Jessica Clara Fealy

18024092

Event Driven and Graphical User INterface Programming

Remote Flight Controller User Interface

Contents

[Remote Controller Interface 1](#_Toc70615666)

[Connecting to Flight Controller 1](#_Toc70615667)

[Receiving and Sending Messages 2](#_Toc70615668)

[Autopilot 3](#_Toc70615669)

[Implementation of Events, Delegates, and Threads 7](#_Toc70615670)

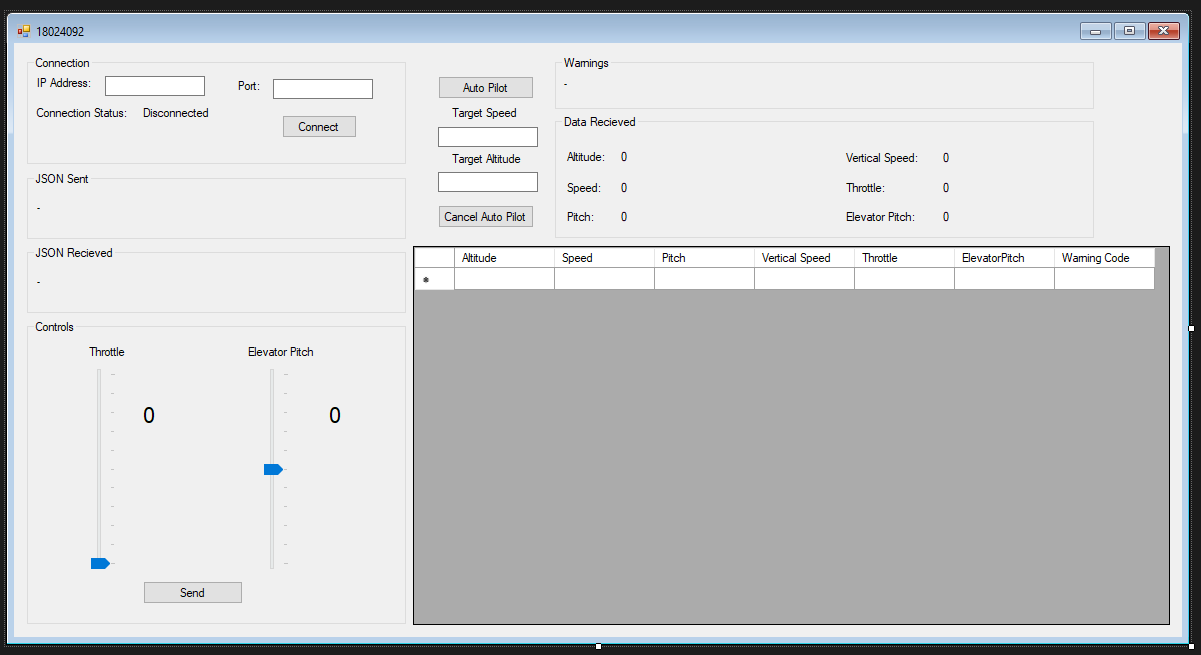
[Events 7](#_Toc70615671)

[Delegates 7](#_Toc70615672)

[Assigning Events with Delegates 8](#_Toc70615673)

[Threads 8](#_Toc70615674)

# Remote Controller Interface

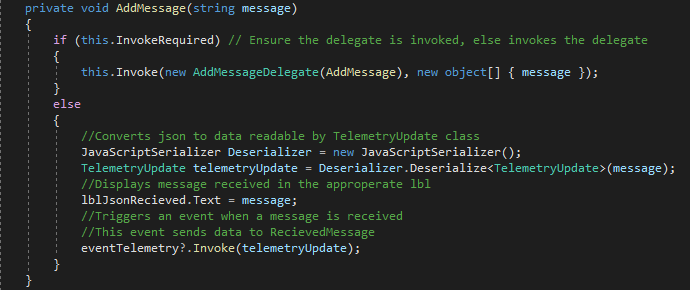


# Connecting to Flight Controller

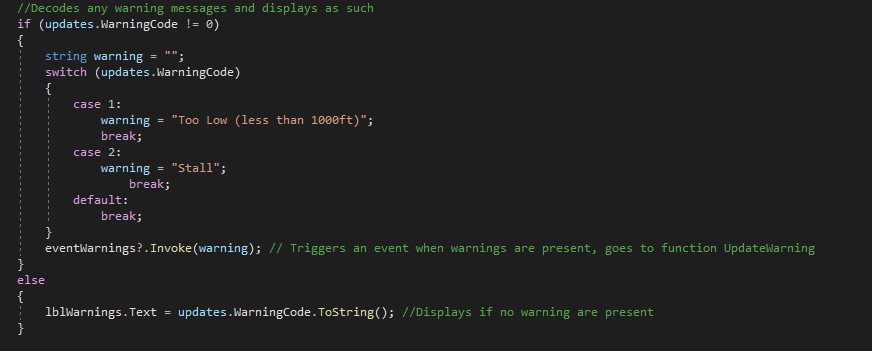
For the remote controller (Client) to connect to the Flight Simulator (Server) the sever must be set to listen for new clients. On the Client interface the server IP address and the desired port must be entered, then pressing connect. Doing so will ensure all commands are correctly sent to and received from the flight simulator. Once the connection has been initialised the user will be indicated as such.

# Receiving and Sending Messages

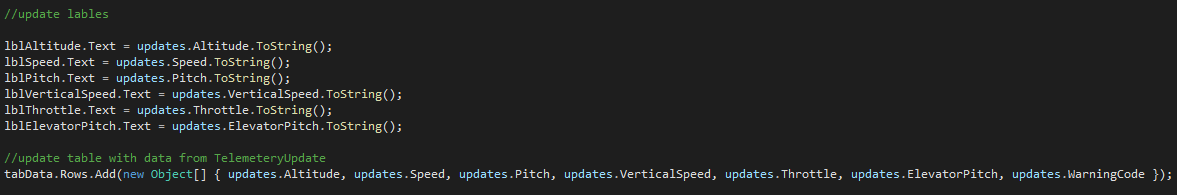
The Client can only understand messages received in serialised JSON format. Message listening is handled in a separate thread which is initialised once a connection to the server has been formed. When a message is first received RecivedJSON label is updated with the JSON message. The message is assumed to be received in serialised JSON format, this message is deserialised into an object of the TelemetryUpdate class. An event is invoked via eventTelementry, this event has the TelemetryUpdated delegate assigned which takes an instance of TelemeteryUpdate as an argument which is used to push the message received to the function ReceivedMessage. This can been seen below:



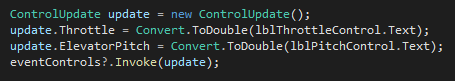
The RecievedMessage function first handles decoding any warning messages received from the Server, if the message received had any warning state other than 0 then a string is defined with the relevant warning received, and the eventWarning is invoked. This event has the warningUpdated delegate assigned which directs the data to the UpdateWarnings function. This function handles updating the warning label with the relevant state. This can be seen below:



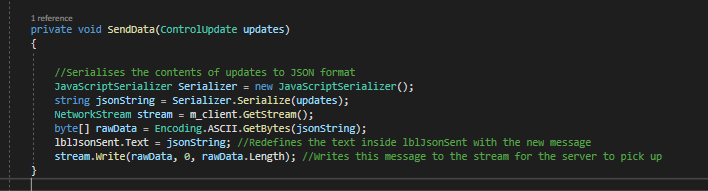
After any warning states has been understood and decoded all other data is updated to their relevant label, and a new entry is made into the data view box. This code is shown below:



For the client to send a message to the server the “Send” button must be pressed. Once this is done the current values of the Throttle and Pitch trackbars is taken and stored to an instance of ControlUpdate class, then eventControls is invoked, this event has the ControlsUpdated delegate assigned which take an instance of ControlUpdate as an argument and passed the data onto the SendData function, this code can be seen below:

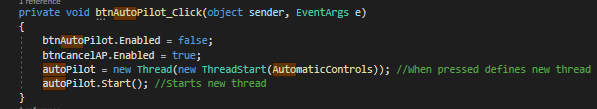


Inside the SendData function the date within update (instance of ControlUpdate) is serialised into JSON format and sent to the server. The data sent to the server is first added to the lblJsonSent label. This code is shown below:

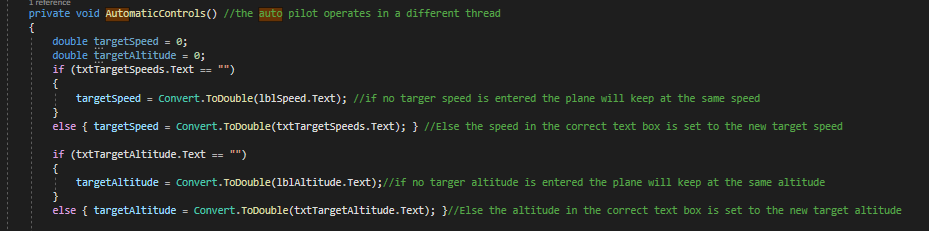


# Autopilot

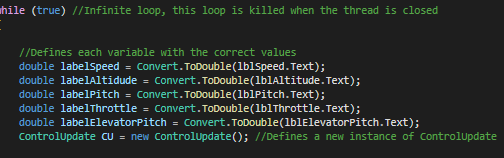
This remote flight controller has the functionality of autopilot, with user defined target speed and target altitude the client attempts to reach these targets and attempts to remain there. When the user starts the autopilot feature a new thread is started, this can be shown below:



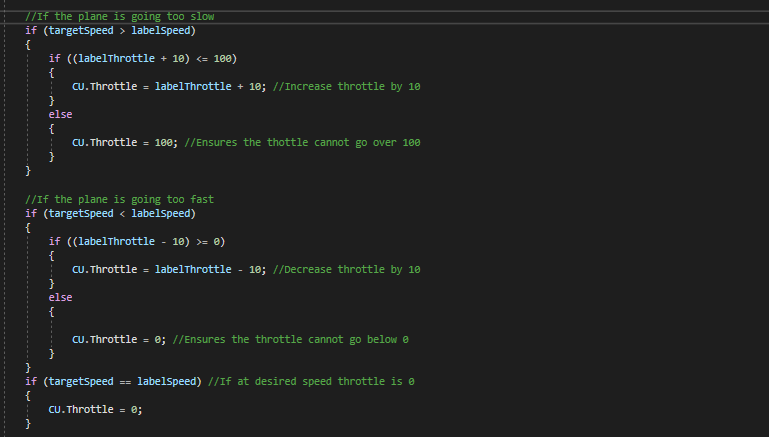
The function of AutomaticControls uses an infinite loop which can only be stopped with the thread being killed. The AutomaticControls function begins by setting the target speed and altitudes that the user wants the plane to achieve and maintain. This code is shown below:



Now the infinite loop begins, the loops begins by updating the currently store value for the system pitch, velocity, altitude, throttle, and elevator pitch by pulling this data from the value store under the labels.



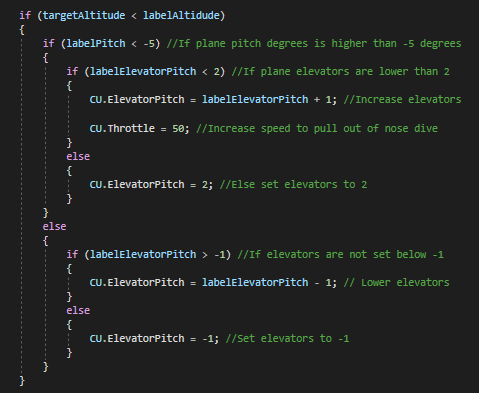
AutomaticControls now has all the required information to begin taking control of the aircraft, firstly the program decides if the throttle of the plane is to be increase, decreased or to remain the same. This is done by determining if the current speed is faster or slower than the target speed, and adjusting the throttle accordingly, this code can be seen below:



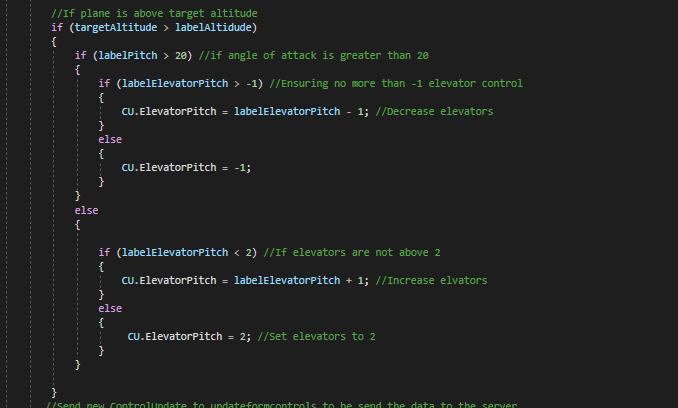
The program will be being adjusting the altitude of the plane by changing the positioning of the elevators control surfaces. The program follow certain rules when changing elevator positions to reduce the risk of the plane losing control, these rules are as follows:

* The planes pitch cannot exceed 20 degrees
* The plane cannot go below -5 degrees in pitch
* The elevators cannot exceed setting 2
* The elevators cannot fall below -1

If the plane is lower than the target altitude then the elevators will be at a higher position, unless the plane exceed 20 degrees in pitch then the elevators will decrease their levels (but not below -1) until the plane falls below 20 degrees pitch, where the elevators will behaviour as normal. This code can be seen below:

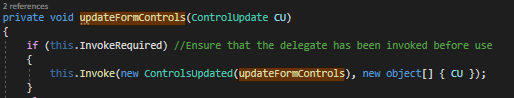


If the plane is higher than the target altitude then the elevators of the plane will be decreased in setting, if the planes pitch falls above -5 degrees then the elevators will be raised in positioning (but not exceeding level 2) until the pitch is not below -5 where the elevators will behave as normal. This code is shown below:

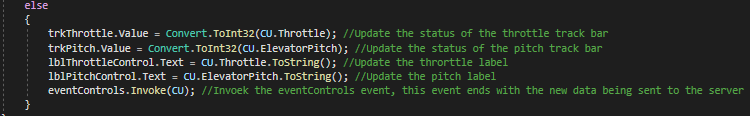


These new control updates are sent to updateFormControls.

Inside this function, a check is preformed to determine if an invoke is required, if so then a new ControlsUpdated delegate is created. Shown below:



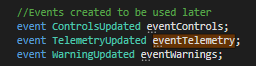
If no invoking is required then the code proceeds normally. The throttle and elevators trackbars are updated using the data given from the passed in ControlUpdate instance CU, the relevant label are also updated. The eventControl is invoked passing in the instance of ControlUpdate, this event leads to the data from ControlUpdate to be sent to the server, similar to how data is sent when the user manually pressed the send button. This is shown below:



# Implementation of Events, Delegates, and Threads

## Events

For this program, all events are created globally, thus can be access anywhere in the program. 3 events have been created for this program to work as intended, however events can be called multiple times. Uses of events in this program:

* Receiving messages
  + The event for this has been called eventTelemetry
    - This event is only invoked once, this happens when a message is received in AddMessage
* Receiving warnings
  + The event for this has been called eventWarning
    - This event is only invoked once, this happens whenever a error code is decoded in RecievedMessage.
* Sending messages
  + The event for this has been called eventControls
    - This event is invoked twice during the course of the project. Once when the user presses “ Send “ and again on update from the autopilot

All these events have delegates assigned to them, so when they are invoked a function is referenced and data is passed along.

## Delegates

For this program 4 delegates are created, this are created globally and can be accessed anywhere within the program. 3 of these delegates have been assigned to happen when their respective events invoke, and are used to reference to another function and pass along the appropriate data. These 3 delegates are:

* ControlsUpdated
  + This delegate takes an instance of ControlUpdate as an argument
* TelemeteryUpdated
  + This delegate take an instance of TelemeteryUpdate as an argument
* WarningUpdated
  + This delegate take a string as an argument

The last delegate is AddMessageDelegate which is used when an invoke is required, then this delegate is assigned.

## Assigning Events with Delegates

When assigning an event with a delegate you must give the delegate a function to return the data to. The 3 events are each assigned a delegate these are:

* Event Controls assigned ControlsUpdated
  + Whenever this event is invoked the values passed into ControlsUpdated (An instance of ControlUpdate is required) then this data is passed to the function SendData.
  + This event is triggered twice
* EventTelemetry assigned TelemeteryUpdated
  + Whenever this event is invoked the values passed into TelemetryUpdated (An instance of TelemetryUpdate is required) is passed to the function ReceivedMessage
* EventWarnings assigned WarningsUpdated
  + Whenever this event is invoked the value passed into WarningsUpdated (in form of a string) is passed into the function UpdateWarnings.

Below can be seen the assigning of events with delegates:



## Threads

This program uses 2 different threads to work correctly. Once Thread being responsible for continuously listening for and updating whenever a new message is received, and another thread to automatically update the controls for the autopilot.

Both these threads are defined in the same manner. An example of this can be found below:



When the thread begins this allows one part of the program is being doing one task, with another part of the program busy with another task.

This threads can be stopped as shown below:

