



Mapping planes using ADS-B

James Coote

April 26th 2012

**Dissertation submitted in partial fulfilment for the degree of
Bachelor of Science in Computer Science**

**Department of Informatics
University of Sussex**

1 Summary

This report gives details about the design, implementation and testing of my project. It contains a background on the technology that I have used and the justifications behind it. Each chapter contains a brief introduction that gives a summary of what is contained within it.

The project itself is about using signals sent from aircraft to track their positions on a map. The map is available for anyone to view over the Internet and can be interacted with to find further information. The map acts like a home radar screen and can be used to view planes from wherever a receiver is, regardless of its location. To use the signals coming from the aircraft I need to decode them so they can be added to the map. The decoded flight data is then recorded into a local database to enable access at a later date.

The users of the website can interact with the data in a number of ways, including plotting the flight paths and viewing the flight from the cockpit of the plane. They can view information about each plane that wouldn't normally be available from just the signals, such as destination and airline. Additionally, the users also have access to old flight data, allowing them to query the database for flights that have already landed and view their flight path on the map.

Users with their own receivers are able to contribute their data to my website by downloading my decoding software and connecting to the Internet. This allows them to use their data on my website and serves to increase the amount of airspace that my site covers.

2 Table of Contents

1 Summary	2
2 Table of Contents	3
3 Introduction	5
3.1 <i>ADS-B background</i>	5
3.2 <i>Project accomplishments</i>	7
3.3 <i>Report structure</i>	7
4 Professional considerations	9
4.1 <i>Ethical Issues</i>	9
4.2 <i>ADS-B security considerations</i>	10
4.3 <i>Project security considerations</i>	11
4.4 <i>Legality concerns</i>	11
5 ADS-B technology	12
5.1 <i>ADS-B Signal Transmission</i>	12
5.2 <i>ADS-B Packet Formation</i>	12
5.3 <i>Compact Position Report (CPR) Decoding</i>	14
5.4 <i>Heading and Velocity decoding</i>	18
5.5 <i>ID Decoding</i>	21
6 Requirements analysis	22
6.1 <i>Competition</i>	22
6.2 <i>Choosing the right development tools</i>	22
6.3 <i>Use cases</i>	27
6.4 <i>End users</i>	30
7 Requirements specification	31
7.1 <i>Functional requirements</i>	31
7.2 <i>Non-functional requirements</i>	33
8 High level design	34
8.1 <i>Design methodology</i>	34
8.2 <i>Decoder overview</i>	34
8.3 <i>Web page overview</i>	34
8.4 <i>Database overview</i>	34
8.5 <i>Abstract system overview</i>	35
8.6 <i>What is Plane Tracker?</i>	35

<i>8.7 Activity diagrams</i>	36
9 Low level design	39
<i>9.1 Receiver</i>	39
<i>9.2 Database</i>	50
<i>9.3 Website and web server</i>	53
10 Implementation	69
<i>10.1 Development environments used</i>	69
<i>10.2 Difficulties faced</i>	69
11 System testing	70
<i>11.1 Requirements specification</i>	70
<i>11.2 Integration testing</i>	71
<i>11.3 Hardware and browser testing</i>	77
<i>11.4 User acceptance testing</i>	80
12 Conclusion	82
13 References	84
14 Appendices	86
<i>14.1 Database setup file (SQL)</i>	86
<i>14.2 CPR Decoder</i>	87
<i>14.3 Velocity over ground packet</i>	88
<i>14.4 Airspeed and Heading packet</i>	89
<i>14.5 ID packet</i>	90
<i>14.6 NL lookup table</i>	91
<i>14.7 Usability questionnaire</i>	93

3 Introduction

Airplanes need to be prevented from crashing into each other and in the past various different technologies have been used to achieve this. This technology has typically been radar, but now there is a new technology called ADS-B that involves airplanes constantly sending, in real time, position and flight information. The airplanes get their position information using GPS and they broadcast it around so anybody who is sufficiently close to the airplane can receive it. The purpose of this information is for every relevant entity, such as other planes, to be aware of where the plane is so they can prevent themselves from crashing into each other.

The information sent via ADS-B is not encrypted and is available for anybody to receive. My project is about intercepting this information, via a commercially available receiver, and visualising the planes information on a map so that it becomes useful for the general public.

3.1 ADS-B background

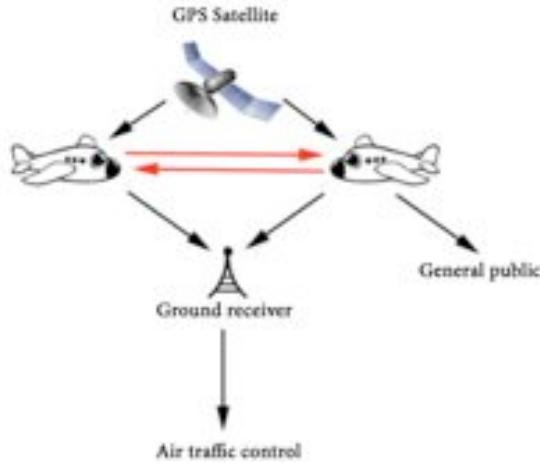
3.1.1 What is it replacing

ADS-B, mandated by the FAA (*Federal Aviation Administration*), has been created to replace radar as the primary method for managing airspaces. The type of radar that was typically used is called PSR (*Primary Surveillance Radar*) and involves a large radar dish that spins around sending out high-powered pulses and waiting for reflected responses. The reflected responses can then be used to calculate the position. The advantage that PSR has over ADS-B is that the planes don't need to be willing to provide their position and only need to be in range of the dish. However, radar cannot uniquely identify a plane and its effectiveness decreases dramatically as the range increases or the weather conditions deteriorate.

3.1.2 How does it work

The information that is needed for sending the ADS-B signals is collected using equipment that is already on-board most commercial planes. The position information is found using normal GPS technology and provides the plane with its latitude, longitude and altitude. This is enough information to uniquely locate the plane at any point on the globe and is referred to as its GNSS (*Global Navigation Satellite System*) data. The planes velocity and heading can be collected using its air speed indicator and compass. The air traffic controllers can then use this information to safely control the planes movements.

The position information is sent once every two seconds and can be collected and decoded by any capable receiver. The plane sends out the information without knowledge of its destination and can receive information from any device capable of sending ADS-B data. The picture below shows the lines of communication between each of the entities involved with ADS-B.



Each plane sends out the information, independent of the other planes, to any entity capable of receiving. The ground receiver collects the signals in real time and sends them to ATC (*Air Traffic Control*) where they are used by the controllers to manage the airspace. Each plane receives the signals from the other plane, allowing the on-board collision detection software to check for dangers and provides the pilots the same view of the airspace as air traffic control.

3.1.3 Why is it useful

- Increased efficiency
- Increased safety
- Improved visibility
- Reduces the impact on the environment

ADS-B gives ATC a clear picture of the airspace, meaning they can better manage it to allow for a shorter time between take offs and landings, without any sacrifice in safety. This reduces the time that an aircraft needs to spend in the air waiting for landing clearances and in turn decreases the time that ATC needs to spend on controlling each plane. A knock on benefit of the reduced waiting time is the elimination of wasted fuel. This drives down flight costs and reduces the environmental impact.

Each aircraft with ADS-B has a complete picture of the airspace surrounding it, something that wasn't possible with traditional radar. The collision prevention systems can use this data and the increased visibility that it provides. Unlike traditional radar, the ADS-B signals do not deteriorate as the range increases and the plane is either available or out of range. Likewise, they are very rarely affected by poor atmospheric conditions, weather and signal deterioration.

3.2 Project accomplishments

- Created a program capable of receiving and processing ADS-B transmissions
- Created decoders for each type of ADS-B packet
- Created client-server application for sending the information over the Internet
- Implemented a website and database backend
- Made a map front end available to the public
- Implemented website features to improve user experience:
 - View a flight through the pilots eyes
 - Plot a single/all flight paths for live and stored flights
 - Display airline, origin, destination and other relevant information

To receive the ADS-B transmissions I am using a commercially available receiver that doesn't perform any decoding and provides a stream of raw data. My software communicates with this device and processes the data so that the useful parts can be found and sent to the decoders. These are decoding modules that I have created from scratch and are necessary for getting the information. The decoded flight information is displayed to the user on their local machine and sent via a client-server connection to a web server that waits for incoming connections.

The web server is capable of receiving data from multiple receivers located anywhere with an Internet connection. The incoming data is used to continually update a list of all the planes that are in range of the receivers. All the incoming data is stored in a backend database. This can be queried using the website to retrieve details about flights that have already landed or to plot the flight paths for planes that are no longer visible.

The map is constantly updated as new data is received from the receivers and each plane can be clicked to present a brief summary. Users can choose to view the flight through the eyes of the pilot using a 3D representation of the world and can view all the information about a certain flight that includes the live flight data, airline information, flight information and the type of plane.

3.3 Report structure

The professional issues associated with my project have been discussed first, including ethical and legal considerations. Next, the results from the requirements analysis show information about existing systems and how my project improves upon my competitors. Following this is a more in-depth explanation of ADS-B and the technical challenges associated with decoding

the transmissions. The next chapter details the high and low level design of the software, followed by the implementation, testing and my conclusions on the finished project.

4 Professional considerations

This chapter outlines the various ethical, legal and safety considerations relevant to my project. I have referenced the BCS code of conduct, which can be found at <http://www.bcs.org/category/6030>, in matters of ethical development and also address the legal concerns surrounding the interception of ADS-B transmissions. I have also explained the security risks that ADS-B poses and how the data I collect could be used maliciously.

4.1 Ethical Issues

4.1.1 BCS Code of Conduct

I have identified clauses 1, 8, 9 and 16 to be of significant importance for my project. I have included abridged versions of the clauses below and have explained why I believe they are important in regards to my project.

4.1.1.1 Clause 1: Regard for public health, safety and environment

This is the most important clause and applies to all aspects of my project. It is imperative that I explicitly state that the data displayed on the map interface and the data stored in the database may not be 100% accurate and should not, under any circumstances, be used for safety critical applications. All data contained within, or referenced from, the web interface cannot be relied upon for any purpose other than casual curiosity and should never be used for navigation. Likewise, this data shouldn't be used to inform decisions about environmental issues such as carbon emission control and should not be used in a situation that has the potential to jeopardise the safety or security of the public or its assets.

4.1.1.2 Clause 8: Do not disclose confidential information

This clause refers to the data uploaded from the public and any information unintentionally collected. It will be explicitly stated that a condition of using the software is that the data about the planes will belong to me and I am free to manipulate and store the data without regard to the Data Protection Act. However, this will not include personal data that I would be at liberty to collect such as IP address, location of receiver and name of the participant. Where collection of this information is unavoidable, I will adhere to the Data Protection Act with regards to the storage of the information and will seek the permission of the participant before collecting and distributing such data.

4.1.1.3 Clause 9: Do not withhold information on the performance of the products

This clause ties in closely with the issues raised by Clause 1 because it will be important that I do not misrepresent the accuracy and reliability of the data and therefore present it in a way that exaggerates the realistic performance of the software.

4.1.1.4 Clause 16: Observe relevant clauses from the BCS GCP and other standards

As a registered member of the BCS it is vital that I follow all the professional standards set out by the institution. I will need to adhere to the relevant sections of the Code of Practice and implement my project in such a way that complements these and other standards set out by the BCS.

4.2 ADS-B security considerations

It is not immediately obvious what security risks ADS-B introduces but I hope to identify these risks and explain why they occur.

4.2.1 Terrorism

This is perhaps the most obvious concern for the FAA because ADS-B makes all of an aircraft's vital data freely available for anyone who wants it. This data can be used to pilot a private aircraft directly into a high value target such as a commercial airliner. It would also not be difficult to fly a model plane, containing explosives, into a commercial airliner as it comes into approach the airport (Phillips, 1999).

4.2.2 Data mining

It is conceivable for a company with an average amount of resources to install ADS-B receivers fairly near to all of their major competitors, with the majority of receivers being able to encompass multiple locations. From there it would be possible to monitor all aircraft seen landing and taking off from their competitors airfield and comparing them to known registered aircraft. This means it would be possible to identify if Company A is visiting Company B or if Company C is visiting their manufacturing plant more than often, possibly indicating problems with the manufacturing process. Data such as this could be analysed and used to inform business decisions such as stock market trading and investments, giving the potential for an unfair industry advantage.

This risk is perhaps not as prevalent or pressing as the risk from terrorism however I believe if harnessed correctly it could provide a good insight into the operations of a large corporation.

4.3 Project security considerations

Although the security implications of ADS-B are vast I do not consider them to be directly related to the service I am going to provide because it would be possible for malicious users to gather the information for themselves or use other sources available on the Internet. There is no effective way for me to guard against the security flaws of ADS-B and so these are not relevant for my project.

The normal safeguards for public websites need to be put in place. One of these is the protection of the database from accidental/malicious damage and the prevention of the uploading of deliberately false information.

4.4 Legality concerns

4.4.1 ADS-B Interception

I had some concerns initially about the legal status of ADS-B decoding in the UK due to the nature of the data I would be at liberty to collect. However, upon researching similar websites and the fact that the ICAO (*International Civilian Aviation Organisation*) technical documents needed for decoding are available online I have come to the conclusion that the ADS-B data is not protected by any laws and I am at liberty to decode and share the data as I see fit.

4.4.2 Web scraping

The web scraping I am performing is on an exceptionally small scale and does not disrupt, alter or in any way affect the normal operation of the website. In addition, I am not doing it for profit or to increase the value of any business and as the data is already publicly available, I am certain that my actions are entirely legal.

5 ADS-B technology

In this chapter I have gone into more detail about the technology behind ADS-B and how the transmissions are structured and decoded. This chapter is not necessary for understanding how my project operates but does provide some background into the general difficulties with decoding ADS-B and the associated technical challenges. The examples I have used are real and are packets that I have collected using my receiver.

5.1 ADS-B Signal Transmission

The ADS-B signals are transmitted at 1090Mhz and can be received by any device capable of receiving these frequencies. The raw data enters my software via a USB port that is acting as a virtual serial port. Due to the nature of a serial port, this means the data is going to be read 1 bit at a time from the USB port. It is significantly easier to receive ADS-B signals than it is to transmit them because transmitting requires expensive equipment. The ADS-B signals received and the signals sent are called ADS-B in and ADS-B out respectively. It is the ADS-B out that will be compulsory for aircraft whereas the ADS-B in, which provides information about other aircraft and weather etc., will be optional. The ADS-B out signals are the ones I will be decoding because they contain the useful information such as location and flight number.

5.2 ADS-B Packet Formation

Packets are just chunks of data that are sent to another entity over a network. The packets are sent in a format called hexadecimal which means the data is represented on an alphabet of 16 characters with 0000 representing ‘0’ and 1111 representing ‘f’. This therefore means that each byte of the packet, i.e. 8 bits, contains two hexadecimal characters.

5.2.1 Extended squitter packets

Each packet is either formed of 7 bytes (for Short Squitter transmissions) and 14 bytes (for ADS-B Extended Squitter messages). The interesting packets are the extended squitter messages because it is these that contain the data about the planes altitude, position, velocity etc. An extended squitter packet always has the format below with each number along the top representing a byte of data (8 bits):

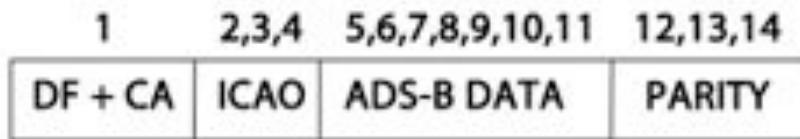


Figure 1 - Extended squitter packet

The 1st byte contains the DF (*Downlink Format*) code and CA (*Capability*). The DF number tells the decoder whether it is dealing with a short or extended squitter packet. Once the hexa-decimal characters have been converted to binary, all extended squitter packets begin with 10001, followed by 3 more bits to indicate the CA code. This means the first byte may take the form of 10001101. The first 5 bits represent the DF code, 17, and the last 3 bits indicate the CA code, 5. From the ICAO technical documents it is possible to see that a DF code of 17 represents an extended 112-bit squitter.

The ICAO number, bytes 2,3,4 in figure 1, is the unique ID assigned to every aircraft by the ICAO and is a 3-byte long hex string. Every ADS-B transmission contains this ICAO code and is used to identify which aircraft the packet originates from.

5.2.2 Decoding extended squitter packets

The flight data itself is contained within the 7 bytes following the ICAO number. The way the flight data is represented in these 7 bytes depends on the type of packet that it is. There are 4 different types of extended squitter packets.

- *Airborne Position message*: Contains the latitude, longitude and altitude
- *Surface Position message*: Contains the latitude, longitude and altitude of grounded planes
- *Airborne Velocity message*: Contains velocity and heading
- *Aircraft ID and Category message*: Contains flight number and aircraft type

Each type of packet requires a different method of decoding.

5.2.3 Breakdown of the structure of extended squitter packets

Figure 2 shows an Airborne Position packet.

8d 40 08 f1 58 37 f2 37 eb e3 a3 89 bf 53

Figure 2 - Airborne Position packet

This packet refers to the aircraft with the ICAO number “4008f1” which by using a website to lookup an aircraft based on its ICAO (AirFrames) can be identified as a British Airways G-EUPW Airbus. The part of the packet that contains the flight information can be broken down further, as shown in figure 3.

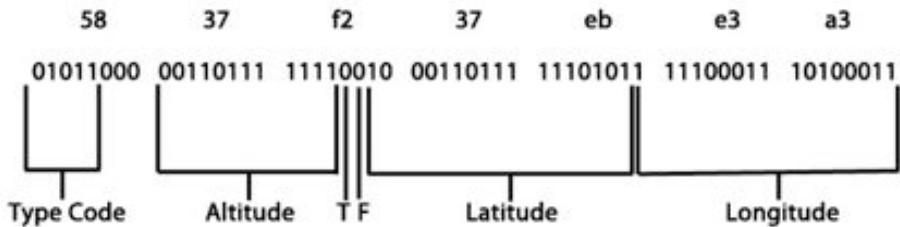


Figure 3 - Airborne Position packet data

The type code is used to further identify the type of packet being received. A type code of 11 means this is an airborne position packet. Bits 6 and 7 indicate the emergency status of the aircraft with anything other than 0 indicating an emergency on board.

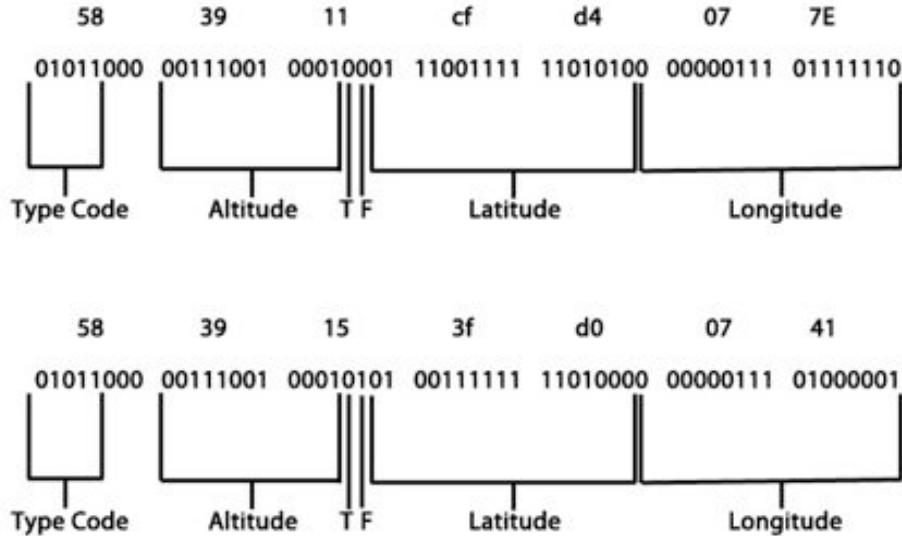
The altitude is encoded either using BA (*Barometric Altitude*) or HAE (*Height Above Ellipsoid*). The difference between the two is that the BA is measured using the current air pressure compared against a default ground air pressure, and the HAE is measured by modelling the earth as an ellipsoid and calculating the altitude based on that. These differences are not really relevant so I shall use whichever altitude the plane is transmitting. There are not enough bits to represent the altitude precisely which means each increase of 1 to the decimal value of the altitude actually represents an increase in 25 feet. This can easily be accounted for by multiplying whatever the decimal value of the altitude is by 25.

UTC (*Universal Time Coordinated*) is the standard by which the world regulates its clocks and the T bit indicates whether the packet is synced to this universal time. The F packet indicates if the packet is an *odd* or *even packet*. This is very important for decoding the actual latitude and longitude of an aircraft because both an odd and an even packet are needed. This is due to the fact that it is encoded using Compact Position Reporting (CPR).

5.3 Compact Position Report (CPR) Decoding

Below is an example of the CPR decoding process that must happen each time an odd and an even packet is received from an aircraft, as long as these packets were received less than 10 seconds apart from each other. The equations that I have used are taken straight from the official documentation (ICAO, 2008) explaining how to decode each type of packet. This means I

haven't created or modified any of the equations in this section and I am basically following the guidelines specified by the ICAO, as this is the only way to successfully decode these types of packets. Figure 6 shows the two packets that are going to be used.



The F-bit shows that the top packet is an *even* packet and the bottom is an *odd* packet. When the lat/long values are converted into their decimal forms, with $\text{lat}(0)$ and $\text{long}(0)$ being the even packet and vice versa, they can be displayed like this:

$$\text{Lat}(0) = 0111001111101010 = 59370$$

$$\text{Lat}(1) = 010011111101000 = 40936$$

$$\text{Long}(0) = 0000001110111110 = 1910$$

$$\text{Long}(1) = 00000011101000001 = 1857$$

5.3.1 Compute latitude index

The next stage is to compute something called the latitude index, or j . This number doesn't represent anything physical and is just a number used as an input to some of the equations later on. The function *floor* simply takes the largest previous Integer i.e. 8.99 would become 8. The latitude index is calculated using the formula below, with the necessary values already substituted in.

$$j = \text{floor} \left(\frac{59^{\circ}59'37.0 - 60^{\circ}10'9.6}{2^{17}} + \frac{1}{2} \right) = 8$$

$$j = \text{floor} \left(\frac{59^{\circ}59'37.0 - 60^{\circ}10'9.6}{2^{17}} + \frac{1}{2} \right) \text{floor} \left(\frac{59^{\circ}59'37.0 - 60^{\circ}10'9.6}{2^{17}} + \frac{1}{2} \right)$$

$$j = \text{floor} \left(\frac{59^{\circ}59'37.0 - 60^{\circ}10'9.6}{2^{17}} + \frac{1}{2} \right) = 8 \text{Lat}(1) = 01101001011011101 = 53981$$

5.3.2 Compute the latitude values

Once the latitude index j has been calculated, the real latitude values can be calculated. The latitude values for both packets must be calculated so that they can be compared. The exact values used for the calculation vary depending on whether it is an odd or even packet. The two calculations below show the results for the even and odd packet respectively.

$$Rlat_0 = 6 * \left(\text{MOD}(0, 59) + \frac{59370}{2^{17}} \right) = 51.71774$$

$$Rlat_1 = \frac{360}{59} * \left(\text{MOD}(0, 59) + \frac{40936}{2^{17}} \right) = 51.71922$$

The function MOD stands for modulo which finds the remainder of the division of one number by another.

5.3.3 Checking the latitude values for errors

The correct latitude value from the results above can be chosen based on whether an odd or even packet was last seen. In this case an even was last seen so Rlat0 would be chosen and the latitude set to 51.71774. However, before this value can be used, both values need to be compared to ensure that they lay in the same latitude zone. The world is split up in latitude and longitude zones and if the packets were not received while the plane was in the same latitude zone then the packets should be discarded. The latitude zones are represented on the image below by the horizontal lines.

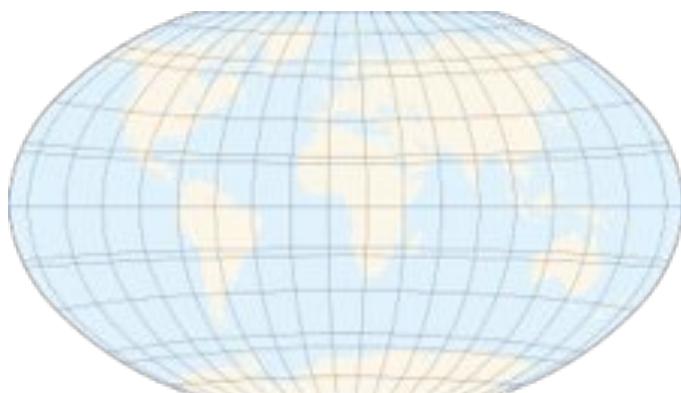


Figure 4 - Latitude zones

To check this I am using a lookup table provided in the technical documents that, when given some latitude, returns which latitude zone it lies in. I therefore pass this function the two latitude values above and check that it returns the correct result. If the result is the same then the longitude can be calculated next, but if they are different then these packets are no longer useful.

In this instance the lookup table returns 37 for both latitude values, meaning the longitude can now be calculated.

5.3.4 Preliminary calculations for longitude

Before the longitude can be calculated a couple of other things need to be calculated first. Both of these don't relate to any physical attribute but do form part of the input to the equation that calculates the longitude. The first value needed is something called dlon. The variable 'i' in the equation below is either a 0 or 1 depending on whether the last seen packet was even or odd.

$$dlon = \frac{360}{n_i} \quad \text{where } n_i = \text{greater of } [\text{Num of Lat zones} - i] \text{ and } 1$$

So in this instance 'i' is 0, making dlon equal to 360/37. This dlon value is now needed to calculate the longitude index m , similar to the latitude index j that was calculated earlier.

$$m = \text{floor} \left(\frac{1918 \times 36 - 1857 \times 37}{2^{17}} + \frac{1}{2} \right) = 0$$

5.3.5 Compute the longitude value

Using the values of dlat and m , 360/37 and 0 respectively, the longitude of the plane is now ready to be calculated as I have shown below.

$$Rlon_i = \frac{360}{37} \times (\text{MOD}(0.37) + \frac{1918}{2^{17}}) = 0.089287$$

5.3.6 CPR Decoding conclusion

The CPR decoding process is now complete. Using the latitude and longitude from the above equations the planes position can be plotted.

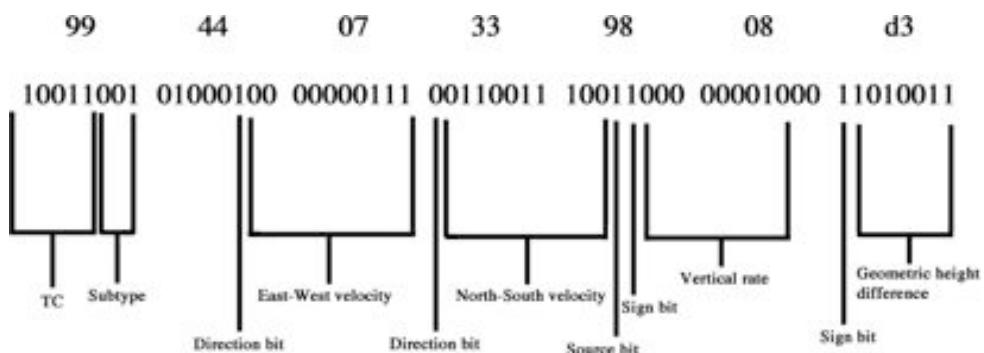


This decoding process must be repeated every time an odd and even packet is received because the planes position is constantly being updated.

5.4 Heading and Velocity decoding

To calculate the velocity and heading some basic algebra needs to be applied to the vectors that are provided in the packet. This decoding method is not listed in the technical documents and so I was required to create the decoding algorithm from scratch. In this section I am going to briefly outline this algorithm.

The velocity is provided as two separate values: one along the North-South line and the other along the East-West line. The packet contains two separate bits that indicate the direction along each line that is being represented, with a 0 on the first bit signalling East and a 1 signalling West, and a 0 on the second bit signalling North and a 1 signalling South. The packet that I'm going to be decoding is shown below.



The formal packet definition taken from the technical documentation can be seen in Appendix 3. There are two different types of Velocity/Heading packets and although the most common is the packet in Appendix 3 it was necessary for me to implement a decoder for the other packet type as well. The formal definition for this can be seen in Appendix 4. For this tutorial I

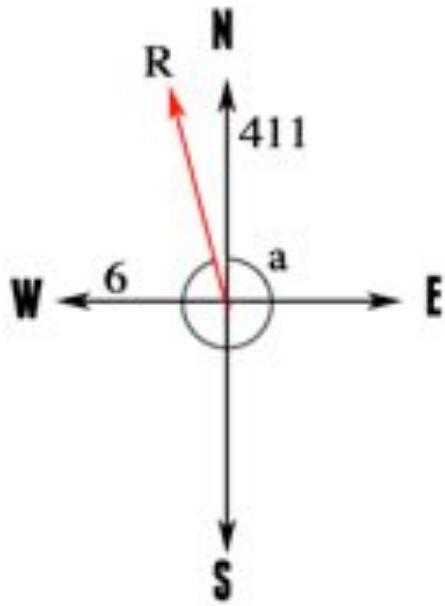
shall only be decoding the '*Extended Squitter Airborne Velocity: Velocity over ground*' packets.

The packets subtype must first be looked at before the East-West and North-South velocities can be converted. If a packets subtype has the decimal value 1 then it represents a normal speed aircraft, whereas if it has decimal value 2 then it is representing a supersonic aircraft and the calculations will need to be performed differently. In this case the packet represents a normal aircraft and so the velocities can be represented in the decimal format like so:

$$\text{East - West velocity} = 0000000111 = 6$$

$$\text{North - South velocity} = 0110011100 = 411$$

By looking at the direction bits, the direction that the velocities represent can be established. The first direction bit is a 1, i.e. west, and the second is a 0, i.e. north. This can visualised like below:



In this diagram, R represents the resultant vector that is produced when the two velocities are combined. The angle 'a' represents the heading of the aircraft in respect to north. Both r and a can be calculated to work out the heading and velocity of the aircraft.

To do this one can use vector addition. Each vector must be formed into a right angle triangle and trigonometry performed on it.

$$Ax = \text{eastWest} \times \cos(0) = 6$$

$$Ay = \text{eastWest} \times \sin(0) = 0$$

$$Rx = northSouth \times \cos(90) = 0$$

$$Ry = northSouth + \sin(90) = 411$$

The above is pretty self-explanatory. The eastWest has a magnitude of 6 along the X-axis and the northSouth has a magnitude of 411 along the Y-axis. Trigonometry can now be performed to calculate the resultant, shown below:

$$Rx = 6 - 0 = 6$$

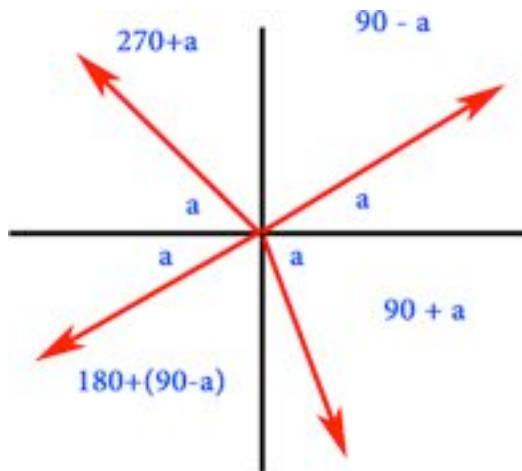
$$Ry = 0 + 411 = 411$$

$$\text{Resultant} = \sqrt{6^2 + 411^2} = 411.044$$

The above shows that the plane was travelling at a velocity of 411.044 knots. From here the heading can be calculated:

$$\theta = \tan^{-1} \frac{Ry}{Rx} = \tan^{-1} \frac{411}{6} = 89.16^\circ$$

This is obviously incorrect in its current format, however care must be taken to accommodate for the 4 ‘parts’ of the compass. The angle produced here, because it is in the 4th quadrant, must be added to 270° to give the angle in relation to north. This is because at the moment the angle has been calculated with respect to the East-West line. Therefore the final heading of the aircraft is actually 359.16°. The value that must be added or subtracted is dependant on the quadrant in which the resultant lies, as demonstrated by the diagram below:



The decoding is completed and these values can now be used to model the aircraft.

5.5 ID Decoding

In this section I shall briefly explain how to retrieve the aircrafts flight number, i.e. BAW223, from the ADS-B data. The aircrafts flight number is transmitted inside its own packet and the decoding process is relatively simple. The flight number can be at most 8 characters long.

Each valid character has a 6 bit binary representation that can be used to retrieve the relevant character from the table below, taken from the official ADS-B technical documents.

B4	B3	B2	B1	B6	0	0	1	1
				B5	0	1	0	1
0	0	0	0			P	SPACE	0
0	0	0	1		A	Q		1
0	0	1	0		B	R		2
0	0	1	1		C	S		3
0	1	0	0		D	T		4
0	1	0	1		E	U		5
0	1	1	0		F	V		6
0	1	1	1		G	W		7
1	0	0	0		H	X		8
1	0	0	1		I	Y		9
1	0	1	0		J	Z		
1	0	1	1		K			
1	1	0	0		L			
1	1	0	1		M			
1	1	1	0		N			
1	1	1	1		O			

Here is an example:

000010 000001 010111 110010 110010 110011 100000 100000

B A W 2 2 3

This means that this aircrafts current flight number is BAW223. The flight number changes a lot and it is not uncommon for the same aircraft to be seen multiple times in one day, such as a shuttle service from London to Edinburgh, each time having a different flight number.

6 Requirements analysis

6.1 Competition

Websites already exist that enable enthusiasts to track aircraft using ADS-B signals and to contribute their own data. These sites don't correlate exactly with what my project is trying to achieve because the majority are commercial ventures whose operations are financed by advertisements and product sales, meaning the range of features available to the public from these products cannot be feasibly accomplished in the time I have. However, I have used an evaluation of the features available on each of the sites to help me create my core requirements to ensure I am being competitive.

6.2 Choosing the right development tools

6.2.1 Mapping software

6.2.1.1 Why Polymaps?

- Open source API and source code
- Supports CloudMade, giving it huge flexibility in terms of themes/skins
- Designed for large datasets, something my software is likely to need
- Smooth, aesthetic interface
- Feature rich
- Javascript based, allowing easy implementation as part of my site



Figure 5 - Some of themes available

6.2.1.2 Alternatives

6.2.1.2.1 Google Maps

Advantages

- Different map views: Satellite, map and terrain.
- Very good API
- Lots of example source code to help development

Disadvantages

- At the time of writing (26/04/2012) Google Maps cannot be skinned with custom themes
- Zooming takes the user to the centre of the map, meaning the user needs to move the map when they zoom
- Doesn't stand out from the competition

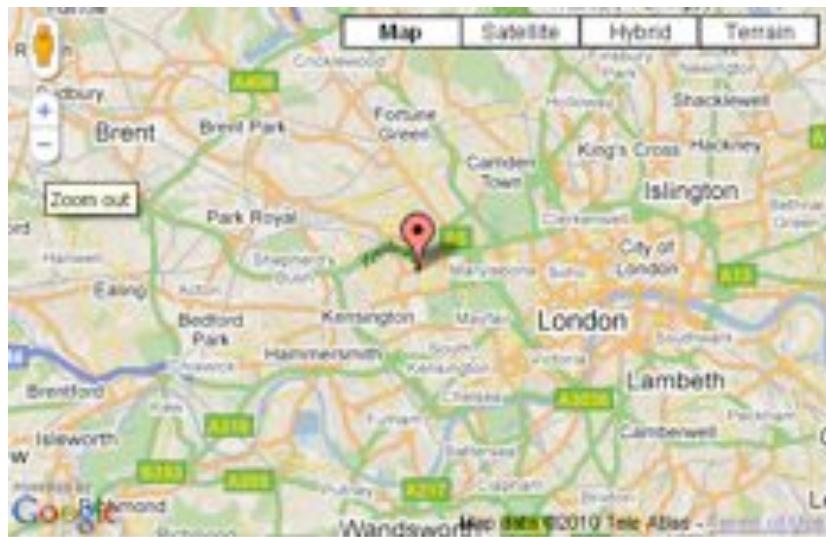


Figure 6 - Typical Google Maps interface

6.2.1.2.2 OpenLayers

Advantages

- Extensive API with lots of examples

Disadvantages

- Boring themes that cannot easily be customised



Figure 7 - Typical OpenLayers interface

6.2.2 Decoder language

Java was the obvious choice for creating the decoding software because it is the language I have the most experience in. Before I started development I made sure I could receive data from the receiver using Java and that all the other features were possible.

6.2.3 Cockpit view

6.2.3.1 Why Google Earth?

- Uses commercial satellite imagery, and thus the data is more recent
- Easy to navigate API with lots of examples
- Intuitive and users are more likely to have the GE plugin installed than its competitors

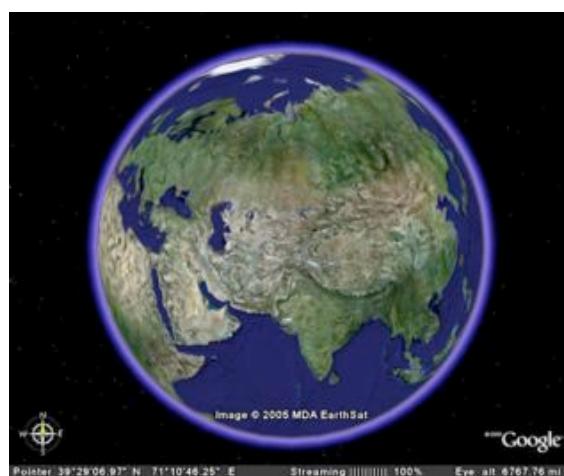


Figure 8 - Google Earth

6.2.3.2 Alternatives

6.2.3.2.1 World Wind

Advantages

- Scientific data visualizer
- Supports plugins and add-ons
- Open source and free

Disadvantages

- Less data coverage and data is out of date
- Less popular, meaning people are more likely to need to download the plugin



Figure 9 - World Wind

6.2.4 Web technology

6.2.4.1 Why AJAX?

- Supports servlet communication
- Lots of external libraries that can be easily added
- Allows page be updated dynamically without the page refreshing
- Supports XMLHttpRequest to make the transfer of data between the servlet and the web page using XML easy

6.2.4.2 Alternatives

6.2.4.2.1 PHP

Advantages

- Already know PHP so development time would have been decreased
- Can interface with the database directly

Disadvantages

- Server side execution, meaning smooth dynamic changes are harder to make
- Less supported libraries to add to make the development easier

6.2.5 Database technology

6.2.5.1 Why MySQL?

- Faster than most of the competition
- Designed for use with web-based servers
- Already have experience dealing with it so development time is decreased
- Supports Windows which is the operating system I am using for the web server
- Provides a native Java database connection to allow me to modify the database programmatically
- Database design is simple

6.2.5.2 Alternatives

6.2.5.2.1 PostgreSQL

Advantages

- Allows complex database design
- Supports complex business rules
- More functionality than MySQL

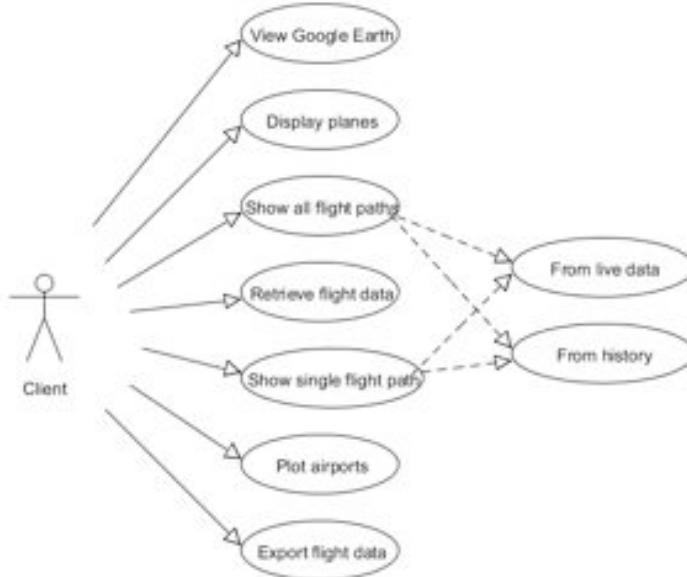
Disadvantages

- The additional features degrade performance
- Not the most efficient choice for a web-server database

6.3 Use cases

6.3.1 High level use case

Each of the activities featured in the use case below are explained in more detail in the following section.



6.3.2 Use case scenarios

6.3.2.1 View Google Earth

Actor: Client

Goal: View a flight through the eyes of a pilot using Google Earth to visualise the view

Preconditions: Plane is in range and is updating its position at acceptable time intervals

Trigger: Client selects a flight to view in Google Earth

Scenario:

1. Client selects a flight from the map to view from the cockpit
2. Server provides the latest position information for that plane
3. The Google Earth view is updated accordingly
4. Repeat from stage 2 until the plane is out of range or has landed

Exception:

1. Client tries to view the cockpit for a flight that doesn't exist: View does not change from default
2. The plane has gone out of range or has landed: Website returns to the previous page

6.3.2.2 Display planes

Actor: Client

Goal: Draw the planes visually onto the map

Preconditions: At least one plane is in range of a receiver

Trigger: Client loads the default page, or chooses to display the planes manually

Scenario:

1. The page is loaded or the client manually chooses to display the planes
2. The latest position data for all the planes is retrieved from the server
3. Each plane is added to the map, formatted and the table updated with this new data
4. Repeat from step 2 until the client navigates away or manually removes the planes

Exception:

1. No planes within range: Nothing is plotted and the table is empty
2. Plane flies out of range or lands: Plane is removed from the map and table

6.3.2.3 Show live flight paths

Actor: Client

Goal: Display either one or all of the flight paths for the planes that are currently visible

Preconditions: At least one plane is in range of a receiver and there have been at least two positions updates for that plane

Trigger: Client wishes to display a flight path

Scenario:

1. Client selects a plane from the map or chooses to display all the flight paths
2. The available flight data is retrieved from the server
3. Each waypoint for each plane is used to construct a flight path that is added to the map and formatted
4. Repeats from step 2 as new data becomes available

Exceptions:

1. No visible planes: Nothing is added to the map
2. Each plane has only had one position update: Nothing is added to the map
3. Plane flies out of range or lands: Flight path is removed

6.3.2.4 Show old flight paths

Actor: Client

Goal: Display a single or all of the flight paths from flights that have been seen previously

Preconditions: At least one plane must have been seen whilst the server has been in operation and it needs to have had at least two position updates

Trigger: Client wishes to display a flight path

Scenario:

1. Client enters a flight number of the flight they want the path displayed for, or they choose to display all flight paths
2. The necessary information is retrieved from the server that queries the database for this information
3. Each waypoint is used to construct the flight path and is added to the map and formatted

Exceptions:

1. No planes have been recorded ever: Nothing is added to the map
2. Only one position update for each plane: Nothing is added to the map

6.3.2.5 Plot airports

Actor: Client

Goal: Plot the airports onto the map

Preconditions: None

Trigger: Client wishes to display the airports

Scenario:

1. Client chooses to display the airports
2. The airport data is retrieved from the server that queries the database for the information
3. The airports are added to the map and formatted

6.4 End users

The users of my website are typically going to be aircraft enthusiasts who have at least a basic understanding of what ADS-B is and would be comfortable navigating their way around the Internet. It is likely to be used either by people who are interested in aircraft in their area and who don't have access to ADS-B equipment or by people who own an ADS-B receiver and require an interface to utilise their data and share it with friends.

My software is ideal for this group of users because it is very simple for them to upload their own data to the website and easy for them to interact with and share the data. The decoder's is so simple because it is a platform independent, stand-alone program that doesn't require any installation and can be up and running with the bare minimum of customisation. The online map interface is intuitive to use and has all the controls from Google Maps that they are probably already competent with.

7 Requirements specification

7.1 Functional requirements

The core features listed in the section below form the specification for the project. All of the features need to be implemented for the website to be successful and for it to offer the necessary functionality. I have included a section on work that, if time permits, I will also implement. These features are typically large and time consuming and as such are not part of my specification.

7.1.1 Core features – Decoder

- Decode ADS-B transmissions correctly
- Display live air traffic in real time in a GUI on the clients local machine
- Decode enough information to accurately represent the planes on a map
- Make the software stable enough to run unattended 24/7

7.1.2 Core features - Website

- Allow multiple receivers to be used to increase the coverage
- Have a display on the homepage of the last x-number of planes to be added to the map
- Have a live twitter feed on the homepage linked to the account PlaneTrackerUK to provide visitors with bug/update information
- Be publically accessible from the URL <http://www.planetracker.co.uk/>
- Display a map with all the visible aircraft on it and have their positions updated automatically without the page refreshing
- Allow each plane to be clicked to bring up a summary of the flight that includes the most relevant information
- Allow the client to plot a flight path for a single flight
- Allow the client to plot the flight paths for all the visible planes
- Allow the client to view a flight through the eyes of the pilot using Google Earth
- Allow the client to plot all the airports in their country on the map
- Allow the airports to be clicked to bring up information about it such as its name and location
- Provide a table along with the map to display a live text display of all the flights
- Allow a flight to be clicked from the table to bring the client to a page that displays all the information available about that flight
- Provide a page that provides the client with all the information available for the given flight
- Provide a picture of the plane and the airline logo to the client
- Provide information about the flights origin and destination
- Provide a page where the users can create queries to retrieve information from the database
- Allow clients to plot a flight path from the database using one of the flights unique ID's
- Allow the client to view all the flight paths for planes stored in the database
- Give the client the ability to export the raw data from the database so it can be used for other applications

7.1.3 Extended features

These are original features that I haven't seen on any of my competitor's sites and although they are not part of my specification I would like to implement them if time permits.

- Show the various protected airspaces in the country
- Tell the client what each plane is currently flying over i.e. “<20NM from Big Ben”
- Allow clients to be notified when flights they are waiting for are added to the map
- Replay the air traffic from any time period in the past

7.1.3.1 Protected airspaces

Over certain places in the country there are things called protected airspaces. These are not physically tangible but represent an area that is off-limits to all aircraft without permission from a certain authority. An example of a protected airspace is the sky surrounding Gatwick Airport. Aircraft are not allowed to enter this space without permission from the ATC tower at the airport and this so they can safely bring incoming flights in to land without interference from smaller private aircraft.

7.1.3.2 ‘Flying Over’ Query

One feature I would really like to implement is the ability to bring up a list of high value targets that a select aircraft is flying near. For example, if a plane flying over London is selected then it may bring up a list similar to: “<10NM Buckingham Palace. <15NM Houses of Parliament”. Although this could be used for security purposes, it is more designed as a feature of interest because it is likely a dangerous plane would not be transmitting their ADS-data.

7.1.3.3 Specific aircraft notification service

I would ideally like to provide a service to the users that allows them to input a flight number to the website and it sends them an email when a plane matching that flight number is seen. This could be used to track their relative's planes or to follow the specific flying habits of a particular plane.

7.1.3.4 Replay previous traffic

A feature that would be useful for data collection purposes is the ability to replay the traffic stored inside the database during time frames specified by the user. This would mean a user could request to see the flights from 3 days ago between 13:00-14:00 and the map would display the flights as if it were in real-time.

7.2 Non-functional requirements

These are the requirements that cannot be quantified in terms of yes they are there or no they are missing. I feel these features are necessary for my project to be competitive and for the project to be fun for use by the clients.

- Website should be easy to navigate and unambiguous
- Data should be accessible in an intuitive fashion
- The decoding software should be platform independent and easy to use

8 High level design

8.1 Design methodology

I have used the agile method for my project because at the beginning I didn't have the full specification and as I explored my options the project aims changed. There was a great deal of uncertainty heading into the creation of the decoding software because it was a completely new area for me and so it would have been inappropriate to use a waterfall based methodology.

My project naturally lends itself to an agile style approach because it is very modular in nature and this allows for components to easily be tested at the end of the iteration. I made an effort to keep the iterations to a maximum of 4 weeks in length, in keeping with the agile methodology.

8.2 Decoder overview

The decoder has been designed to be a platform-independent, self-contained program that can be executed by clients who have downloaded it. Its purpose is to receive, process, decode and send the data to a remote web server and also to provide a simple local display of the aircraft traffic.

8.3 Web page overview

The purpose of the web page and its associated underlying server technology is to provide an interface for the public so that they can browse and interact with the live aircraft data. More specifically it allows users to upload their own data using my software and then interact with this data online. The web page should also provide extra information that wouldn't normally be automatically available to them from the ADS-B data alone.

8.4 Database overview

The purpose of the database is to record all the flight data so that machine learning and data mining can be performed on it. This information is used to plot the flight paths for flights that have already landed and can be downloaded by the public.

It also contains airline and airport information that is used to provide extra information to users of the website.

8.5 Abstract system overview

Below is an abstract representation of how each component interacts with each other to form the complete project.

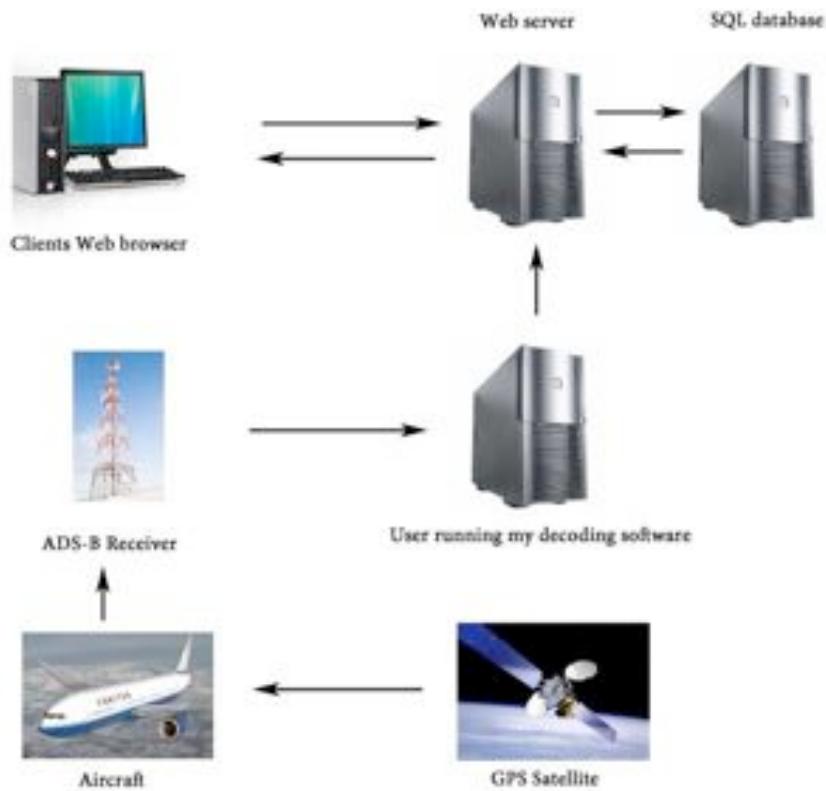


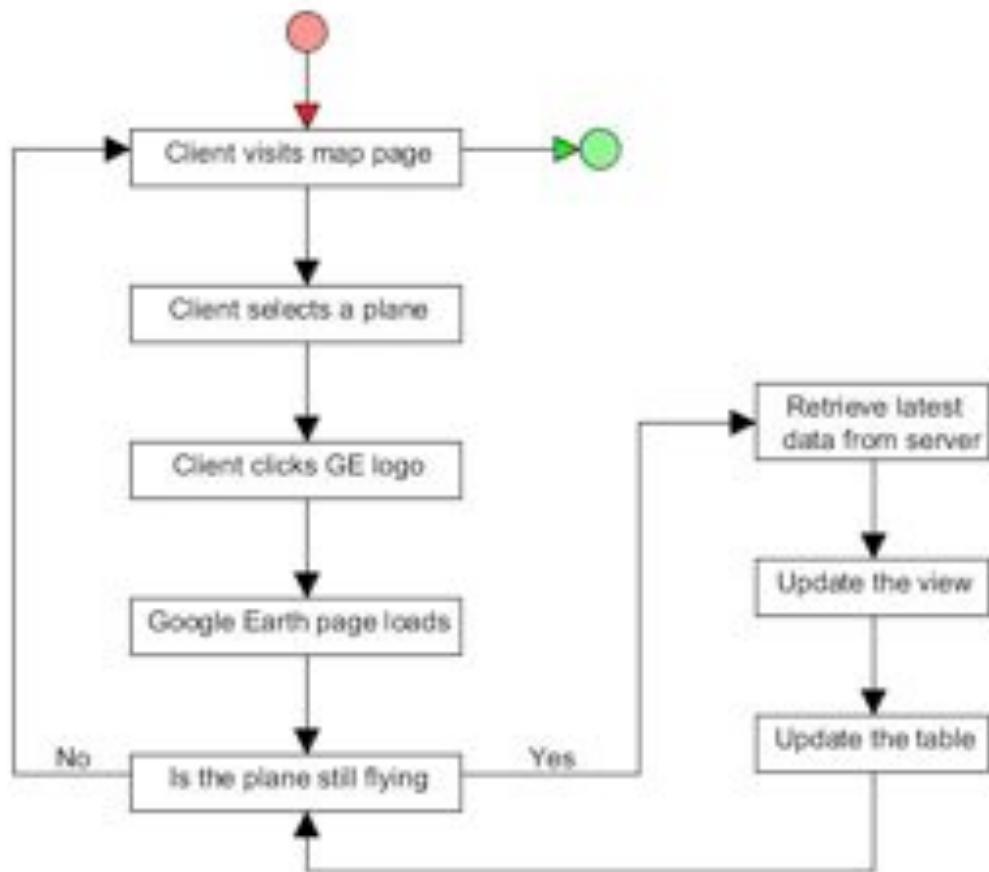
Figure 2 - System design

8.6 What is Plane Tracker?

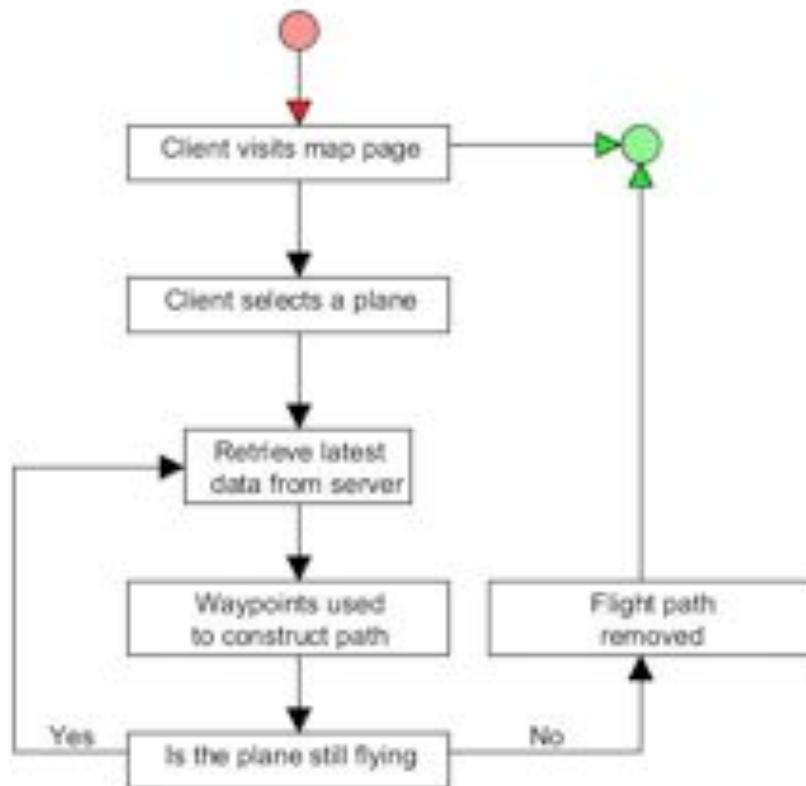
Plane Tracker was a name for the project that I thought of myself. I registered the domain <http://www.planetracker.co.uk> to provide a point of access for the public. I thought of various names before Plane Tracker but based my decision on which domains were available to buy.

8.7 Activity diagrams

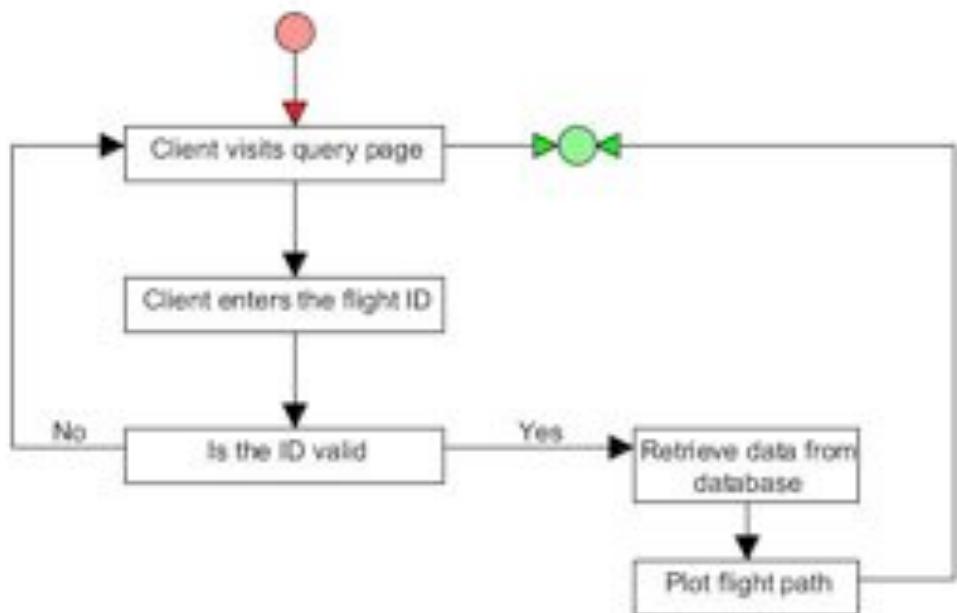
8.7.1 View flight in Google Earth



8.7.2 Plot a live flight path



8.7.3 Plot a flight path from history



9 Low level design

9.1 Receiver

9.1.1 Introduction

The decoder relies upon the data received from the ADS-B device plugged into the USB port. This data is sent from the device in a serial format where it is read into an input buffer that can then be accessed using my software. As the software is reliant upon live data, the decoding process cannot be allowed to ‘slow down’ the execution loop, causing data to be lost from the serial port when the input buffer reaches capacity and starts discarding data. To prevent this my software needs to be very efficient and utilise multi-threading where applicable.

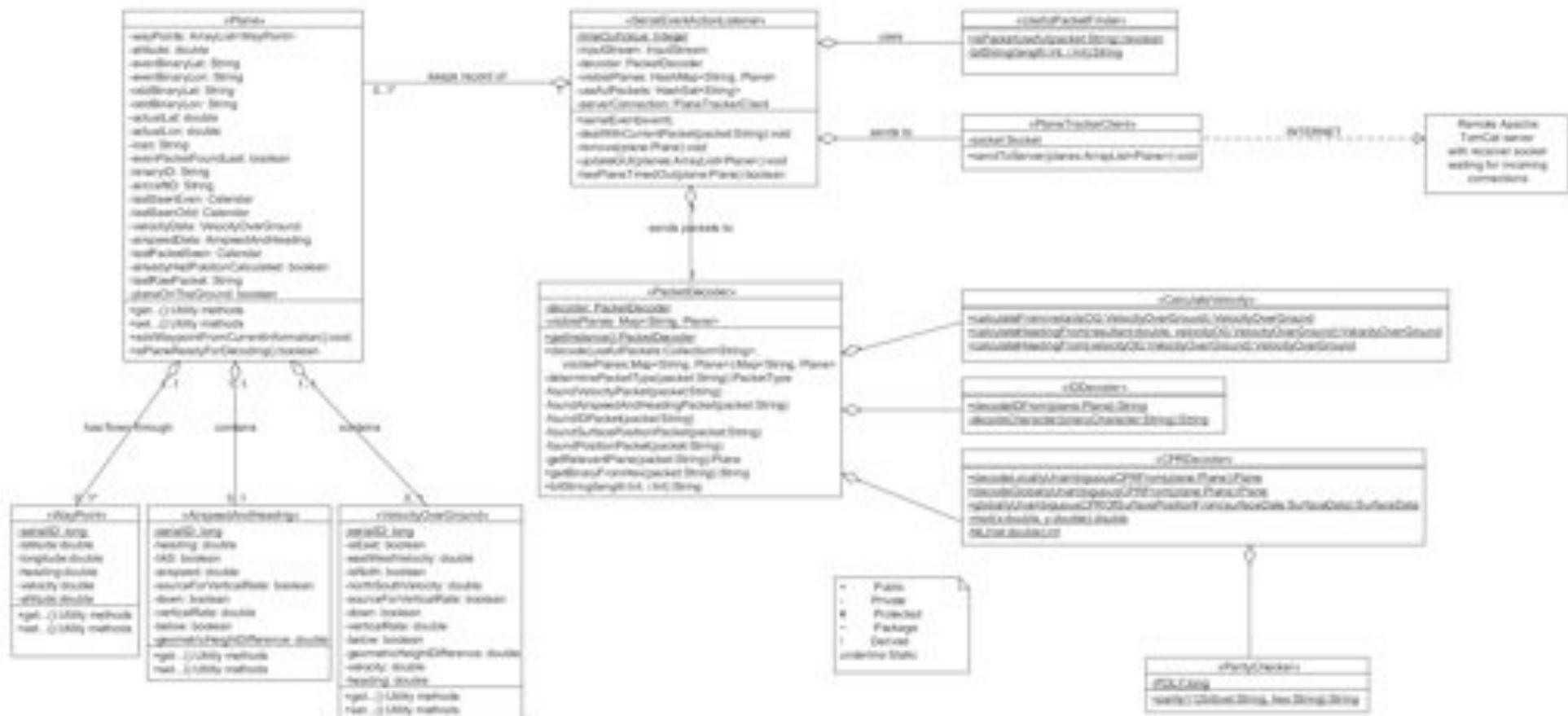
In keeping with the agile nature of this project I designed the software in such a way that features could be ‘bolted on’ at any time throughout the development as my requirements developed. The greatest hurdle I had to overcome before beginning the design was understanding the lengthy technical documents that, although were in the public domain, did not allow themselves to be easily converted into decoding algorithms. Each decoding algorithm only needs a maximum of 2 packets as input to perform the decoding, meaning each method could be designed as a separate module. This allowed me to develop and test each module independently and ‘bolt on’ to the main software when they were ready.

The following sections explain the implementation details of the decoding software and describe the operation of the system using a variety of UML diagrams.

9.1.2 File system layout

- Source
 - Decoders
 - CalculateVelocity – Calculates velocity and heading
 - CPRDecoder – Calculates position and altitude
 - IDDecoder – Decodes flight number
 - PacketDecoder – Passes the packets to the correct decoding modules
 - Accessory classes
 - ParityChecker – Checks packet is not corrupt
 - PacketType – Represents the type of packet i.e. ID packet
 - UsefulPacketFinder – Filters out the useful packets to send to the packet decoder
 - Action Listeners
 - SerialEventActionListener – Processes the data coming from the receiver
 - ServerActionListener – Used to start the web server on the local machine
 - StartActionListener – Starts the decoding
 - StopActionListener – Stops the program
 - Client
 - PlaneTrackerClient – Sends the latest planes to the server

- Wrapper classes
 - AirspeedAndHeading – Represents airspeed and heading data
 - ReceiverLocation – Stores the location of the receiver
 - VelocityOverGround – Represents the velocity over ground data
 - WayPoint – Represents a single position update
 - Plane – Represents a plane and contains the other wrapper classes
 - SurfaceData – Represents the position information when the plane is on the ground
- Test classes
 - TestCalculateVelocity – Unit tests for CalculateVelocity
 - TestCPRDecoder – Unit tests for CPRDecoder
 - TestIDDecoder – Unit tests for IDDecoder
 - TestPacketDecoder – Unit tests for PacketDecoder
 - TestUsefulPacketFinder – Unit tests for UsefulPacketFinder
- External libraries
 - RXTXcomm.jar – Assists with JAXB
 - JAXB2_20110601.jar – Allows communication with the receiver
 - Junit-4.10.jar – Gives the ability to run automated unit tests



9.1.3 Software architecture

9.1.3.1 How I model a plane

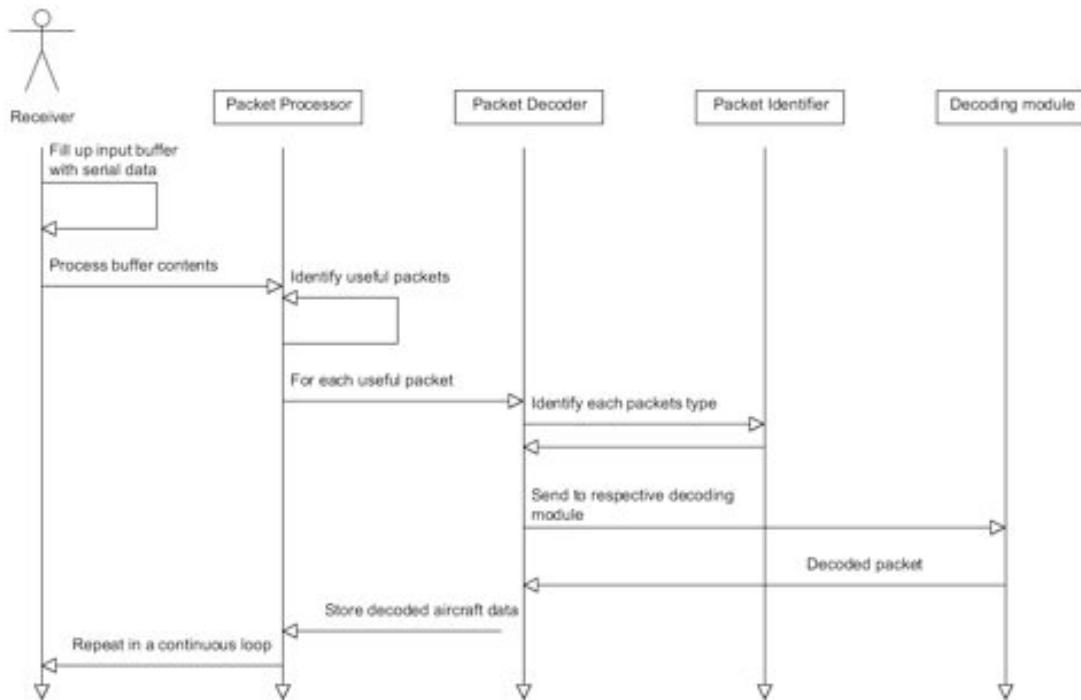
The left of the diagram shows the classes that represent a plane: *Plane*, *Waypoint*, *AirspeedAndHeading* and *VelocityOverGround*. Each one is a wrapper class containing the necessary variables with getters and setters for each one. Each time a planes position is updated, a Waypoint is created and added to the Plane. These waypoints can then be used to populate the database and plot flight paths. The AirspeedAndHeading and VelocityOverGround both represent the heading and velocity of the plane and are different methods for expressing the same thing. They require different decoding methods and, although VelocityOverGround is almost always the one that is transmitted, it was necessary for me to implement both. Although unlikely, a plane may also switch transmission method mid flight if needed, meaning it must be checked every time a packet is found. All 4 classes extend Serializable to allow them to be serialized and sent as an object via a Java socket.

9.1.3.2 How I identify useful packets

The UsefulPacketFinder class is responsible for identifying the packets that can be used for decoding. It checks the DF code and performs a parity check on the packets to ensure they have not been corrupted. It contains an accessory method for converting an Integer into its respective binary string (Neely, 2009). I did not write this method or any of the other occurrences of this method in my software.

9.1.4 Software operation

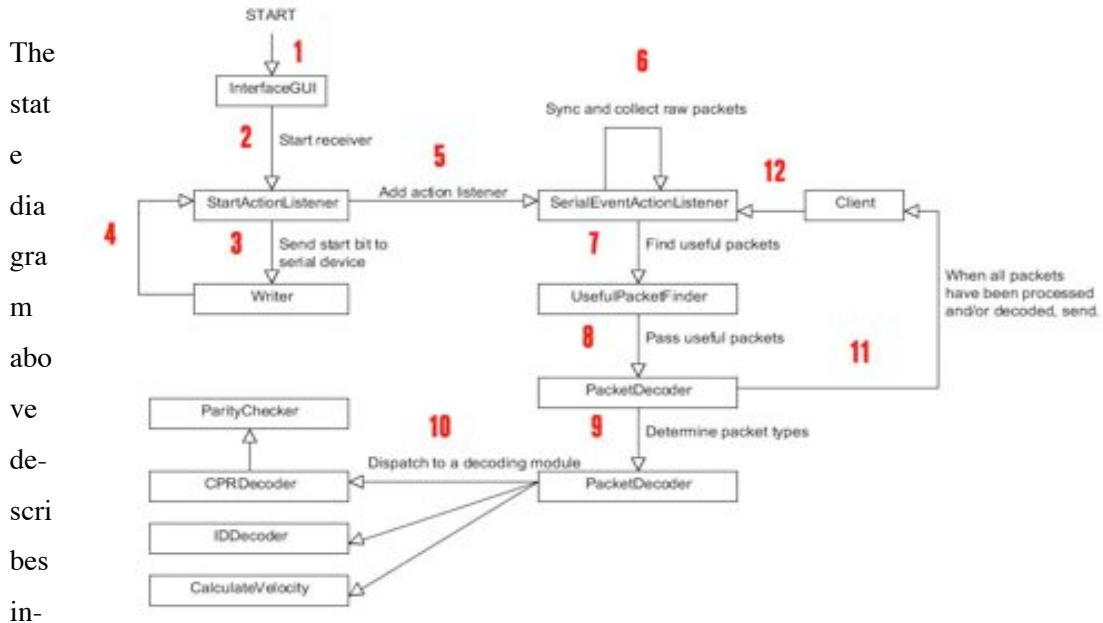
9.1.4.1 Sequence diagram



The sequence diagram above shows the structure of the software. It has a single thread of execution with the majority of the work being done in the respective decoding modules. The following decoding modules are essential: *CPR, Flight Number, Altitude, Heading, Velocity and Airspeed*.

One action that is not represented on this diagram is the sending of the data to the web server. It is important that the client be multi-threaded because if there is a large amount of data to send or if the server is highly congested with incoming data then the sending process has the potential to delay the thread, possibly causing data from the serial port to be lost. To ensure this doesn't happen I shall send the data to the client each time the loop completes. This will start another thread to handle the sending, whilst the main thread of execution continues to wait for data to arrive from the device.

9.1.4.2 State diagram



State change	Explanation
1	User executes the standalone .jar file. This starts up the user interface, InterfaceGUI.
2	When 'Start reader' is clicked the StartActionListener onClick method is triggered.
3	Action listener instructs the Write class to write a bit string to the receiver to start the transmission of serial data via the virtual COMM port.
4	Writer completes its action and returns to the action listener.
5	An action listener, SerialEventActionListener, is added to the serial port and awaits incoming data.
6	When data becomes available the actionEvent is triggered and the data is read into a byte array acting as a buffer.
7	When all the data is inside the buffer, it is iterated over to sync with the data stream and identify individual packets. Each packet is then sent to the UsefulPacketFinder and returns a Boolean value.
8	The packets identified as being useful are then collated into a Java Collection of Strings and passed to the PacketDecoder.
9	Before a packet can be decoded its type must be determined i.e. position, velocity data etc.
10	The individual packet, represented as a String is then sent to its respective decoding module where the majority of the computation occurs.
11	Once all packets have been decoded they are then sent to the client where they can be sent to the server.
12	The thread of execution then waits until more data becomes available from the serial port and begins again from step 6.

9.1.5 Decoding modules

9.1.5.1 CPR

This module was by far the most troublesome module to implement. The greatest difficulty I faced, as I did to a lesser extent with some of the other modules, was bit manipulation and the management of Java types. This was the first module I wrote which meant a lot of the mistakes I made initially I was careful not to reproduce. However I repeatedly ran into the problem of Java rounding values up or down unless I explicitly declared every number to be a double. Although this problem is obvious with hindsight, I feel a CPR decoder would have been a lot simpler to implement in a language designed for mathematical operations.

The source code for the CPR decoder can be found in Appendix 2. This is the code for the function `decodeGloballyUnambiguousCPRFrom(Plane planes)` and is only 1 of the 3 CPR functions. I made extensive use of the Math class for all 3 functions and tested the decoder vigorously using TDD (*Test Driven Development*). This meant I wrote JUnit tests that I knew should work before I started writing the decoder, allowing me to test it at every stage of the development. More information on TDD can be found in the testing chapter.

The latitude zone lookup table was implemented as a separate static method that takes latitude as input and returns an Integer representing the number of latitude zones. Within this method is a series of switch-case statements that looks at the value of the latitude and compares it with hard-coded values until it finds a match. Switch-case statements were more suitable for this task than if-then-else statements because they required less code and are more human-readable.

9.1.5.2 Heading and Velocity

Very little documentation exists on how to convert the raw decimal values for the heading and velocity into their true values. This meant I needed to create the algorithm from scratch. I followed the procedures outlined in the design chapter and converted the mathematical statements into their equivalent Java statements. Like the other modules, all the methods were static so that they can be accessed without an object of that class. Although this is not essential it is good practice because another component of the software may wish to calculate the velocity without having a respective Plane object.

9.1.5.3 Flight number

This was the simplest module to implement. First, each 6-bit character is converted into its decimal value. This is then passed to a method acting as a lookup table and its corresponding

character is returned. Each of these characters can then be concatenated to form the full flight number and the decoding is complete.

9.1.5.4 Parity checker

The last 3 bytes of a packet contain the parity data that can be used to check the integrity of the received packet. I researched how to implement a parity checker and could not find any documentation on how the algorithm was supposed to work. I discovered a ready-made parity checker (Coupe, 2011) already implemented in Java that would fit my needs perfectly. I implemented it within my software and it worked very well in filtering out the erroneous packets I had been receiving. I did not write any of the code within the ParityChecker and copied the source code from the public domain.

9.1.6 Physical design

I am using a commercially available receiver purchased from www.radargadgets.com/ to receive the ADS-B signals.



Figure 12 - ADS-B receiver

To be in the best position for receiving the ADS-B packets the aerial needs to be as high as possible. It should have no physical obstruction and also be placed upon a ground plane. This ground plane serves as an earth plane for the mobile antenna, boosting the range significantly.



Figure 13 - Antenna and ground plane



Figure 14 - 24/7 decoding machine



Figure 15 - Full ADS-B setup

In the setup above I have set the receiver up in the loft space. This bungalow sits in a relatively flat area and so by placing it at the highest point in the building I have managed to achieve a large receiving range that encompasses Gatwick, Luton, Heathrow and Stansted. My receiver software runs on a netbook 24/7 and continually tries to send the data to the URL planetracker.co.uk. This is the domain that I have registered and is also the point of access for the public website. The TomCat server is run on my personal laptop and so naturally the IP address of it changes regularly. To counter this I am using a free DNS routing service that continually up-

tinually updates the IP address that the URL points to and have setup my router to allow all incoming HTTP requests to be forwarded to the virtual machine that is running the server. Running the server in a virtual machine also has the added benefit of slightly increased security. The setup above is for when the receiver is placed in Hampshire. The Brighton setup can be seen below.



Figure 16 - Brighton receiver



Figure 17 - Increased field of view

Due to the added height of the receiver and the increased field of the vision, the range has increased dramatically. The receiver still encompasses the 4 major London airports, but also reaches the north coast of France and a significant portion of the English Channel. If I had two receivers I would like to have placed one in each location so that I could extend the range even further.

I created a basic GUI, seen below, for the user to control the software. No thought has been given to the aesthetics and is designed with just the functionality in mind.

ICAO	Latitude	Longitude	Altitude	Reg	Flight	Heading	Speed	Type	Route	Squawk	Time
4008e8	51.5387878417	-0.0485992431	32000.0	N/A		103.471917595	467.8739	N/A	N/A	N/A	Thu Apr 05 14:1...
4ca9c4	51.5640106201	-0.02998352050	28000.0	N/A		254.818581987	330.6289	N/A	N/A	N/A	Thu Apr 05 14:1...
4ca6c4	51.5095825195	-0.9125518798	4400.0	N/A		99.4126294368	189.5521	N/A	N/A	N/A	Thu Apr 05 14:1...
4008b0	51.8712546484	-1.1839294433	15025.0	N/A	SHT2961	149.924270392	365.1643	N/A	N/A	N/A	Thu Apr 05 14:1...
400935	51.3197937011	-0.3856656935	7000.0	N/A	BAW345	277.796991110	243.2488	N/A	N/A	N/A	Thu Apr 05 14:1...
400771	51.1199798583	-0.8363713444	13975.0	N/A		248.166419950	330.7234	N/A	N/A	N/A	Thu Apr 05 14:1...
4004c4	51.2563018798	-0.3989967139	8025.0	N/A	BAW242	304.009795119	248.5095	N/A	N/A	N/A	Thu Apr 05 14:1...
4009da	51.3252410838	-0.4566007047	5275.0	N/A		256.905941941	222.3240	N/A	N/A	N/A	Thu Apr 05 14:1...

Figure 18 - Decoder GUI

9.2 Database

I am using MySQL v5.5.22 with MySQL Workbench v5.2 as the DBMS. I have chosen MySQL because it is the language I am most comfortable with and I chose the MySQL workbench because it is widely supported and the most logical choice.

9.2.1 Database structure

9.2.1.1 Conceptual schema

Each **PLANE** must have *flown through* one or more **WAYPOINTS**

Each **WAYPOINT** must have been *flown through* by one and only one **PLANE**

Plane (Icao, Flight number, Destination, Departed)

Waypoint (Id, Icao, Lat, Lon, Heading, Altitude, Velocity, Time)

Airline (Id, Iata, Icao, Name Callsign, Country, Comment)

Airport (Id, Name, Location, Country, Code)

9.2.1.2 Physical schema

Table	Columns	Data type	Notes
Plane	Icao	VARCHAR(6)	Composite key, Not null
	Flight Number	VARCHAR(10)	Composite key
	Destination	VARCHAR(30)	
	Departed	VARCHAR(30)	
Waypoint	Id	Int	Primary key, Auto increment
	Icao	VARCHAR(6)	Not null
	Lat	VARCHAR(40)	Not null
	Lon	VARCHAR(40)	Not null
	Heading	DOUBLE	
	Altitude	DOUBLE	
	Velocity	DOUBLE	
	Time	TIMESTAMP	Not null
Airline	Id	Int	Primary key, Auto increment
	Iata	VARCHAR(3)	
	Icao	VARCHAR(6)	
	Name	VARCHAR(100)	
	Callsign	VARCHAR(100)	
	Country	VARCHAR(100)	
	Comment	VARCHAR(300)	
Airport	Id	Int	Primary key, auto increment
	Name	VARCHAR(100)	
	Location	VARCHAR(100)	
	Country	VARCHAR(100)	
	Code	VARCHAR(6)	
	Full code	VARCHAR(6)	
	Lat	VARCHAR(40)	
	Lon	VARCHAR(40)	

9.2.1.3 Implementation details

The SQL for the database setup can be found in Appendix 1 and it has been created using the schemas above.

Each Waypoint is identified using the primary key field ID. This is necessary because a record may differ from another in only one column, meaning a composite primary key would need to be created from all columns in the table which isn't necessary. The ICAO field can be used to match up Waypoint records with their respective Plane entities. To recreate the captured flight the Waypoint ID can be ordered to give the correct sequence of events.

TIMESTAMP has been chosen to represent the Waypoint time because it correlates with the TimeStamp class in Java, making converting between the SQL-type and the Java-type easier.

9.2.2 Database interaction

MySQL provide a native Java driver that can act as a DMS and allow standard CRUD functionality (*Create, Read, Update, Delete*). Using this library allows me to create standard SQL queries, represented as Strings, and execute them. The majority of the time I have used PreparedStatements to represent my queries. These take the form:

```
PreparedStatement prest = conn.prepareStatement("insert into planetracker.waypoint values(?, ?, ?, ?, ?, ?, ?, ?);");
```

This allows me to inject values into the statement and then execute it. SELECT queries can be performed in a similar fashion and their results are stored in a ResultSet:

```
ResultSet rs = stmt.executeQuery("SELECT icao, flight_number FROM planetracker.plane");
```

The result set can then be iterated over and each column value extracted using its name.

9.2.3 Airport data source

To get the airport information I was initially using a web service that when queried via a Javascript XMLHttpRequest object would return an XML representation of the data. This could then be transformed into usable map data. This approach worked successfully and I was able to implement it into my site. However, I noticed the data is incomplete and occasionally incorrect, meaning I needed to find a different source for the data.

I found another source that seemed more complete and did not share the same errors as the previous source. Additionally, this data came in a well-formatted text file, meaning it could easily be parsed using Java and added into my own database. To accomplish this I examined its format to identify how I could traverse it automatically and extract the relevant information. I then wrote a stand-alone Java application that parsed the text file line by line and added the

data to my database. This process only needs to be performed one. This approach has the advantage over using the web service because it does not rely upon the availability of the web service and allows me more control over the data that I can provide.

9.2.4 Airline data source

A similar approach was needed to collect data about the worlds airlines because although the data was only available online in a web format. After researching various websites I identified Wikipedia to be the easiest to mine the data from because the data was represented in a HTML table. This allowed me to get the raw HTML for the page and copy the relevant table data into a text document. I then used the same method as I did for the airport data and parsed the data from the text document and updated the database with each airline entity. Having this data readily available was essential for many of the websites features.

9.3 Website and web server

9.3.1 Introduction

The purpose of the server side software is three-fold: communicate with multiple receivers; serve client requests for data and interface with a SQL database. The web platform I have chosen is Apache TomCat, integrated with Eclipse Indigo. Using Eclipse allows all aspects of the project to be brought together inside one package and makes deployment of the application to the web server seamless.

9.3.2 File system layout

- Java resources
 - AddAirlineToDatabase - Reads the airline text file and adds each one to the database
 - AddAirportsToDatabase – Reads the airports text file and adds each one to the database
 - Airports.txt
 - Airlines.txt
 - DatabaseConnect – Updates the database with the latest plane data
 - DatabaseConnectFlightPaths
 - DatabaseExport
 - FullFlightInfo
 - GetAirports
 - GetAllFlightPaths
 - GetFlightPath
 - GetPlanes
 - LatestData – Updates and provides access to the central store of the currently visible planes
 - De-serialize classes– Necessary for reconstructing the objects sent from the clients. Must be identical to the receiver classes
 - Plane
 - SurfaceData
 - VelocityOverGround
 - WayPoint
 - AirspeedAndHeading
 - WebScrapers
 - WebScraper
 - WebScraperPost
 - WriteToFile
- WebContent
 - Javascript – Contains the many scripts needed for the site to function
 - Libraries – Contains the files necessary for the external libraries I've used to function
 - Fonts – Contains the various fonts needed for the site
 - Icons – Contains the icons
 - Images – Contains the images needed for various pages
 - Styles – CSS styles used for the map etc.
 - *Index.jsp*
 - *Planes.jsp*
 - *Data.jsp*

- *Earth.jsp*
- *Contact.jsp*
- *Single.jsp*
- *Fullinfo.jsp*
- External JARs
 - Mysql-connector-java-5.1.18 – Needed to connect to the back end database

9.3.3 Servlet layout and purpose

All the servlets use XML to send the response back to the client

- *DatabaseConnectFlightPaths.java*: This servlet provides either one or all of the flight paths from the database.
- *DatabaseExport.java*: This servlet creates a text file based on the data stored in the database.
- *FullFlightInfo.java*: This servlet provides all the information needed to populate the full info page.
- *GetAirports.java*: Provides the airport data so they can plotted onto the map
- *GetAllFlightPaths.java*: Retrieves all of the flight paths for the currently visible planes
- *GetFlightPath.java*: Retrieves a single flight path for a plane that is currently visible
- *GetPlanes.java*: Provides the data about all the currently visible planes, allowing them to plotted on the map

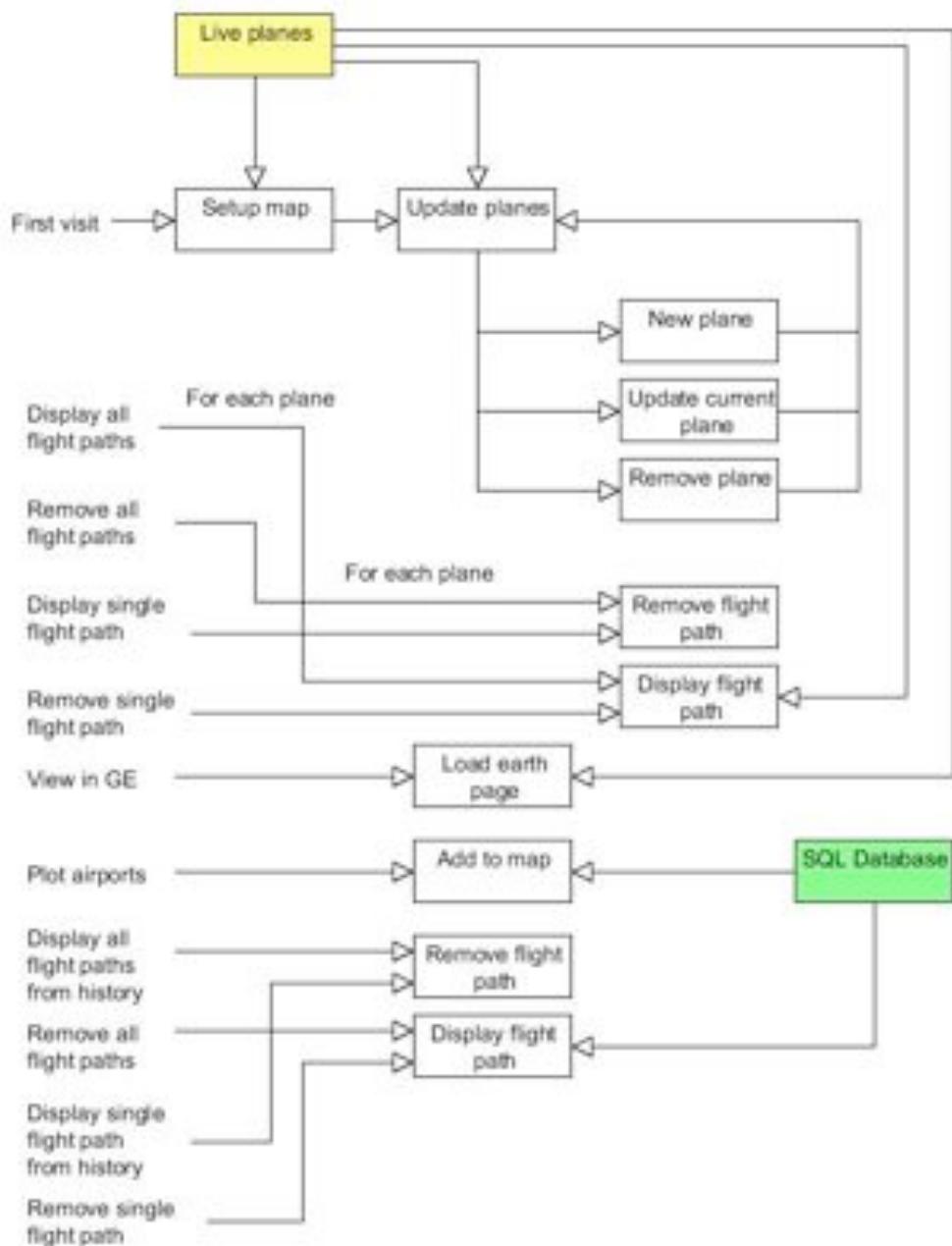
9.3.4 Website pages and their purpose

I initially used a free template available online for the web page but the design changed so much during development that the final design is unrecognisable from the original template and so I do not class the website a template created by someone else.

- *Index.jsp*: This page is the welcome page that the user is first taken to. It contains a brief section about what the website achieves, the last 5 planes that have been added to the map and a live twitter feed that updates dynamically as new tweets are posted.
- *Planes.jsp*: This is the main page for the site and it contains the core functionality. It contains a map with the planes changing dynamically as new data is received.
- *Data.jsp*: This page is the query builder and allows the users to retrieve data from the database.
- *Earth.jsp*: Allows the user to view a flight through the eyes of the pilot using Google Earth.

- *Contact.jsp*: Provides a page that the users can use to report bugs or contact me with questions.
- *Single.jsp*: Used to display the entire raw flight data for a flight from history.
- *Fullinfo.jsp*: Displays all the information available for a flight that hasn't landed yet.

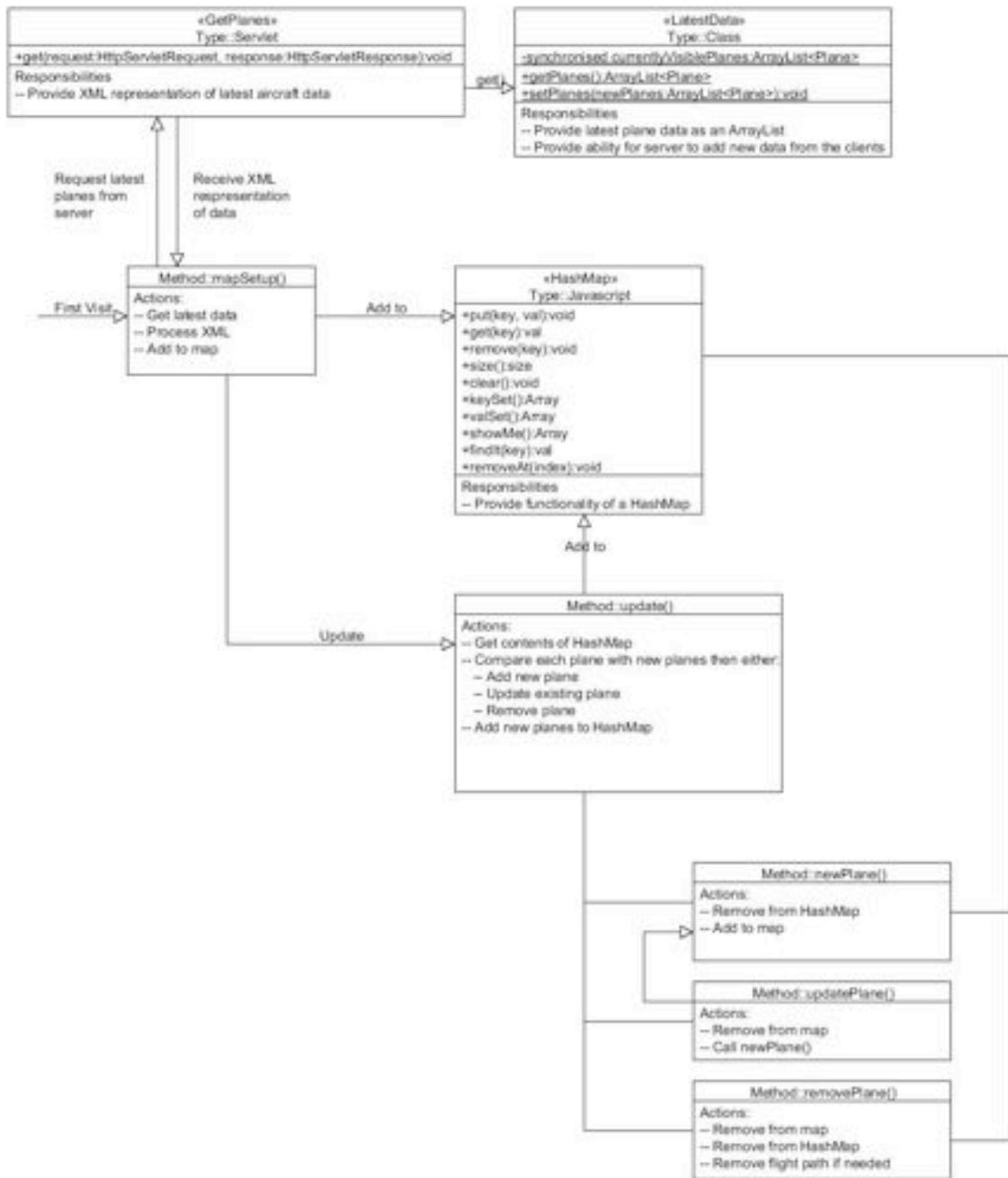
9.3.5 High level software architecture



The above diagram is a high level representation of the design I have implemented.

9.3.6 Low level software architecture

9.3.6.1 Core functionality architecture



This shows the core architecture of the *planes.jsp* page. It is responsible for the initial setup and updating of the planes position and data. The *update()* function repeats every 4000ms regardless of whether the data changes more than once in that time frame. 4000ms was chosen because it makes the updates appear frequent enough to be useful but not too frequent so as to put unnecessary strain upon the server when it is serving data to multiple clients.

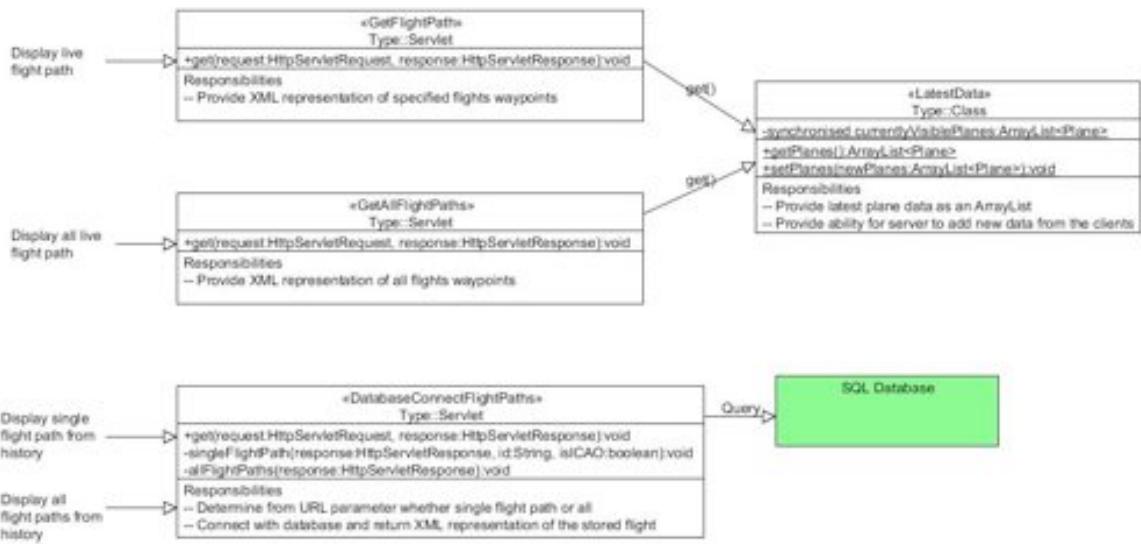
An important feature of my implementation is my use of a **HashMap** (FreeCode, 2007) to hold the planes data that was last returned from the servlet. It contains a map between the planes ICAO, which is unique and acts like a String, and the planes XML data. This allows oper-

ations to be performed on the data without additional calls to the server being made, thus reducing network traffic and increasing the speed that operations are performed at. The key set, in this case all the ICAO codes of the currently visible planes, is also very useful when removing flight paths from the map. This is because all the flight paths are added with the unique ID of “*flightPath*”+*icao*, allowing the key set to be iterated over to remove any occurrences of an element with that ID from the map. As Javascript doesn’t support HashMap as standard I needed to use a 3rd party library to emulate one. I chose the HashMap implementation from FreeCode that has all the necessary features and is free for me to use as I see fit.

The GetPlanes servlet is implemented as a standard Java web servlet with a get and post method. I have chosen to only use the get method because the processing is idempotent, that is its used to only retrieve data and doesn’t alter anything. Using the get method also has the added advantage of encoding any parameters in the URL, meaning a redirection URL can be created manually. An example of when this would be useful is when a client wants information for a specific flight and instead of finding the plane in the table or on the map they can enter the flight number directly into the URL bar, taking them to the information straight away.

You will notice in the diagram above that the update process has been split into 3 distinct functions: *Add*, *Update* and *Remove*. This isn’t the most efficient method or entirely necessary, but I have chosen to do this because it makes the software more easily extendable. It would now be easy to add additional events to each function, such as a notification service when a new plane has been added to the map.

Where relevant I have also used the external library jQuery to process the XML. This is because jQuery provides a few convenience methods for traversing XML structures and because it is also a super class for the plugin TableSorter. I am using TableSorter to display the latest plane data and it is updated each time the map is updated. By using this library I have the ability to add custom sorting rules that allow the clients to sort the table columns regardless of the type of data they contain. The table is also interactive, that is if the clients click a planes flight number it takes them to *fullinfo.jsp* where they can view extra information. The flight number is passed as a parameter in the URL so that the page displays the correct plane.



9.3.6.2 Flight paths

When a user opts to display a currently active flight path, the request is sent via GET to an underlying servlet that processes the request and returns an XML representation of the specified flight. It does this by retrieving the relevant plane from `LatestData` and iterating through each of its waypoints, building the XML as it does so. The client side Javascript can then parse the XML and format it to model a flight path. To model a flight path, the Javascript constructs a geoJSON array to represent multiple points on a single line. This takes the form $[(lat, lon), \dots, (lat, lon)]$. This array can then be added to the map as a PolyLine and post-processed to modify its presentation via CSS i.e. its colour and opacity. As the flight path is added to the map it is given a unique ID that takes the format "`flightPath`" + `ICAO`. This allows the flight paths to be added and removed easily, improving flexibility.

When a user chooses to display all the flight paths for currently active planes the same process occurs, except that the XML returned contains more than one plane element. The Javascript code therefore needs to iterate through every plane element in the XML and add it to the map as above.

The users have the ability to plot flight paths for planes that have been seen before but are no longer active. When a user enters a flight number or ICAO into the query builder, this is sent as a parameter to the `DatabaseConnectFlightPaths` along with a Boolean value. The Boolean value dictates whether all flight paths should be returned or not. If it is false, the ICAO of the aircraft is passed to the `singleFlightPath` method where the database is queried and the XML is created based on the result. If it is true the ICAO is ignored, because it will be null, and the

database is queried to retrieve every waypoint that has ever been recorded. This data is then processed to create the XML and then sent back to the client side Javascript page. Once the data is back on the client side, the Javascript function processes the data as if it were live using the same process as outlined previously.

9.3.6.3 Google earth

The *earth.jsp* page contains one of the core features from my original specification: the ability to view a flight in real time in 3D. It contains a Google Earth instance that is updated dynamically as the data for that plane is updated.

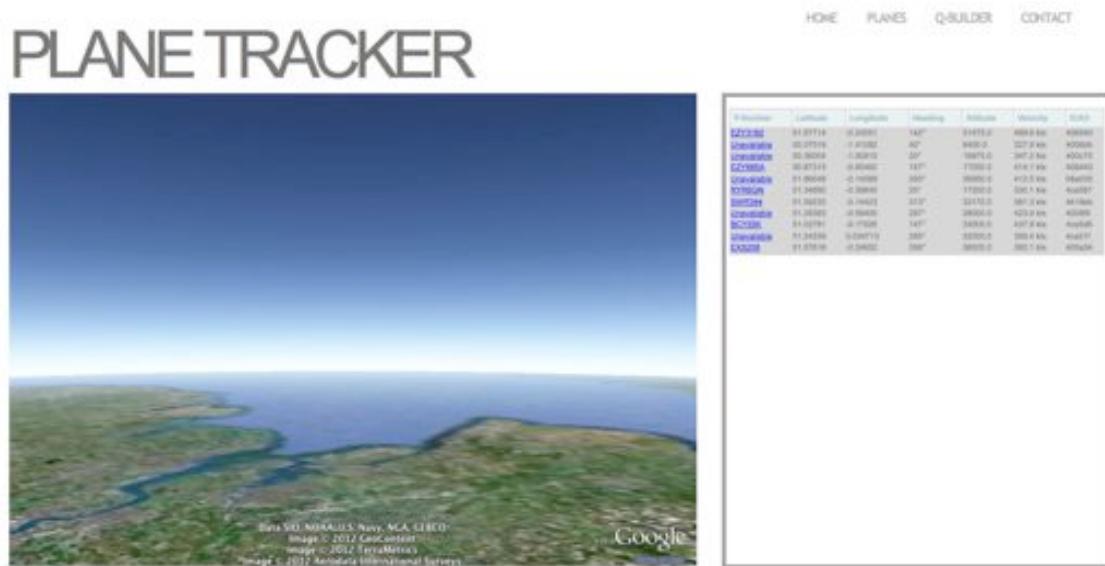


Figure 19 – Cockpit view

It makes use of the GetPlanes servlet because the XML returned from that servlet contains all the information that it needs. These fields are: *latitude*, *longitude*, *heading* and *altitude*. The API for GE is very comprehensive and all that is required to present a cockpit view is the redirection of the camera using the planes data. The update function can be called repeatedly so that each time the data changes the view is updated instantly. It was not necessary to use the 4000ms delay like before because I am only modelling one plane and by having instant feedback it enforces the fact that it is happening in real time. The *earth.jsp* page takes a parameter called ICAO that specifies the ID of the plane that should be displayed. By encoding the ID in the URL I have made it easier to link to it programmatically and have allowed users to enter a flight number that they already know. I make use of a utility function *gup()* (Matteis, 2009) that extracts the parameter from the URL based on its ID. I didn't write this or other occurrences of this method in other files and have only used it to extract the parameters from the URL.

9.3.6.4 Query builder

This page allows the users to view the flight paths for all previous flights, get the raw data from the database as a comma separated list and display the flight path for a flight they have specified.

To export the data as a comma separated list I query the database for all waypoints and separate them based on their ICAO. I then construct a HTML page manually that is returned as the output from the servlet. This means one can visit the servlet directly without having any parameters passed to it. The waypoints are displayed in the format [(lat,lon)...] to allow them to be easily added to other mapping packages or imported into another database.

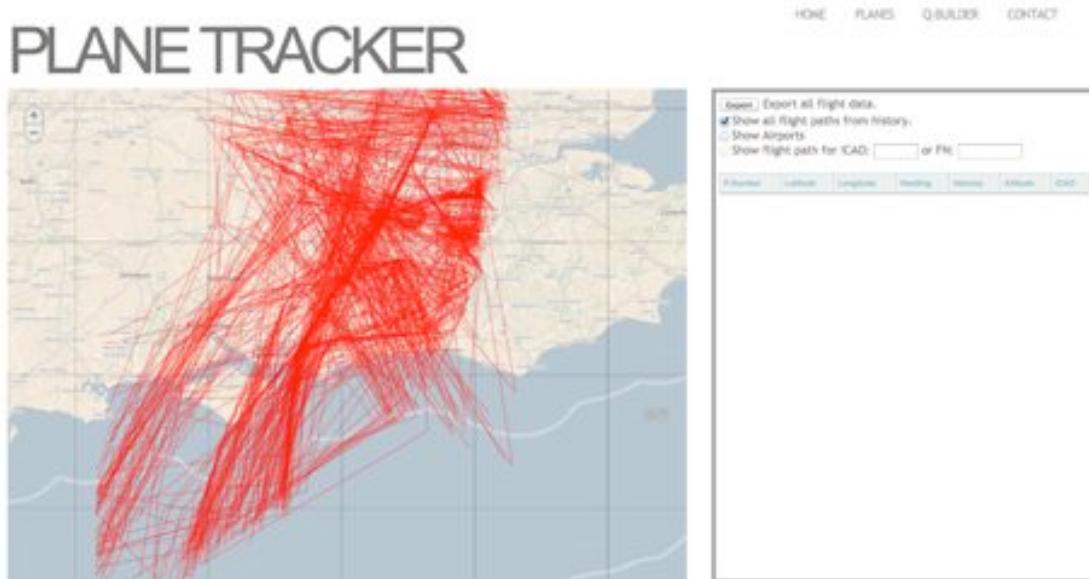


Figure 20 - Display all flight paths

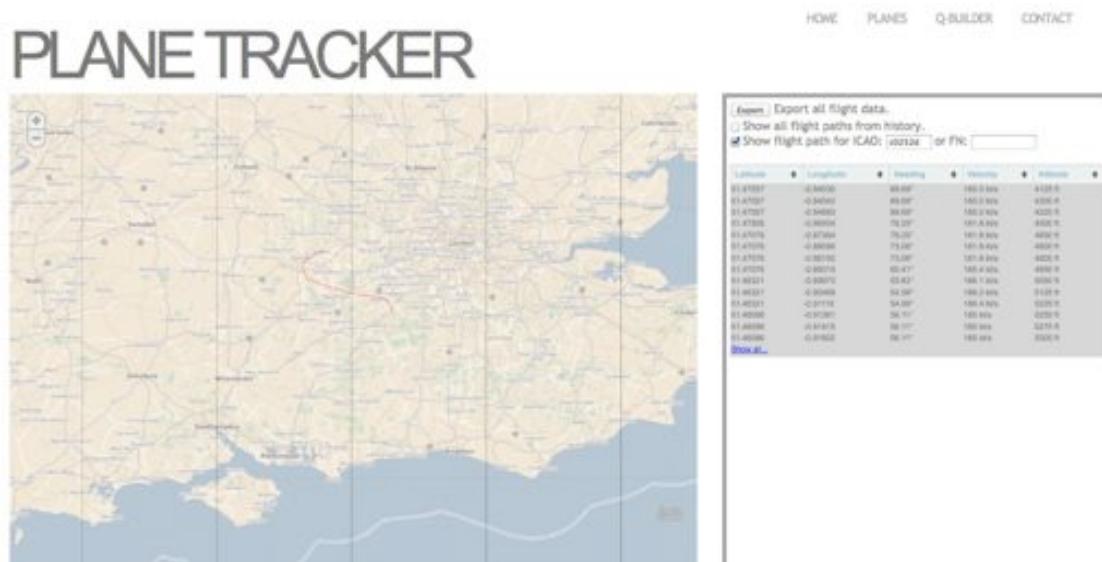


Figure 21 - Display a single flight path

```

"3dc699": {
    "id": 50,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"51": {
    "id": 69,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"52": {
    "id": 70,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"53": {
    "id": 71,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"54": {
    "id": 72,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"55": {
    "id": 73,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"56": {
    "id": 74,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"57": {
    "id": 75,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"58": {
    "id": 76,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"59": {
    "id": 77,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"60": {
    "id": 78,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"61": {
    "id": 79,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"62": {
    "id": 80,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"63": {
    "id": 81,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"64": {
    "id": 82,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"65": {
    "id": 83,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"66": {
    "id": 84,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"67": {
    "id": 85,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"68": {
    "id": 86,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"69": {
    "id": 87,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"70": {
    "id": 88,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"71": {
    "id": 89,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"72": {
    "id": 90,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"73": {
    "id": 91,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"74": {
    "id": 92,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"75": {
    "id": 93,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"76": {
    "id": 94,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"77": {
    "id": 95,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"78": {
    "id": 96,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"79": {
    "id": 97,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"80": {
    "id": 98,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"81": {
    "id": 99,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
},
"82": {
    "id": 100,
    "flightNumber": "0125",
    "airline": "01456085265078125",
    "date": "2010-07-01T12:15:00Z",
    "origin": "10456085265078125",
    "destination": "10456085265078125",
    "gate": "121",
    "status": "Arrived"
}
]

```

Figure 22 - Comma-separated database exportation

9.3.6.5 Full info

This displays all the known information about a flight to the user in a convenient format. It makes use of two web scrapers, database information and live data. The diagram below details the interaction between each of the components.



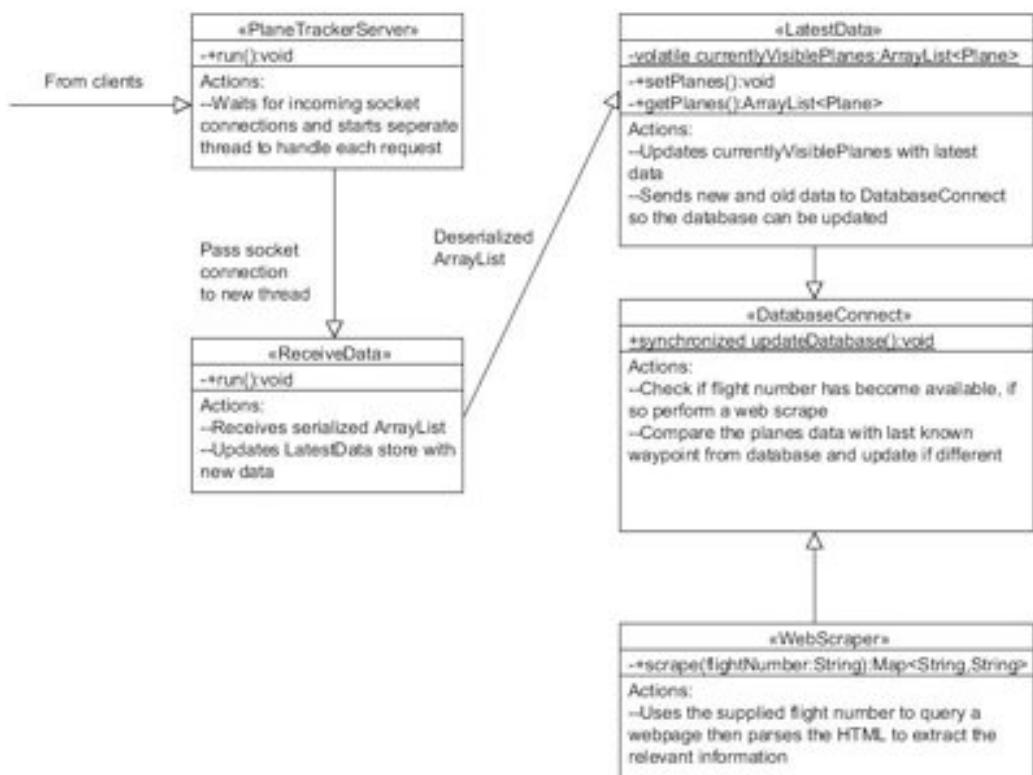
The majority of the computation work occurs in the **WebScraper** class. In essence it parses the HTML from an external web page and extracts the relevant data. Please refer to the web scraper implementation chapter for more information. By choosing a **HashMap** as the method of transferring the data back to the servlet I have made it easy to check whether the data is missing or incomplete and allows multiple datasets to be combined into one **HashMap**, such as the data returned from scraping two different websites.

PLANE TRACKER



Figure 23 - Full info

9.3.6.6 LatestData



My original specification stated that the server needed to be able to handle multiple clients simultaneously because it is quite likely multiple receivers will be feeding the single server. I was thus required to implement the server in a multi-threaded environment and this introduced the need to protect against issues such as race conditions. At each new client request, a new Socket connection is created and passed to another thread to handle the receiving and processing of the data. Once the data object, in this case an **ArrayList** of **Plane**, has been de-serialized it can then be sent to the **LatestData** class for storage. This **LatestData** class acts as a central repository for the latest aircraft data and could potentially be accessed by multiple threads.

simultaneously. To protect against race conditions, where one thread overwrites the changes from another, I have used the *volatile* keyword for the ArrayList. This is essentially the same as placing it within a synchronised block and means the value of the variable will never be cached within the thread and instead all changes will be stored straight into main memory.

Before the ArrayList is updated with the latest data, both lists are sent to DatabaseConnect for comparison. The method *updateDatabase()* within DatabaseConnect is synchronised to prevent thread interference and memory consistency errors, meaning only one thread will be allowed to update the database at any one time.

The two ArrayList's are compared to detect new planes or changed flight numbers. If a new plane is found or its flight number has just been decoded then the database is updated accordingly by inserting either a new Plane or updating an existing entry. In the special case that the flight number becomes available, a web scraper is used to retrieve that flights departure and destination location and, if they can be found, the database is updated with these values as well. Please refer to the Web Scraper chapter for more information on this process. From here, the latitude, longitude, heading and velocity of each plane are compared to the latest waypoint in the database for that plane and, if it differs, a new waypoint is added. If it isn't different then the data has not changed from the previous update and no action is needed.

9.3.6.7 Google AJAX

To retrieve an approximate location for the user without any user input I am using a feature of the Google AJAX API called ClientLocation. When the Google API is loaded into a page the ClientLocation is automatically populated with the users location data. This contains the following fields: *latitude*, *longitude*, *city*, *region*, *country* and *country code*. I then use these to pinpoint the users approximate location and centre the map based on this information.

This should improve the users experience because it will initially show them the planes within range of their location, and they still have the ability to view any part of the map they wish.

I have included a marker that indicates his or her approximate location along with an icon for each receiver. This allows the clients to identify their closest receiver.

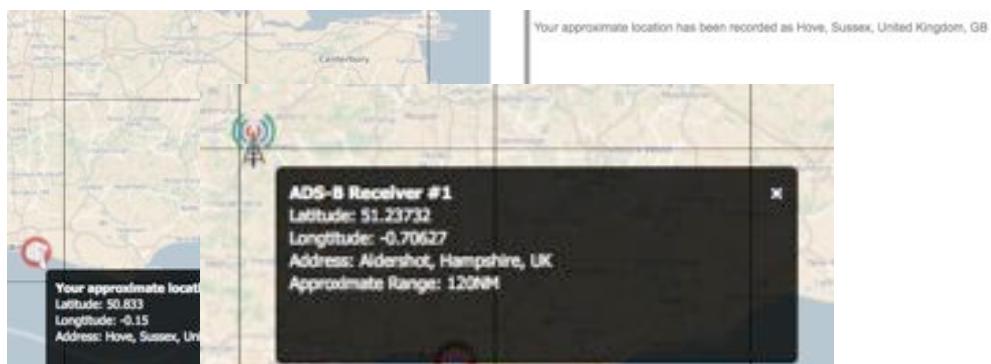
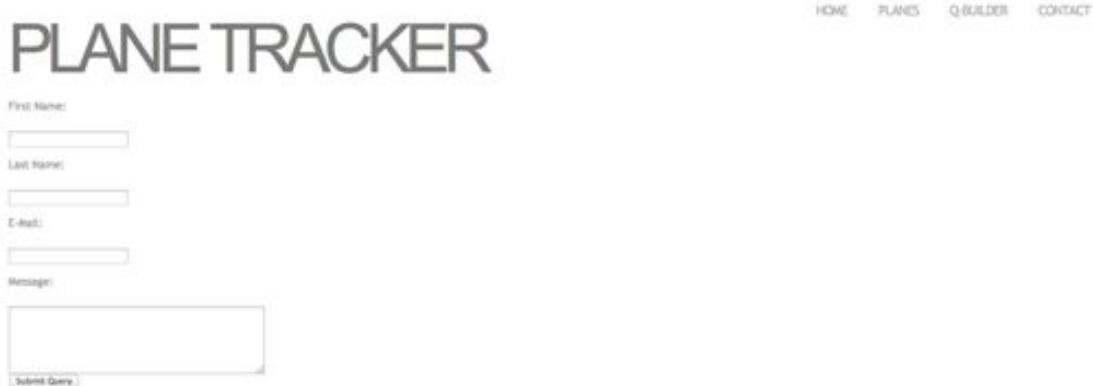


Figure 25 - Shows the location of the receiver(s)

9.3.6.8 Contact page

There are various open source contact page libraries that I could have chosen and I experimented briefly with a few of them. I chose the library that provided only the minimally required features and was extremely lightweight (roScripts). Due to the fact that this library requires PHP to operate I was required to integrate PHP into my TomCat server to handle the clients requests and process the PHP.



The screenshot shows the Plane Tracker contact page. At the top right, there are navigation links: HOME, PLANS, Q-BUILDER, and CONTACT. The main title 'PLANE TRACKER' is centered above a form. The form fields include 'First Name' (with an empty input box), 'Last Name' (with an empty input box), 'E-mail' (with an empty input box), and a large 'Message' area (with an empty text area). Below the message area is a 'Submit Query' button.

Figure 26 - Contact page

9.3.6.9 Twitter feed

To provide the live Twitter feed I am using the official Twitter widget and styling it using a 3rd party style sheet (Oliveira, 2011). This was a relatively simple process and only required some small changes to the default configuration to get the desired behaviour.

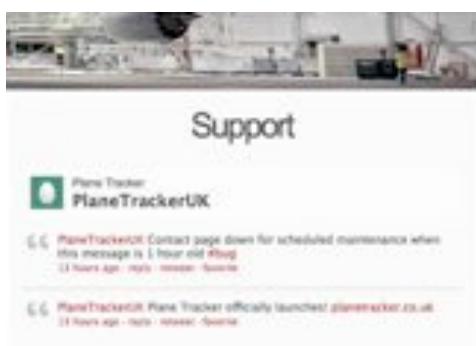


Figure 27 - Live twitter feed

9.3.6.10 Home page

Contains the last 5 planes added to the map and displays the live twitter feed.

The screenshot shows the homepage of Plane Tracker. At the top, there is a navigation bar with links to HOME, PLANES, Q-BUILDER, and CONTACT. Below the navigation is a large image of a Lufthansa Boeing 747 aircraft parked inside a hangar. To the left of the main content area is a sidebar with a heading 'About' and a paragraph of text explaining the site's purpose and data source. The main content area is divided into three sections: 'Live data' (listing five flight entries), 'Support' (with a Plane Tracker UK link), and a live Twitter feed at the bottom.

About

This site is the culmination of 8 months of research and has been produced as part of a dissertation for the University of Sussex. The data that feeds this site is sourced from a personal ADS-B receiver and is being decoded using bespoke decoding software. To upload your own data to this site please visit this page and execute the downloaded file.

Live data

- TOM37P - 491.4kts 33000.0ft - James Coote
- Unavailable - 411.1kts 18500.0ft - James Coote
- RYR6AP - 481.0kts 32925.0ft - James Coote
- CFG518 - 452.3kts 35000.0ft - James Coote
- Unavailable - 372.0kts 20900.0ft - James Coote

Support

[Plane Tracker](#)
[PlaneTrackerUK](#)

PlaneTracker0 Contact page down for scheduled maintenance when this message is 1 hour old [#bug](#)
13 hours ago · reply · retweet · favorite

PlaneTracker0 Plane Tracker officially launched [planeTracker.co.uk](#)
13 hours ago · reply · retweet · favorite

Figure 28 - Home page

9.3.6.11 Web scrapers

The definition of a web scraper is “*An application set to automatically extract information from web pages.*” (Information activism). I have created a web scraping class that scrapes information from two separate websites (flight24) (Gatwick Aviation Society) and presents this data to the clients. To process the returned HTML I am using the external library JSoup that contain methods for traversing the documents DOM and is relatively forgiving with poorly formatted HTML. The implementation details are in the sections below.

9.3.6.11.1 Origin and destination

The flight24 website provides live flight information, in particular the Origin and Destination of the specified flight. The flight number can be appended to the URL and this returns a

HTML page with the necessary information on. However, the flight number needs to use the airlines IATA code rather than its ICAO code. For example, the flight number decoded from the ADS-B transmission uses its ICAO code such as BAW223, whereas to execute a query on the website it would need to be in the format of BA223. To convert the ICAO code to the IATA I need to use the airline data from the database to look up the corresponding IATA code. JSoup can now retrieve the HTML document based on this new URL and convert it into a traversable tree. By knowing the structure of the document I can extract the necessary data and add it to a HashMap that maps the name to the value. The next stage is to retrieve the aircraft data.

9.3.6.11.2 Aircraft data

This process is similar to flight24 but with one key difference: POST rather than GET. This means that the query parameters are not passed in the URL and are encoded within the request. I therefore need to use HttpURLConnection to inject the aircrafts ICAO code into the request and retrieve a String representation of the resultant page. From here I can use JSoup to parse the string and extract the data using the ID of the element. These values are then added to the existing HashMap, ready to be sent back to the servlet so that the XML can be generated.

9.3.6.11.3 Image search

To retrieve the airline logo and the images of the aircraft I make use of Google's image search API. To retrieve a list of images it is possible to pass a search string to the library and receive back the results as images. To create the search string I use the aircraft type that is found from the Gatwick aviation society site and for the airline logo I use the name of airline.

9.3.6.11.4 Combined display

PLANE TRACKER

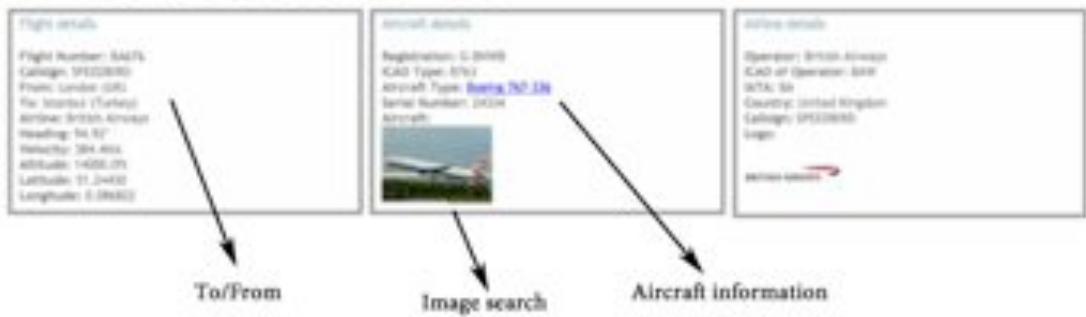


Figure 29 - Full info page

10 Implementation

10.1 Development environments used

The entire software package for both the decoder and the web server was implemented in Eclipse Helios. The SQL and web server were hosted on a Windows XP box and the web server was deployed as part of Eclipse to allow dynamic editing of the hosted data.

10.2 Difficulties faced

10.2.1 Decoder

The greatest challenge was creating the decoding modules. This is because the technical documents where the instructions for decoding are located don't explain how to fully decode all the types of packets, meaning for the velocity and heading I needed to figure out the decoding process myself. The documents are also necessarily complex which made it difficult to understand how best to implement the algorithms in a programming language.

Management of data types and automatic rounding of values in Java was also an issue because of the amount of mathematical functions I needed to use and my relative inexperience with the Math class.

10.2.2 Web server

The first difficulty that I faced was when the web site I was using to retrieve the flight data from, <http://flight24.com>, changed the format they used to display their information. This meant that the HTML changed and I had to change the way the results page was parsed.

The second major difficulty I faced was in understanding the Polymaps API. It seems to be missing vast amounts of information and a lot of the methods I needed to use were not listed at all. This resulted in a lot of wasted time experimenting and trying to find an example that did something close to what I was trying to achieve. This difficulty with this approach is the lack of example code available due to the relative infancy of Polymaps. If I had used Google Maps this would have not have been a problem but would have resulted in a different UI and was something I wasn't prepared to compromise on.

11 System testing

11.1 Requirements specification

In this chapter I have compared the finished project to my original specification to help me reach a conclusion on how successful the project has been.

11.1.1 Core features

Original requirement	Present?
Decode ADS-B transmissions correctly	✓
Display live air traffic in real time in a GUI on the clients local machine	✓
Decode enough information to accurately represent the planes on a map	✓
Make the software stable enough to run unattended 24/7	✓
Allow multiple receivers to be used to increase the coverage	✓
Have a display on the homepage of the last x-number of planes to be added to the map	✓
Have a live twitter feed on the homepage linked to the account PlaneTrackerUK to provide visitors with bug/update information	✓
Be publically accessible from the URL http://www.planetracker.co.uk/	✓
Display a map with all the visible aircraft on it and have their positions updated automatically without the page refreshing	✓
Allow each plane to be clicked to bring up a summary of the flight that includes the most relevant information	✓
Allow the client to plot a flight path for a single flight	✓
Allow the client to plot the flight paths for all the visible planes	✓
Allow the client to view a flight through the eyes of the pilot using Google Earth	✓
Allow the client to plot all the airports in their country on the map	✓
Allow the airports to be clicked to bring up information about it such as its name and location	✓
Provide a table along with the map to display a live text display of all the flights	✓
Allow a flight to be clicked from the table to bring the client to a page that displays all the information available about that flight	✓
Provide a page that provides the client with all the information available for the given flight	✓
Provide a picture of the plane and the airline logo to the client	✓
Provide information about the flights origin and destination	✓
Provide a page where the users can create queries to retrieve information from the database	✓
Allow clients to plot a flight path from the database using one of the flights unique ID's	✓
Allow the client to view all the flight paths for planes stored in the database	✓
Give the client the ability to export the raw data from the database so it can be used for other applications	✓

11.1.2 Non-functional requirements

Original requirement	Present
Website should be easy to navigate and unambiguous	✓
Data should be accessible in an intuitive fashion	✓
The decoding software should be platform independent and easy to use	✓

- **Website should be easy to navigate and unambiguous**
 - No issues were discovered during the usability tests described in the following sections
 - The minimalist design reduces the clutter and makes navigation simpler
 - No unnecessary features have been added that can confuse the users
- **Data should be accessible in an intuitive fashion**
 - Features such as the plane changing colour when the mouse is hovering over it indicates to the users when and where they can click to receive more information
 - No additional plugins are required to see the data making it easily accessible for people with older browsers or who are reluctant to install additional browser software
- **The decoding software should be platform independent and easy to use**
 - Usability testing was performed on a variety of platforms without any problems
 - The usability testing was unguided and none of the participants had trouble operating the software

11.1.3 Conclusion of requirements

I completed all of the core specification and believe my project fulfilled the non-functional requirements to a satisfactory level.

I didn't have time to implement any of my additional features due to the complexity of the tasks and my time constraints. However if I had extra time I still believe that these extra features have the potential to add significant value to the website.

11.2 Integration testing

Integration testing can be defined as “individual software modules that are combined and tested as a group”. This means bringing together all the parts of the software and testing it as a whole so test the overall stability of the software.

Some of the results can be checked visually i.e. look at how the data is presented on the website or stored in the database and some can be checked using unit testing. Unit testing involves writing Java code to test other Java code. This can be useful for checking the output from a certain class, method or line is what is expected and these tests can be automated so they are all run at once. The advantage of unit testing is that as new software is written the backwards compatibility can easier be checked by running the unit tests regularly i.e. during the nightly build.

11.2.1 Receiver testing

Action	Location	Expected	Result	Method
Useful packets being identified	UsefulPacketFinder	Correct packets identified, rest ignored	✓	JUnit
Each packet identified correctly	UsefulPacketFinder	Type of packet correctly identified	✓	JUnit
Corrupted packets discarded	UsefulPacketFinder	Parity check removes packet	✓	JUnit
Packets sent to correct decoder	PacketDecoder	Each packet is sent to correct decoder	✓	Manual/Debug
Position and altitude decoding	CPRDecoder	Correct position and altitude outputted	✓	JUnit
ID decoding	IDDecoder	Correct flight number outputted	✓	JUnit
Heading and Velocity decoding	CalculateVelocity	Correct heading and velocity outputted	✓	JUnit
Data not lost from the receiver	SerialEventActionListener	No data lost	N/A	Cannot be tested
Data sent to the client	PlaneTrackerClient	Data received correctly on the server	✓	Manual/Debug

11.2.2 Database testing

Action	Location	Expected	Result	Method
Add airports to database	AddAirportsToDatabase	Airports are correctly formatted inside the database	✓	Visual
Add airlines to database	AddAirlineToDatabase	Airlines are correctly formatted inside the database	✓	Visual
Update a planes waypoint	DatabaseConnect	No duplicate waypoints added and none missing	✓	Visual

11.2.3 Website testing

11.2.3.1 index.jsp

Action	Expected	Result	Method
Home link pressed	User taken back to home page	✓	Visual
Planes link pressed	User taken to the planes page	✓	Visual

Query link pressed	User taken to the query page	✓	Visual
Contact link pressed	User taken to the contact page	✓	Visual
Live data updated	List of planes changes dynamically	✓	Visual
Twitter username pressed	User taken to twitter profile	✓	Visual

11.2.3.2 planes.jsp

Action	Expected	Result	Method
Home link pressed	User taken back to home page	✓	Visual
Planes link pressed	User taken to the planes page	✓	Visual
Query link pressed	User taken to the query page	✓	Visual
Contact link pressed	User taken to the contact page	✓	Visual
Plane clicked	Summary of plane displayed slightly offset from the planes position	✓	Visual
Google Earth logo clicked	User taken to earth.jsp which displays the selected flight	✓	Visual
Multiple planes clicked	Multiple summaries displayed neatly slightly offset from the last planes clicked position	✓	Visual
Close notification clicked	Notification bubble fades away	✓	Visual
'Close All' notifications clicked	All notification bubbles fade away	✓	Visual
Plane clicked	Flight path displayed	✓	Visual
Plane clicked	Summary contains correct information	✓	Visual
All flight paths checked	All flight paths correctly displayed	✓	Visual
All flight paths checked	Paths added and removed when a plane is added or removed	✓	Visual
All flight paths unchecked	All flight paths removed from the map	✓	Visual
Display airports checked	Airports plotted correctly onto the map	✓	Visual
Airport clicked	Summary of the airport displayed in the same style as plane notifications	✓	Visual
Airport clicked	Summary contains correct information	✓	Visual
Display planes unchecked	Planes removed from map	✓	Visual
Display planes checked	Planes added to the map	✓	Visual
N/A	Planes updated dynamically without page refresh	✓	Visual
N/A	Table updated dynamically without page refresh	✓	Visual

Flight number in table clicked	User is taken to fullinfo.jsp that displays information about the selected flight	✓	Visual
--------------------------------	---	---	--------

11.2.3.3 data.jsp

Action	Expected	Result	Method
Home link pressed	User taken back to home page	✓	Visual
Planes link pressed	User taken to the planes page	✓	Visual
Query link pressed	User taken to the query page	✓	Visual
Contact link pressed	User taken to the contact page	✓	Visual
User attempts to enter no flight number	User is asked to enter a valid flight number	✓	Visual
User attempts to enter no ICAO	User is asked to enter a valid ICAO	✓	Visual
User attempts to enter invalid data	User is asked to enter valid data	✓	Visual
User attempts to retrieve data for a non-existent flight	User is alerted that that flight doesn't exist	✓	Visual
User enters a flight number	Flight path is displayed on map	✓	Visual
User enters a flight number	Flight data is displayed in the table up to a maximum of 15 entries	✓	Visual
User enters an ICAO number	Flight data is displayed on the map	✓	Visual
User enters an ICAO number	Flight data is displayed in the table up to a maximum of 15 entries	✓	Visual
User clicks view more info	User is taken to single.jsp where the entire flight data for a single flight is displayed	✓	Visual
View all flight paths checked	All flight paths added to the map	✓	Visual
Flight path clicked	The ICAO and flight number is displayed in a notification	✓	Visual
Export data clicked	User is taken to the text file with all the database data presented as a comma separated list	✓	Visual

11.2.3.4 fullinfo.jsp

Action	Expected	Result	Method
Home link pressed	User taken back to home page	✓	Visual
Planes link pressed	User taken to the planes page	✓	Visual
Query link pressed	User taken to the query page	✓	Visual
Contact link pressed	User taken to the contact page	✓	Visual
Aircraft type clicked	User taken to Google images with the results show pictures of that aircraft	✓	Visual
N/A	Flight data updated dynamically	✓	Visual
N/A	Flight data notifies when plane has gone out of range	✓	Visual
N/A	Origin and destination fails correctly if unavailable	✓	Visual
N/A	Image of the plane correctly displayed	✓	Visual
N/A	Correct airline logo displayed	✓	Visual
N/A	Information correctly formatted even if unavailable	✓	Visual
User attempts to enter an invalid ICAO	No information displayed	✓	Visual
User attempts to enter an invalid flight number	No information displayed	✓	Visual

11.2.3.5 earth.jsp

Action	Expected	Result	Method
Home link pressed	User taken back to home page	✓	Visual
Planes link pressed	User taken to the planes page	✓	Visual
Query link pressed	User taken to the query page	✓	Visual
Contact link pressed	User taken to the contact page	✓	Visual
User attempts to enter an invalid ICAO	GE view not updated	✓	Visual
Flight moves out of range	User is taken back to planes.jsp	✓	Visual
N/A	Flight information updated dynamically in the table	✓	Visual

Flight clicked in the table	User is taken to fullinfo.jsp for the specified flight	✓	Visual
-----------------------------	--	---	--------

11.2.3.6 contact.jsp

Action	Expected	Result	Method
Home link pressed	User taken back to home page	✓	Visual
Planes link pressed	User taken to the planes page	✓	Visual
Query link pressed	User taken to the query page	✓	Visual
Contact link pressed	User taken to the contact page	✓	Visual
User submits with missing fields	User is notified of the missing fields	✓	Visual
User submits correct message	User is notified it was a success and the email is sent to me	✓	Visual

11.2.3.7 single.jsp

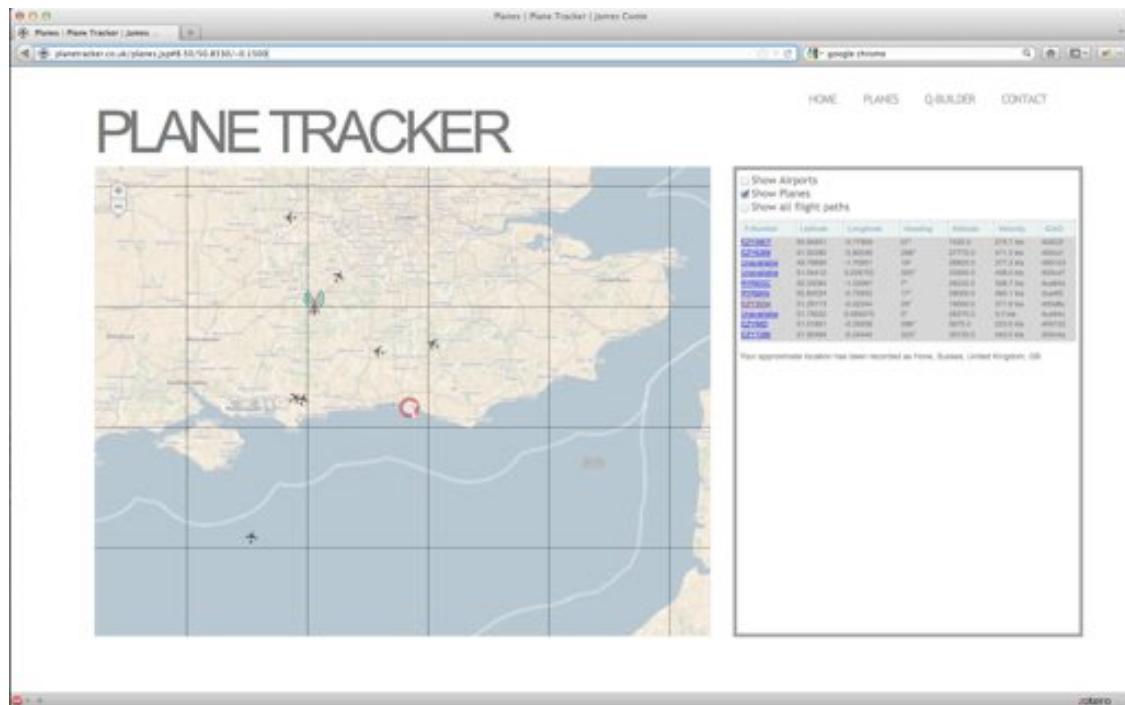
Action	Expected	Result	Method
First visit to page	Table displays all the flight data for the given flight	✓	Visual
User attempts to manually enter an invalid ICAO	Table doesn't displayed any data	✓	Visual

11.3 Hardware and browser testing

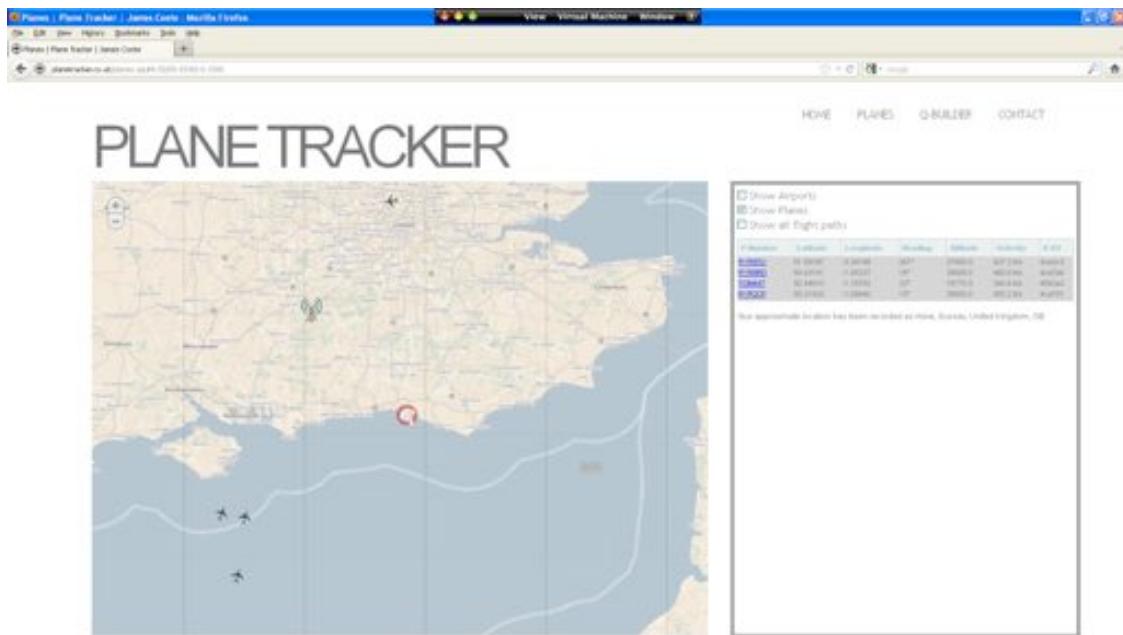
The way web browser's display and structure HTML pages vary greatly and it's an essential part of testing to ensure the site is consistent regardless of browser or operating system. I have tested the receiver software using the integration tests above on Mac OSX Lion and Windows XP. The web browsers I have used to test the website functionality are the latest stable versions of Firefox, Safari, Opera and Chrome.

11.3.1 Operating system testing

These tests are performed using Firefox as the web browser.

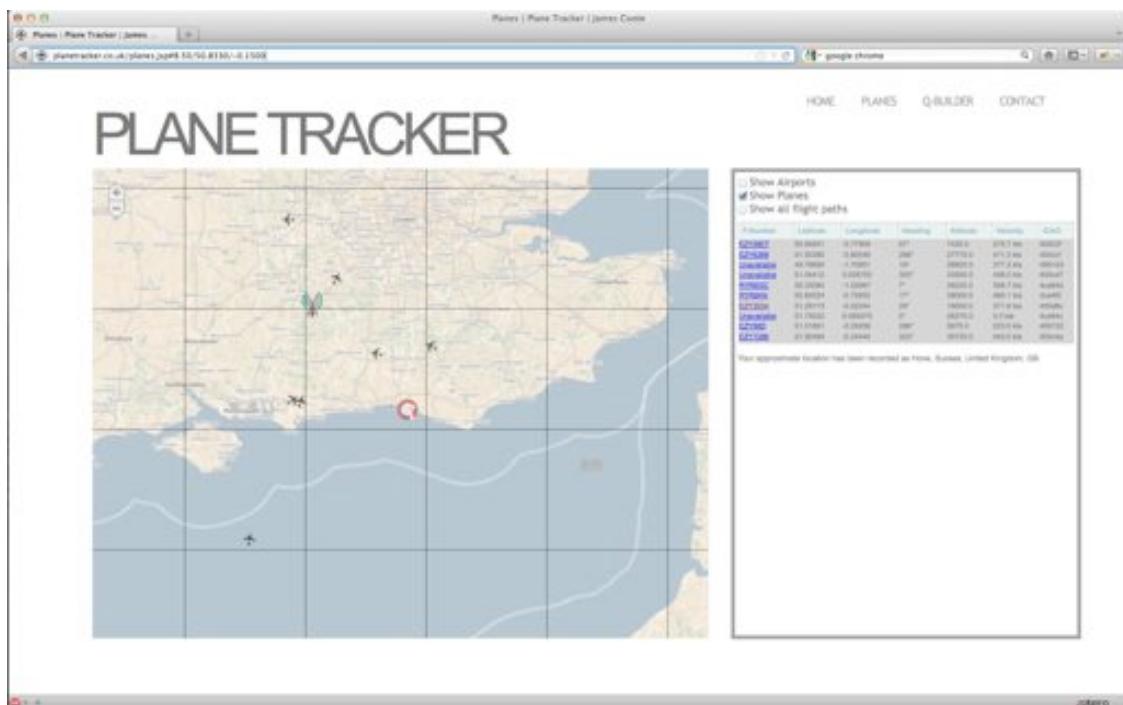


Firefox on Mac OSX Lion

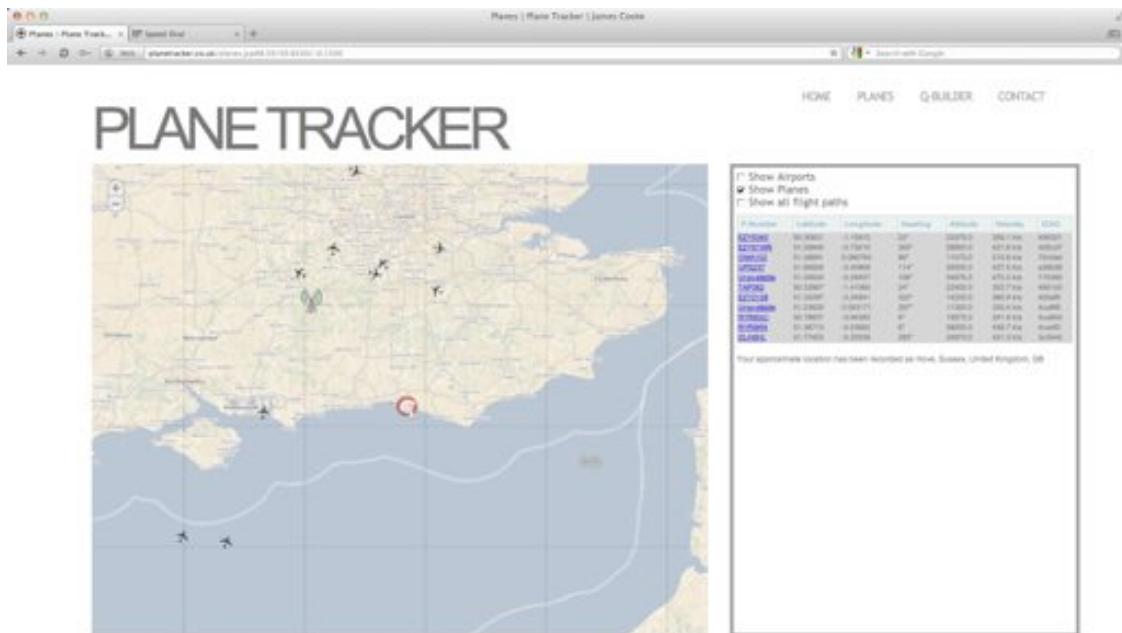


Firefox on Windows XP Service Pack 3 (Virtual Machine)

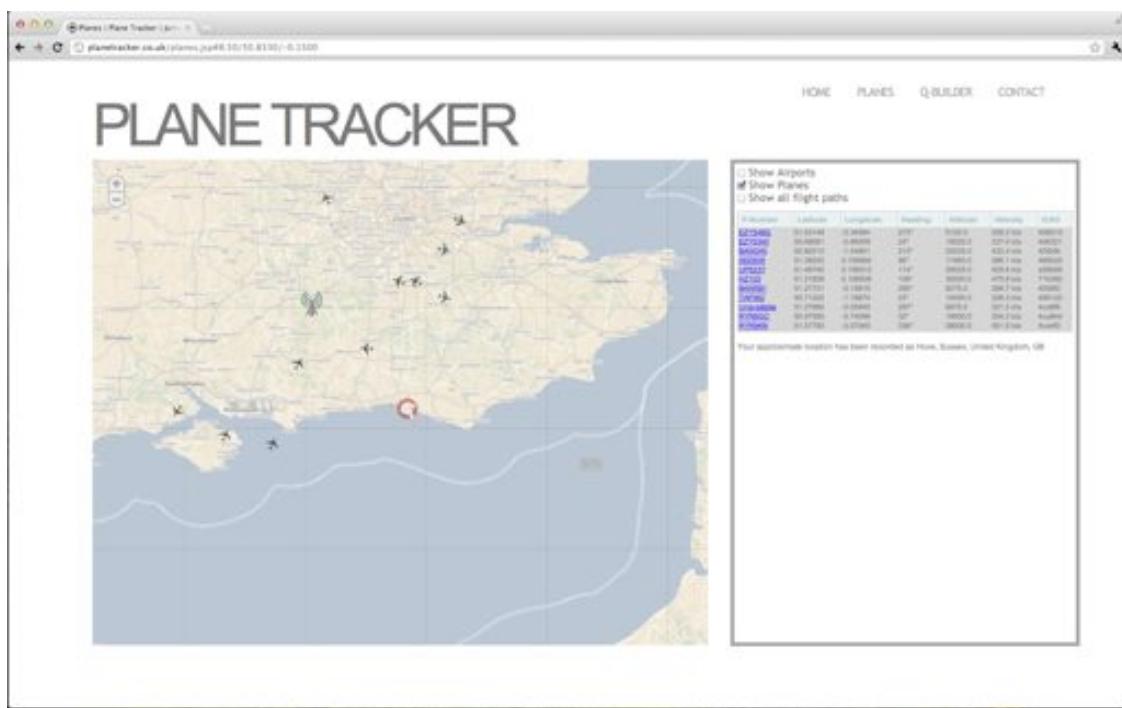
11.3.2 Web browser testing



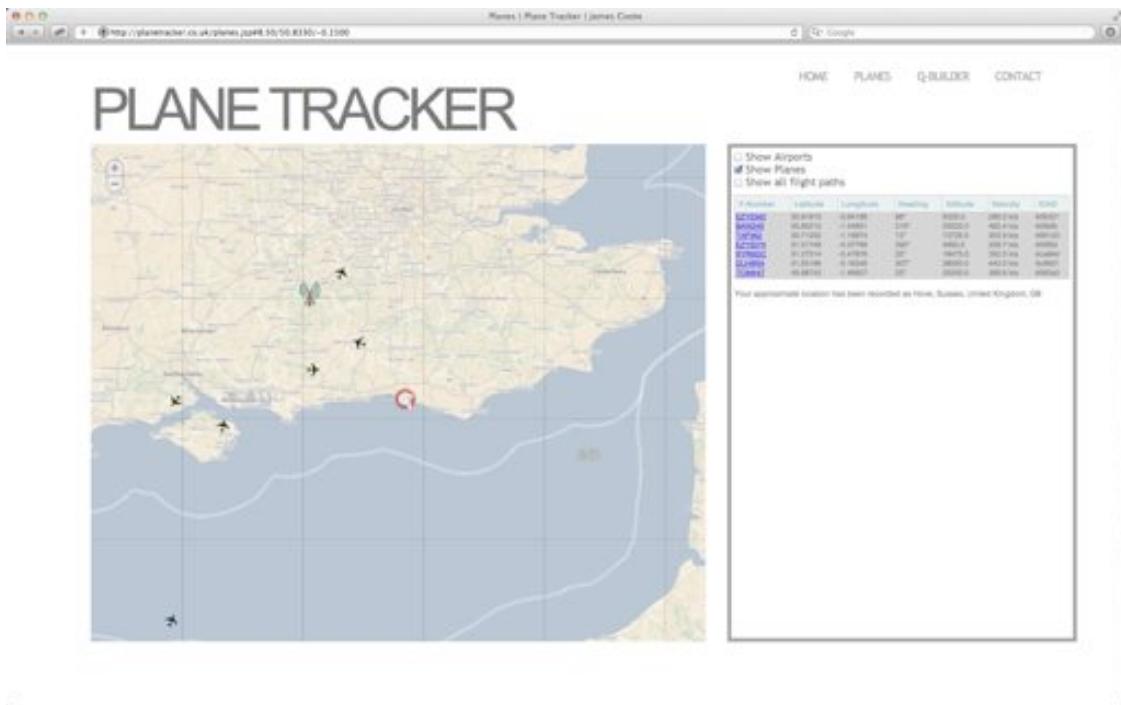
Firefox on Mac OSX Lion



Opera on Mac OSX Lion



Chrome on Mac OSX Lion



Safari on Mac OSX Lion

11.4 User acceptance testing

When selecting the users for the user acceptance testing I was careful to chose people who would accurately represent my chosen demographic. The majority of the users were technically literate university students however I also featured a couple of older users with no technical experience.

These non-technical users tended to have slightly less positive results than the students which indicates that perhaps my website could benefit from more focus on the UI design.

The users identified a number of features that I hadn't included in the core functionality that I believe would add significant value to the site. These features were:

- Display a flight path all the way from the origin to the destination. The reason I didn't implement this is because it would require guesswork about the planes route and I didn't want to represent false data. If the coverage of the map were increased with more receivers then this task would become more feasible.
 - Search the map for a flight. This would be useful when there are too many planes on the map to display in the table and the user is looking for a specific flight.
 - Have the ability to browse the flights in the database rather than having to specify a flight.
 - Update the planes more often. At the moment this isn't possible because it is already been updated as soon as the position update is received. With more receivers, more position reports would be received and this would become more feasible.

The original questionnaire can be found in the appendix, along with the users responses. These responses matched the responses I was expecting and did not highlight any serious flaws in my software.

12 Conclusion

The project started out with a hazy specification and it wasn't until I started to understand the capabilities of ADS-B that I could create the formal specification for the project. If the features contained in the finished project are compared solely against the original specification then my project has been a complete success. I implemented all of the functionality I set out to and although I didn't add any additional features I have every confidence in the sites ability to compete with the commercial products available. With additional resources there is definite potential for my projects capabilities to be extended far beyond my competitors and therefore be at level where it could enter the commercial market.

I set out at the beginning with two clear aims: to create a standalone decoding program that could be used by the public and to create a public facing website where users could interact with their own and others data. I have been successful on both fronts and have met the requirements at every stage of the development. The decoding software can be used to provide a local display to users who have downloaded it and feeds the decoded data to the website so it can be made useful for others as well. The website allows the public to find out more information about a flight than would not normally be available and has huge potential to be used for machine learning and data mining purposes. This is because I have made all of the stored data publicly available and when the website increases in popularity and more receivers are being used this data will become even more useful.

I believe that my choice to use the agile method of development was absolutely correct and gave me the ability to add features at any point, without the complications that would have occurred if I had used the waterfall method. It allowed me to create a base application, test it, and then add components on top of it, confident in the integrity of the previous modules. Without the flexibility that the agile method supports I would have been unable to implement some of the most useful features of my website due to the changes that were required in the original designs.

If I had more development time I would have liked to implement more of the features mentioned in the design chapter. More specifically, the ability for a user to enter a flight number into the website so that when that flight is found for the first time they are notified that it has been seen. This would not have taken much extra development time and, if I had a longer time frame, is the feature I would most like to have added. Other features that aren't present have been excluded because they did not add enough value to the website to justify the time it would take to implement them.

Due to the project budget I was unable to purchase another receiver to fully test the multi-receiver environment that my software supports. With an increase in budget I would setup two or more receivers in geographically distributed locations to increase the coverage of the receiver and make the website more useful for its users.

13 References

1. Unknown. (2009). *Airports Data Service*. Retrieved Jan 2012 from <http://airports.pidgets.com/v1/>
2. ADS-B Technologies. (n.d.). *ADS-B Technologies - What is ADS-B?* Retrieved September 22, 2011 from ADS-B Technologies: <http://www.ads-b.com/>
3. AirFrames. (n.d.). *Aircraft Database*. Retrieved September 22, 2011 from AirFrames: <http://airframes.org>
4. BCS Trustee Board. (2006, November 29). *Code of Conduct for BCS Members*. Retrieved December 1, 2011 from British Computer Society: <http://www.bcs.org/category/6030>
5. Coupe. (2011, September 3). Retrieved 01 10, 2012 from RadarSpotters: <http://radarspotters.eu/forum/index.php?action=printpage;topic=5617.0>
6. FAA, CAA. (2009). *1090-WP-9-14 - ADS-B 1090 MHz MOPS*. FAA, CAA.
7. flight24. (n.d.). *Aviation database*. Retrieved April 04, 2012 from flight24: <http://data.flight24.com>
8. FreeCode. (2007, June 11). *A HashMap object in Javascript like the HashMap in Java*. Retrieved 12 10, 2011 from FreeCode: <http://freecode-freecode.blogspot.co.uk/2007/06/hashmap-object-in-javascript-like.html>
9. Gatwick Aviation Society. (n.d.). *Mode S Codes*. Retrieved April 04, 2012 from Gatwick aviation society: <http://www.gatwickaviationsociety.org.uk/modeslookup.asp>
10. ICAO. (2007). *Annex 10 To The Convention On Internal Civil Aviation*. ICAO.
11. ICAO. (2008). *Doc 9871 AN/460 - Technical Provisions for Mode S Services and Extended Squitter*. ICAO.
12. Information activism. (n.d.). *Glossary of terms*. Retrieved April 04, 2012 from Information Activism: <http://www.informationactivism.org/glossary>
13. Nave, R. (n.d.). *Hyper Physics*. Retrieved March 10, 2012 from <http://hyperphysics.phy-astr.gsu.edu/hbase/vect.html#vec4>
14. Neely, J. (2009, March 09). *Converting binary number to string*. Retrieved 01 10, 2012 from StackOverflow: <http://stackoverflow.com/questions/625838/java-specify-number-of-bits-length-when-converting-binary-number-to-string>

15. Matteis, L. (2009, February 7). *How can I get a specific parameter from location.search?* Retrieved Feb 22, 2012 from Stack Overflow: <http://stackoverflow.com/questions/523266/how-can-i-get-a-specific-parameter-from-location-search>
16. Oliveira, R. (2011, June 11). *How to Customize Twitter Search and Profile Widgets.* Retrieved April 12, 2012 from 1st Web Designer: <http://www.1stwebdesigner.com/css/customize-twitter-search-widgets/>
17. Phillips, D. H. (1999, March). *ADS-B...Terroists dream, security's nightmare.* Retrieved September 22, 2011 from Airport Corporation: <http://www.airsport-corp.com/adsb2.htm>
18. roScripts. (n.d.). *AJAX Contact Form.* Retrieved April 12, 2012 from roScripts: http://www.roscripts.com/AJAX_contact_form-144.html
19. RTCA. *ADS-B 1090 MHz MOPS Meeting 9.* RTCA.

14 Appendices

14.1 Database setup file (SQL)

```
CREATE TABLE plane
(
    icao VARCHAR(6) NOT NULL,
    flight_number VARCHAR(10),
    destination VARCHAR(30),
    departed VARCHAR(30),
    PRIMARY KEY (icao)
);
CREATE TABLE waypoint
(
    id int NOT NULL AUTO_INCREMENT,
    icao VARCHAR(6) NOT NULL,
    lat VARCHAR(40) NOT NULL,
    lon VARCHAR(40) NOT NULL,
    heading DOUBLE PRECISION,
    altitude DOUBLE PRECISION,
    speed DOUBLE PRECISION,
    time TIMESTAMP NOT NULL,
    PRIMARY KEY (id)
);
CREATE TABLE airline
(
    airline_id int NOT NULL AUTO_INCREMENT,
    airline_iata VARCHAR(100),
    airline_icao VARCHAR(100),
    airline_name VARCHAR(100),
    airline_call_sign VARCHAR(100),
    airline_country VARCHAR(100),
    airline_comment VARCHAR(300),
    PRIMARY KEY (airline_id)
);
CREATE TABLE airport
(
    id int NOT NULL AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    city VARCHAR(50) NOT NULL,
    country VARCHAR(50) NOT NULL,
    iata VARCHAR(3),
    icao VARCHAR(6),
    lat VARCHAR(40) NOT NULL,
    lon VARCHAR(40) NOT NULL,
    PRIMARY KEY (id)
}
```

14.2 CPR Decoder

```
double YZ0 = Integer.parseInt(plane.getEvenBinaryLat(), 2);
double YZ1 = Integer.parseInt(plane.getOddBinaryLat(), 2);
double XZ0 = Integer.parseInt(plane.getEvenBinaryLon(), 2);
double XZ1 = Integer.parseInt(plane.getOddBinaryLon(), 2);

//Calculates the Latitude
double j = Math.floor(((double)59*YZ0-
    (double)60*YZ1)/(double)131072)+(double)0.5;
double rlat0 = ((double) 360/(double)60) * (mod(j,59) +
    (YZ0/(double)131072));
double rlat1 = ((double) 360/(double)59) * (mod(j,59) +
    (YZ1/(double)131072));

//Checks the NL value is equal otherwise discard
if(NL(rlat0)!=NL(rlat1)) {
    return plane;
}
//Calculates the Longitude
double ni = 1;
double nl = 0;
if(plane.isFoundLast()) {
    if(NL(rlat0) > 1 ) {
        ni = NL(rlat0);
        nl =NL(rlat0);
    }
} else {
    if(NL(rlat1)> 1){
        ni = NL(rlat1)-1;
        nl =NL(rlat1);
    }
}

double dlon= (double)360/(double)ni;
double m = Math.floor(((XZ0 * ((double)nl - 1)) - (XZ1 * (double)nl)) /
    (double)131072) + (double)0.5;

double rlon=0;
if(plane.isFoundLast()){
    rlon = dlon * (mod(m,ni)+(XZ0/131072));
} else {
    rlon = dlon * (mod(m,ni)+(XZ1/131072));
}

if(rlat0 > 47 && rlat0 < 53 && rlon > -2 && rlon < 3) {
    plane.setActualLat(rlat0);
    plane.setActualLon(rlon);
    plane.setPositionBeenCalculated(true);
}
plane.addWaypointFromCurrentInformation();
return plane;
```

14.3 Velocity over ground packet

Figure C-5. Extended Squitter Airborne Velocity
(Subtypes 1 and 2: Velocity Over Ground)

Register 09₁₆

1	MSS	1	Purpose: To provide additional state information for both normal and supersonic flight.
2		0	
3	FORMAT TYPE CODE = 19	0	
4	(SC 2.3.1)	1	
5	LSS	1	
6	Subtype 1	0	Subtype Coding:
7		0	
8		1	
9	INTENT CHANGE FLAG (SC 2.3.5.3)		
10	RESERVED-A		
11	NAVIGATION ACCURACY CATEGORY FOR VELOCITY		
12	(NAC) (SC 2.3.5.4)		
13	DIRECTION BIT for E/W Velocity (0=East, 1=West)		
14	EAST-WEST VELOCITY (10 kmh)		
15	NORMAL : LSS = 1 knot	SUPERSONIC : LSS = 4 knots	Reference ARINC Labels for Velocity
16	All zeros = no velocity info	All zeros = no velocity info	East - West
17	Value	Velocity	North - South
18	000000	000000	GPS: 174
19	1	0 Ma	GPS: 188
20	2	1 Ma	TAS: 367
21	3	2 Ma	
22	—	—	
23	1022	1021 kts	1022
24	1023	>1021.5 kts	>1022 kts
25	DIRECTION BIT for N/S Velocity (0=North, 1=South)		
26	NORTH - SOUTH VELOCITY (10 kmh)		
27	NORMAL : LSS = 1 knot	SUPERSONIC : LSS = 4 knots	Reference ARINC Labels:
28	All zeros = no velocity info	All zeros = no velocity info	GNSS Height (HAE): GPS 270
29	Value	Velocity	GNSS Altitude (MSL): GPS 276
30	000000	000000	
31	1	0 Ma	
32	2	1 Ma	
33	3	2 Ma	
34	—	—	
35	1022	1021 kts	1022
36	1023	>1021.5 kts	>1022 kts
37	SOURCE BIT FOR VERTICAL RATE (0=Geometric, 1=Rate)		
38	SIGN BIT FOR VERTICAL RATE (0=Up, 1=Down)		
39	VERTICAL RATE (9 kmh)		
40	All zeros = no vertical rate info, LSS = 64 feet/min		
41	Value	Vertical Rate	
42	000000	000000	
43	1	0 ft/min	ARINC 429 16DFR
44	2	64 ft/min	GPS: 165
45	—	—	TAS: 369
46	510	323.75 ft/min	
47	511	>323.75 ft/min	
48	RESERVED-B		
49	DIFFERENCE SIGN BIT (0 = Above Baro, 1 = Below Baro AD)		
50	GEOMETRIC HEIGHT DIFFERENCE FROM BARO ALT		
51	(7 bits) (SC 2.3.5.6) (All zeros = no info) (LSS = 25 feet)		
52	Value	Difference	
53	000000	0 feet	
54	1	25 feet	
55	2	—	
56	—	—	
57	128	3125 feet	
58	127	>3125.5 feet	

14.4 Airspeed and Heading packet

Figure C-6. Extended Squitter Airborne Velocity (Subtypes 3 and 4; Airspeed and Heading)

Register 09_h

1	MSB	1
2	FORMAT TYPE CODE = 19	0
3	(SC 2.3.7)	0
4	LSS	1
5	Subtype 3	0
6		Subtype 4
7	1	0
8	1	0
9	1	0
10	INTENT CHANGE FLAG (SC 2.3.5.3) RESERVED-A	
11	NAVIGATION ACCURACY CATEGORY FOR VELOCITY (NAC) (SC 2.3.5.4)	
12	STATUS BIT (1 = Heading available, 0 = Not available)	
13	MSB	
14	HEADING (10 bits) (SC 2.3.6.6) Resolution = 360/1024 degrees	
15	Reference ARINC Label PN: 320	
16	LSS	
17	AIRSPEED TYPE (0 = TAS, 1 =IAS)	
18	AIRSPEED (10 bits)	
19	NORMAL: LSS = 1 knot All zeros = no velocity info	SUPERSONIC: LSS = 4 knots All zeros = no velocity info
20	Value	Value
21	1	0 Kts
22	2	1 Kts
23	3	2 Kts
24	—	3 Kts
25	1022	1022 Kts
26	1023	> 1021.5 Kts
27	SOURCE BIT FOR VERTICAL RATE (0=One, 1=Both)	
28	SIGN BIT FOR VERTICAL RATE (0=Up, 1=Down)	
29	VERTICAL RATE (8 bits)	
30	All zeros = no vertical rate information	
31	LSS = 64 fpm/min	
32	32648	Vertical_Rate
33	1	0 fpm
34	2	64 fpm
35	—	—
36	510	32576 fpm
37	511	= 32604 fpm
38	RESERVED-B	
39	DIFFERENCE SIGN BIT (0 = Above Bars, 1 = Below Bars AND GEOMETRIC HEIGHT DIFFERENCE FROM BARO ALT 17 ft) (SC 2.3.5.6) (All zeros = no info) (LSB = 25 feet)	
40	Value	
41	1	0 ft
42	2	25 ft
43	—	—
44	126	3125 ft
45	127	= 31375 ft

Purpose: To provide additional state information for both normal and supersonic flight based on airspeed and heading.

Note: — This format is only used if velocity over ground is not available.

Subtype Coding:

Code	Velocity	Type
0	Reserved	
1	Ground	Normal
2	Speed	Supersonic
3	Airspeed	Normal
4	Heading	Supersonic
5	Non-Assigned	Normal
6	Non-Assigned	Supersonic
7	Non-Assigned	Normal/Supersonic

Reference ARINC 429 Labels:
for Air Data Source:

IAS: 206

TAS: 210

Reference ARINC Labels:
GNSS Height (HAE): GPS 370
GNSS Altitude (MSL): GPS 376

14.5 ID packet

Figure C-4. Extended Squitter Identification and Category

Register 0B₁₄

1	FORMAT TYPE CODE (§C.2.3.1)	Purpose: To provide aircraft identification and category.
2	AIRCRAFT Emitter CATEGORY	TYPE Coding: 1 = Aircraft identification, Category Set D 2 = Aircraft identification, Category Set C 3 = Aircraft identification, Category Set B 4 = Aircraft identification, Category Set A
3	MSS	ADS-B Aircraft Emitter Category coding:
4	CHARACTER 1	Set A 0 = No ADS-B Emitter Category Information 1 = Light (< 15000 lbs) 2 = Small (15000 to 75000 lbs) 3 = Large (75000 to 300000 lbs) 4 = High-Vortex Large (aircraft such as B-707) 5 = Heavy (> 300000 lbs) 6 = High Performance (> 5g acceleration and 400 kts) 7 = Rotorcraft
5	LSS	
6	MSS	
7	CHARACTER 2	
8	LSS	Set B
9	MSS	0 = No ADS-B Emitter Category Information 1 = Glider / sailplane 2 = Lighter-than-air 3 = Parachute / Skydiver 4 = Ultralight / hang-glider / paraglider 5 = Reserved 6 = Unmanned Aerial Vehicle 7 = Space / Transatmospheric vehicle
10	CHARACTER 3	
11	LSS	
12	MSS	
13	CHARACTER 4	
14	LSS	Set C
15	MSS	0 = No ADS-B Emitter Category Information 1 = Surface Vehicle - Emergency Vehicle 2 = Surface Vehicle - Service Vehicle 3 = Point Obstacle (includes tethered balloons) 4 = Cluster Obstacle 5 = Line Obstacle 6 = Reserved 7 = Reserved
16	CHARACTER 5	
17	LSS	
18	MSS	
19	CHARACTER 6	Set D (Reserved)
20	LSS	
21	MSS	
22	CHARACTER 7	
23	LSS	Aircraft Identification coding: Character coding as specified in §C.2.3.4.
24	MSS	
25	CHARACTER 8	
26	LSS	

14.6 NL lookup table

Table A-21. Look-Up Table for Number of Longitude Zones, NL.

Condition	Transition Latitude		Number of Longitude Zones, NL	
	Degrees (decimal)	32-bit AWB (hexadecimal)		
If $ \text{lat} <$	10.47047130	07 72 17 54	Then $\text{NL}(\text{lat}) =$	59
Else if $ \text{lat} <$	14.82817437	0A 8B 63 03	Then $\text{NL}(\text{lat}) =$	58
Else if $ \text{lat} <$	18.18626357	0C EE B5 50	Then $\text{NL}(\text{lat}) =$	57
Else if $ \text{lat} <$	21.02939493	0E F4 4B D6	Then $\text{NL}(\text{lat}) =$	56
Else if $ \text{lat} <$	23.54504487	10 BE 3E 9F	Then $\text{NL}(\text{lat}) =$	55
Else if $ \text{lat} <$	25.82924707	12 5B 12 29	Then $\text{NL}(\text{lat}) =$	54
Else if $ \text{lat} <$	27.93888710	13 DB 23 2C	Then $\text{NL}(\text{lat}) =$	53
Else if $ \text{lat} <$	29.911135686	15 45 32 43	Then $\text{NL}(\text{lat}) =$	52
Else if $ \text{lat} <$	31.77209708	16 97 E7 0B	Then $\text{NL}(\text{lat}) =$	51
Else if $ \text{lat} <$	33.53993436	17 D9 C2 3B	Then $\text{NL}(\text{lat}) =$	50
Else if $ \text{lat} <$	35.22899598	19 0D 3B 35	Then $\text{NL}(\text{lat}) =$	49
Else if $ \text{lat} <$	36.85025108	1A 34 62 2C	Then $\text{NL}(\text{lat}) =$	48
Else if $ \text{lat} <$	38.41241892	1B 50 C4 78	Then $\text{NL}(\text{lat}) =$	47
Else if $ \text{lat} <$	39.92256684	1C 63 AE 77	Then $\text{NL}(\text{lat}) =$	46
Else if $ \text{lat} <$	41.58651832	1D 6E 2F 8C	Then $\text{NL}(\text{lat}) =$	45
Else if $ \text{lat} <$	42.80914012	1E 71 2A 89	Then $\text{NL}(\text{lat}) =$	44
Else if $ \text{lat} <$	44.19454951	1F 6D 5F 49	Then $\text{NL}(\text{lat}) =$	43
Else if $ \text{lat} <$	45.54626723	20 63 71 B6	Then $\text{NL}(\text{lat}) =$	42
Else if $ \text{lat} <$	46.86733252	21 53 F0 01	Then $\text{NL}(\text{lat}) =$	41
Else if $ \text{lat} <$	48.16039128	22 3F 54 B9	Then $\text{NL}(\text{lat}) =$	40
Else if $ \text{lat} <$	49.42776439	23 26 0C C7	Then $\text{NL}(\text{lat}) =$	39
Else if $ \text{lat} <$	50.67150166	24 08 77 22	Then $\text{NL}(\text{lat}) =$	38
Else if $ \text{lat} <$	51.89342469	24 E6 8B B0	Then $\text{NL}(\text{lat}) =$	37
Else if $ \text{lat} <$	53.09516153	25 C1 AD 0F	Then $\text{NL}(\text{lat}) =$	36
Else if $ \text{lat} <$	54.27817472	26 99 0A 48	Then $\text{NL}(\text{lat}) =$	35
Else if $ \text{lat} <$	55.44378444	27 60 3B A2	Then $\text{NL}(\text{lat}) =$	34
Else if $ \text{lat} <$	56.59318756	28 3B 79 B3	Then $\text{NL}(\text{lat}) =$	33
Else if $ \text{lat} <$	57.72747354	29 0C F7 42	Then $\text{NL}(\text{lat}) =$	31
Else if $ \text{lat} <$	58.84763776	29 DB E2 B2	Then $\text{NL}(\text{lat}) =$	30
Else if $ \text{lat} <$	59.95459277	2A A2 66 89	Then $\text{NL}(\text{lat}) =$	30
Else if $ \text{lat} <$	61.04917774	2B 69 A9 B5	Then $\text{NL}(\text{lat}) =$	29
Else if $ \text{lat} <$	62.13216659	2C 2E D0 D5	Then $\text{NL}(\text{lat}) =$	28
Else if $ \text{lat} <$	63.20427479	2D F1 FC B2	Then $\text{NL}(\text{lat}) =$	27

Condition	Transition Latitude		Number of Longitude Zones, NL	
	Degrees (decimal)	32-bit AWB (hexadecimal)		
Else if lati <	64.26616523	2D 83 4C 6D	Then NL(lati) =	26
Else if lati <	65.31845310	2E 72 DC BC	Then NL(lati) =	25
Else if lati <	66.36171008	2F 30 C7 D8	Then NL(lati) =	24
Else if lati <	67.398646734	2F ED 27 0C	Then NL(lati) =	23
Else if lati <	68.42322022	30 A8 11 2E	Then NL(lati) =	22
Else if lati <	69.44242631	31 61 9B A1	Then NL(lati) =	21
Else if lati <	70.45451075	32 19 DA 2E	Then NL(lati) =	20
Else if lati <	71.45986473	32 D0 DF 12	Then NL(lati) =	19
Else if lati <	72.45884545	33 86 BA F3	Then NL(lati) =	18
Else if lati <	73.45177442	34 3B 7C CB	Then NL(lati) =	17
Else if lati <	74.43893416	34 EF 31 C5	Then NL(lati) =	16
Else if lati <	75.42056257	35 A1 E4 FB	Then NL(lati) =	15
Else if lati <	76.39684391	36 53 9E FA	Then NL(lati) =	14
Else if lati <	77.36799461	37 04 65 38	Then NL(lati) =	13
Else if lati <	78.33374083	37 84 38 EB	Then NL(lati) =	12
Else if lati <	79.29428225	38 63 15 64	Then NL(lati) =	11
Else if lati <	80.24923213	39 10 ED 48	Then NL(lati) =	10
Else if lati <	81.19801349	39 BD A5 B3	Then NL(lati) =	9
Else if lati <	82.15956981	3A 69 CD 67	Then NL(lati) =	8
Else if lati <	83.07199445	3B 12 CB 8A	Then NL(lati) =	7
Else if lati <	83.99173563	3B 3A 3A 96	Then NL(lati) =	6
Else if lati <	84.89166391	3C 5E 0E 31	Then NL(lati) =	5
Else if lati <	85.75541621	3C FB 4C 0F	Then NL(lati) =	4
Else if lati <	86.53536998	3D 89 48 8A	Then NL(lati) =	3
Else if lati <	87.00000000	3D 00 00 DE	Then NL(lati) =	2
Else			NL(lati) =	1

14.7 Usability questionnaire

14.7.1 Questionnaire

1. The website is intuitive and easy to use
 - a. Strongly Agree
 - b. Mildly Agree
 - c. Passive
 - d. Mildly disagree
 - e. Strong disagree
2. I understood all the information displayed on the website
 - a. Strongly Agree
 - b. Mildly Agree
 - c. Passive
 - d. Mildly disagree
 - e. Strong disagree
3. The website was easy to navigate
 - a. Strongly Agree
 - b. Mildly Agree
 - c. Passive
 - d. Mildly disagree
 - e. Strong disagree
4. I would use this website
 - a. Regularly
 - b. Occasionally
 - c. Once
 - d. Never
5. Did you feel there was information you wanted to know that wasn't available?
 - a. Yes
 - b. No
6. If Yes to the previous question, what information would you like displayed?
7. The website is too minimalist
 - a. Strongly Agree
 - b. Mildly Agree
 - c. Passive
 - d. Mildly disagree
 - e. Strong disagree
8. What additional features would you add if you had the choice?

14.7.2 Results

User	Question	Result
1	1	Strongly agree
2		Mildly agree
3		Strongly agree
4		Regularly
5		No
6		
7		No
8		Flight path displayed all the way from origin to destination Increased coverage of the map

2	1	Strongly agree
2	2	Strongly agree
3	3	Strongly agree
4	4	Occasionally
5	5	No
6	6	
7	7	No
8	8	Let me search for a plane that's on the map

3	1	Strongly agree
2	2	Strongly agree
3	3	Strongly agree
4	4	Occasionally
5	5	No
6	6	
7	7	No
8	8	Let me browse the flights from the database in a text format rather than having to specify a flight

4	1	Mildly agree
2	2	Mildly agree
3	3	Passive
4	4	Once
5	5	Yes
6	6	More information about the aircraft such as capacity, number of engines etc.
7	7	No
8	8	Add more information to the full info page Find a way to have the planes updating more often

5	1	Strongly agree
2	2	Strongly agree
3	3	Strongly agree
4	4	Occasionally
5	5	No
6	6	
7	7	No
8	8	No

6	1	Strongly agree
2	2	Strongly agree
3	3	Strongly agree
4	4	Occasionally
5	5	No
6	6	
7	7	No
8	8	No