DOCUMENTATIONS and CODE EXPLANATIONS

Bùi Nhã Đạt

Poitiers, January 2014

Introduction

This technical report is a part of the PFE Quadricopter. There are two parts in this report:

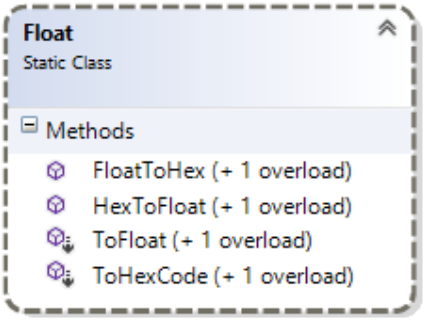
- **Part 1** explains the methods in the DLL libraries: the ***HexCodec.dll*** – extension methods for encoding/decoding a float variable to/from a string; and the ***AHcommLib.dll*** – library for preparing data stream in communication with the Artificial Horizontal via UDP protocol.
- **Part 2** explains the ***Quadricopter Ground Station source codes.***

1 DLL libraries

1.1 HexCodec.dll

HexCodec is a library that includes methods for encoding/decoding a float variable to/from a string.

1.1.1 Class diagram



1.1.2 Methods

```
public static float ToFloat(this string hexCode,
                            bool swapEndianness)
```

- **Input:** `this string hexCode`, `bool swapEndianness`.
- **Return:** parse the `hexCode` string into float value. If `swapEndianness` is true, the endianness will be changed by reversing the byte order of the returned variable.
- **Example:**

```
string hexCode = "DB0F4940";
Console.WriteLine(hexCode.ToFloat(true));
Console.WriteLine(hexCode.ToFloat(false));
```

Results on console:

```
3.141593
-4.033146E+16
```

- **Overload method and related methods:**

```
public static float HexToFloat(string hexCode)
  { return hexCode.ToFloat(false); }
public static float HexToFloat(string hexCode, bool swapEndianness)
  { return hexCode.ToFloat(swapEndianness); }
/// Overload method(s):
public static float ToFloat(this string hexCode)
  { return ToFloat(hexCode, false); }
```

```
public static string ToHexCode(this float value,
                               bool swapEndianness)
```

- **Input:** `this float value`, `bool swapEndianness`.
- **Return:** convert the `value` into hex string. If `swapEndianness` is true, the endianness will be changed by reversing the byte order. The returned string is in uppercase.
- **Example:**

```
float value = 3.141593f;
Console.WriteLine(value.ToHexCode(true));
Console.WriteLine(value.ToHexCode(false));
```

Results on console:

```
40490FDC
DC0F4940
```

- **Overload method and related methods:**

```
public static string FloatToHex(float value)
  { return value.ToHexCode(false); }
public static string FloatToHex(float value, bool swapEndianness)
  { return value.ToHexCode(swapEndianness); }
/// Overload method(s):
public static string ToHexCode(this float value)
  { return ToHexCode(value, false); }
```

1.2 AHcommLib.dll

The free a340gc (**A**irbus A**340** **G**lass **C**ockpit) project is used as a part of the Quadricopter Ground Station: the artificial horizontal.

> *"The Glass Cockpit Library (libGC) is created for the Airbus A340 Glass Cockpit (a340gc) which is an Open Source project. The a340gc project is part of the Airbus A340 simulator project of the IRADIS Foundation. The goal of libGC and a340gc is to create a free framework that can be used to build a glass cockpit upon."*
>
> —a340gc.iradis.org

The Ground Station communicates with a340gc by sending RDDP packet via UDP. AHcommLib.dll is a library for preparing data stream in form of RDDP packet.

This technical report focuses on documenting the code rather than explaining how to use the external resources. Since then, for further informations that concern to the libGC, a340gc, or RDDP packet, please refer the documentations on a340gc.iradis.org/documentation/.

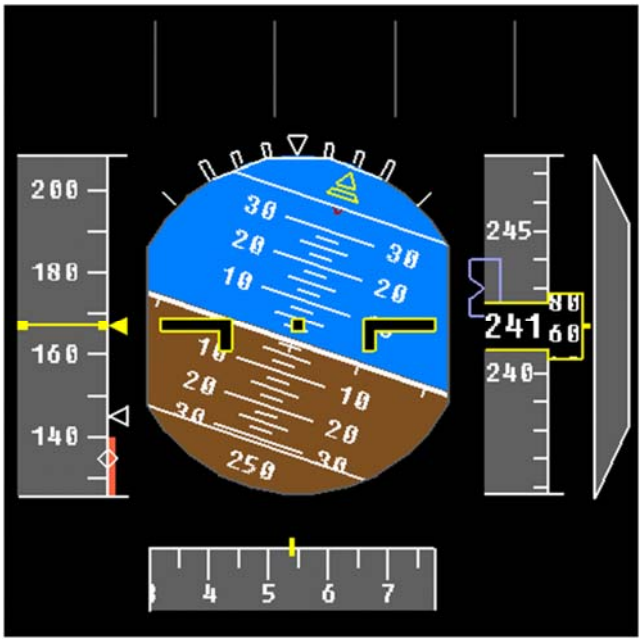


Figure 1. The Artificial Horizontal a340gc

1.2.1 RDDP Packet

The documentation of the RDDP packet will be repeated here in order to understand the AHcommLib.dll more easily. Content of this part (RDDP packet) is a part of

http://a340gc.iradis.org/documentation/libGC/libgc.html#Raw%20Distributed%20Data%20Protocol.

***

The RDDP packets can be sent using any method. Currently only an UDP/IP connection will be supported. With very little effort, it is possible to use other network protocols or pipes.

An RDDP packet is constructed of 1 RDDP packet header and 0 to 255 RDDP Data Records.

Figure 2. RDDP packet

1.2.1.1 Packet Header

Each RDDP packet starts with the RDDP packet header.

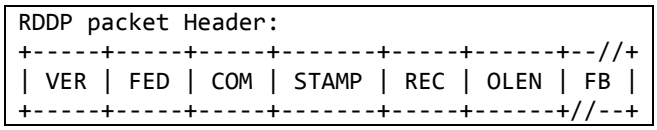


Figure 3. RDDP packet header

Filed	Type	Description
VER	8bit UInteger	Version number of RDDP
FED	32bit UInteger	Federate ID number
COM	32bit UInteger	Component ID number
STAMP	32bit UInteger	Timestamp
REC	8bit UInteger	The number of RDDP data records
OLEN	8bit UInteger	Length of the future block
FB #tab Variable	Future block	

VER The version number of the RDDP specification/implementation. This version number should be used to determine the contents of the FB.

FED The unique identification number of the federate the component belongs to

COM The unique identification number of the component.

STAMP The timestamp field identifies the Attribute change number for the component. The first attribute change is marked as '0'.

REC The number of RDDP Data Records that follow the RDDP Header.

OLEN This field indicates the length of the FB. The first RDDP Data Record can be found by adding this value to the length of the RDDP Header excluding the length of the known BF size. For the first RDDP implementation, this value will be '0' since the FB is not utilized.

FB The Future Block will be used in future RDDP Specification that require the RDDP Header to be expanded for new information. Future expansions of the Data Records will be appended after the base RDDP Data Record array.

1.2.1.2 Data Records

After the RDDP packet header there can be 0 to 255 data record.

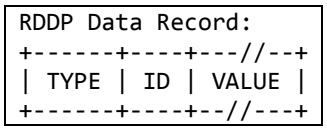


Figure 4. RDDP data record

Field	Type	Description
TYPE	8bit UInteger	Data type of variable
ID	32bit SInteger	Variable identifier
VALUE	[N] bytes	New values for variable

1.2.1.3 RDDP Variable Types

This section lists the definition of RDDP variable types.

Type	Data Record	Size	Range
boolean	10	8bit	0=FALSE, 1=TRUE
integer	40	32bit	–2147483648 to +2147483647
float	80	16bit	Standard ANSI float
double	84	32bit	Standard ANSI double
string	120	1+[N] byte	Length byte + 0 to 255 characters

1.2.2 Enumerators

```
public enum AHtypeID:byte
{
  BOOL    = 16,
  INT     = 40,
  FLOAT   = 84, // 32-bit float ---- double in DIOM
  FLOAT16 = 80, // 16-bit float ----  float in DIOM
  STRING  = 120
}
```

This enumerator defines the data type identifiers (TYPE filed in RDDP data record), based on Part 1.2.1.3 (RDDP Variable Types).

```
public enum AHID:int
{
  heading  = 1000,
  pitch    = 1001,
  roll     = 1002,
  altitude = 1010
}
```

This enumerator defines some variable identifiers (ID filed in RDDP data record): heading, pitch, roll, and altitude. For other identifiers, please refer the configuration file a340gc.conf.

```
public enum AHsize:int
{
  INT    = 9,
  FLOAT  = 9,
  header = 15
}
```

This enumerator defines the size of a type-defined RDDP data record.

1.2.3 Methods

```
public static byte[] AHheader(byte VER, int FED, int COM,
                              int STAMP, byte REC, byte OLEN)
```

This method generates a RDDP packet header as a byte array.

```
public static byte[] AHint(AHID ID, int VAL)
```

This method generates a RDDP data record for integer type.

```
public static byte[] AHfloat(AHID ID, float VAL) //32 bit floating point
```

This method generates a RDDP data record for float type.

1.2.4 Usage example

```
01 using System.Net.Sockets;
02 using AHcommLib;
03 // UDP socket variables
04 string senderIP = "127.0.0.255";
05 int senderPort = 9000; //Artifical Horizon UDP listening port
06 IPEndPoint ArtificalHorizonUDPEndPoint;
```

```
07  Socket ArtificalHorizonUDPSocket;
08  // UPD socket initializing
09  ArtificalHorizonUDPEndPoint = new IPEndPoint(IPAddress.Parse(senderIP),
    senderPort);
10  ArtificalHorizonUDPSocket = new Socket(AddressFamily.InterNetwork,
    SocketType.Dgram, ProtocolType.Udp);
11  private void BroadcastUDP(int heading, int roll, int pitch, int altitude)
12  {
13      int headerSize = (int)AHcomm.AHsize.header;
14      int intRecordSize = (int)AHcomm.AHsize.INT;
15      //int fltRecordSize = (int)AHcomm.AHsize.FLOAT;
16      byte[] AHheaderV1 = AHcomm.AHheader(1, 1, 1, 0, 1, 0);
17      byte[] buff = new byte[headerSize + intRecordSize];
18      Buffer.BlockCopy(AHheaderV1, 0, buff, 0, headerSize);
19
20      Buffer.BlockCopy(AHcomm.AHint(AHcomm.AHID.heading, heading + 180), 0,
    buff, headerSize, intRecordSize);
21      ArtificalHorizonUDPSocket.SendTo(buff, buff.Length, SocketFlags.None,
    ArtificalHorizonUDPEndPoint);
22
23      Buffer.BlockCopy(AHcomm.AHint(AHcomm.AHID.roll, roll), 0, buff,
    headerSize, intRecordSize);
24      ArtificalHorizonUDPSocket.SendTo(buff, buff.Length, SocketFlags.None,
    ArtificalHorizonUDPEndPoint);
25
26      Buffer.BlockCopy(AHcomm.AHint(AHcomm.AHID.pitch, pitch), 0, buff,
    headerSize, intRecordSize);
27      ArtificalHorizonUDPSocket.SendTo(buff, buff.Length, SocketFlags.None,
    ArtificalHorizonUDPEndPoint);
28
29      Buffer.BlockCopy(AHcomm.AHint(AHcomm.AHID.altitude, altitude * 100),
    0, buff, headerSize, intRecordSize);
30      ArtificalHorizonUDPSocket.SendTo(buff, buff.Length, SocketFlags.None,
    ArtificalHorizonUDPEndPoint);
31  }
```

- Line 16 generates a byte array of RDDP packet header.
- Line 17 creates a byte array for buffering the complete RDDP packet.
- Line 18 puts the RDDP header into the buffer.
- Line 20 (23, 26, 29) generates a data record for the heading (roll, pitch, yaw, altitude), puts it into the buffer, behind the RDDP header.
- Line 21 (24, 27, 30) sends the buffer as a UDP package via an UDP socket.

This example is a part of the *Quadricopter Ground Station.Networking.cs* source code. Here, the method `BroadcastUDP` sends four RDDP packets; each packet holds only one data record (heading/roll/pitch/altitude). Since one RDDP packet can hold up to 255 data record, this `BroadcastUDP` method can be optimized by putting all 4 data records of heading, roll, pitch and altitude in a single RDDP packet.

2 Quadricopter Ground Station Source Code

2.1 Definitions & Global Variables

Different from C or C++, C# does not use macros. So public constants are used instead. These constants are used mostly as an array index.

Source file: **Quadricopter Ground Station.Definitions.cs**

```
/* === DEFINITIONS ========= */
public const byte ROLL  = 0;
public const byte PITCH = 1;
public const byte YAW   = 2;
public const byte ALTITUDE = 3;
public const byte KP = 0;
public const byte KI = 1;
public const byte KD = 2;

public const byte ASSISTED = 0;
public const byte AUTO     = 1;

public const char STX = '\x02';
public const char EOT = '\x04';
public const char ETB = '\x17';

///
/// === GPGGA STRUCTURE ============
///
public struct GPGGA
{
    public float time ; // HHMMSS.SSS
    public float lat;
    public float lon;
    public float alt;
    public byte quality;
    public byte sat_num;
}
```

Source file: **Quadricopter Ground Station.GlobalVariables.cs**

```
///
/// === ORDER LIMITATIONS ==========
///
public int[] limMax = { 3, 3, 360 }; // ROLL PITCH YAW LIMITERS
public int[] limMin = { -3, -3, 0 };
///
/// === STATE VARIABLES ==========
///
/// the current Euler Angles
public float[] eulerAngles = new float[3];
```

```
/// the current altitude
public float altitude;
/// the current GPS informations
public GPGGA gpsPosition;

/// the current MCU time (micros) that received from the Quadricopter
public uint micros = 0;
/// the time that the Ground Station started
DateTime startTime = DateTime.Now;

/// the orders: [ROLL, PITCH, YAW, ALTITUDE]
public float[] orders = { 0, 0, 0, 0 };
/// the default gains for quick resetting gains --- Should be reviewed!!!
public float[] defaultGains = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

/// the current gains --- Should be reviewed!!!
public float[] currentGains = { 9, 8, 7, 6, 5, 4, 3, 2, 1 };
/// the new gains will be sent to the Quadricopter
public float[] gainsToSet = { 0, 0, 0, 0, 0, 0, 0, 0, 0 };
/// the current mode
public byte mode = ASSISTED;

///
/// === NETWORK VARIABLES ==========
///
public string ordersToSend = "";
public string[] gainsToSend = new string[3];
public string modeToSend = "";
public string messageReceived = "";
```

2.2 Nested type

Source file: **Quadricopter Ground Station.Definitions.cs**

```
public struct GPGGA
{
    public float time ; // HHMMSS.SSS
    public float lat;
    public float lon;
    public float alt;
    public byte quality;
    public byte sat_num;
}
```

2.3 Monitor & Control

Monitor & Control: helps user to monitor and control the Quadricopter.

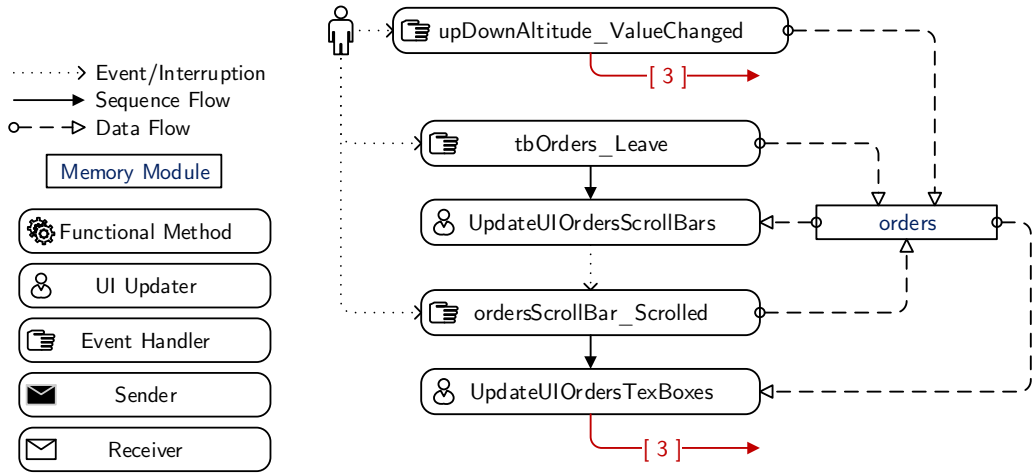
- The current date time and MCU time are display on the top.
- The panel *Altitude and Attitude* displays the current state of the Quadricopter, gathers the desired values via both text boxes and scroll bars; there is the *gains tuner* for tuning the gains of the Quadricopter PID controllers.

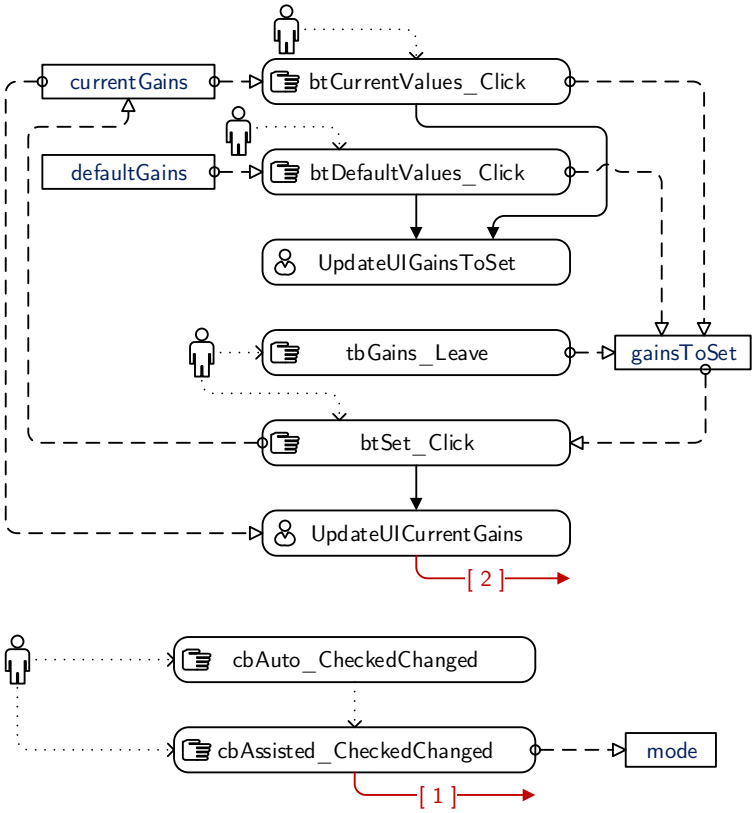
- The panel *GPS* displays the current GPS information.
- On the bottom, there are two toggle buttons for displaying and toggling the current operating mode.
- On the right hand side, the *AHRS Display* is left blank for later use if necessary. Currently, the free module Artificial Horizon is used instead.
- The *GPS Map* display the current position of the Quadricopter from its longitude and latitude. This map uses Google Maps API. With Google Maps API, we can do path tracking with intractable map. However, due to the limitation of time, this GPS is no more than a static map centered at the current position of the Quadricopter.

The interface will be updated by the *UI Updaters* – the methods: `UpdateUIClocks`, `UpdateUIAltitude`, `UpdateUIAttitude`, and `UpdateUIGPS`. These UI Updaters are triggered periodically every 100 ms by the timer `timerInterface`.

When user interacts with the UI (change text in the text boxes, scroll the scroll bars, and toggle the toggle buttons...), the related *event handler* will be invoked. The following figures show the sequence/data flows when user triggers an event on the UI. The orders, gains, or mode will be parsed from UI components and put into related memory module (the global variable `orders`, `gains`, or `mode`), related *message updater* will then be invoked (the sequence flows [1], [2] and [3]).

Annotations





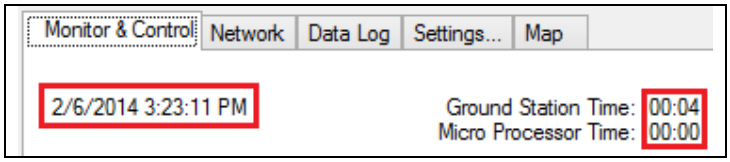
2.3.1 Interface updaters

Source file: **Quadricopter Ground Station.UIProcessing.cs**

`private void UpdateUIAltitude()`
Read the current altitude from `altitude` and update the relative text box.

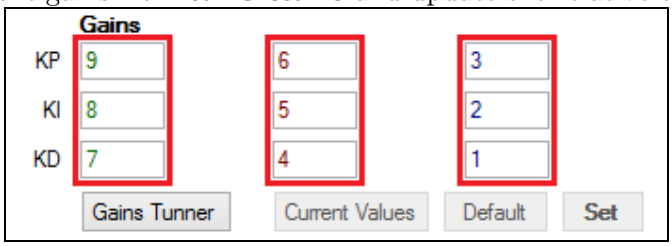
`private void UpdateUIAttitude()`
Read the current roll, pitch, yaw values from `eulerAngles` and update the relative text boxes.

`private void UpdateUIClocks()`
Update the clocks on UI.

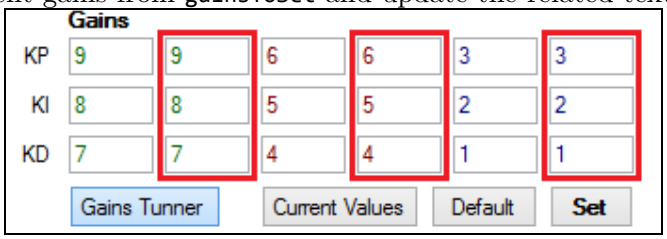


Source file: **Quadricopter Ground Station.UIProcessing.cs**

`private void UpdateUICurrentGains()`
Read the current gains from `currentGains` and update the relative text boxes.



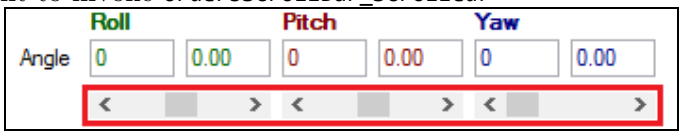
`private void UpdateUIGainsToSet()`
Read the current gains from `gainsToSet` and update the related text boxes.



`private void UpdateUIGPS()`
Read values from `gpsPosition` and update the related text boxes.

`private void UpdateUIOrdersTextBoxes()`
Read the current orders from `orders` and update the related text boxes.

`private void UpdateUIOrdersScrollBars()`
Read the current orders from `orders` and update the related scroll bars. This can trigger an event to invoke `ordersScrollBar_Scrolled`.



2.3.2 Event handlers

Source file: **Quadricopter Ground Station.EventHandlers.AltAtt.cs**

`private void ordersScrollBar_Scrolled(object sender, EventArgs e)`

- Parse the current position of the sender scroll bar into reference value of the related attitude and update it into the `orders`,
- Call `UpdateUIOrdersTextBoxes()` to update the orders text boxes,
- Call `UpdateOrdersToSend()`.

Senders link to this handler:

`private void tbOrders_Leave(object sender, EventArgs e)`

- Try to parse the text in the sender text box and update it into `orders`. If an invalid value were entered, the text will be changed back to the previous valid value.
- Call `UpdateUIOrdersScrollBars()`. This can trigger an event to `ordersScrollBar_Scrolled`.

Senders link to this handler:

`private void upDownAltitude_ValueChanged(object sender, EventArgs e)`

- Get the reference altitude value in the numeric up down box and update it into `orders`.
- Call `UpdateOrdersToSend()`.

Sender links to this handler:

Source file: **Quadricopter Ground Station.EventHandlers.GainsTuner.cs**

Additional method(s):

`private void LoadGainsToSet(float[] inGains)`

- Load the values of `inGains` into `gainsToSet`,
- Update the related text boxes by calling `UpdateUIGainsToSet()`.

`public int GainsIndex(byte channel, byte PID)`

This method returns `channel * 3 + PID`.

Source file: **Quadricopter Ground Station.EventHandlers.GainsTuner.cs**

`private void btCurrentValues_Click(object sender, EventArgs e)`

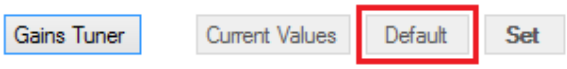
- Call `LoadGainsToSet(currentGains)` (this will load the `currentGains` into `gainsToSet` and update the related text boxes).

Sender links to this handler:

`private void btDefault_Click(object sender, EventArgs e)`

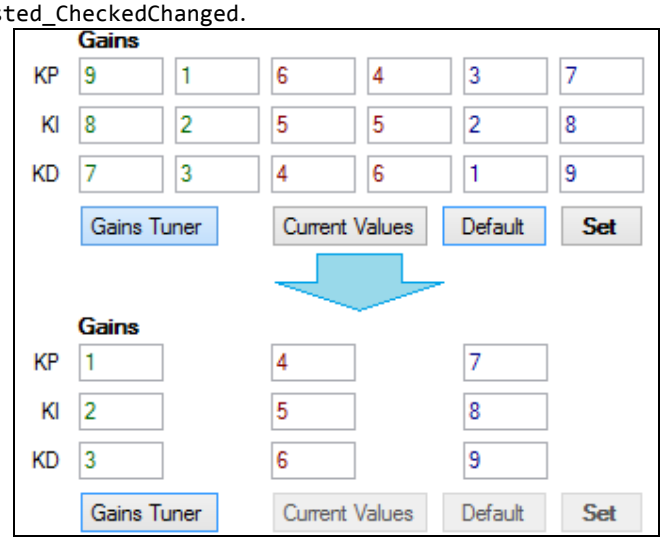
- Call `LoadGainsToSet(defaultGains)` (this will load the `defaultGains` into `gainsToSet` and update the related text boxes).

Sender links to this handler:



`private void btSet_Click(object sender, EventArgs e)`

- Copy `gainsToSet` to `currentGains`,
- Call `UpdateUICurrentGains()` (this will update the current gains tool boxes),
- Call `UpdateGainsToSend()`.
- Toggle the check box `cbGainsTuner` off. This will triggers an event to `cbAssisted_CheckedChanged`.

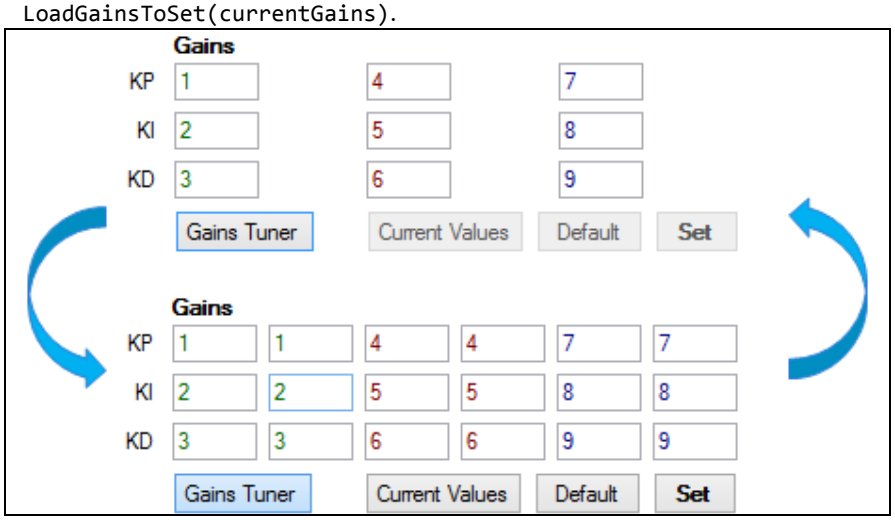


Sender links to this handler:

Source file: **Quadricopter Ground Station.EventHandlers.GainsTuner.cs**

`private void cbGainsTuner_CheckedChanged(object sender, EventArgs e)`

- Toggle the text boxes `tbGains---`, the `btCurrentValue`, `btDefault` and `btSet` on/off.
- If toggling from off to on, this method also calls `LoadGainsToSet(currentGains)`.

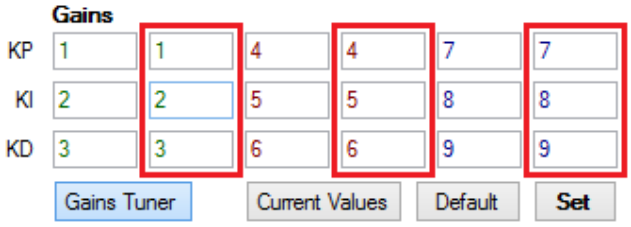


Sender links to this handler:

`private void tbGains_Leave(object sender, EventArgs e)`

Try to parse the text in the sender text box and update that value into `gainsToSet`. If an invalid value were entered, the text will be changed back to the previous valid value.

Senders link to this handler:



Source file: **Quadricopter Ground Station.EventHandlers.Mode.cs**

`private void cbAssisted_CheckedChanged(object sender, EventArgs e)`

- Toggle the `cbAuto`,
- Update the `mode` from the current state of `cbAssisted.Checked`,
- Call `UpdateModeToSend()`.

Source file: **Quadricopter Ground Station.EventHandlers.Mode.cs**

```
private void cbAuto_CheckedChanged(object sender, EventArgs e)
```

Toggle the `cbAssisted`. This will trigger an event to `cbAssisted_CheckedChanged`.

2.4 Networking

Networking module consists of the `TransceiverDoWork` method (hereafter *transceiver*), which is invoked periodically by the timer `timerInterface`, a `SenderDoWork` method (hereafter *sender*), a `ReceiverDoWork` method (hereafter *receiver*), three functional methods (hereafter *message updaters*) `UpdateModeToSend`, `UpdateGainsToSend`, and `UpdateOrdersToSend`, and two control methods Connect and Disconnect.

These **message updaters** are invoked by UI event handlers, whenever orders or mode is changed, or the new gains are set by user via UI. The invoked message updater prepare the message, following the transmission protocol, and put the message(s) in the related *message holder* (the strings `orderToSend`, `gainsToSend`, `modeToSend`).

Once invoked by the timer, the **transceiver** checks the connection status (connected/disconnected, timeout). If the connection is OK, it will invoke the `SenderDoWork` method and then the `ReceverDoWork` method.

- `ReceiverDoWork` will check the network stream. If the stream is available, the receiver waits for dollar sign '$' on the data stream for synchronizing. Once synchronized, the complete message will be acquired and parsed; the global variables (`micros`, `altitude`, `eulerAngles`, and `gpsPosition`) will then be updated. If synchronization cannot be done for an interval of time, the timeout flag `isTimeout` will be turned on.
- `SenderDoWork` will check the length of the message holders. If one of these strings is not empty, it will be sent to the Quadricopter via network stream. After being sent, the related message holder will be cleared.

If timeout occurs, the flag `disconnectNow` will be turned on. `Disconnect` method will then be invoked.

The `Connect` method will try to establish a TCP connection to the Quadricopter via the IP and port number entered on the UI. If this TCP connection can be established, the timer `timerNetworking` will then be started, and an UDP socket (for sending data to Artificial Horizon module) will be opened.

The `Disconnect` method closes the connection and stops the timer.

2.4.1 Network variables

Source file: **Quadricopter Ground Station.Networking.cs**

```
public byte timeout = 10; // seconds
public bool connected = false;
public bool disconnectNow = false;
public bool isTimeout = false;
DateTime lastActivity;

// TCP socket -- Communicate with the Quadricopter
public TcpClient tcpClient = new TcpClient();
public NetworkStream networkStream;
public StreamReader streamReader;
public StreamWriter streamWriter;

// UDP socket -- Communicate with the Artificial Horizon
string senderIP = "127.0.0.255";
int senderPort = 9000; //Artifical Horizon UDP listening port
IPEndPoint artificalHorizonUDPEndPoint;
Socket artificalHorizonUDPSocket;
```

2.4.2 Interface updaters

Source file: **Quadricopter Ground Station.Networking.cs**

`private bool Connect()`

- Try to connect to the sever address specified on UI.
- Return the connection status.

`private void Disconnect()`

- Close the current connection,
- Update status to UI,
- Reset the flag (`connected = false`), turn off the networking timer.

`private void TransceiverDoWork()`

- Read timeout interval from UI,
- Check if `connected` and not `disconnetNow` and not `isTimeout`, then call the receiver (`ReceiverDoWork()`) and sender (`SenderDoWork()`).
- The receiver will update the `isTimeout` flag.
- If `isTimeout`, turn on the flag `disconnectNow`.
- If `disconnectNow`, call `Disconnect`().

`private void SenderDoWork()`

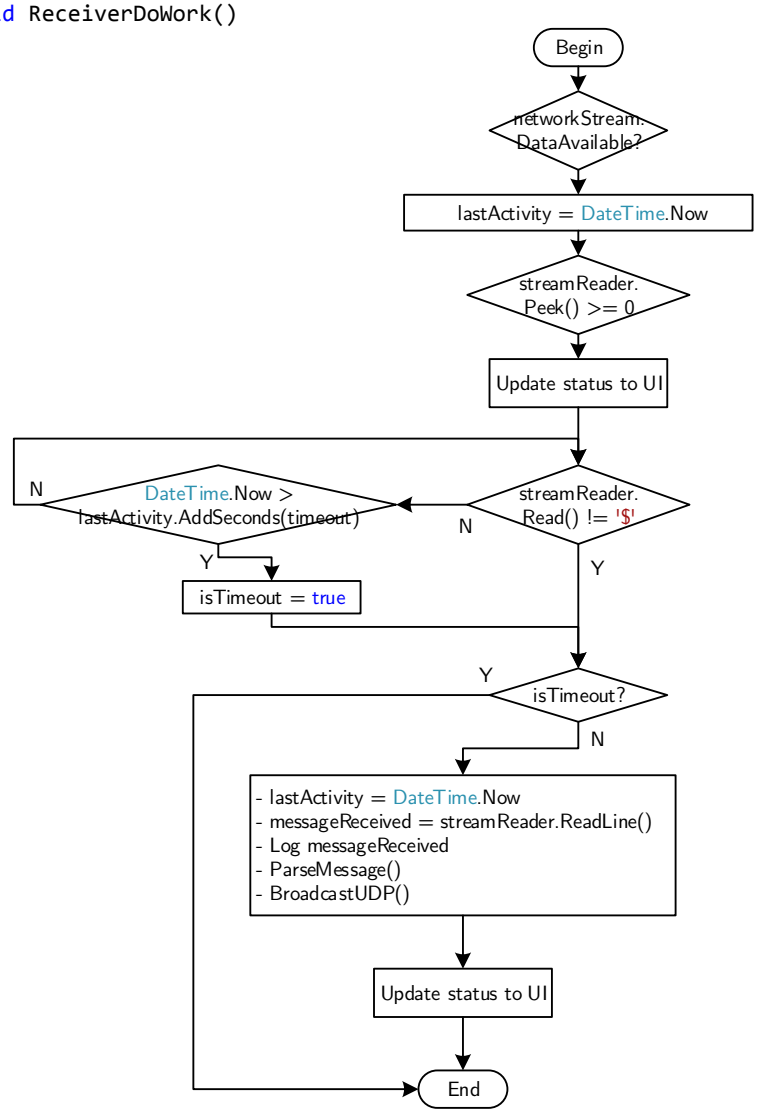
Check if one of the message holders `ordersToSend`, `gainsToSend` or `modeToSend` is not null, then send them to the network. If more than one message holder are not empty, only one will be sent at a time. Related message holder will be emptied after being sent.

Source file: **Quadricopter Ground Station.Networking.cs**

```
private void UpdateOrdersToSend()
private void UpdateGainsToSend()
private void UpdateModeToSend()
```

Read data from `orders`, `gainsToSet` or `mode`, convert into string (message) following the communication protocol. The generated strings will be put in the message holder `ordersToSend`, `gainsToSend` or `modeToSend`.

```
private void ReceiverDoWork()
```




```
private void ParseMessage()
```

Parse the received message into values and update into the global variables (`micros`, `eulerAngles`, `altitude`, `gpsPosition`).

```
private void BroadcastUDP(int heading, int roll, int pitch, int altitude)
```

Send RDDP packet via UDP protocol to update the Artificial Horizon.

2.4.3 Event handlers

```
private void btConnect_Click(object sender, EventArgs e)

private void btDisconnect_Click(object sender, EventArgs e)
```

2.5 Timers