A graphic tool for air traffic control

Flight simulation of historic traffic

Corina Yanes Yanes

Grado en Ingeniería Aeroespacial

Escuela Técnica Superior de Ingeniería de Diseño

Universidad Politécnica de Valencia

Tutor: Joan Vila Carbó

Julio 2018

PAGE INTENTIONALLY LEFT IN BLANK

Abstract

The aim of the present project is to provide a graphical representation tool for traffic simulation. This tool is supposed to be able to simulate historic traffics from .so6 files provided by Eurocontrol. The purpose is to visualize and analyze Merge Point approaches and to provide the basis of a future tool for air traffic controllers which includes a relative position indicator. In the first part of this work, the objectives and program requirements for the tool to develop are specified. The second part gives the theoretical background of the Merge Point approaches and lists its advantages based on recent studies. For the total understanding of Merge Point and to provide some background knowledge, the concept of air space management is introduced and airspace structure and the kind of holdings that exist depending on the navigation method used (traditional or RNAV) are explained. To give an example, the Merge Point approach of Dublin Airport for runway 28 is described in more detail, as the same example is used as well for the use case in the third part of this work. All the tools and sources used for the development of the program are also introduced in this second part, including a short introduction in object oriented programming since the language used for the development of the program is Java 8. Finally, in the third part, the developed tool is explained and class interaction visualized by UMLs. A use case of Dublin Airport runway 28 was carried out and is included at the end of the program illustration in order to demonstrate its functioning.

TABLE OF CONTENTS

PAGE INTENTIONALLY LEFT IN BLANK

LIST OF FIGURES

LIST OF TABLES

PAGE INTENTIONALLY LEFT IN BLANK

PART 1 - INTRODUCTION

1. Introduction

A current discussion about the increment of take-offs and landings during night hours in Germany reveals that in 2017 a new record of night flights was established with a total of 215.843 starts and landings. “On the one hand, the reason for the increase in night flights is the overall increase in air traffic. On the other hand, it is due to the inactivity of authorities who fail to do enough against increasing delays and early arrivals” [1].

Certainly, there is not one single approach for solving this kind of problems. Interventions need to be implemented at different levels and in collaboration with all affected entities. The Point Merge Approach can definitely contribute to the solution of problems caused by an increase of air traffic by absorbing a higher volume of traffic. This new procedure for sequencing arrival flows was developed by Eurocontrol in 2006 and first implemented in 2011 at Oslo airport. There are several studies which show the advantages of Point Merge. And moreover, the advantages are not reduced to higher traffic capacity but also allow continuous descent, reduced holding, fuel saving and noise reduction. Studies have also shown that safety can be increased with Point Merge procedures. Furthermore, it has various advantages from the controllers’ point of view.

The present work is composed of two main parts. A theoretical part with a revision of Point Merge, introducing all relevant concepts, and a practical part. The practical part focuses on the controller's point of view. A graphical tool has been developed to simulate historic traffic from Eurocontrol, a first approach to optimize the controller's monitor. Since controllers play a crucial role as far as safe and efficient operations are concerned, it is important to provide effective tools that help to improve the controllers’ performance. That is why in the traffic simulation tool a visual aid called relative position indicator (RPI) has been implemented. It indicates the time left for each aircraft to reach the Merge Point and so take the full advantage of Point Merge operations.

The project includes a use case to visualize the tool developed and to show its correct functioning as well as the possibility to make any suggestions in order to improve the RPI representation and the arrangement of the monitor.

2. Objectives

The objective of this project is to develop a graphical tool for traffic representation which can be used by air traffic controllers in the terminal area (TMA). The tool is mainly intended for use at TMAs with Merge Point approaches, in addition to the routes over which the aircrafts are represented. The tool is intended for providing a platform, open to introduce and test innovations in console of ATC.

This objective comprises some additional partial objectives:

Objective 1

A revision of holding patterns, with special emphasis in the simulation of the Merge Point approach. The objective is to carve out the advantages of the Merge Point pattern and to find approaches to tap the full potential of this procedure.

Objective 2

The decoding and simulation of historical traffic from .so6 files provided by Eurocontrol data repository. It can be used for traffic analysation and serves as a first approach to the final design.

Objective 3

A graphical environment for the simulation of traffics where new features for helping the air traffic controller can be tested. In this project, a Relative Position Indicator (RPI) is included on the controller's console, indicating for each aircraft the *time* left until reaching the specified Merge Point.

The project includes the presentation of a use case. The objective of this use case is not only to show the correct functioning of the program but also to demonstrate the design of the GUI and the way information is represented. This allows to comment requests concerning design changes or remarks about any missing information. For the use case Dublin Runway 28 was chosen.

As the development of such a tool needs to undergo various validation and verification processes and due to the critical environment of implementation, such a tool cannot be developed in one single step. Therefore, in this project only the first part of the whole project is developed and presented.

3. Program Requirements

The global requirement of the project is providing a tool that provides:

- Time simulation of historical traffic provided by Eurocontrol: at each instant of time it should provide a list of flights with their positions and the time left to reach a specified point. It can provide additional information of interest.
- Traffic position will be expressed as a 4D point with longitude and latitude in degree decimal, flight level expressed in feet and time in seconds.

These global requirements convey the following partial requirements.

File Decoding and Information Input

- The tool should decode .so6 files from Eurocontrol to get the data of real historic traffic.
  - It should filter the traffics in the .so6 file and only store the information after certain (indicated) waypoints.
- The information about the routes to simulate that is given in the AIP of the RNAV standard arrival chart should be provided to the program.
- Information about Initial Fix and Merge Point for different approaches can be provided to the tool to facilitate simulation parameter input.
- Documents with relevant information should be stored in a way easy to access in order to facilitate file addition and modification.

User Interface (GUI)

- The user interface should be designed to process historic traffic, simulated as well as real traffic.
- The program should show the routes of the approach (defined in the AIP).
- Over the routes, the program should provide a graphical representation of the position of all flights in the air at the time simulated.
- The flights should be identified by their callsigns.
- The graphic interface should include a Relative Position Indicator (RPI).
- The time of simulations should be shown to the user.

- The program should provide detailed information of the flights simulated if required.

Simulation parameter input

- The following data should be provided / selected by the user:
    - name of .so6 file from Eurocontrol with the historic traffic of one day
    - list of waypoints to filter the traffics
    - name of Merge Point to make the time reference
    - timeframe to simulate
    - desired simulation speed
    - the name of an excel file with the desired route data
- The tool should inform the user about the following errors in the input data:
    - no .so6 file chosen for simulation
    - no Merge Point indicated
    - the indicated Merge Point does not correspond to the .so6 file chosen (it is not included in the file)
    - no waypoints to filter indicated
    - none of the waypoints is included in the .so6 file
    - simulation start time is set after end time
- The program should not start if there is any error in the simulation parameters.

Communication Protocols and Interfaces

- The program should implement two types of interfaces: a programmatic interface and a graphical interface. The interfaces are important to implement for future usage and development of the program.
    - Programmatic interface
        - The following attributes of traffic will be provided by the programmatic interface:
            - 4D Point
            - velocity
            - status (climb, descent or levelled)

        - time left to reach Merge Point
    - Graphical interface
        - A graphical interface should be implemented in the program which provides all the simulation parameters and traffics for simulation in each instant.

Constrains and Limitations

- The traffic that can be simulated is reduced to approaches.
- The simulation is reduced to one approach (one Merge Point).

PART 2 – BACKGROUND AND TOOLS

4. Air Traffic Management, ATM

Air Traffic Management can be defined as "…the dynamic and integrated management of air traffic and airspace, safely, economically and efficiently, through the provision of facilities and seamless services, in collaboration with all partners." (ICAO) [2]. ATM, along with the aeronautic information service and meteorological service, forms part of the air navigation services. ATM activities are targeted to ensure save and orderly flow of traffic to guide aircrafts safely through the sky and on the ground and to manage airspace in a way it can easily adapt to changing needs of air traffic [3]. These objectives are achieved in collaboration with partners like airlines or airport operators.

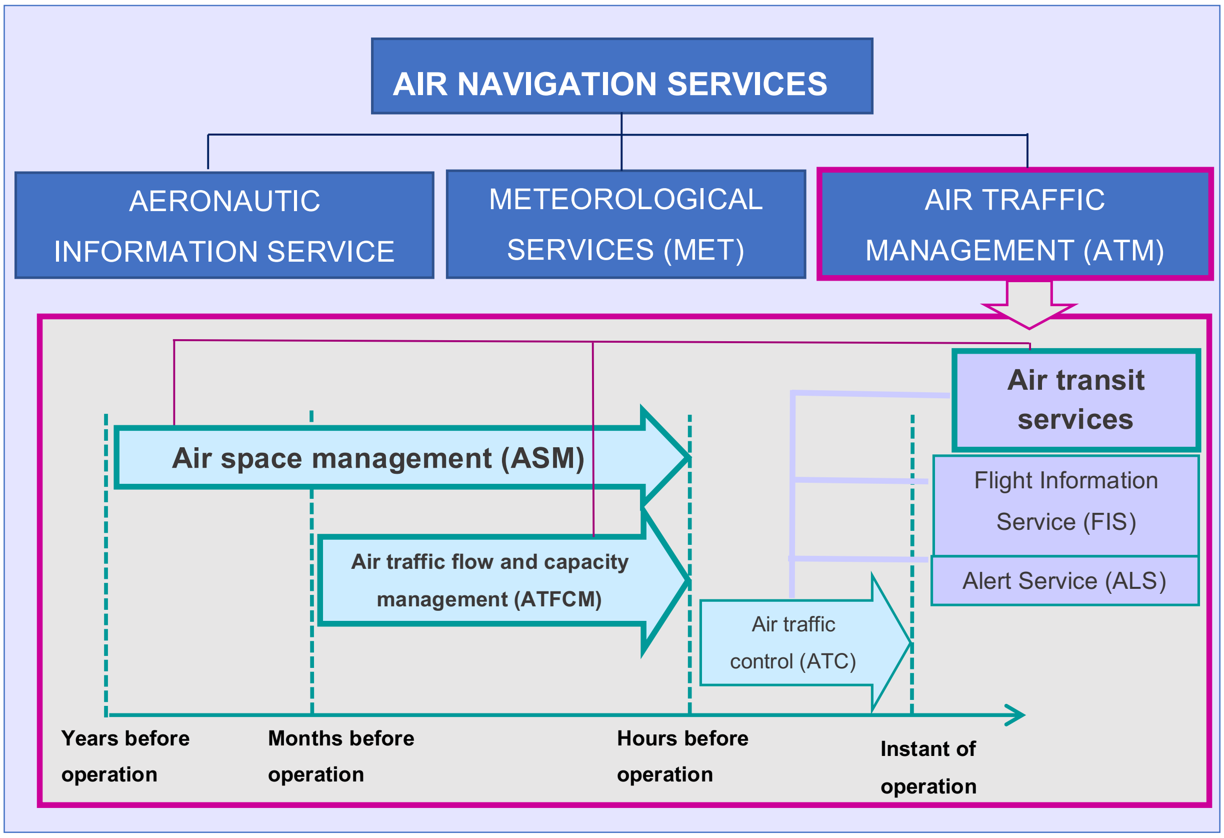

AIR NAVIGATION SERVICES
AERONAUTIC INFORMATION SERVICE
METEOROLOGICAL SERVICES (MET)
AIR TRAFFIC MANAGEMENT (ATM)
Air transit services
Air space management (ASM)
Flight Information Service (FIS)
Air traffic flow and capacity management (ATFCM)
Alert Service (ALS)
Air traffic control (ATC)
Years before operation
Months before operation
Hours before operation
Instant of operation

Figure 1. Air Navigation Services.

Activities can be divided into three main services covering different time phases: Airspace Management (ASM), Air Traffic Flow Management (ATFM) and Air Traffic Services (ATS) including Air Traffic Control (ATC) [4]. In figure 1 the time phases with the corresponding services are illustrates.

4.1. Airspace Management

Airspace Management is "the coordination, integration, and regulation of the use of airspace of defined dimensions" [5]. It solves traffic capacity problems based on long-term planning where the structure of airspace is modified. It is performed at national and international level and can be divided into three different stages: Strategic, Pre-Tactical and Tactical. The *Strategic* level includes airspace design and airspace structure assignation to maximize efficient airspace use. In order to achieve this maximization during the strategic level the following activities are carried out [6]:

- the airspace structures including airport control zones and terminal areas, sectors and sector configurations for en-route services, proposals for airspace classifications, etc .
- standard arrivals, standard departures and instrument approach procedures to/from the airports
- high performance procedures to achieve higher traffic densities, RNAV, RVSM Reduced Vertical Separation
- air traffic services routes
- areas and zones with restricted air traffic, restricted areas, danger areas, prohibited areas, temporary segregated/reserved areas
- definition of restrictions and conditions for the use of the route network
- flexible use of airspace

In the *Pre-Tactical* stage, the established designs are planned and assigned in accordance to the users' needs. An Airspace Use Plan is developed, by which the planning and occupancy of airspace is defined.

Finally, at the Tactical level, the airspace structures are dynamically managed based on real-time usage as well as the Airspace Use Plan, making use of the Flexible Use of Airspace.

4.2. Air Traffic Flow Management

The purpose of ATFM is to balance traffic demand and available capacity to ensure safe densities of traffic by minimizing traffic surges. It is based on the clear definition of capacities and the analysis of predicted traffic flows. Therefore, an efficient exchange of information including flight plans, airspace availability and capacity is essential [7]. Air Traffic Flow management similar to ASM is divided in three phases. During the Strategic phase, taking place up to two days before the operation, traffic forecasts are made based on the flight plans obtained, statistics and simulations. Together with capacity information of each sector, systematic capacity deficits can be detected as well as those able to receive additional traffic. In the Pre-Tactical phase, during the two days before the operation takes place, capacities and demand are studied and those areas are identified where demand is likely to exceed capacity. Operators and ATS are informed about possible regulations for the following day. Finally, in the Tactical phase - which is the same day of operation - ATFM calculates where an aircraft will be at any given moment during its operation and checks that the controllers in that airspace can safely cope with the flight and there will be no overload. If they cannot cope with the flight, the aircraft has to wait on the ground until it is safe to take off [8], pp GEAII.

4.3. Air Traffic Service

This service is provided hours before and during the operation. It requires radiotelephony and/ or data link of RCP type, assuring continuity, availability and integrity. Included in ATS are [9]:

- Air Traffic Control: During the flight and on ground ATC takes care of a save separation between aircrafts, including Tower Control at airports and Air Traffic Control Centres. The objective is to prevent collisions between aircrafts and to maintain an orderly flow of air traffic.

- Flight Information Service: The objective is to provide advice and information useful for the performance of a safe an efficient operation. The FIS provides information and updates related to safety, navigation, technical, administrative or legal topics aimed to improve safety, regularity and efficiency. It also informs about meteorological and atmospheric conditions which may affect safety. This information can be distributed in different ways, for example via maps, notices or publications.
- Alert Service: The ALS objective is to inform authorities about aircraft in need of search and rescue and to support the responsible organisation as required. It informs about aircraft emergencies or unlawful interferences, differentiating the actuation required in *Uncertainty*, *Alert* and *Distress*.

5. Airspace Structure

Eurocontrol defines airspace as "a defined three dimensional region of space relevant to air traffic" [10]. Due to the continuously increasing volume of air traffic, this airspace needs to be structured and classified. The existing classifications by ICAO structure airspace relative to sovereignty, responsibility and information service as well as control service.

5.1. Worldwide Air Navigation Regions

The need for a division in Worldwide Air Navigation Regions goes back to the 40's of last century, where standards and procedures for safe and efficient operations of international air services were not shared by all areas of the world. That is why in 1945 ICAO established ten air navigation regions with equivalent aerial development, infrastructures and traffic problems. These regions were reduced to a number of eight in 1954 [11].

Over time, the regional character of operational and technical problems was lost. Today, the regional division of airspace serves for the planning of facilities and services needed for the international air routes network. It provides a starting point for regional air navigation meetings and the basis for facilities and service distribution. Summarizing, its goal is supporting the provision of air services and planning the implementation of essential ground facilities for international air transport [11].

The air navigation regions are visualized in figure 2 and an official map from ICAO is represented in figure 3. The eight regions with their corresponding identification are the following:

- Africa-Indian Ocean (AFI) Region
- Asia (ASIA) Region
- Caribbean (CAR) Region
- European (EUR) Region
- Middle East (MID) Region
- North American (NAM) Region
- North Atlantic (NAT) Region
- Pacific (PAC) Region
- South American (SAM) Region

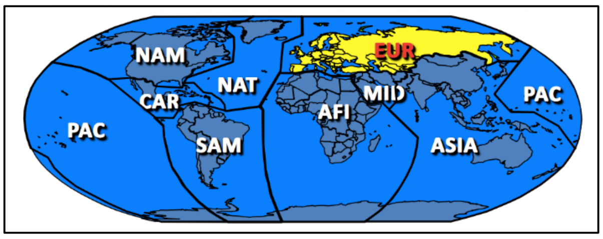


Figure 2. Worldwide Air Navigation Regions.

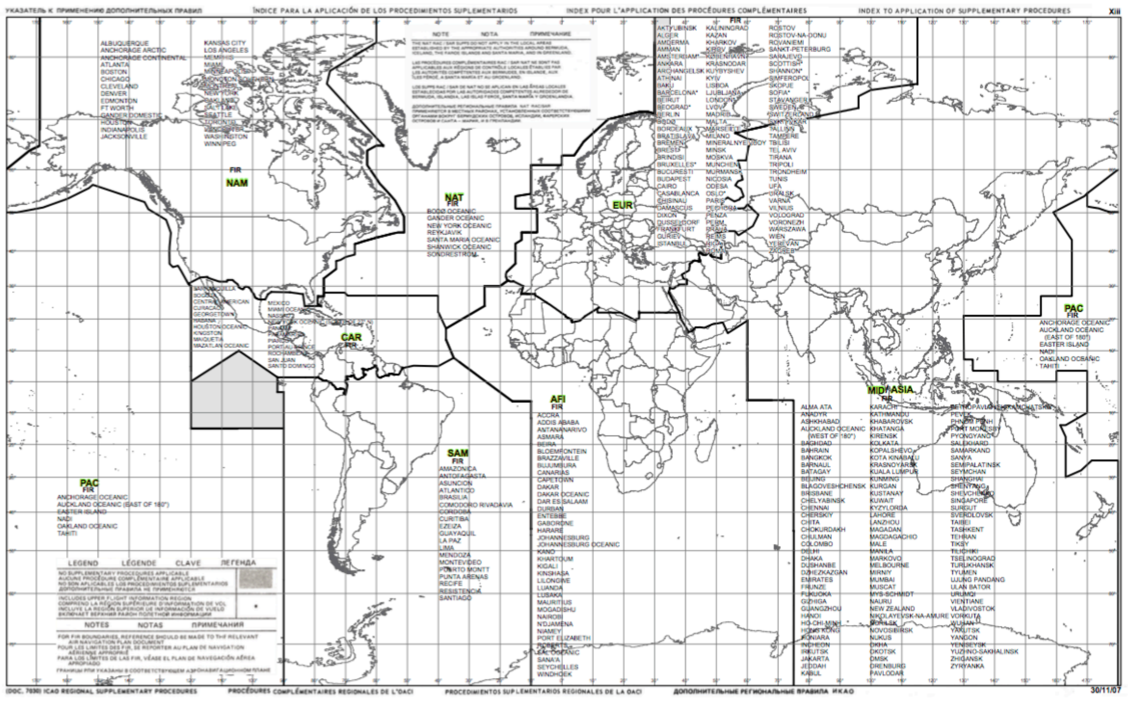


Figure 3. Worldwide Air Navigation Regions ICAO. *Source: ICAO*

5.2. Lower and Upper Airspace

The separation between lower and upper airspace was performed after the appearance of jets. In contrast to internal combustion engines, these are more effective when flying above FL245.

The lower airspace is arranged below a variable vertical limit. In most countries - including Spain - it is from ground level to FL245 (7300m). The airspace is controlled and includes airways linking the airport with upper airspace but excluding terminal or airport airspace [12], [9].

The upper part of the airspace is defined above a variable vertical limit. Its lower limit is the lower airspace and it is formed by upper airspace airways. It is a controlled airspace used mainly by jets in the cruise flight phase [13], [9].

5.3. Flight Information Regions and Upper Information Regions

ICAO divides all airspace around the world into areas of responsibility for providing Flight Information Services. These areas are called Flight Information Regions (FIR). The controlling authority in each region has the responsibility to ensure that air traffic services are provided to the aircraft flying within it [14].

FIRs may be splitted vertically into lower and upper sections.

- The lower section remains referred to as a FIR. In this region, Flight Information Services (FIS) and Alerting Service (ALS) are provided.
- The upper section is called Upper Information Region (UIR). In the UIR Flight Information Services are provided.

In the following two cards (graph 4 and 5), the FIR/UIR above the Eurocontrol Member States are shown. When comparing both maps, you can observe that the Upper- and Lower Airspace over Spain and Ireland are exactly the same, meaning that the FIR and UIR regions coincide, whereas the ones over France for example are different.

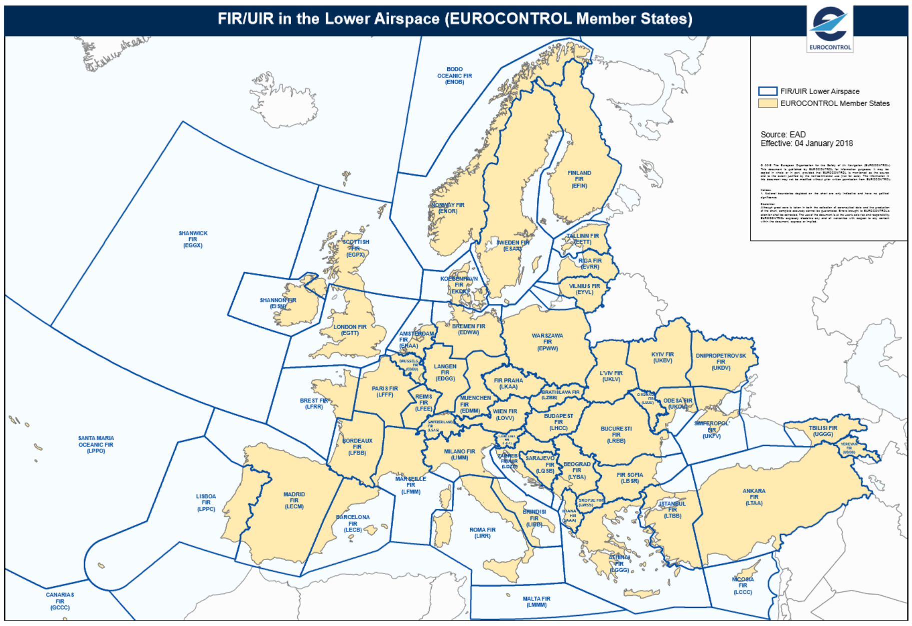


Figure 4. FIR/UIR Lower Airspace. Source: http://www.eurocontrol.int/sites/default/files/content/documents/nm/cartography/04012018-firuir-lower-airspace-ectl.pdf

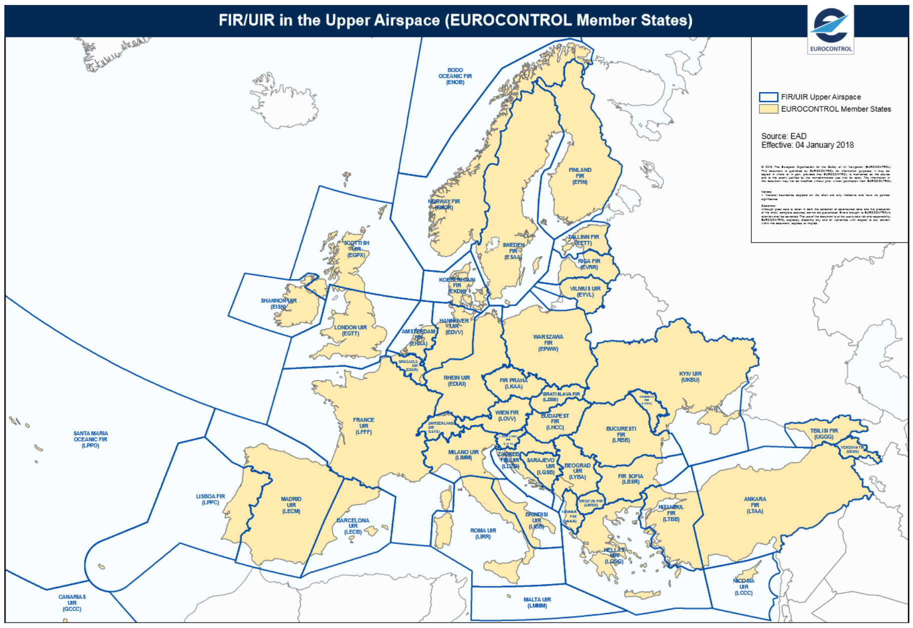


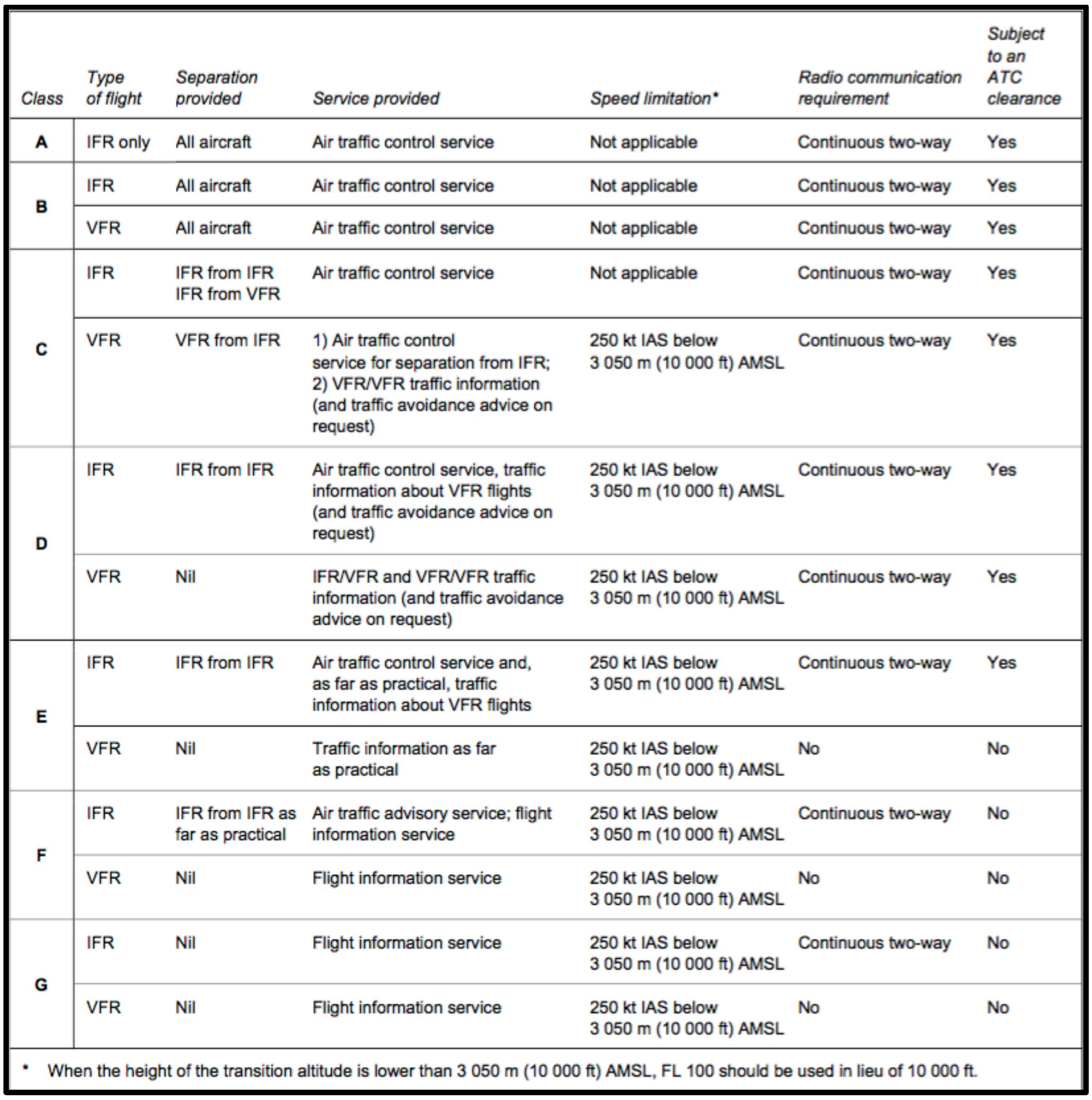
Figure 5. FIR/UIR Upper Airspace. Source: http://www.eurocontrol.int/sites/default/files/content/documents/nm/cartography/04012018-firuir-upper-airspace-ectl.pdf

5.4. ICAO airspace classes

Airspace within a FIR/UIR is divided into areas that vary in function, size and classification. Depending on the classification, the rules for flying within the specified airspace and whether it is ‘controlled’ or ‘uncontrolled’ vary. Information

Regions are uncontrolled airspaces, where FIS and ALS services are provided. In controlled airspace, in addition to FIS and ALS, air traffic control services are provided and aircraft flying in controlled airspace must follow instructions from air traffic controllers.

The airspace classes defined by ICAO (Annex 11, [15]) are resumed in table 1. Only in classes A and B separation for all flights (IFR and VFR) is provided by air traffic control service. Class C separates IFR from other IFR flights and VFR flights and VFR from IFR flights. In class D and E only IFR flights are separated from IFR flights. In class F and G air traffic control services are not available for any flight. In class F, air traffic advisory service is provided for IFR flights and in class F and G flight information service is available for all flights.



Class	*Type of flight*	*Separation provided*	*Service provided*	*Speed limitation**	*Radio communication requirement*	*Subject to an ATC clearance*
A	IFR only	All aircraft	Air traffic control service	Not applicable	Continuous two-way	Yes
B	IFR	All aircraft	Air traffic control service	Not applicable	Continuous two-way	Yes
	VFR	All aircraft	Air traffic control service	Not applicable	Continuous two-way	Yes
C	IFR	IFR from IFR IFR from VFR	Air traffic control service	Not applicable	Continuous two-way	Yes
	VFR	VFR from IFR	1) Air traffic control service for separation from IFR; 2) VFR/VFR traffic information (and traffic avoidance advice on request)	250 kt IAS below 3 050 m (10 000 ft) AMSL	Continuous two-way	Yes
D	IFR	IFR from IFR	Air traffic control service, traffic information about VFR flights (and traffic avoidance advice on request)	250 kt IAS below 3 050 m (10 000 ft) AMSL	Continuous two-way	Yes
	VFR	Nil	IFR/VFR and VFR/VFR traffic information (and traffic avoidance advice on request)	250 kt IAS below 3 050 m (10 000 ft) AMSL	Continuous two-way	Yes
E	IFR	IFR from IFR	Air traffic control service and, as far as practical, traffic information about VFR flights	250 kt IAS below 3 050 m (10 000 ft) AMSL	Continuous two-way	Yes
	VFR	Nil	Traffic information as far as practical	250 kt IAS below 3 050 m (10 000 ft) AMSL	No	No
F	IFR	IFR from IFR as far as practical	Air traffic advisory service; flight information service	250 kt IAS below 3 050 m (10 000 ft) AMSL	Continuous two-way	No
	VFR	Nil	Flight information service	250 kt IAS below 3 050 m (10 000 ft) AMSL	No	No
G	IFR	Nil	Flight information service	250 kt IAS below 3 050 m (10 000 ft) AMSL	Continuous two-way	No
	VFR	Nil	Flight information service	250 kt IAS below 3 050 m (10 000 ft) AMSL	No	No

* When the height of the transition altitude is lower than 3 050 m (10 000 ft) AMSL, FL 100 should be used in lieu of 10 000 ft.

Table 1. ICAO airspace classes.

5.4.1. Controlled Airspace

Controlled airspace covers ATS airspace classes A, B, C, D and E and is divided into different areas with specific ATC requirements. It is differentiated between Controlled Area (CTA) – an area which does not touch the ground - and Controlled Airports, areas which touch the ground. Both are again divided into different areas and zones, as resumed in table 2:

Controlled Traffic Area	In route airspace	Oceanic Control Area	OCA / UTA	UTA: Upper Terminal Area. Controlled area in upper airspace. AWY: Route in the air in the form of corridor defined with segments or legs between fixes.
		Airways	AWY / UTA	
	Terminal area	Terminal Maneuvering Area	TMA / CTA	CTA: It usually spans over several close airports whose dimensions or traffic volume do not justify separate TMA. TMA: junction of several airways; the ones in the lower airspace are inside the TMA .
Controlled Airports		Controlled Traffic Region	CTR	Associated to an aerodrome intended to protect the arrival and departure paths of controlled IFR flights and holding patterns. It includes the departures and final approach areas. They contain some corridors for VFR flights in their final approach to the aerodrome traffic circuit and corridors for overflights (of class G).
		Aerodrome Traffic Zone	ATZ	It is contained in the CTR and delimits the airspace where the tower is in control of the aircraft flying in the aerodrome traffic circuit. Used when VFR air traffic is intense.

Table 2. Controlled airspaces and their ATC requirements.

Each of the previously defined areas and zones is controlled by a specific air traffic control unit. The en route airspace which includes all the airways is controlled by Area Control Centers (ACC). The Approach control service for TMAs is provided by the Approach and Terminal Area Control Center if available. Otherwise ACC or the Aerodrome Tower (TWR) which also controls the CTR, offer control in the TMA. Once the aircraft is on the ground, Ground Movement Control (GMC) is responsible for aircraft guidance (see graph 6).

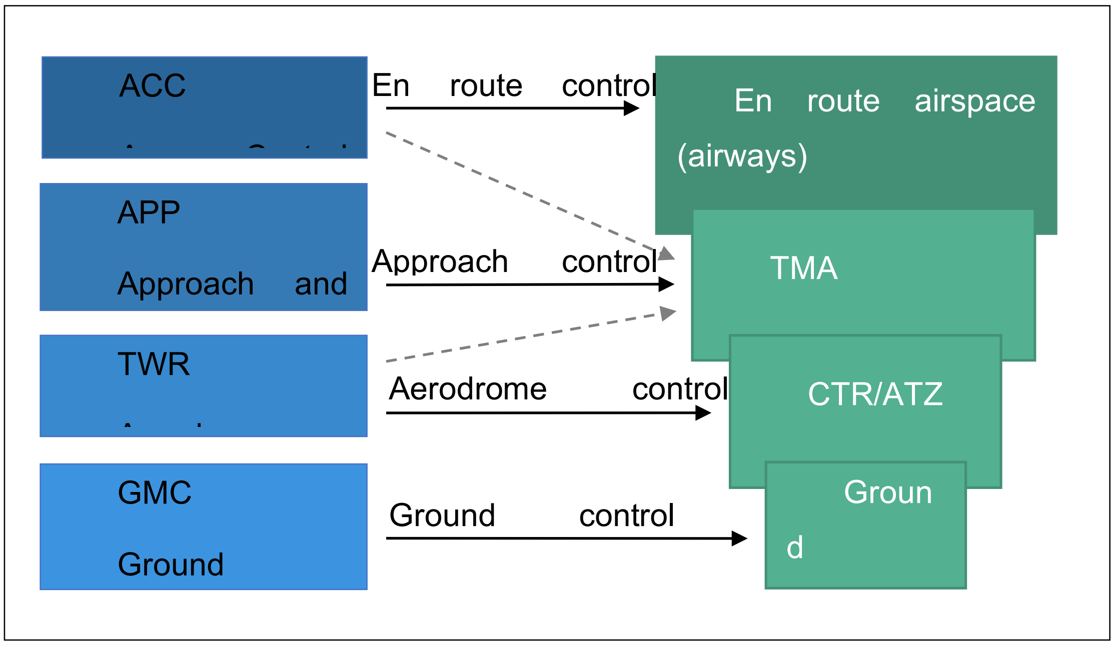


Figure 6. Air traffic control units.

Graph 7 shows once more the different control centers related to the corresponding flight phase they are responsible for. Graph 8 places the different control areas in space, indicates the corresponding class as well as the altitudes for each.

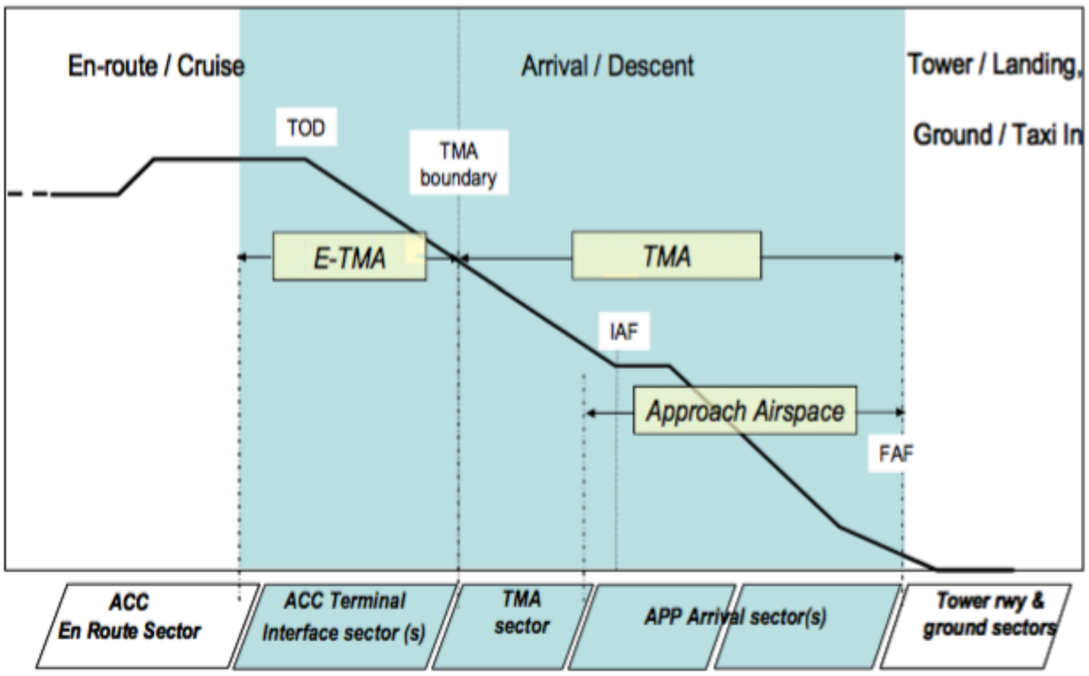


Figure 7. Control centres related to flight phases.

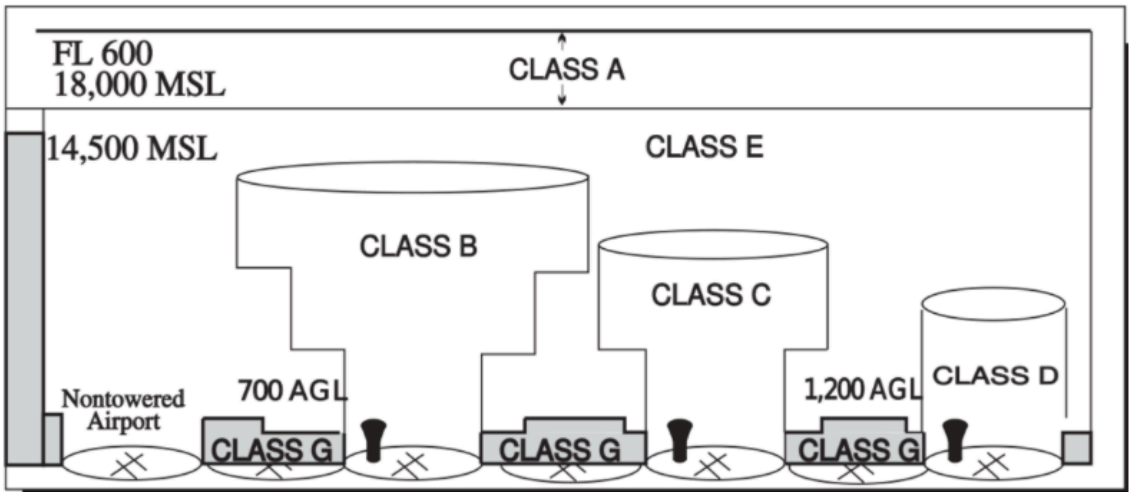


Figure 8. Control areas with classes and altitudes.

6. Holding Patterns

Holding patterns are used to delay an aircraft by flying a predetermined manoeuver keeping it within a specified airspace. They can be requested by the pilots or used when ATC needs to delay the progress of a flight due to different reasons which can be an emergency at the airport, weather conditions or runway unavailability preventing aircrafts from landing. The most common reason for holding is caused by too much traffic when an airport has attained its traffic capacity and aircrafts are required to hold in order to maintain a safe and orderly flow of traffic. Hence, holding is especially important for arrivals at congested airports.

6.1. Holding Pattern IFR

For instrumental flight rules (IFR) the standard holding pattern is performed with right turns, left turns in non-standard holding patterns. Each holding pattern begins and ends at a holding fix. Additionally, it is defined by a direction to hold from the fix defined by a bearing, course or radial. The distance to fly can be measured by timing or with a DME. Figure 9 shows the structure and elements of a standard holding pattern with right turn [16].

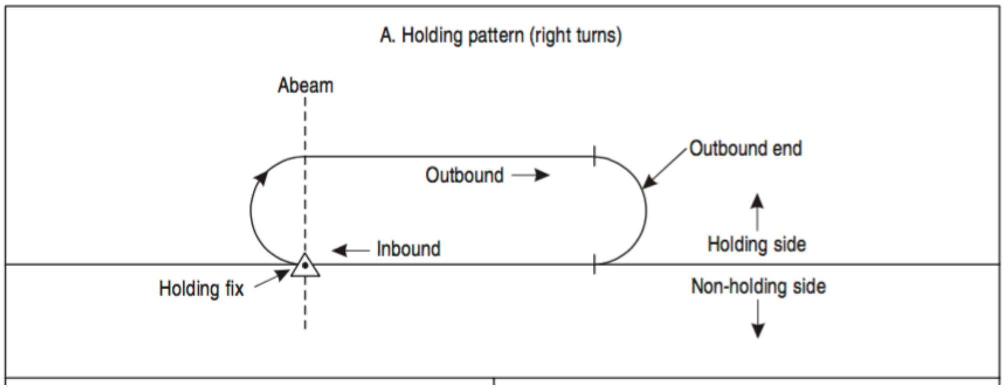


Figure 9. Holding pattern right turn. *Source: OACI 8168 Vol. II*

If there is more than one aircraft flying the same holding pattern, they are separated vertically by at least 1000 feet. This is called a vertical holding stack, where new arriving aircraft are added at the top and then spiraled down to be vectored into final approach [17]. Depending on the configuration of the airport and the runway on use, one airport can have more than one holding pattern. That can be seen in figure 10 where two vertical holding stacks can be observed in white color.

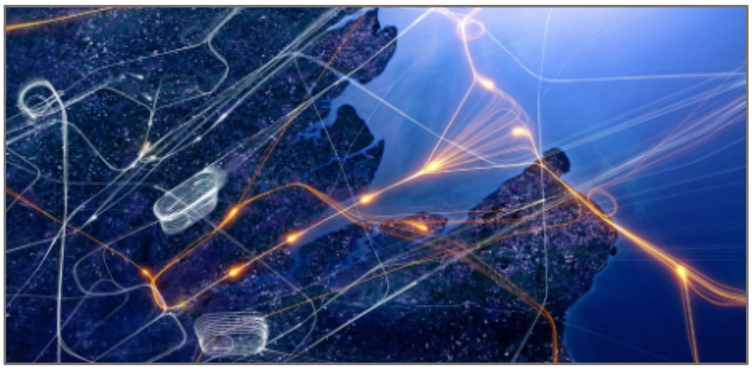


Figure 10. Holding stacks in South East England Airspace. Source: NATS [17]

In figure 10, additionally to the vertical holding stacks, a different kind of holding can be detected. This is a type of linear holding performed with RNAV (aRea NAVigation) which will be illustrated in the next section.

6.2. Linear RNAV Holding Patterns

Area navigation is a method allowing aircrafts to operate on any flight path without the need of a track directly to or from any specific radio navigation aid and that way optimizing air routes. They have been made possible by the huge variety of high performance RNAV systems based on ground- / space navigation aids, self-contained aids, or a combination of these. RNAV approaches, containing the analyzed holding patterns, are described by waypoints (3D artificial reference points), legs, speed and altitude constrains. These data are all stored on the onboard navigation database [18], [19]. Because of the lack of accuracy for terminal areas, Basic Area Navigation (B-RNAV), introduced for en route airspace, could not be used in the more complex terminal areas. The development of Precision Area Navigation (P-RNAV) with a track-keeping accuracy of ± 1nm made a re-design of terminal area airspace possible, changing terminal operation procedures [20]. During the past years, more and more terminal P-RNAV arrival routes have been defined, replacing Standard Terminal

Arrival Routes (STAR) with given vectors to guide aircraft to the runway when leaving the STAR and entering the terminal area [21]. The change of arrival procedures implicated the modification of holding patterns. The traditional vertical holding stacks were replaced by linear holdings. There are two different types of linear hold, trombone and Point Merge. In both procedures, all arriving aircrafts are kept on the same level. The separation occurs on the horizontal plane using satellite navigation tracks. The aircraft stays on the linear hold until it is vectored to the final approach. The traditional holding stacks still remain for use but only in exceptional circumstances. In figure 11 , the two linear holdings are represented [17]. In the following paragraphs, the two holding patterns are amplified, going into more detail and focusing on the Point Merge Holding.

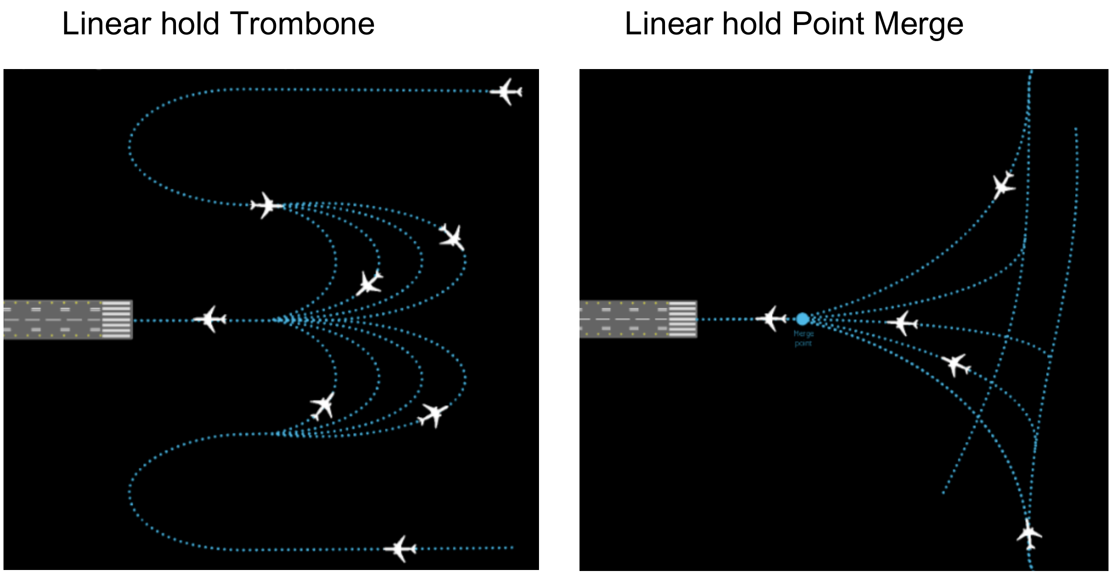


Figure 11. Linear holdings, Trombone and Point Merge. Source: [17]

6.2.1. Trombone Holding

In this holding pattern "Aircraft [are] sequenced by the timing of the turn into the final approach" [17]. The design of Trombone procedures includes an outbound leg, parallel and opposed to the final approach. The aircraft fly the outbound leg, then performing a turn to the inbound leg to reach the final approach. The instruction to turn is given by the air traffic controller. It can extend the outbound leg until there is enough spacing, incrementing the turn to base in half mile steps. That way, spacing conflicts can be solved in combination with speed reductions. As this procedure can extend the outbound leg for various nautical miles, it has the capability of absorbing large amounts of delay.

Examples of airports with Trombone holding patterns are for instance Hartsfield-Jackson Atlanta International Airport or Paris Charles De Gaulle Airport.

6.2.2. Merge Point Holding

This holding pattern can be designed in many different ways. Considering its configuration factors like airspace and ground configuration, capacity and efficiency need to be taken into account. All designs are composed of one Merge Point, that is the point where all possible routes encounter and which is the last waypoint before being guided to the IAF. Around this Merge Point, placed at iso-distance from the same, are sequencing legs forming an envelope of possible paths to the Merge Point (figure 12). Merging is achieved by a "direct to" instruction, given to the pilot, and telling him to fly directly to the Merge Point.

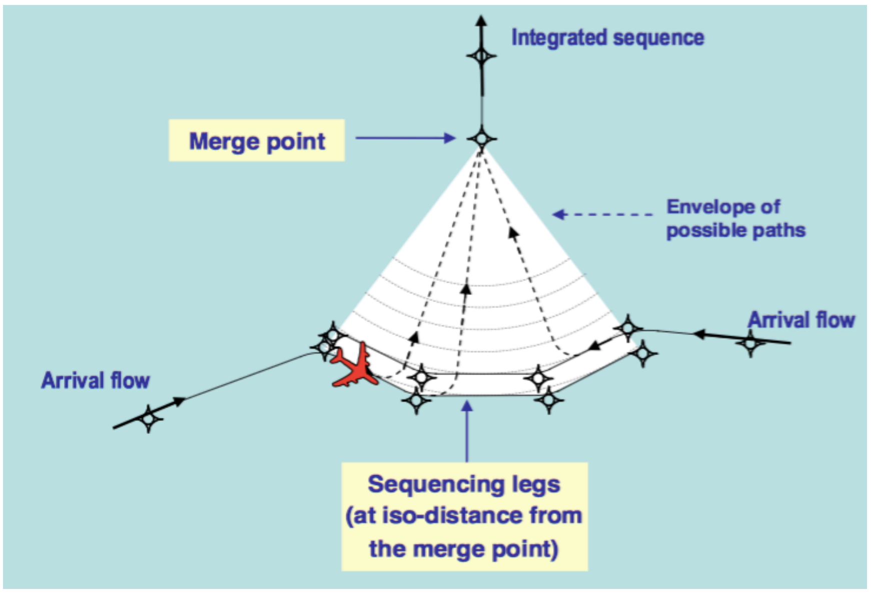

Integrated sequence
Merge point
Envelope of possible paths
Arrival flow
Arrival flow
Sequencing legs (at iso-distance from the merge point)

Figure 12. Point Merge Concept.   Source: Eurocontrol [22]

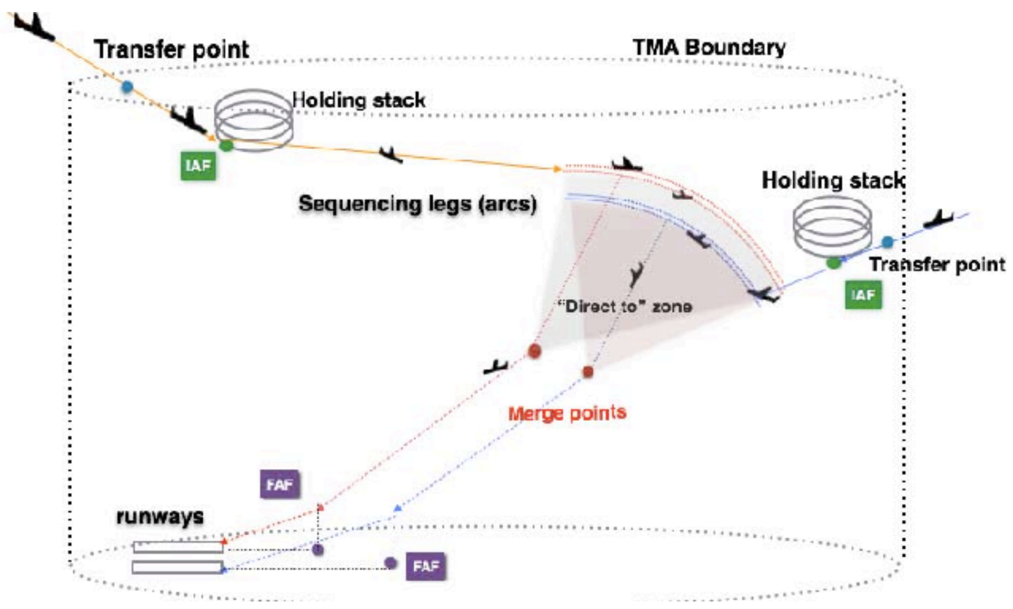


Figure 13. Point Merge located inside the TMA.

Graph 13 shows how the Merge Point is embedded in the TMA. It can also be observed that traditional holding stacks are placed at the IAF, whose usage should be kept as low as possible.

Point Merge arrivals are of special interest for this work, as they will be analyzed in detail and furthermore they are object of the use case in part 4 of this work. In the next chapter, the Point Merge arrivals will be described in more detail and studies of this recent arriving method will be analyzed.

7. Merge Point Pattern

The Merge Point procedure was developed by the Eurocontrol Experimental Centre (EEC). Its aim is to merge aircraft arrival flows improving arrival operations based on P-RNAV. Point Merge operations allow continuous descent approaches, contributing to a fuel saving procedure in the terminal area [22]. It also allows to reduce holdings and controller workload. All these advantages of Merge Point will be analyzed and proven in more detail in the subsequent section.

7.1. Studies of Point Merge Operations

Many studies have been carried out based on real scenario simulations to show the advantages and disadvantages of Point Merge procedures. In the following subchapters, the results of the here presented studies will be summarized.

- Merging arrival flows without heading instructions [23]

This paper, presented at the 7th USA/Europe Air Traffic Management R&D Seminar in 2007, summarizes the results of a series of small-scale experiments to get an initial overview of its benefits and limits. Three experiments were performed with the purpose of improving the method and another one to collect data and compare to the traditional procedure.

- Real Time Simulation Dublin TMA2012 Phase 2 [24]

This study was carried out in March 2010 at the EUROCONTROL Experimental Centre. It is a real-time simulation (RTS) based on the implementation of a Point Merge System in Dublin TMA. A total of 10 controllers participated in this study, performing scenario based exercises and reporting their experience in form of questionnaires. The study was based on the framework of the Dublin TMA2012 carried out by the Irish Aviation Authority.

- PMS-TE simulations joint study DSNA and EUROCONTROL [25]

The study, a series of real time simulations, was conducted jointly by DSNA (Operations Directorate and Paris ACC) and EUROCONTROL (Network Development Pillar and Experimental Centre) in the period between 2009 and 2010. The aim of this study was to investigate the potential applicability, benefits and limitations of Point Merge for ACC arrival sectors.

7.1.1. Controller side

After the simulation, controllers indicated a higher perception of awareness of the traffic situation in their sector under Point Merge Operations attributed to the predictability of traffic flows. All controllers coincided, that Point Merge

Operations allowed a more structured way of working coming along with low work load. Nevertheless, some of them expressed concern about the loss of vectoring skills when working with Point Merge approach [24].

In the third study presented, controllers reported as main benefits [25]:

- easiness and robustness of the procedure
- better and clearer division of tasks and roles
- reduction of workload and communication
- enhanced safety and capacity
- better delivery to approach and a better view of the arrival sequence

Limitations identified were:

- sensitivity to vertical aspects with the need to strictly respect levels on the legs (analogy with holding patterns often highlighted) and compatibility with existing receiving conditions (aircraft sometimes transferred late and at high altitude)
- sequencing of the secondary arrival flow not always intuitive or optimized

From a service provider perspective, other benefits were identified:

- better airspace management (best use of available airspace, clear determination of airspace capacity)
- continuous initial descent (although level-offs along the legs)
- no need for any system modification or new technology

7.1.2. Controller Activity and Workload

All studies came to the conclusion that workload was reduced with Point Merge Operations. Coordination between sectors was reduced and simplified as traffic flows became more predictable. Fewer messages, clearer and better task distribution and less instructions were reported. Figure 14 compares instructions between Baseline and Point Merge operations. While Heading Direct and Level instructions decrease with Point Merge, speed instructions are slightly higher as the task of the APC essentially consisted in achieving homogeneous speeds when aircraft join the sequencing legs. There were also differences between the operational designs [23], [24], [25].

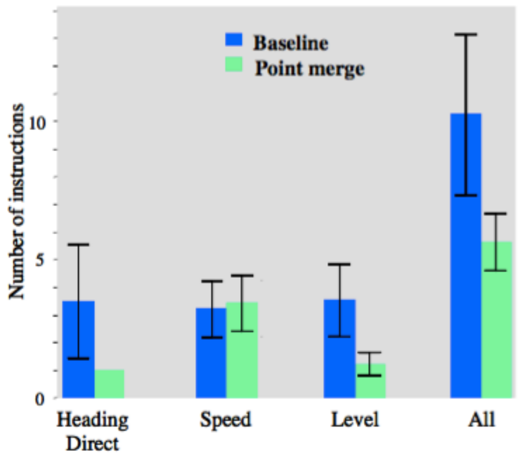


Figure 14. Instructions per aircraft.

In post-simulation self-assessment rations, controllers indicated that workload was rated significantly higher in Vectoring conditions in Approach, as shown in figure 15 [26].

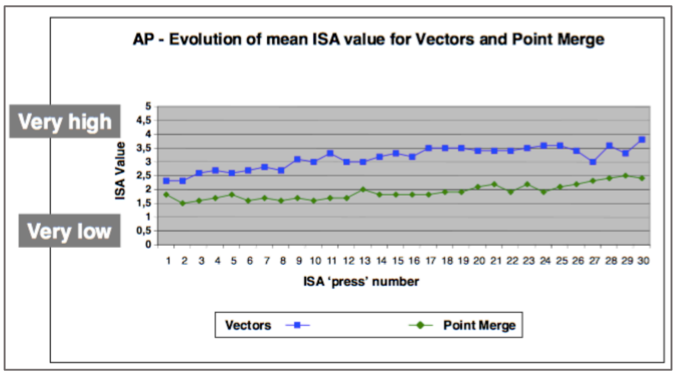


Figure 15. AP - Evolution of mean ISA value for Vectors and Point Merge.

7.1.3. Capacity

The simulation analysis indicated an increase in the sector's capacity limit of 40%. This increase could be observed on both sides, the controllers' and the system's. Up to 50 movements per hour could be achieved and still the controllers' workload was reported to be low [24]. In a final post-simulation questionnaire controllers saw a potential for capacity increase due to the gain in comfort and frequency load [25].

7.1.4. Descent Profiles - Continuous Descent Operation

A posterior analysis of descent profiles (figure 16) for both methods (Point Merge (green) and traditional (blue)) shows a difference between both. Due to the increased predictability of aircraft trajectories for Point Merge, aircraft are able to remain somewhat higher when flying from the legs to the final approach fix (FAF). In this study (in 2007) controllers commented that descent could be better managed if the flight crew knew the distance left and that the structure could be improved by placing the legs at higher altitudes (FL100 or FL120) to allow aircraft to perform a continuous descent until the ILS [23].

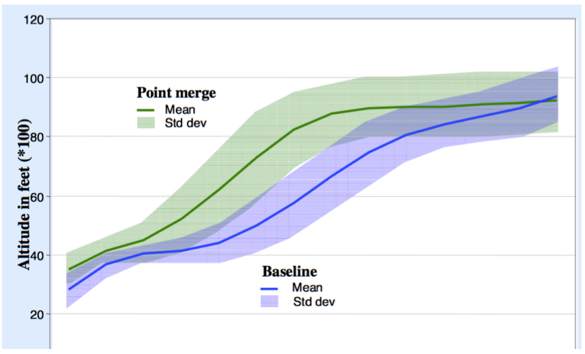


Figure 16. Descent profile for Baseline and Point Merge.

Step descents were no longer necessary to space and sequence the traffic. Besides, in later designs with sequencing legs at FL100-FL120, the performance of Continuous Descent Operation (CDO) from the level flown along the sequencing legs towards the ILS localiser could be achieved [24].

7.1.5. Reduced Holding

The study concerning the Point Merge System in Dublin revealed a significant decrease of holding aircraft in Point Merge Operations (figure 17).

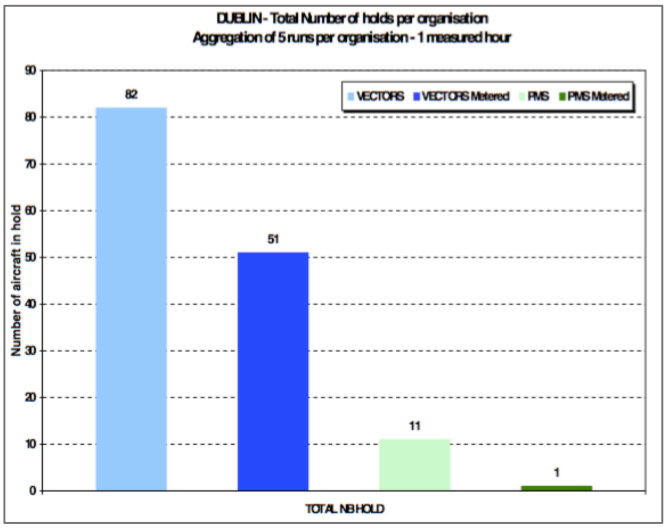


Figure 17. DUBLIN - Total Number of holds in 1 measured hour.

7.1.1. Fuel saving and noise

Regarding the potential of fuel saving and noise issues, David Curtis, Head of future ATM and Policy of NATS, comments that “the big difference is that these linear holds can be much higher than a traditional stack, potentially up to 20,000 feet, and are therefore quieter for people living underneath and more fuel efficient for the airlines” [27].

7.1.2. Safety

Safety could be improved with the Merge Point Procedure due to more anticipation, less workload and the reduced risks of misunderstandings as less communication is needed. This provides more time for conflict detection and resolution. From a controller’s point of view an improvement can be obtained, as trajectories were more structured and less dispersive reducing the probability of conflicts and that way increasing situational awareness and predictability [24]. Also, the objective perception of safety under Point Merge was rated as equal or increased by the controllers. They reported the capacity of safely and efficiently handling more traffic under Point Merge condition [24].

7.2. Airports with Point Merge arrivals

The first airport in implementing Point Merge Approaches was Oslo airport in 2011. Since then various airports followed, see figure 18. Today, Point Merge operations are implemented at three Norwegian regional airports (2014), Dublin (2012), Seoul (2012), Paris ACC (2013), Kuala Lumpur (2014), Lagos (2014), Canary Islands (2014), Hannover (2014), Leipzig (2015) as well as London City and Biggin Hill (2016) [28].

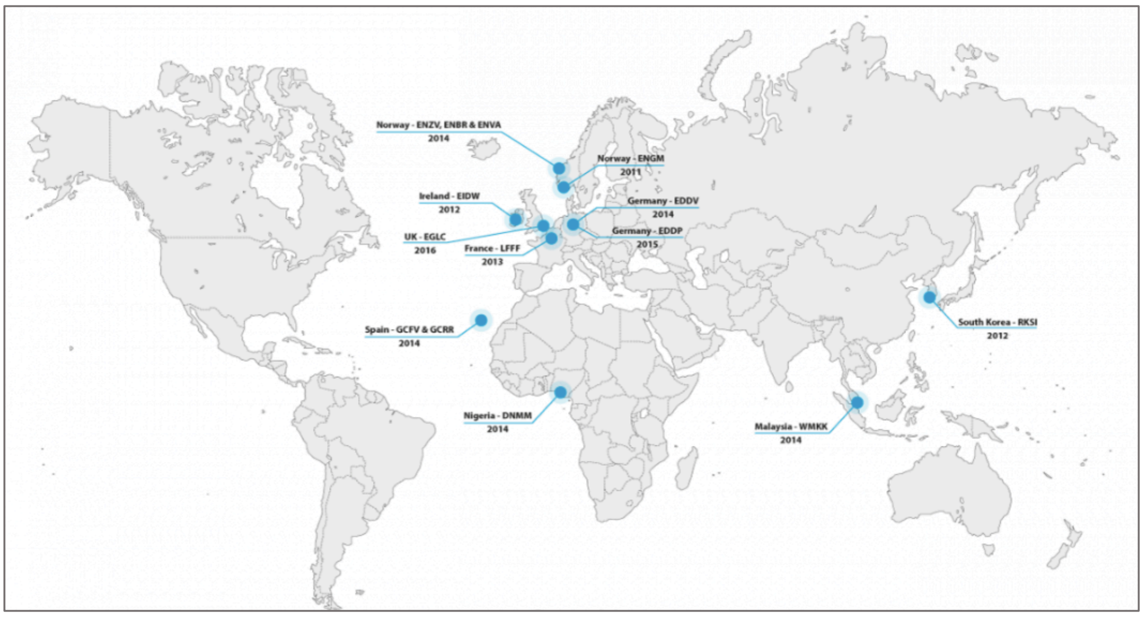


Figure 18. Point Merge Deployment Status. *Source: Eurocontrol [28]*

8. Relative Position Indicator

The benefits of Point Merge Operations can be restricted by wind conditions and speed variation during the approach and between aircraft. Different geometries of the routes to the Merge Point also contribute to the difficulty for controllers to identify possible merge problems on time. That inhibits early intervention and avoidance of vectoring an aircraft off the RNAV procedure. If controllers were able to identify potential merge problems with anticipation, they could intervene with speed control and let aircraft get to the Merge Point with the required separation. The MITRE Corporation has been developing a tool called Relative Position Indicator (RPI), a near term tactical tool to assist managing traffic flows in the TMA. This tool does not need any special equipment to be installed on the aircraft nor does it require procedural changes for ATC.

8.1. RPI tool

The RPI is shown on the controller's monitor indicating the position of aircraft relative to a Merging Point as if all aircraft were flying in one line or on one route. To identify the relative position of all aircraft to each other, the flight path distance to the Merge Point is calculated, taking into account all the non-linear segments and the exact distances flown by turns.

The expected benefits are first of all the application of speed control on an early stage of approach to build gaps more precisely and in consequence, when arriving at the Merge Point, aircraft have the necessary separation. That would lead to a reduction of vectoring for delaying aircraft.

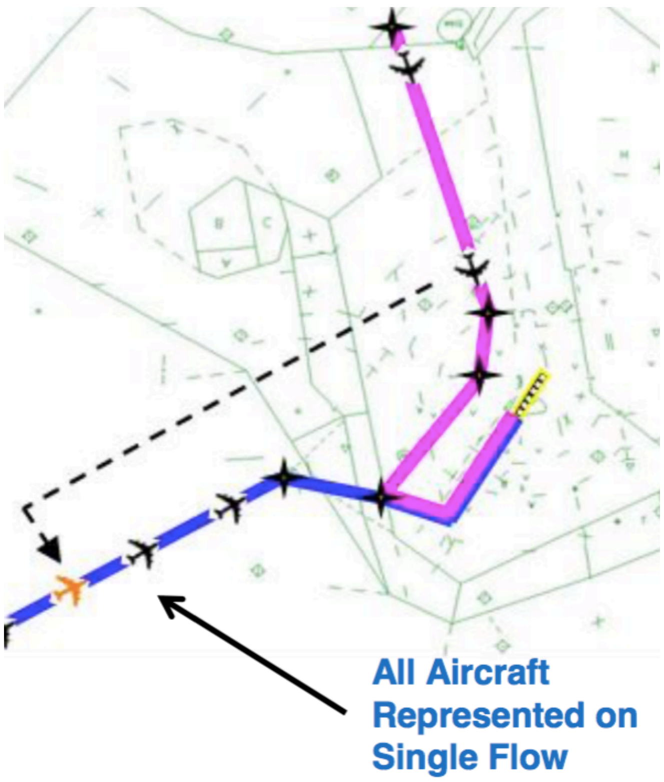


Figure 19. RPI basic idea.

9. EUROCONTROL Network Manager

Eurocontrol is an intergovernmental organization founded in 1960 with 41 member states, including Ireland and therefore offering relevant information and data for Dublin Airport. Its commitment is to build a Single European Sky (SES), a project launched by the European Commission in 1999, by reforming the architecture of European air traffic management (ATM). Eurocontrol also takes the role of Network Manager, bringing together the different aviation and air traffic management protagonists involved in the design, planning and management of the European ATM network. The services provided by Eurocontrol include strategic and tactical flow management, regional control of airspace, controller training, safety-proofed technologies and procedures, collection of air navigation charges, Airspace Management (ASM) processes, aeronautical information management (AIM), the collection, analysis and publication of data and statistics and much more.

The Network Manager (NM) manages air traffic management network functions (developing and creating Route Network Design, providing a central function for Frequency Allocation, coordinating improvements to SSR Code Allocation, carrying out the Air Traffic Flow management function) [29] and supports the global performance of the European aviation network by monitoring traffic and providing delay forecasts and analysis. These offer an approach for planning and operational activities. The three activities realized by NM in order to support global performance of European aviation network are listed in detail in table 3.

Statistics and Forecast Service (STATFOR)	The STATFOR forecasts are used as direct inputs into the NSP, NOP and the Network and local Performance Plans as required by the NMF IR. These forecasts are also a prerequisite for the establishment of the unit rates used to calculate the route and terminal charges. Traffic forecasts are also used by an extensive number of planning departments of airlines, ANSPs, airports, government authorities, etc. for general planning.
The Operational Analysis and Reporting (OAR)	The Network Manager annual report describes the implementation of the Network Strategy Plan and the Network Operations Plan, as well as the performance of all aspects of the network compared to the performance targets and performance plans. A comprehensive set of more detailed reports - covering all operations and aspects of performance and compliance concerning the network - is also published.

The Central Office for Delay Analysis (CODA)	In addition to monitoring and reporting on the performance of the ATM network in terms of delays from flow management regulations, the Network Manager provides a monitoring and analysis function for all delay reasons (ATFM, airline, airport, etc.). This enables correlation between airline and network reported delays, and is used in schedule and turnaround planning, enabling better punctuality.

Table 3. Activities Network Manager. *Source: [30]*

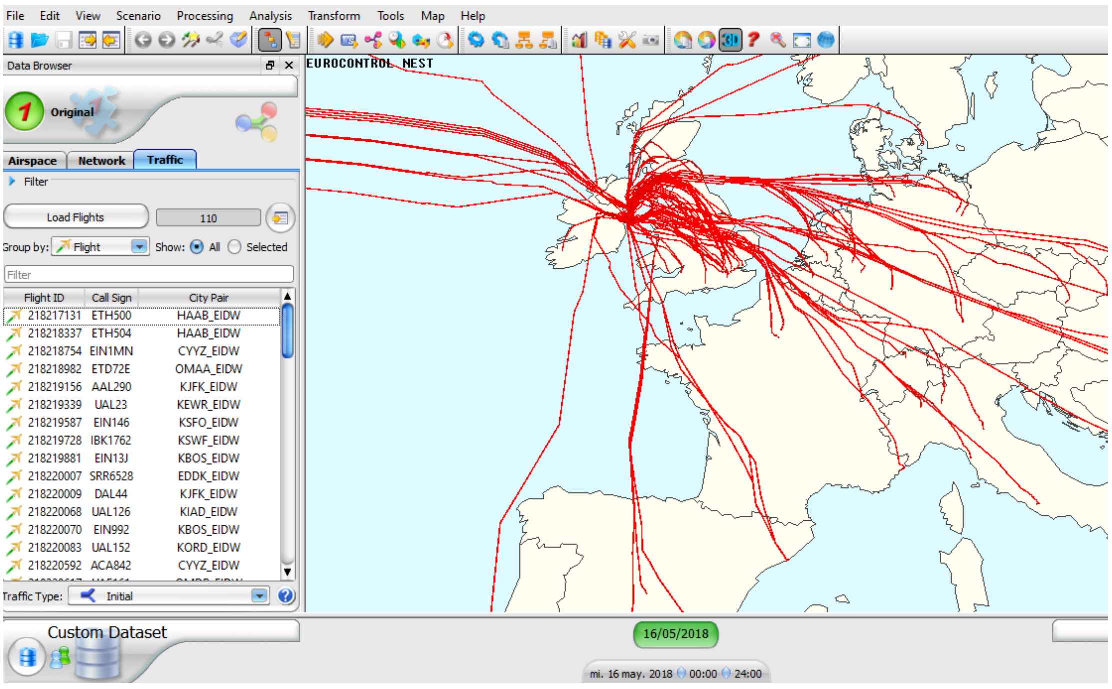
For the airspace and network design NM makes use of tools and data to perform modelling and simulation. These tools allow operational analysis of airspace structures and traffic distribution. Data used for these analysis are stored in a database of past, present and future airspace and demand data, DDR (Demand Data Repository) [30].

9.1. NEST (Network Strategic Tool)

The desktop application NEST (Network Strategic Tool), see figure 20, is a scenario based modelling tool used for airspace structure design and development, capacity planning and post-operations analysis, the organization of traffic flows and the preparation of scenarios for fast time simulations. It is used by the Eurocontrol NM to optimize the available resources and improve performance at network level but also by Area Control Centres or Airports [30], [31].

The simulation algorithm of the program provides future traffic samples and delay simulations as well as 4D traffic distribution, configuration optimizer and regulation builder. The tool allows a wide range of analysis and traffic visualization. With NEST the different scenarios can not only be simulated. The data representing a chosen scenario, for example, show all the traffics flying over a certain way point and can also be exported in different formats [31].

The traffic data and other datasets to be used for planning and analysis with NEST- including future and historic traffic - can be downloaded from the DDR2 web portal, described in more detail in the next section.



Flight ID	Call Sign	City Pair
218217131	ETH500	HAAB_EIDW
218218337	ETH504	HAAB_EIDW
218218754	EIN1MN	CYYZ_EIDW
218218982	ETD72E	OMAA_EIDW
218219156	AAL290	KJFK_EIDW
218219339	UAL23	KEWR_EIDW
218219587	EIN146	KSFO_EIDW
218219728	IBK1762	KSWF_EIDW
218219881	EIN13J	KBOS_EIDW
218220007	SRR6528	EDDK_EIDW
218220009	DAL44	KJFK_EIDW
218220068	UAL126	KIAD_EIDW
218220070	EIN992	KBOS_EIDW
218220083	UAL152	KORD_EIDW
218220592	ACA842	CYYZ_EIDW

Figure 20. NEST tool showing traffics flying through LAPMO.

9.2. DDR2 (Demand Data Repository)

The source for data and tools of EUROCONTROL is DDR2 - standing for phase 2 of Demand Data Repository – which is promoted as DDR service. It is a web portal interface for generating and downloading traffic. Depending on their license, users can download future traffic samples for planning purposes or / and past traffic samples for post-operations traffic trend analysis and for identifying best practices for future operations [30]. Historic traffic is available from July 2011 and future traffic can be consulted from the day of operation until up to 5 to 7 years in advance. But the DDR service not only offers traffic data, it also produces datasets describing the European Route Network structure environment including ATS route network, RAD restrictions, Free Route Airspace implementation, Airspace Closure, Flight level constraints and more in order to be used for generating 4D flight trajectories for future traffic demand [30]. In figure 21 the structure of DDR service and its connection to NEST are illustrated.

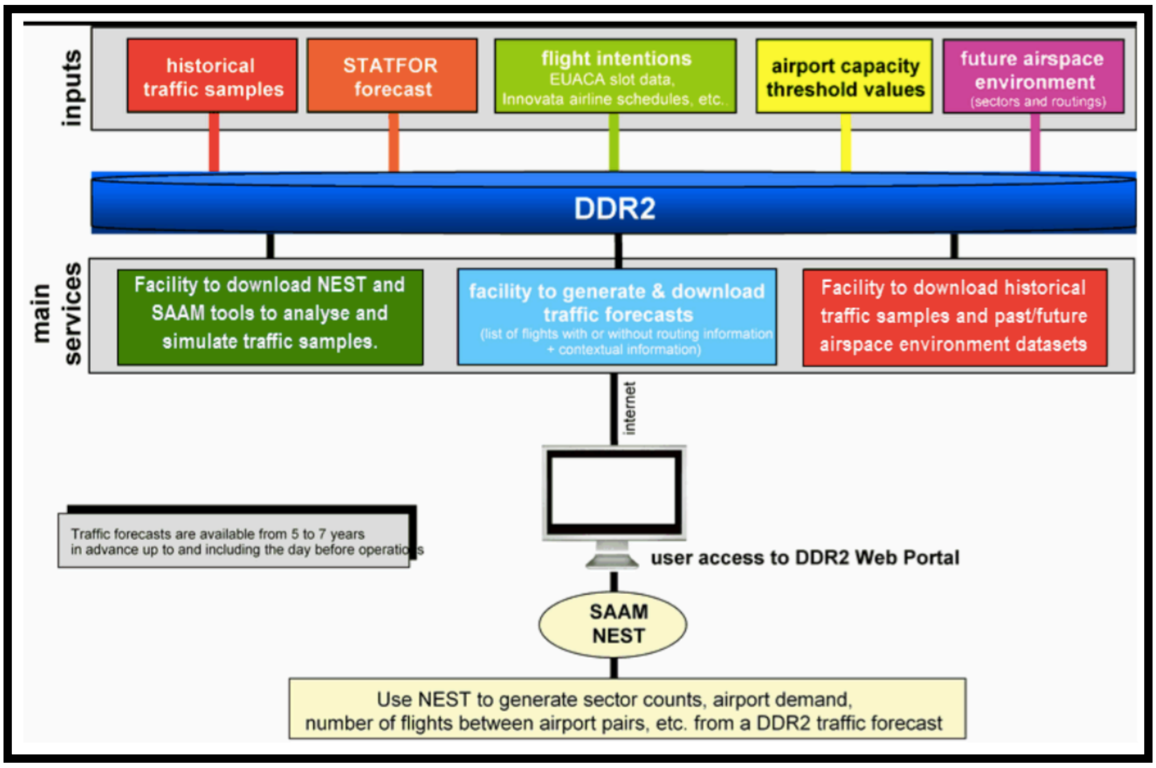

inputs
historical traffic samples
STATFOR forecast
flight intentions
EUACA slot data, Innovata airline schedules, etc.
airport capacity threshold values
future airspace environment
(sectors and routings)
DDR2
main services
Facility to download NEST and SAAM tools to analyse and simulate traffic samples.
facility to generate & download traffic forecasts
(list of flights with or without routing information + contextual information)
Facility to download historical traffic samples and past/future airspace environment datasets
internet
Traffic forecasts are available from 5 to 7 years in advance up to and including the day before operations
user access to DDR2 Web Portal
SAAM
NEST
Use NEST to generate sector counts, airport demand, number of flights between airport pairs, etc. from a DDR2 traffic forecast

Figure 21. Overview DDR2 service. Source: [33]

The wide range of data and tooling options makes this web portal interesting for a many users. The NM develops the Network Operation Plan including route and airspace design, demand and capacity balancing for different situations. ANSPs use the portal to prepare and optimise their capacity plans, Airlines to detect flight efficiency improvements based on past operations. ASM actors are able to manage and coordinate airspace processes with this service and airports integrate their local plans with the Network Operational Plan [32].

In in this work historical traffic is analyzed. It is important to outline that in the DDR2 version the traffic sample for one day includes all flights flying within ECAC during that day. This means that flights overflying the ECAC area over midnight will be counted twice. Although there will be some duplicated flight from one day to the other, accurate airspace load will be achieved with these data. Furthermore, the number of flights is based on all flights with a last filed and accepted flight plan. Leading to the inclusion of some cancelled flights [32].

Historical traffic is available in different file types, users can choose between NEST official, SO6 Model1, SO6 Model3, EXP2, ALLFT and ALLFT+. The files are available after two to four days on the DDR2 Web Portal. The architecture of the SegOut6 files is explained in more detail in the following segment.

9.3. SegOut6 files

The SegOut6 files, whose extension is so6, describe flight 4D trajectories segment by segment, sorted by flight segment sequence from origin to destination. The described trajectories can have different sources [33]:

- SAAM, calculating vertical profiles possible from operational constrains and mixing information from BADA and CFMU aircraft performance
- CFMU, distinguishing between m1 and m3; M1 is based on the last filed flight plan and m3 on the flight plan enriched with radar data
- Pure Radar Data (CPR data)
- Other sources

As already mentioned, there exist two different models of SegOut6 files available containing different information. Users can choose between historical traffic based on last filed flight plan data, model 1, or actual trajectories, model3, [34]:

- <u>Enhanced Tactical Flow Management System (ETFMS) **Model 1** flight information</u>: This information is the information captured in ETFMS based on the last filed flight plan from the airline (FTFM).
- <u>ETFMS **Model 3** flight information</u>: It is the information captured in ETFMS after the flight has been operated. It is based on the last filed flight plan, whenever a flight deviates from its filed flight plan by more than one of the pre-determined NMOC thresholds of 5 minutes, 7FL or 20NM these data are updated with available CPR information. The trajectory stored in the file is not equivalent to the one indicated in the flight plan, but converges with the 4D trajectory actually flown.

The following table, table 4, explains the information contained in one line of the so6 file.

#	Field	Type	Size	Comment
1	segment identifier	char		first point name ”_” last point name
2	origin of flight	char	4	ICAO code
3	destination of flight	char	4	ICAO code
4	aircraft type	char	4	
5	time begin segment	num	6	HHMMSS padded with 0’s
6	time end segment	num	6	HHMMSS padded with 0’s
7	FL begin segment	num	1-3	
8	FL end segment	num	1-3	
9	status	char	1	0=climb, 1=descent, 2=cruise
10	callsign	char		
11	date begin segment	num	6	YYNNDD padded with 0’s
12	date end segment	num	6	YYNNDD padded with 0’s
13	lat begin segment	float		in minute decimals
14	lon begin segment	float		in minute decimals
15	lat end segment	float		in minute decimals
16	lon end segment	float		in minute decimals
17	flight identifier	num		same as the one provided in expand file (must be uniq). in case of flight option, it is >=1000000000)
18	sequence	num		start at 1 for every new flight, incremented at each line. IMPORTANT!
19	segment length	float		in nautical miles
20	segment party/colour	num		0=NO (grey, R=102, G=102, B=102), 1=ODD (green, R=60, G=255, B=60), 2=EVEN (blue, R=100, G=100, B=255), 3=ODD_LOW (dark green, R=0, G=200, B=0), 4=EVEN_LOW (light blue, R=160,G=160,B=255), 5=ODD_HIGH(light green, R=160, G=255, B=160), 6=EVEN_HIGH (dark blue, R=0, G=0, B=200), 7=General Purpose Red Color (R=255, G=0, B=0), 8=General Purpose Orange Color (R=255, G=128, B=0), 9=General Purpose Yellow Color (R=255, G=255, B=0)

Table 4. SO6 traffic file structure. *Source: [34]*

In figure 22 an extract of a real .so6 file is shown. There are two different flights represented in this extract. The information of the indicated line is the following:

The flight, with identifier 217186613, was performed on 19th April, 2018, and the actual line is the 81st line recorded of this flight. The aircraft, an A320 with callsign EIN63PT, enters the segment over waypoint LAPMO at 11:38:11 and exits it at point $UNTM at 11:38:54. The origin of the flight is Barcelona–El Prat Airport and the destination Dublin airport. The aircraft is actually descending, starting from FL35 when entering the segment, and leaving the segment at FL26.

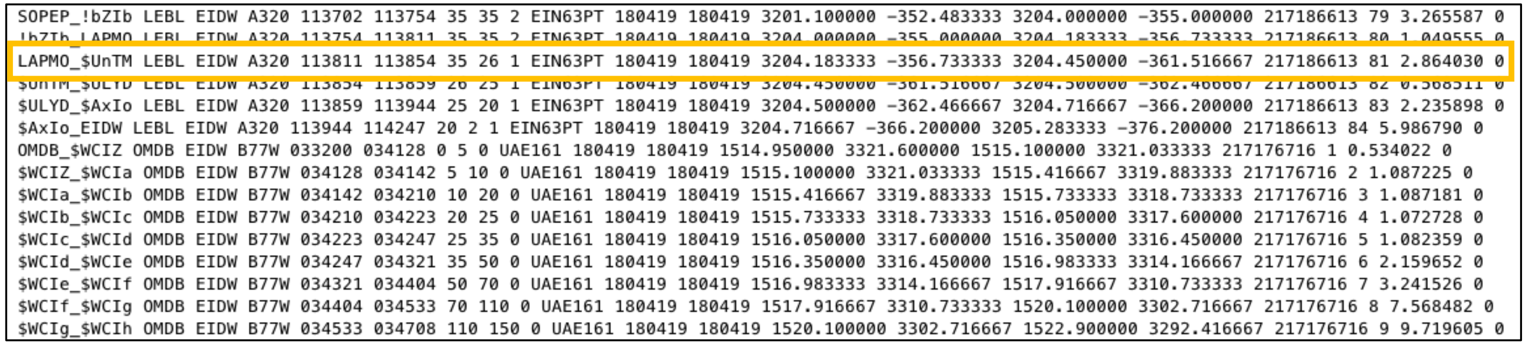
The exact position of the entering point is:

- latitude = 3204.183333 and longitude = -356.733333

And the position of the exit point is:

- latitude = 3204.45 and longitude = -361.516667

The position are given in minute decimals and the distance between the two points are 2.864030 nautical miles.



```
SOPEP_!bZIb LEBL EIDW A320 113702 113754 35 35 2 EIN63PT 180419 180419 3201.100000 -352.483333 3204.000000 -355.000000 217186613 79 3.265587 0
!bZIb_LAPMO LEBL EIDW A320 113754 113811 35 35 2 EIN63PT 180419 180419 3204.000000 -355.000000 3204.183333 -356.733333 217186613 80 1.049555 0
LAPMO_$UnTM LEBL EIDW A320 113811 113854 35 26 1 EIN63PT 180419 180419 3204.183333 -356.733333 3204.450000 -361.516667 217186613 81 2.864030 0
$UnTM_$ULYD LEBL EIDW A320 113854 113859 26 25 1 EIN63PT 180419 180419 3204.450000 -361.516667 3204.500000 -362.466667 217186613 82 0.568511 0
$ULYD_$AxIo LEBL EIDW A320 113859 113944 25 20 1 EIN63PT 180419 180419 3204.500000 -362.466667 3204.716667 -366.200000 217186613 83 2.235898 0
$AxIo_EIDW LEBL EIDW A320 113944 114247 20 2 1 EIN63PT 180419 180419 3204.716667 -366.200000 3205.283333 -376.200000 217186613 84 5.986790 0
OMDB_$WCIZ OMDB EIDW B77W 033200 034128 0 5 0 UAE161 180419 180419 1514.950000 3321.600000 1515.100000 3321.033333 217176716 1 0.534022 0
$WCIZ_$WCIa OMDB EIDW B77W 034128 034142 5 10 0 UAE161 180419 180419 1515.100000 3321.033333 1515.416667 3319.883333 217176716 2 1.087225 0
$WCIa_$WCIb OMDB EIDW B77W 034142 034210 10 20 0 UAE161 180419 180419 1515.416667 3319.883333 1515.733333 3318.733333 217176716 3 1.087181 0
$WCIb_$WCIc OMDB EIDW B77W 034210 034223 20 25 0 UAE161 180419 180419 1515.733333 3318.733333 1516.050000 3317.600000 217176716 4 1.072728 0
$WCIc_$WCId OMDB EIDW B77W 034223 034247 25 35 0 UAE161 180419 180419 1516.050000 3317.600000 1516.350000 3316.450000 217176716 5 1.082359 0
$WCId_$WCIe OMDB EIDW B77W 034247 034321 35 50 0 UAE161 180419 180419 1516.350000 3316.450000 1516.983333 3314.166667 217176716 6 2.159652 0
$WCIe_$WCIf OMDB EIDW B77W 034321 034404 50 70 0 UAE161 180419 180419 1516.983333 3314.166667 1517.916667 3310.733333 217176716 7 3.241526 0
$WCIf_$WCIg OMDB EIDW B77W 034404 034533 70 110 0 UAE161 180419 180419 1517.916667 3310.733333 1520.100000 3302.716667 217176716 8 7.568482 0
$WCIg_$WCIh OMDB EIDW B77W 034533 034708 110 150 0 UAE161 180419 180419 1520.100000 3302.716667 1522.900000 3292.416667 217176716 9 9.719605 0
```

Figure 22. SO6 file extract.

PART 3 – PROGRAMMING TOOL

10. Object Oriented Programming

Object oriented programming (OOP) models objects and their interactions in the production of a system. Since the real-world problem domain is characterized by objects and their interactions, a software application developed using the object-oriented programming approach will have a closer representation of the real-world problem domain than any other programming approach [35].

10.1. Advantages of Object Oriented Programming

The reasons for choosing OOP for the development of this project are the following:

- It is suitable for large and complex projects with more than one developer working on it.
- It has a modular structure, meaning the complexity is reduced by assigning the tasks to different classes which communicate among each other.
- The creation and interaction of classes allows to model real world scenarios in an easy way.
- Multiplatform – The source code only has to be written once and can be used in different platforms.
- The use of design patterns simplifies the programing of a GUI. SWING, the graphical Java tool, uses the Model-View-Controller pattern.

10.2. Basics of Object Oriented Programming

The basics of object oriented programming are classes and objects, with the focus on data implied in the objects and not on the code that manipulates the data. A class is defined by instance variables, constructors and methods. An object is created by calling the constructors. Each object created from a class has a *copy* of the instance variables and can assign them its individual values and manipulate them independently using the methods of the class. While variables represent an object, the methods are used to modify and work with these variables and to accomplish different type of tasks [36].

There are three principles of object oriented programming: abstraction, inheritance and polymorphism [36].

10.2.1. Abstraction

Abstraction is the process of representing the essential features of a system without getting involved with its complexities. Once a class is defined, it can be created and its methods can be called without going into detail on how they are implemented. By that way, complexities are hidden and only the essential is exposed and necessary to know in order to create and work with an object.

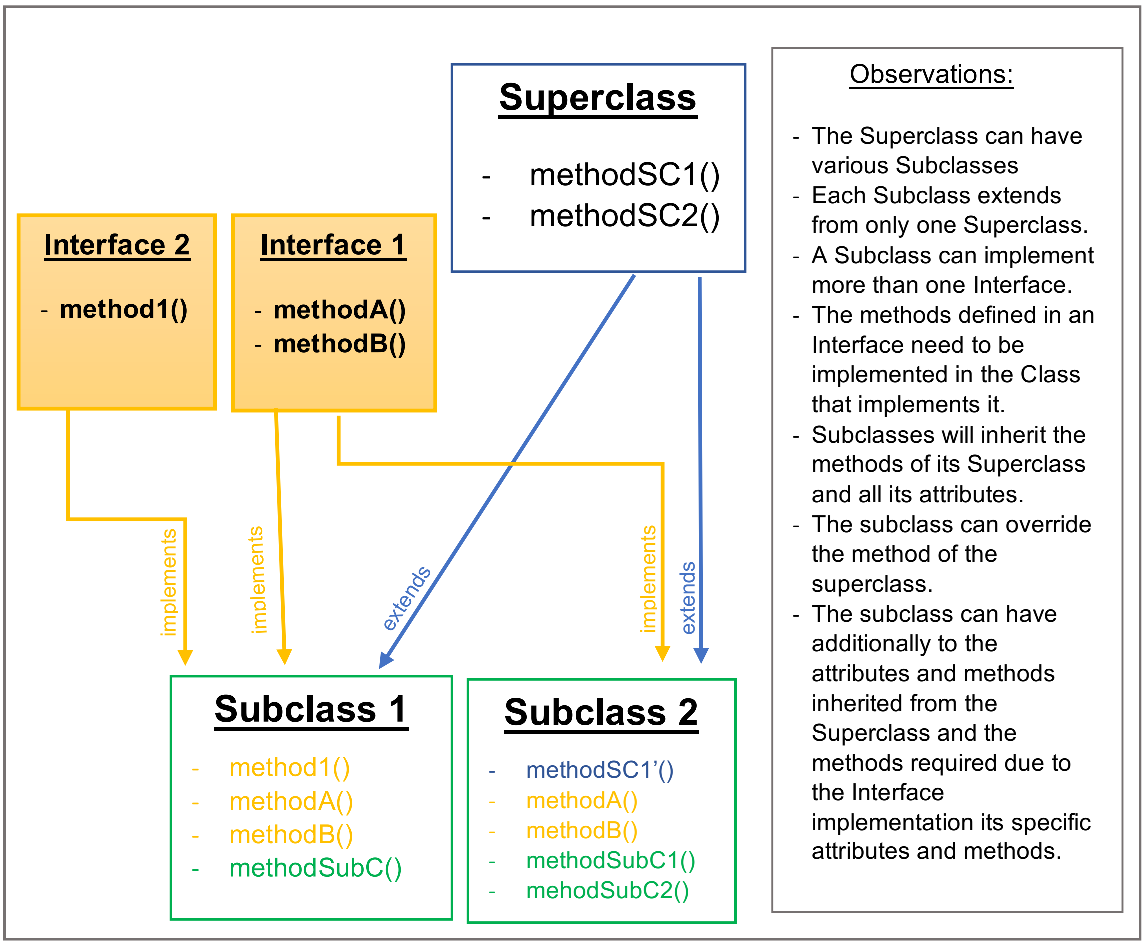

Superclass
- methodSC1()
- methodSC2()
Interface 2
- method1()
Interface 1
- methodA()
- methodB()
implements
implements
extends
implements
extends
Subclass 1
- method1()
- methodA()
- methodB()
- methodSubC()
Subclass 2
- methodSC1'()
- methodA()
- methodB()
- methodSubC1()
- mehodSubC2()
Observations:
- The Superclass can have various Subclasses
- Each Subclass extends from only one Superclass.
- A Subclass can implement more than one Interface.
- The methods defined in an Interface need to be implemented in the Class that implements it.
- Subclasses will inherit the methods of its Superclass and all its attributes.
- The subclass can override the method of the superclass.
- The subclass can have additionally to the attributes and methods inherited from the Superclass and the methods required due to the Interface implementation its specific attributes and methods.

Figure 23. OOP class and interface relations.

10.2.2. Inheritance

Inheritance is the capability of a class to inherit the variables and methods of other classes, called super-classes. The new class can define its own variables and methods additionally to what has been inherited. -> Representation diagram

10.2.3. Polymorphism

Polymorphism describes the property of different objects being able to react to the same method call. Subclasses can define their own unique behaviors and yet share some of the same functionality of the parent class. To make polymorphism possible, interfaces and overriding are important concepts to introduce.

10.2.4. Interfaces

Interfaces are kind of contracts between the class and the outside world. They can only be implemented by classes or extended by other interfaces, never instantiated. In an interface, the methods a class implementing this interface must provide, are specified. These methods have to appear in the source code of the class to be compiled successfully. In this way, it allows a class to become more formal about the behavior it provides. The interface only contains the signature of the methods, never its implementation because an interface specifies what a class must do, but not how it has to be done. It can also contain variables, these need to be public, static and final.

As already mentioned, interfaces allow polymorphism and full abstraction. One class can implement more than one interface, unlike the concept of inheritance where a class can only have one super-class not allowing multiple inheritance. The relation between subclasses, super-classes and interfaces is visualized in figure 23.

10.2.5. Overriding

The concept of overriding allows a subclass to modify the implementation of a method provided by its super-class. Name, parameters and return type need to be the same, only the way the return value is obtained or calculated will be modified. To determine which version of an overridden method will be executed, it will be checked first if the method is defined in the object's class. If there is no such method, the one of the super-class will be used, see figure 23.

10.3. Threads

OOP features thread objects that can be executed in parallel with other threads. That allows the handling of concurrent work processes. Every process has at least one thread and the creation of a new thread requires fewer resources compared to the creation of a new process [37].

The present project has two classes extending from a thread. *"timeThread"* is the class which takes charge of the simulation time representation depending on the indicated simulation speed. The other class extending from a thread is the "*simulation"* class. This thread prepares a list of all aircraft flying at each instant of simulation and passes it to the listener for representation.

10.4. Collections

A collection can be defined as an object that represents a group of objects, called "the collection's elements". In Java, there are three different kinds of collections: Lists, Maps and Iterators.

In this project, Lists and Maps are of special interest and created in the following way:

- List: *List<E>* E stays for element and is the class indicating the kind of objects contained in the list. It is an ordered collection, elements can be placed at a certain position and they can be accessed by their index. An example of a list in the developed program is: *List<TrafficFlight>*.

- Map: *Map<K, V>* is a key-value mapping. Values can be accessed indicating the corresponding key provided that no key is duplicated. An example of a map in the developed program is: *Map<Integer, TrafficPoint>* or *Map< String, List<Traffic>>* where the value is a list containing Traffics.

11. NetBeans IDE

NetBeans is a free, open-source integrated development environment (IDE) for Java programming language. The latest version is NetBeans IDE 8.2, which was released on October 3, 2016 [38].

All the IDE's functions are provided by modules. The modules are defined to provide one specific function, such as support for the Java language or editing [39]. The GUI design tool is one of the included modules to design Swing GUIs and it is of special interest for the development of the present project.

11.1. Swing GUI Builder

NetBeans IDE includes a GUI (Graphical User Interface) builder. The feature for automatic code generation simplifies the GUI development process. Additionally, the newer versions (starting from NetBeans 7.4) include a Swing GUI Builder. In the Design window, the class can be edited in the Swing GUI Builder, where buttons, labels, test fields, panels and other objects can be chosen and dragged from palette onto the frame. Code is automatically generated for the objects and designs made in the Design window and saves writing the user interface entirely by hand. Where the Builder cannot automatically generate code, for example when creating an event, it provides empty methods that can be completed to implement the desired logic [38].

PART 4 - DESIGN

12. Design overview

Figure 24 is a schematic representation of the general process of the program. The user starts choosing the kind of traffic to simulate. In this project only the historical traffic was implemented. In the second window, the parameters for simulation and the input files are chosen. The program engine treats all the data so they can be used for simulation and then - if all the inputs are consistent – it passes to the third window where the simulation can be started.

The different windows are explained in more detail in the following sections.

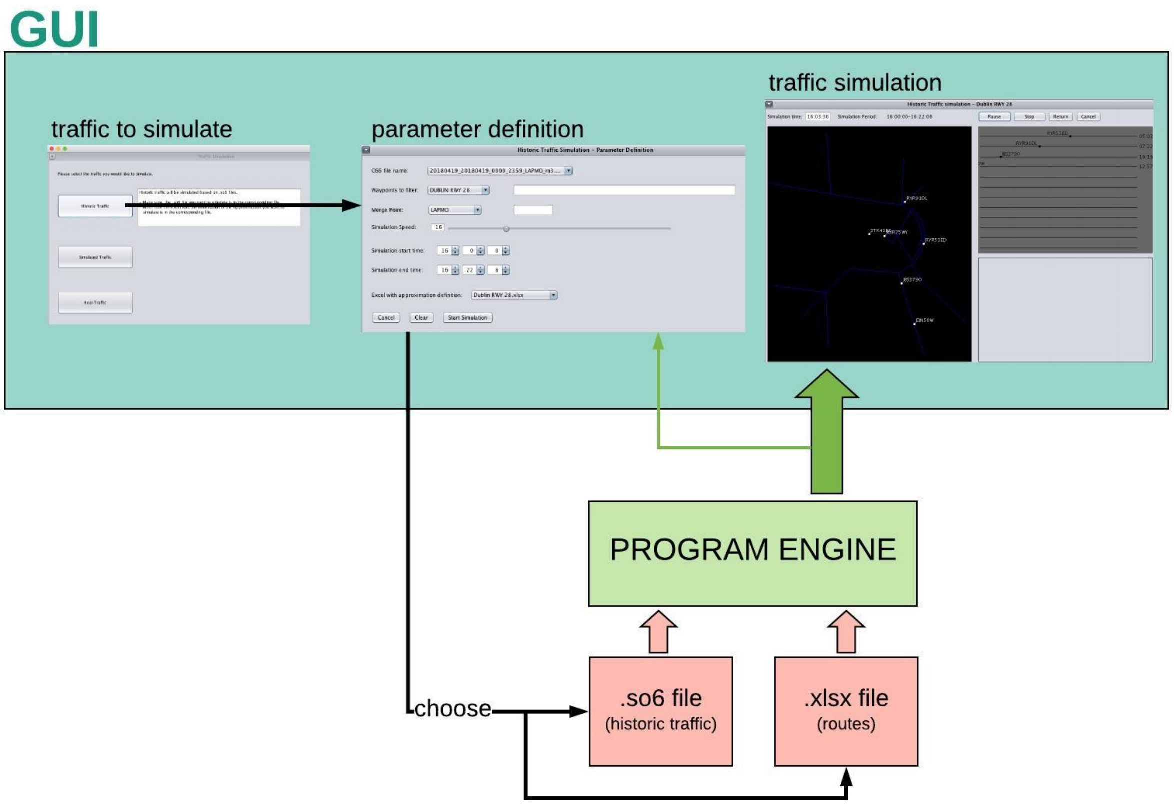
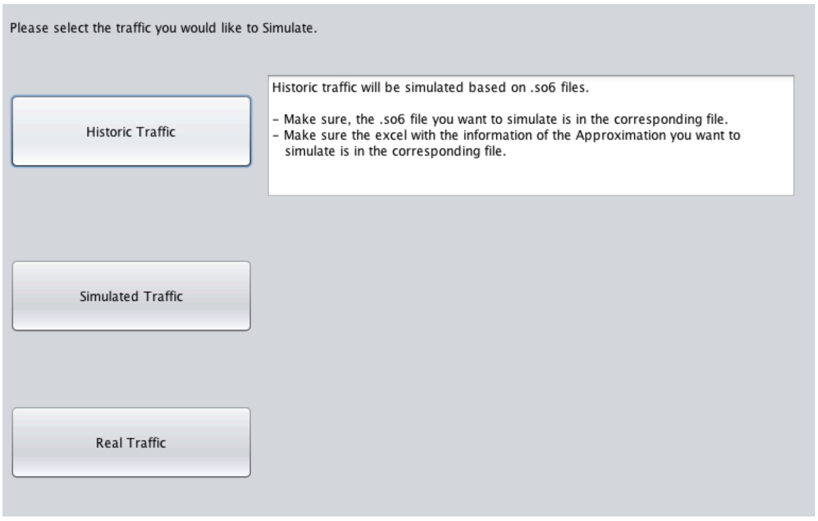


Figure 24. Program design overview.

13. Traffic selection

In the first window visible when starting the application (see figure 25), the program asks to choose the kind of traffic that you would like to simulate. By now, only Historic Traffic can be selected. The tool is not yet developed for Simulated or Real Traffic. By clicking on the Historic Traffic button the program changes to the parameter selection window.


Please select the traffic you would like to Simulate.

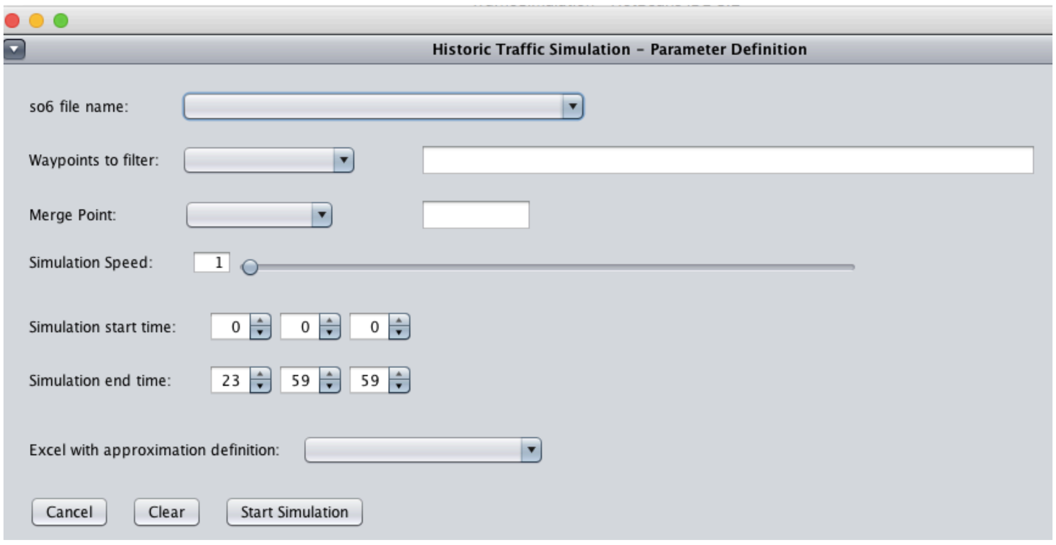
Historic traffic will be simulated based on .so6 files.

- Make sure, the .so6 file you want to simulate is in the corresponding file.
- Make sure the excel with the information of the Approximation you want to simulate is in the corresponding file.

Figure 25. Window one - chose traffic to simulate.

14. Parameter definition

In this window, figure 26, all parameters for the simulation are introduced. The .so6 file with the historic traffic information and the excel file containing the different routes for the approach are directly read from the folders containing them. So, if you want to add an additional file, it needs to be placed in the corresponding folder before starting the program. The waypoints to filter the traffic and the Merge Point of the approach can be chosen from the combo Box, alternatively they can also be introduced manually. In case, an approach is going to be used frequently, the corresponding waypoints and the Merge Point can be added to the text file the program is reading the data for the combo Boxes from. Simulation speed can be chosen with the slider from a range between 1 and 60. If the start and end time of the simulation are not modified, the whole day/file will be simulated.



Historic Traffic Simulation – Parameter Definition

so6 file name:

Waypoints to filter:

Merge Point:

Simulation Speed: 1

Simulation start time: 0 0 0

Simulation end time: 23 59 59

Excel with approximation definition:

Figure 26. Window 2 - Parameter Definition.

14.1. Buttons available

The following three buttons (see figure 27) are available in the parameter selection window:

- Cancel: All inputs are deleted and the program redirects to the traffic selection window.
- Clear: All inputs are deleted.
- Start Simulation: After confirming the correctness of the parameters, the program passes to the traffic simulation window.

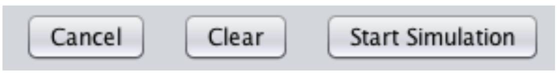


Figure 27. Buttons window 2.

14.2. Error messages

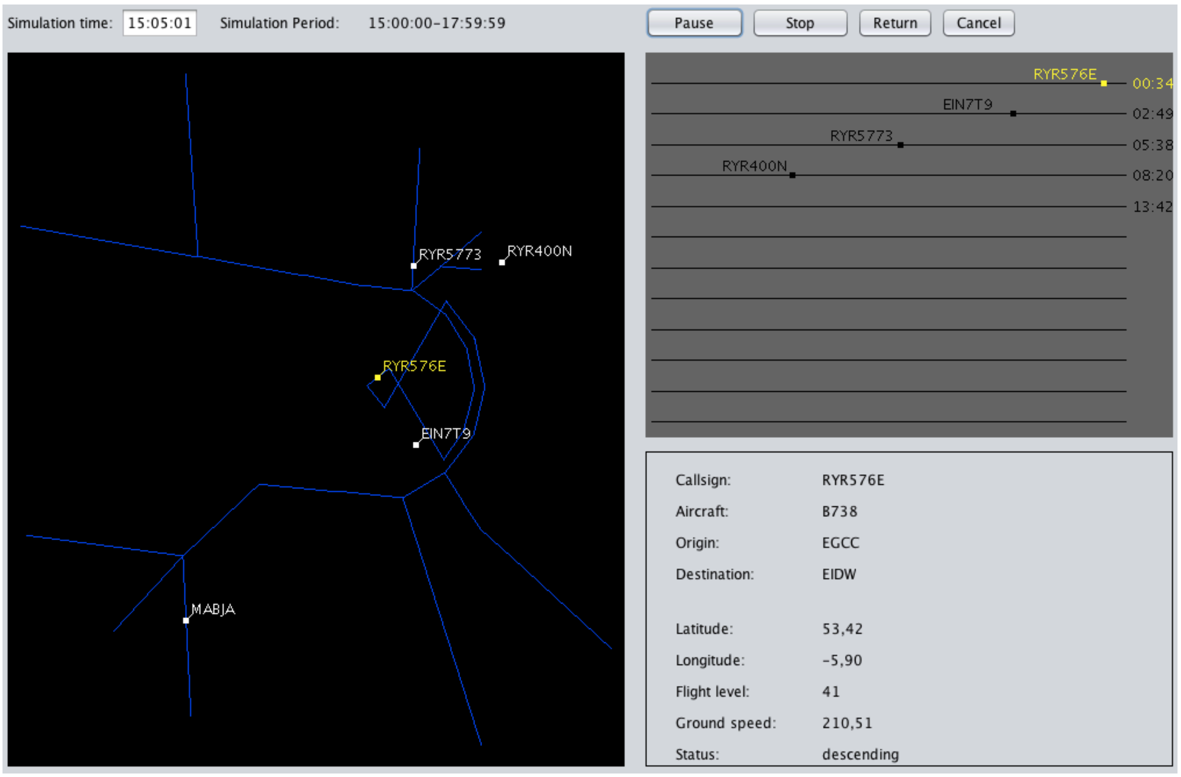
Once the parameters for the simulation are introduced and the start button is pressed, the program verifies that all parameter are chosen and that they are consistent. If any error is detected, the program will inform about the error and it will not continue until it is corrected. All the possible errors, their signification and the required actions are resumed in table 5.

ERROR MESSAGE	MEANING	CORRECTIVE ACTION
No file name entered	There was no file name selected from the combo Box.	Choose a file name from the combo Box.
No Merge Point selected	There was no Merge Point selected from the combo Box or introduced manually.	Choose a Merge Point from the combo Box or introduce one manually. If manually introduced make sure the spelling is right.
No waypoints selected to be filtered	There were no waypoints selected from the combo Box or introduced manually.	Choose an approach from the combo Box or introduce the waypoints manually. If manually introduced make sure the spelling is right.
Start time needs to be before End time	The indicated start time for simulation is later than the indicated end time. Simulation is only working forward, not backward.	Change start and/or end time so that start time is chronologically before end time.
Merge Point is not in the selected file and/or none of the waypoints to filter is in the selected file	In this case, the chosen .so6 file does not match with the selected waypoints. The indicated Merge Point is not contained in the chosen .so6 file and/or none of the indicated waypoints for which the traffics should be filtered are contained in the .so6 file.	Choose another .so6 file and make sure, this file contains flights performing the approach matching with the entered waypoints. Or change the waypoints so they match the chosen file. If manually introduced make sure the spelling is right.
The routes for the approach selected does not correspond to the Merge Point and/or the waypoints to filter.	In this case, the chosen excel file with the routes information does not match with the selected waypoints. Either the indicated Merge Point is not contained in it and/or none of the indicated waypoints for which the traffics should be filtered.	Choose another .so6 file and make sure, this file contains flights performing the approach matching with the entered waypoints. Or change the waypoints so they match the chosen file. If manually introduced make sure the spelling is right.

Table 5. Error messages.

15. Traffic simulation

This last window is the main one and presents the simulation. It consists of three panels, one with the map of the approach, another one with the RPI and one with information about the chosen aircraft. In the upper part of the window on the left side, the actual simulation time and the simulation period are shown and on the right side, there are four buttons available. In figure 28, there is one aircraft over the map and on the RPI that is represented in yellow. This is called the *chosen aircraft* and it is the one whose information is shown on the aircraft information panel. The aircraft can be chosen with the mouse on the map or on the RPI. By clicking on the chosen aircraft or on a point of the map or RPI where no aircraft is placed, the chosen aircraft is set to null and no information is shown on the aircraft information panel.



Simulation time: 15:05:01   Simulation Period: 15:00:00-17:59:59

Callsign:	RYR576E
Aircraft:	B738
Origin:	EGCC
Destination:	EIDW
Latitude:	53,42
Longitude:	-5,90
Flight level:	41
Ground speed:	210,51
Status:	descending

Figure 28. Window 3 - Simulation.

15.1. Map

The map shows the routes given to the program through the excel file (see figure 29). Over these routes the historic traffic of the so6 file is represented.

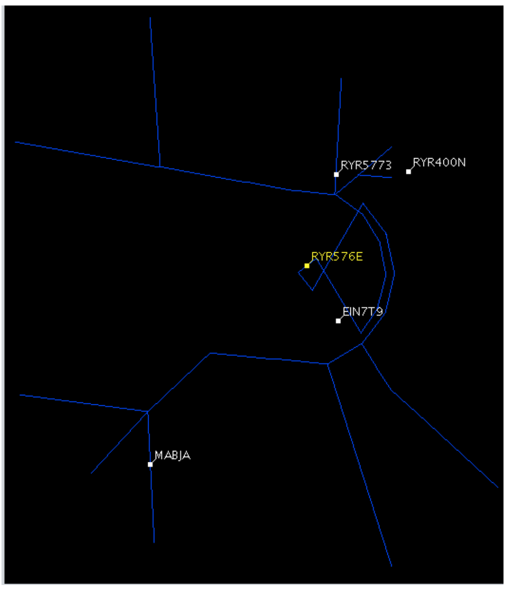


Figure 29. Map panel.

15.2. RPI implementation

In this work, the RPI will be shown on the controller's display on the left of the map. Each aircraft will be represented on one line moving from left to the right. Once the aircraft arrives at the end of the line, it has reached the Merge Point. Additionally, at the right end of the line, the time left to reach the Merge Point is indicated (see figure 30).

As the simulation is based on historic traffic, the time left to reach the Merge Point can be easily calculated. The time shown on the RPI is simply the subtraction of the actual time minus the time when the Merge Point is overflown.

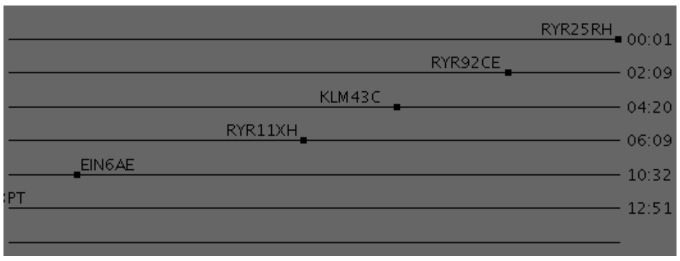
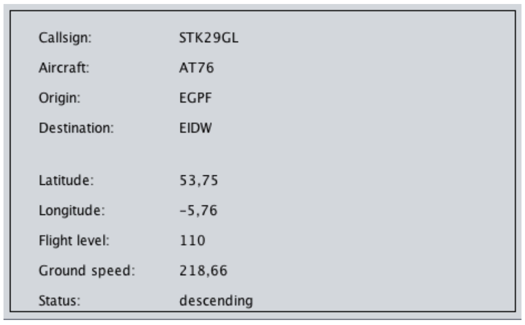


Figure 30. RPI panel.

15.3. Aircraft Information

This panel contains information only if an aircraft is chosen (it is represented in yellow over the map and the RPI). The information can be divided into two (see figure 31). At the top, the fix information is shown (Callsign, Aircraft type, Origin and Destination). The airport of origin and destination are given in ICAO code.

At the lower part, variable information about the actual state of the aircraft is given. Latitude and Longitude are shown in degree decimal, the flight level is given as FL (= feet*100) and the ground speed in knots. The status can be climbing, descending or cruise.



Callsign:	STK29GL
Aircraft:	AT76
Origin:	EGPF
Destination:	EIDW
Latitude:	53,75
Longitude:	-5,76
Flight level:	110
Ground speed:	218,66
Status:	descending

Figure 31. Aircraft information panel.

When the chosen aircraft has landed and is no longer shown on the map, the information from this panel also disappears.

15.4. Buttons available

When the traffic simulation window opens, the following four buttons are available:

- *Start*: It starts the simulation; the two threads are started (timeThread and simulation).
- *Stop*: The simulation is stopped, as well as the threads. The time shown at simulation time coincides with the start time. The RPI and aircraft information panels are shown in blank and the map contains the routes only.
- *Return*: The simulation is stopped and the program is redirected to the parameter input window. The last parameters introduced are still shown. This is to make the change of only one parameter easier - for example simulation speed in case it was chosen incorrectly – and it saves a new input of all parameter.
- *Cancel*: The simulation is stopped and the program is redirected to the first window. If now historic traffic is chosen, the parameter input window does not show the last inputs, they all need to be chosen again.

As you can see in figure 32, the *Start* button can convert into a *Pause* and *Continue* button. Once the simulation is started, the *Start* button converts into a *Pause* button. If the user clicks this button, the two threads are paused and the button converts into a *Continue* button. Map, RPI and aircraft information panels are not cleared like with the *Stop* button but show the information from the moment the *Pause* button was clicked. By clicking *Continue*, the threads start running again, the button converts into a *Pause* button and the aircraft start moving over the map and on the RPI. After clicking the *Stop* button, the *Start* button is shown again.

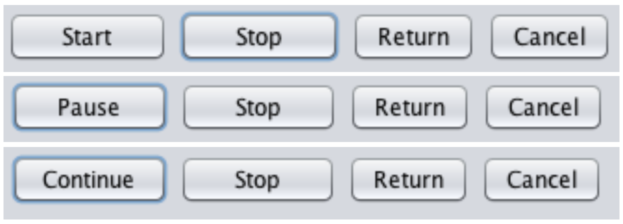


Figure 32. Buttons window 3.

16. Use case DUBLIN RWY 28

The main objective of this use case is to confirm and demonstrate the right functioning of the developed tool. The present use case aims to simulate historic traffic over the Point Merge approach procedure of runway 28 at Dublin airport.

16.1. Dublin Point Merge

The ICAO identification for Dublin Airport is EIDW. The airport has two Merge Point procedures. In December 2012, the Irish Aviation Authority (IAA) introduced Point Merge on Runway 28 supported by Eurocontrol and in April 2015 on Runway 10. In figure 32 the STAR for both Point Merges are illustrated.

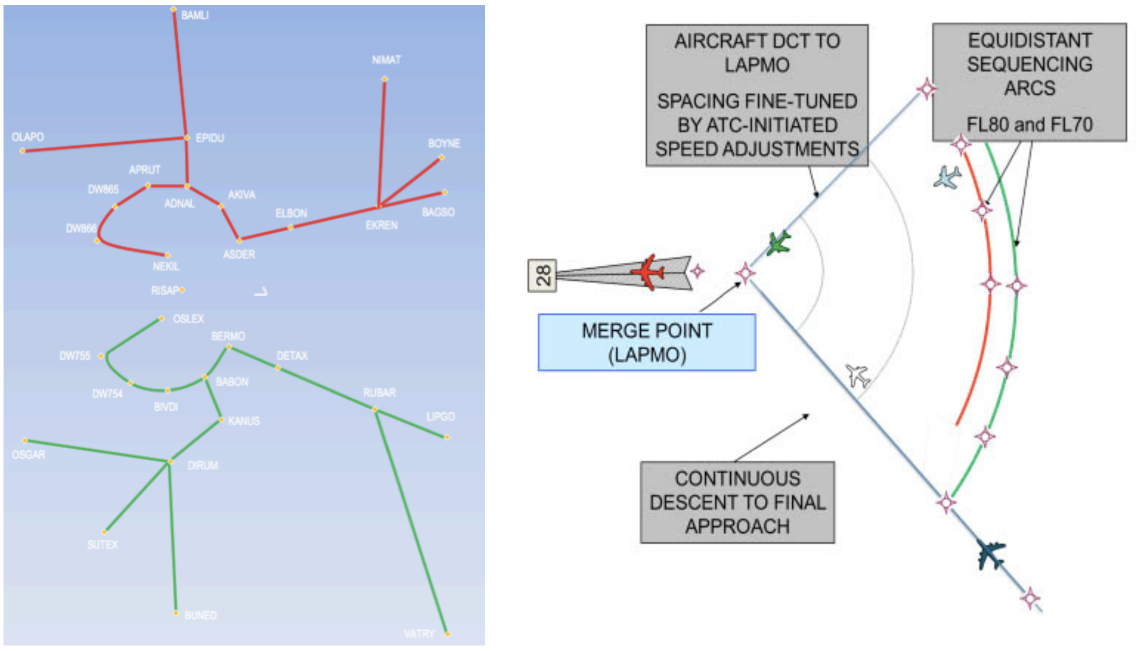


Figure 33. Point Merge in Dublin – runway 10 (left) and runway 28 (right).

Focusing on Runway 28, rather than using traditional holding stacks the system involves the placing of arriving aircraft onto defined equidistant arcs or tracks, from which they can make a continuous descent to the runway. The northern sequence legs need to be flown at FL80 and the southern at FL70. For both, IAS is fixed at 230 knots. The Merge Point (LAPMO) should not be overflown under 3000 feet. As the Point Merge Concept had originally been designed to increase ATC capacity and improved sequencing, sequence leg length was optimised for ATC use rather than flight and fuel efficiency.

Nevertheless, Point Merge and CDO make both possible: overall track miles flown by the aircraft and associated CO2 emissions reduction.

After the implementation of Point Merge for Runway 28 positive feedback from airlines and controllers were received. Even during the peak arrival flows controllers had been able to offer more aircraft direct routes and continuous descent approaches. The previously reported results of simulation studies for Dublin Point Merge, like enhanced situational awareness and reduced radio frequency transmissions, could be confirmed and allowed an increasing availability for tactical instructions [40], [41].

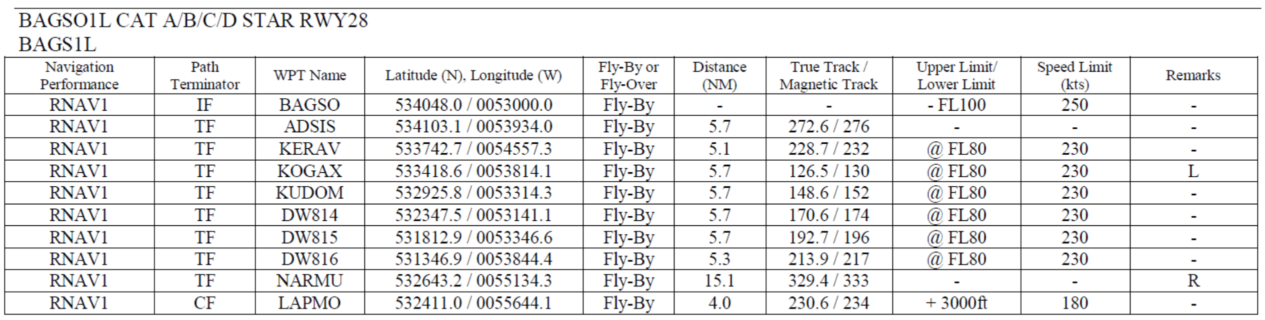
Despite an extensive awareness campaign for airlines informing about the flight planning and fueling, some problems arose. While the concept of Point Merge is easy to understand, fuel planning issues are more complicated. There were crews uplifting up to 400 kg of fuel per arrival in order to avoid any FMGC (Flight Management Guidance Computer) messages. As a consequence, fuel benefits of Point Merge were not achieved by all airlines, but would have, if the uncalled-for extra fuel had not been carried [41].

16.1.1. Dublin Merge Point Runway 28

The cart contained in the AIP of the approach can be seen in figure 34 (The whole AIP is available in appendix 1).

The waypoint name of the Merge Point is LAPMO and there are ten arrival routes available with the following initial fixes (IF): BAGSO, BAMLI, BOYNE, BUNED, ABLIN, NIMAT, OLAPO, OLAPO, SUTEX and VATRY.

An example of how the routes are defined in the AIP is given in table 6. where in each line a waypoint of the route is specified.



BAGSO1L CAT A/B/C/D STAR RWY28
BAGS1L

Navigation Performance	Path Terminator	WPT Name	Latitude (N), Longitude (W)	Fly-By or Fly-Over	Distance (NM)	True Track / Magnetic Track	Upper Limit/ Lower Limit	Speed Limit (kts)	Remarks
RNAV1	IF	BAGSO	534048.0 / 0053000.0	Fly-By	-	-	- FL100	250	-
RNAV1	TF	ADSIS	534103.1 / 0053934.0	Fly-By	5.7	272.6 / 276	-	-	-
RNAV1	TF	KERAV	533742.7 / 0054557.3	Fly-By	5.1	228.7 / 232	@ FL80	230	-
RNAV1	TF	KOGAX	533418.6 / 0053814.1	Fly-By	5.7	126.5 / 130	@ FL80	230	L
RNAV1	TF	KUDOM	532925.8 / 0053314.3	Fly-By	5.7	148.6 / 152	@ FL80	230	-
RNAV1	TF	DW814	532347.5 / 0053141.1	Fly-By	5.7	170.6 / 174	@ FL80	230	-
RNAV1	TF	DW815	531812.9 / 0053346.6	Fly-By	5.7	192.7 / 196	@ FL80	230	-
RNAV1	TF	DW816	531346.9 / 0053844.4	Fly-By	5.3	213.9 / 217	@ FL80	230	-
RNAV1	TF	NARMU	532643.2 / 0055134.3	Fly-By	15.1	329.4 / 333	-	-	R
RNAV1	CF	LAPMO	532411.0 / 0055644.1	Fly-By	4.0	230.6 / 234	+ 3000ft	180	-

Table 6. Route definition in the AIP.

It can be observed on the cart, that additionally to the merge holding, there are two traditional holding patterns whose holding fixes are over waypoint KERAV and SORIN.

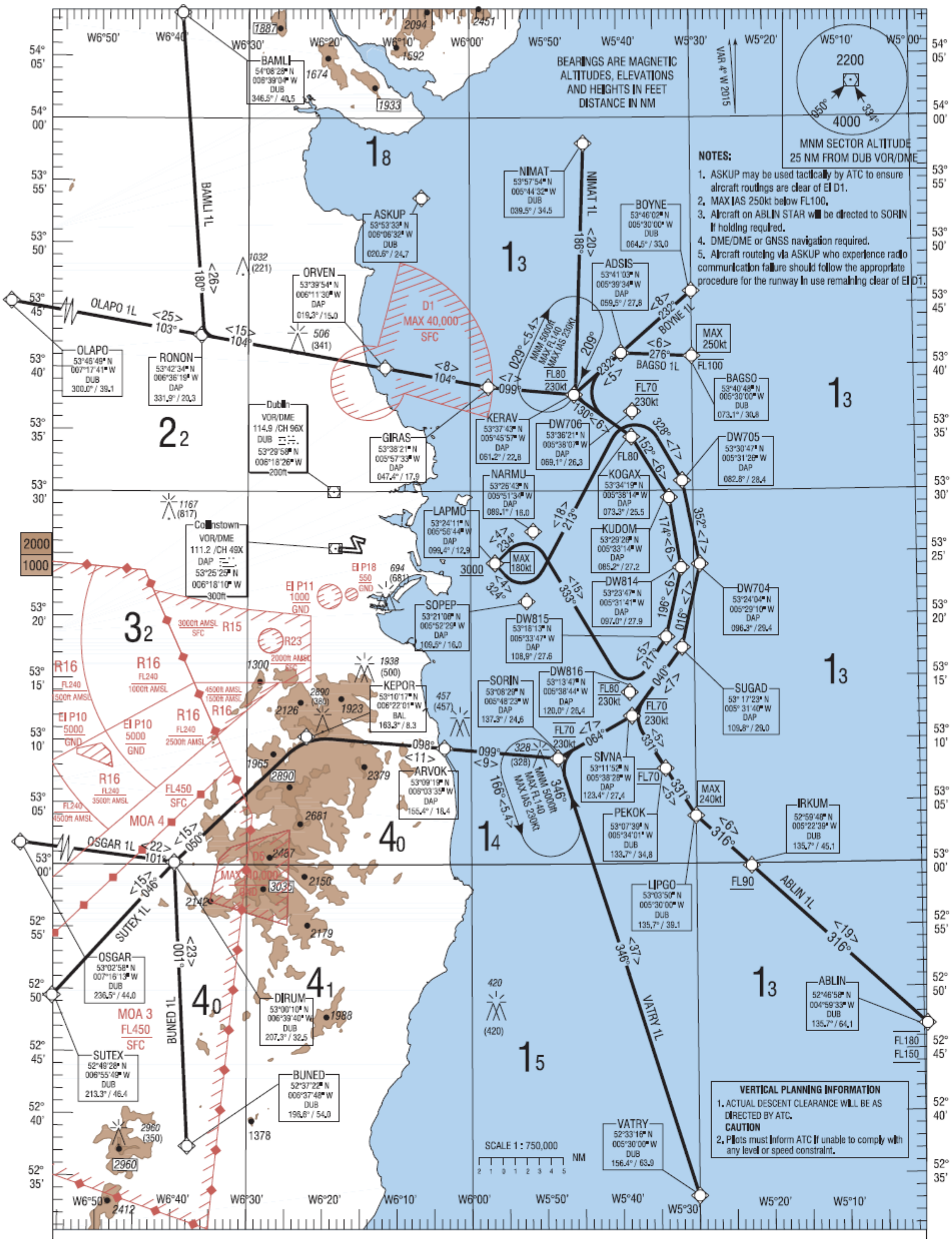

BEARINGS ARE MAGNETIC
ALTITUDES, ELEVATIONS AND HEIGHTS IN FEET
DISTANCE IN NM
MNM SECTOR ALTITUDE 25 NM FROM DUB VOR/DME
NOTES:
1. ASKUP may be used tactically by ATC to ensure aircraft routings are clear of EI D1.
2. MAX IAS 250kt below FL100.
3. Aircraft on ABLIN STAR will be directed to SORIN if holding required.
4. DME/DME or GNSS navigation required.
5. Aircraft routeing via ASKUP who experience radio communication failure should follow the appropriate procedure for the runway in use remaining clear of EI D1.
VERTICAL PLANNING INFORMATION
1. ACTUAL DESCENT CLEARANCE WILL BE AS DIRECTED BY ATC.
CAUTION
2. Pilots must inform ATC if unable to comply with any level or speed constraint.
SCALE 1 : 750,000
BAMLI
OLAPO
RONON
ASKUP
NIMAT
BOYNE
ADSIS
BAGSO
KERAV
DW706
GIRAS
NARMU
KOGAX
DW705
LAPMO
KUDOM
DW814
DW704
SOPEP
DW815
SORIN
DW816
SUGAD
KEPOR
ARVOK
SIVNA
PEKOK
IRKUM
LIPGO
OSGAR
DIRUM
SUTEX
BUNED
VATRY
ABLIN
D1 MAX 40,000 SFC
Dublin VOR/DME 114.9 /CH 96X DUB
Cornstown VOR/DME 111.2 /CH 49X DAP

Figure 34. AIP IRELAND AC 2.24 - 17.1 DUBLIN RWY 28.

16.2. Input files

The following files contain relevant information used by the tool for simulation of the desired approach. They can be divided into obligatory and recommended information:

Obligatory:

- .so6 file from Eurocontrol containing historic traffic which is filtered by the waypoint LAPMO, meaning they contain flights which performed the STAR of the use case
- .xlsx file with the routes information from the AIP: the information of the AIP

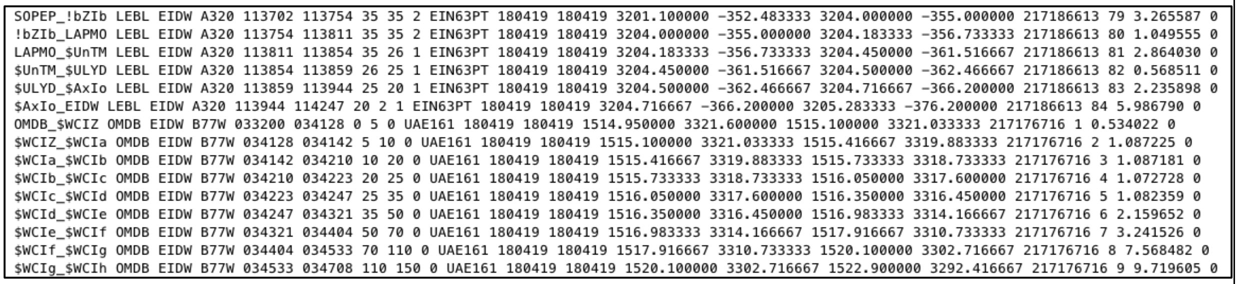
Recommended:

- .txt file with IFs information
- .txt file with Merge Point information
- .txt file with max time for RPI representation

The files are stored in the directory "Files". In the following table (table 7), the way information is stored in the different files, as well as the exact location of each file inside the "Files" carpet, is shown.

HistoricTrafficFiles / 20180419_20180419_0000_2359_LAPMO_m3.so6

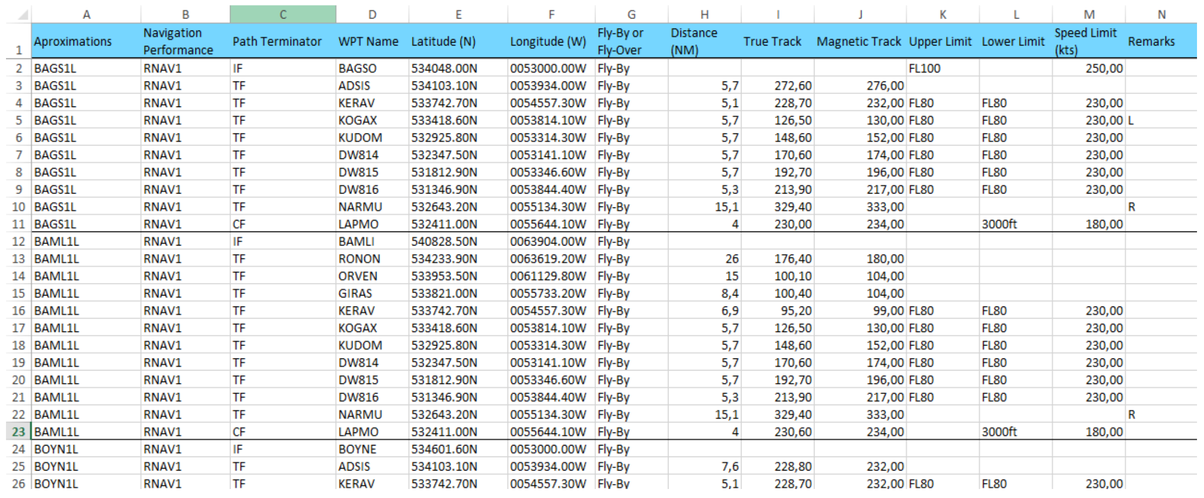
The name of the file indicates the following: the traffic is from 19th April, 2018, from 00:00 until 23:59. It contains all the traffics of this day which passed over the waypoint LAPMO and the file contains traffics obtained by method 3.



```
SOPEP_!bZIb LEBL EIDW A320 113702 113754 35 35 2 EIN63PT 180419 180419 3201.100000 -352.483333 3204.000000 -355.000000 217186613 79 3.265587 0
!bZIb_LAPMO LEBL EIDW A320 113754 113811 35 35 2 EIN63PT 180419 180419 3204.000000 -355.000000 3204.183333 -356.733333 217186613 80 1.049555 0
LAPMO_$UnTM LEBL EIDW A320 113811 113854 35 26 1 EIN63PT 180419 180419 3204.183333 -356.733333 3204.450000 -361.516667 217186613 81 2.864030 0
$UnTM_$ULYD LEBL EIDW A320 113854 113859 26 25 1 EIN63PT 180419 180419 3204.450000 -361.516667 3204.500000 -362.466667 217186613 82 0.568511 0
$ULYD_$AxIo LEBL EIDW A320 113859 113944 25 20 1 EIN63PT 180419 180419 3204.500000 -362.466667 3204.716667 -366.200000 217186613 83 2.235898 0
$AxIo_EIDW LEBL EIDW A320 113944 114247 20 2 1 EIN63PT 180419 180419 3204.716667 -366.200000 3205.283333 -376.200000 217186613 84 5.986790 0
OMDB_$WCIZ OMDB EIDW B77W 033200 034128 0 5 0 UAE161 180419 180419 1514.950000 3321.600000 1515.100000 3321.033333 217176716 1 0.534022 0
$WCIZ_$WCIa OMDB EIDW B77W 034128 034142 5 10 0 UAE161 180419 180419 1515.100000 3321.033333 1515.416667 3319.883333 217176716 2 1.087225 0
$WCIa_$WCIb OMDB EIDW B77W 034142 034210 10 20 0 UAE161 180419 180419 1515.416667 3319.883333 1515.733333 3318.733333 217176716 3 1.087181 0
$WCIb_$WCIc OMDB EIDW B77W 034210 034223 20 25 0 UAE161 180419 180419 1515.733333 3318.733333 1516.050000 3317.600000 217176716 4 1.072728 0
$WCIc_$WCId OMDB EIDW B77W 034223 034247 25 35 0 UAE161 180419 180419 1516.050000 3317.600000 1516.350000 3316.450000 217176716 5 1.082359 0
$WCId_$WCIe OMDB EIDW B77W 034247 034321 35 50 0 UAE161 180419 180419 1516.350000 3316.450000 1516.983333 3314.166667 217176716 6 2.159652 0
$WCIe_$WCIf OMDB EIDW B77W 034321 034404 50 70 0 UAE161 180419 180419 1516.983333 3314.166667 1517.916667 3310.733333 217176716 7 3.241526 0
$WCIf_$WCIg OMDB EIDW B77W 034404 034533 70 110 0 UAE161 180419 180419 1517.916667 3310.733333 1520.100000 3302.716667 217176716 8 7.568482 0
$WCIg_$WCIh OMDB EIDW B77W 034533 034708 110 150 0 UAE161 180419 180419 1520.100000 3302.716667 1522.900000 3292.416667 217176716 9 9.719605 0
```

Routes / Dublin RWY 28.xlsx

The whole file is to be seen in Appendix 2.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Aproximations	Navigation Performance	Path Terminator	WPT Name	Latitude (N)	Longitude (W)	Fly-By or Fly-Over	Distance (NM)	True Track	Magnetic Track	Upper Limit	Lower Limit	Speed Limit (kts)	Remarks
2	BAGS1L	RNAV1	IF	BAGSO	534048.00N	0053000.00W	Fly-By				FL100		250,00	
3	BAGS1L	RNAV1	TF	ADSIS	534103.10N	0053934.00W	Fly-By	5,7	272,60	276,00				
4	BAGS1L	RNAV1	TF	KERAV	533742.70N	0054557.30W	Fly-By	5,1	228,70	232,00	FL80	FL80	230,00	
5	BAGS1L	RNAV1	TF	KOGAX	533418.60N	0053814.10W	Fly-By	5,7	126,50	130,00	FL80	FL80	230,00	L
6	BAGS1L	RNAV1	TF	KUDOM	532925.80N	0053314.30W	Fly-By	5,7	148,60	152,00	FL80	FL80	230,00	
7	BAGS1L	RNAV1	TF	DW814	532347.50N	0053141.10W	Fly-By	5,7	170,60	174,00	FL80	FL80	230,00	
8	BAGS1L	RNAV1	TF	DW815	531812.90N	0053346.60W	Fly-By	5,7	192,70	196,00	FL80	FL80	230,00	
9	BAGS1L	RNAV1	TF	DW816	531346.90N	0053844.40W	Fly-By	5,3	213,90	217,00	FL80	FL80	230,00	
10	BAGS1L	RNAV1	TF	NARMU	532643.20N	0055134.30W	Fly-By	15,1	329,40	333,00				R
11	BAGS1L	RNAV1	CF	LAPMO	532411.00N	0055644.10W	Fly-By	4	230,00	234,00		3000ft	180,00	
12	BAML1L	RNAV1	IF	BAMLI	540828.50N	0063904.00W	Fly-By							
13	BAML1L	RNAV1	TF	RONON	534233.90N	0063619.20W	Fly-By	26	176,40	180,00				
14	BAML1L	RNAV1	TF	ORVEN	533953.50N	0061129.80W	Fly-By	15	100,10	104,00				
15	BAML1L	RNAV1	TF	GIRAS	533821.00N	0055733.20W	Fly-By	8,4	100,40	104,00				
16	BAML1L	RNAV1	TF	KERAV	533742.70N	0054557.30W	Fly-By	6,9	95,20	99,00	FL80	FL80	230,00	
17	BAML1L	RNAV1	TF	KOGAX	533418.60N	0053814.10W	Fly-By	5,7	126,50	130,00	FL80	FL80	230,00	
18	BAML1L	RNAV1	TF	KUDOM	532925.80N	0053314.30W	Fly-By	5,7	148,60	152,00	FL80	FL80	230,00	
19	BAML1L	RNAV1	TF	DW814	532347.50N	0053141.10W	Fly-By	5,7	170,60	174,00	FL80	FL80	230,00	
20	BAML1L	RNAV1	TF	DW815	531812.90N	0053346.60W	Fly-By	5,7	192,70	196,00	FL80	FL80	230,00	
21	BAML1L	RNAV1	TF	DW816	531346.90N	0053844.40W	Fly-By	5,3	213,90	217,00	FL80	FL80	230,00	
22	BAML1L	RNAV1	TF	NARMU	532643.20N	0055134.30W	Fly-By	15,1	329,40	333,00				R
23	BAML1L	RNAV1	CF	LAPMO	532411.00N	0055644.10W	Fly-By	4	230,60	234,00		3000ft	180,00	
24	BOYN1L	RNAV1	IF	BOYNE	534601.60N	0053000.00W	Fly-By							
25	BOYN1L	RNAV1	TF	ADSIS	534103.10N	0053934.00W	Fly-By	7,6	228,80	232,00				
26	BOYN1L	RNAV1	TF	KERAV	533742.70N	0054557.30W	Fly-By	5,1	228,70	232,00	FL80	FL80	230,00	

Data / mergePoints.txt

```
LAPMO; LAPMO
DUBLIN RWY 28; LAPMO
DUBLIN RWY 08
```

Data / RPI_times.txt

```
Dublin RWY 28.xlsx: 720
```

Data / waypointsIF.txt

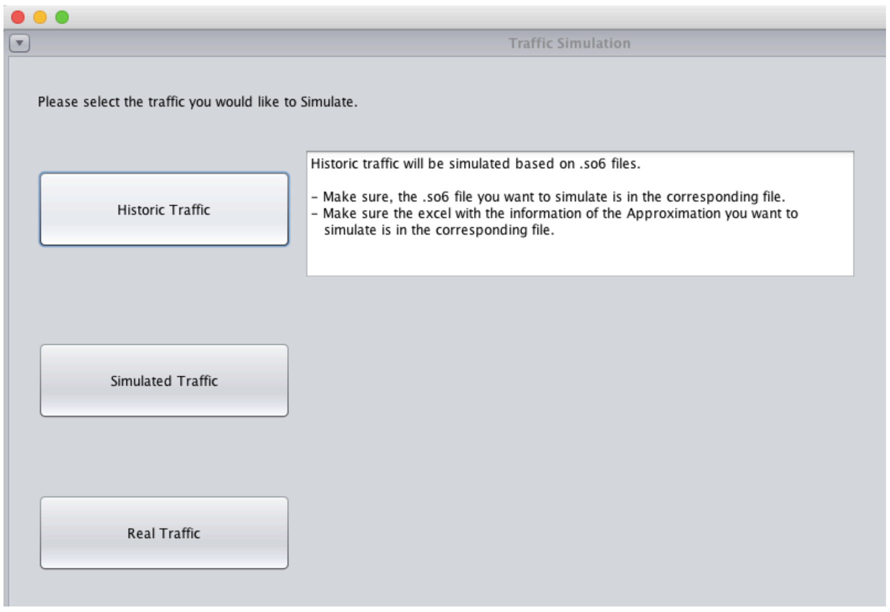
```
DUBLIN RWY 28;  NIMAT, OLAPO, OSGAR, SUTEX, VATRY, ABLIN, BUNED, BOYNE, BAMLI, BAGSOZUM
```

Table 7. File Structures and location.

Once it is confirmed that all the files contain the correct information, introduced in the required format and stored in the right carpet, the program execution can be started.

16.3. Program execution

After starting the program, in the first window shown “Historic Traffic” is chosen, as it is the kind of traffic to be simulated in this use case (figure 35).



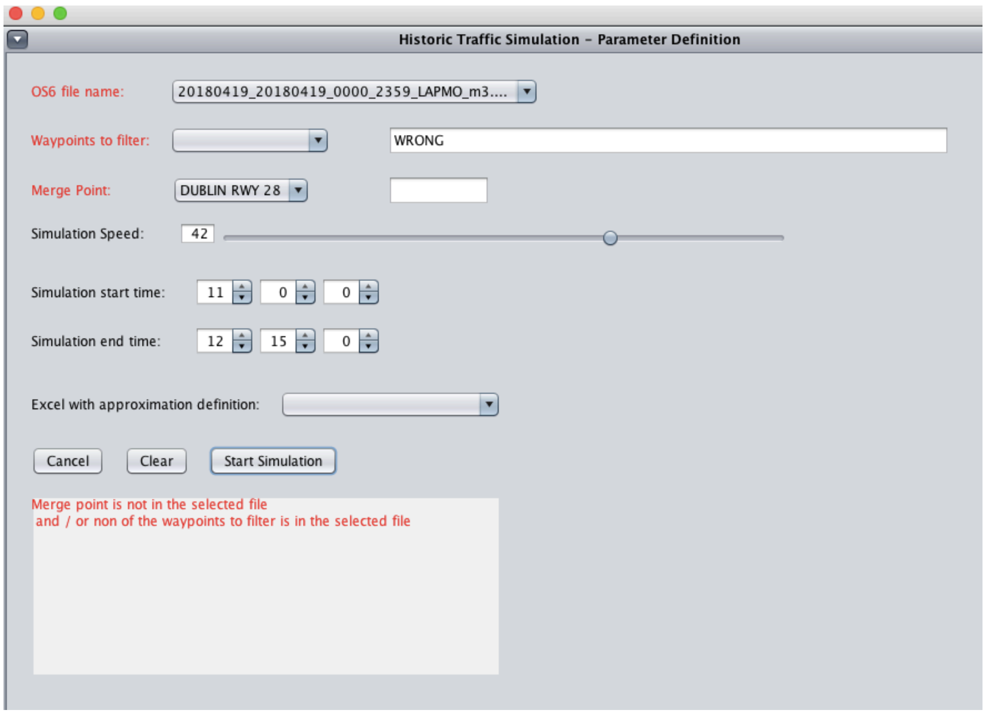
Please select the traffic you would like to Simulate.

Historic traffic will be simulated based on .so6 files.

- Make sure, the .so6 file you want to simulate is in the corresponding file.
- Make sure the excel with the information of the Approximation you want to simulate is in the corresponding file.

Figure 35. Chosing historic traffic.

This leads to a second window asking to specify the simulation parameters. It can be observed, that in the combo boxes Waypoints to filter and Merge Point the information contained in the files about IFs and Merge Points is shown. In the combo boxes, we can chooseSO6 file name and Excel with routes definition among all the files available in the corresponding carpet. As for confirmation of the right functioning of the program, in a first step an erroneous parameter is chosen for simulation: The IFs do not correspond to the chosen so6 file.


Historic Traffic Simulation – Parameter Definition

OS6 file name: 20180419_20180419_0000_2359_LAPMO_m3....

Waypoints to filter: WRONG

Merge Point: DUBLIN RWY 28

Simulation Speed: 42

Simulation start time: 11 0 0

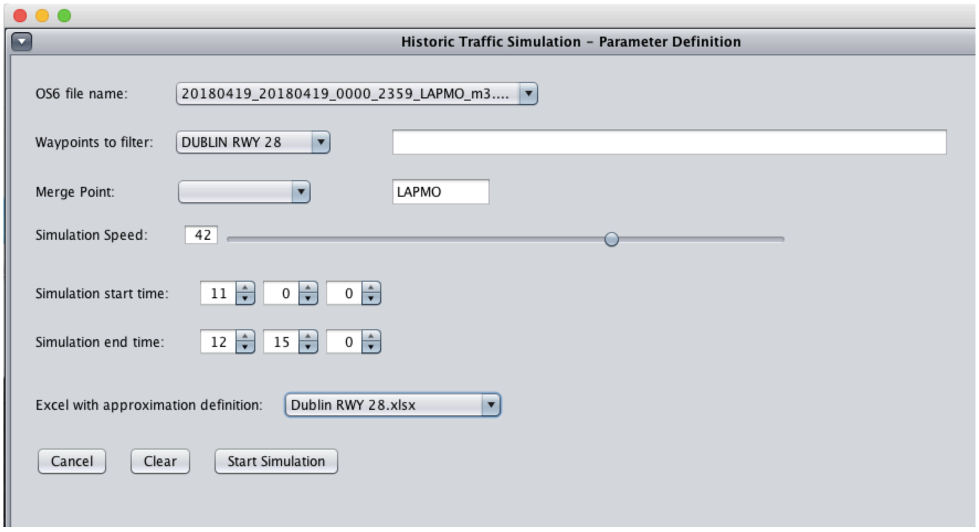
Simulation end time: 12 15 0

Excel with approximation definition:

Merge point is not in the selected file
and / or non of the waypoints to filter is in the selected file

Figure 36. Example wrong parameter input.

The error is indicated in the test area and parameters to be modified are marked in red (figure 36). The right functioning of the input error identification and notification can be confirmed. The input is cleared, correct parameters are introduced (figure 37) and simulation started.


Historic Traffic Simulation – Parameter Definition

OS6 file name: 20180419_20180419_0000_2359_LAPMO_m3....

Waypoints to filter: DUBLIN RWY 28

Merge Point: LAPMO

Simulation Speed: 42

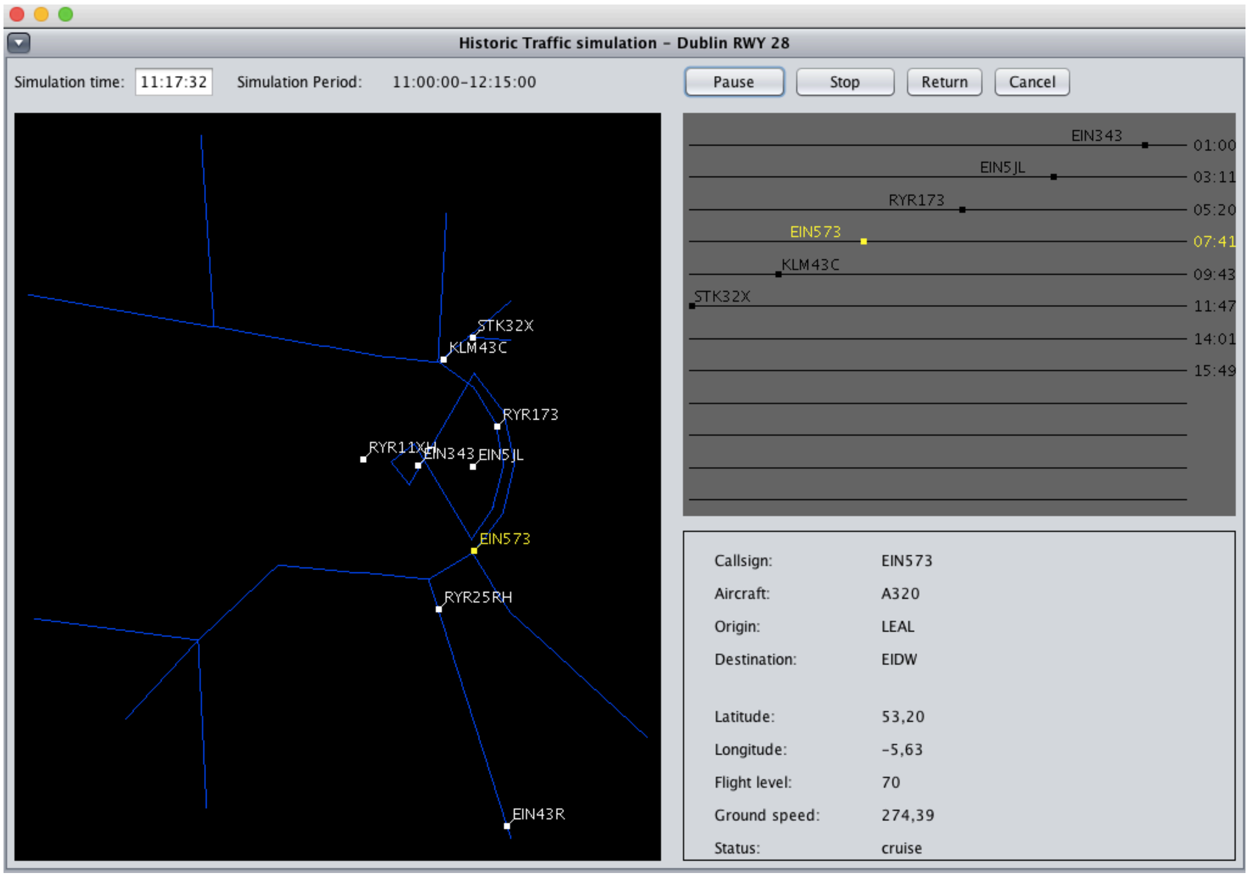
Simulation start time: 11 0 0

Simulation end time: 12 15 0

Excel with approximation definition: Dublin RWY 28.xlsx

Figure 37. Parameter input.

This leads to the main window where the simulation is shown. The simulation does not start until the button “Start” is pressed. That is when aircraft, identified by their callsign, start moving and can be observed over the defined approach (the left side of the window) and the RPI (right upper part of the window). Also, the simulation time starts running (upper left position). If an aircraft is chosen by a mouse click over the map or the RPI during the simulation, the colour of the corresponding aircraft over the map and the RPI as well as the time left to reach the Merge Point LAPMO are changed to yellow. Additionally, in the lower right part, more information about the flight chosen is shown (see figure 38). During the program execution, the right functioning of all the buttons (Start, Pause, Continue, Stop, Return, and Cancel) could be confirmed.



Simulation time: 11:17:32 Simulation Period: 11:00:00-12:15:00

Callsign:	EIN573
Aircraft:	A320
Origin:	LEAL
Destination:	EIDW
Latitude:	53,20
Longitude:	-5,63
Flight level:	70
Ground speed:	274,39
Status:	cruise

Figure 38. Traffic simulation with a chosen aircraft.

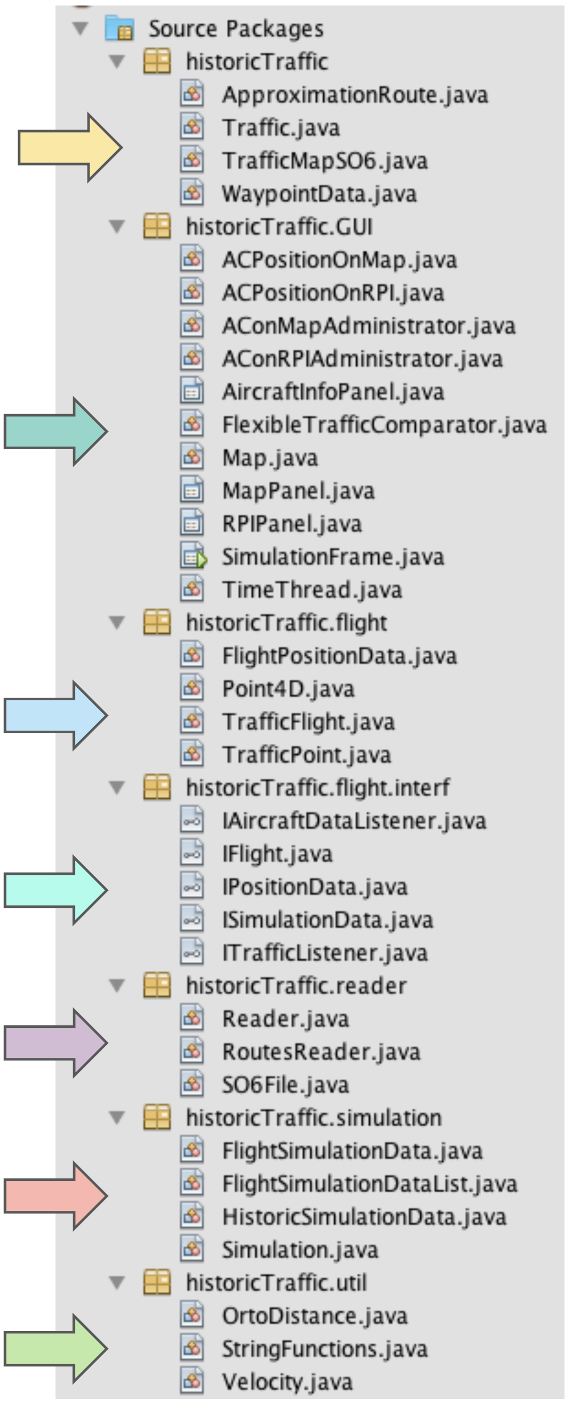
16.4. Conclusion of the Use Case

The present use case helped to confirm the right functioning of the developed tool. It could be checked, that:

- all buttons work the way they should
- simulation and time threads run properly and can be paused and stopped
- the routes of the approach are represented correctly
- among the simulated aircraft, you can choose one to obtain further information

PART 5 – PROGRAM IMPLEMENTATION

In this section, the internal functioning of the program will be explained. After explaining the packages, their main function and important classes, the interfaces created for the program will be presented briefly and finally all the classes with their methods will be expounded.



```
Source Packages
    historicTraffic
        ApproximationRoute.java
        Traffic.java
        TrafficMapSO6.java
        WaypointData.java
    historicTraffic.GUI
        ACPositionOnMap.java
        ACPositionOnRPI.java
        AConMapAdministrator.java
        AConRPIAdministrator.java
        AircraftInfoPanel.java
        FlexibleTrafficComparator.java
        Map.java
        MapPanel.java
        RPIPanel.java
        SimulationFrame.java
        TimeThread.java
    historicTraffic.flight
        FlightPositionData.java
        Point4D.java
        TrafficFlight.java
        TrafficPoint.java
    historicTraffic.flight.interf
        IAircraftDataListener.java
        IFlight.java
        IPositionData.java
        ISimulationData.java
        ITrafficListener.java
    historicTraffic.reader
        Reader.java
        RoutesReader.java
        SO6File.java
    historicTraffic.simulation
        FlightSimulationData.java
        FlightSimulationDataList.java
        HistoricSimulationData.java
        Simulation.java
    historicTraffic.util
        OrtoDistance.java
        StringFunctions.java
        Velocity.java
```

Figure 39. Package structure.

17. Packages

The classes are grouped in seven packages (see figure 39):

historicTraffic

This package contains the decoded information provided by the input files (.so6 and .xlsx).

historiTraffic.GUI

The classes in this package are those necessary for the graphic representation. This package contains the *main* method which is inside the *SimulationFrame* class, also containing the three windows (traffic to simulate, parameter definition and traffic simulation).

The classes *MapPanel*, *RPIPanel* and *AircraftInfoPanel* are placed over the traffic simulation window and have been explained in detail in the previous section.

TimeThread shows the simulation time running from start to end time.

historicTraffic.flight

It is composed by the classes necessary to define an airplane containing all the important information needed for the simulation and representation in the GUI.

historicTraffic.flight.interf

This package contains all the interfaces of the program. The interfaces are explained in detail in a posterior section.

historicTraffic.reader

The classes in this package have methods to decode different files. *Reader* has methods to read all the text files with the Merge Point, maxRPItime and waypoint information. *RoutesReader* can read the excel file with the routes information and *SO6File* has a method to decode .so6 files.

historicTraffic.simulation

The simulation package contains all the information necessary for the simulation. The class *Simulation* is a thread that provides for each simulation instant the information about all aircraft and sends through the *ITrafficListener* the information to the GUI.

historicTraffic.util

It contains auxiliary functions to work with strings and calculate velocities or distances.

17.1. Package interaction

The following graph represents the interaction between the different packages indicating which is the method they use for interaction (figure 40). It can also be read from the figure, which interface is used in which package. Figure 41 goes more into detail, showing the interaction between the classes and the methods they are using.

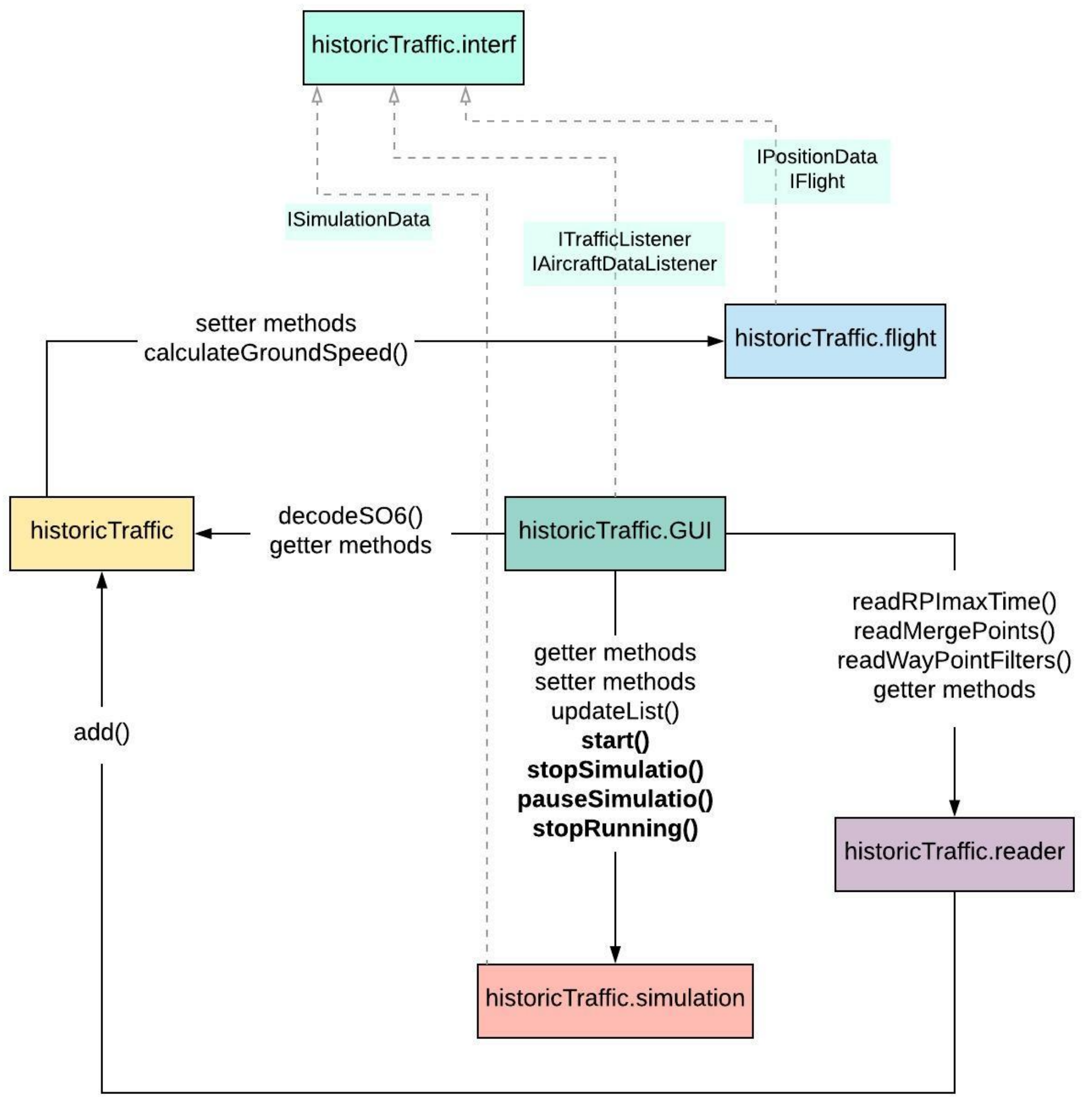


Figure 40. Package interaction and interfaces.

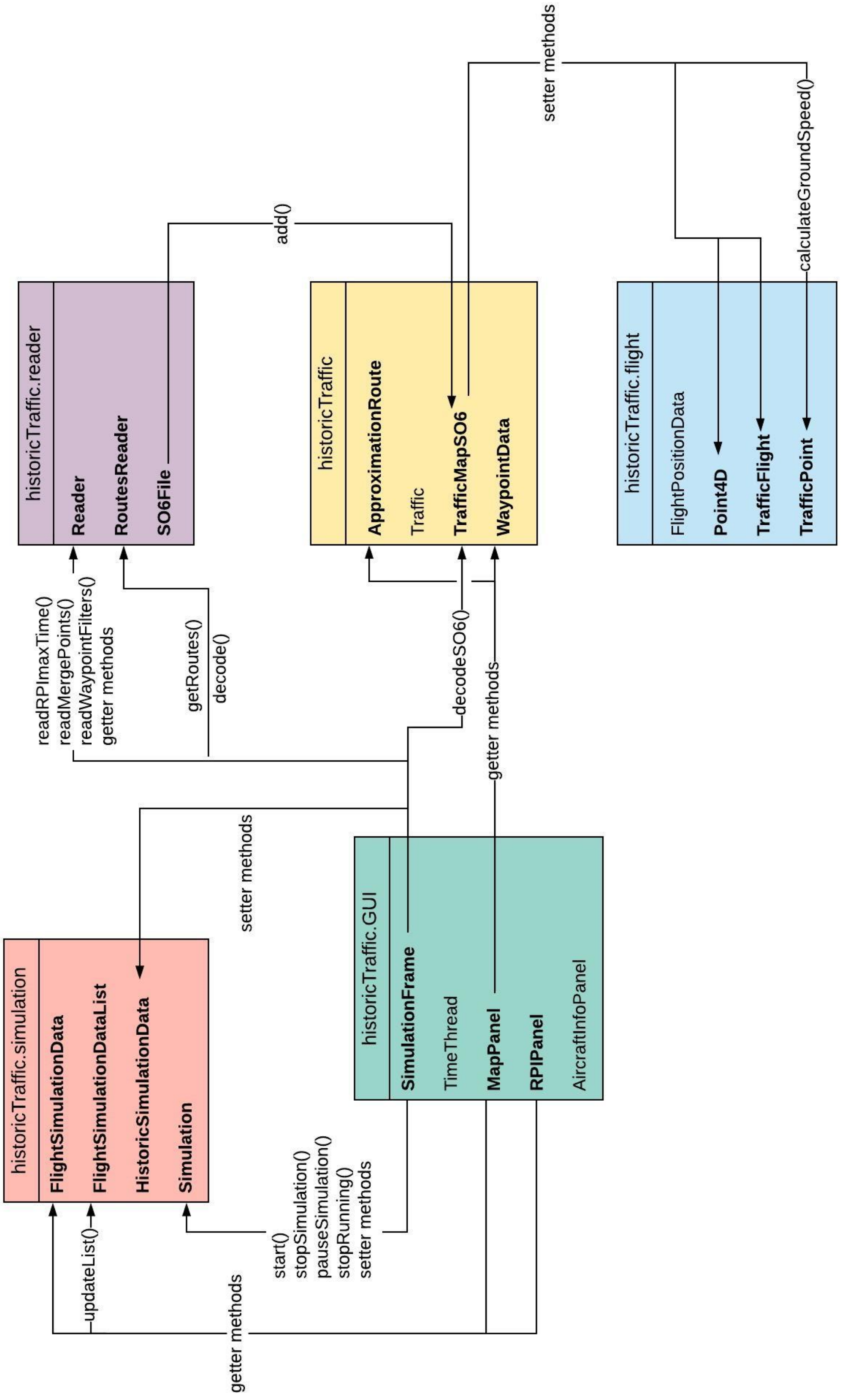


Figure 41. Class interactions.

18. Interfaces

In this section, all the interfaces created for the program are described. The attached graphs show all the methods of the corresponding interface and the classes implementing it.

18.1. ISimulationData

The interface ISimulationData is implemented by the class HistoricSimulationData, figure 42. This interface was created to be used by the three kinds of traffics to simulate, historic-, simulated- and real traffic. It defines the minimum methods required for the simulation.

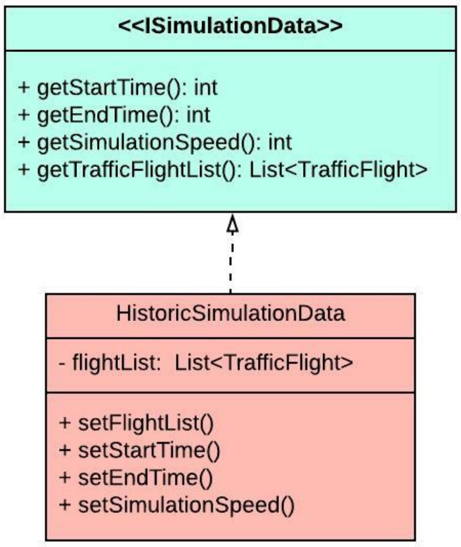


Figure 42. ISimulationData.

18.2. IPositionData

The interface IPositionData is implemented by the FlightPositionData class, figure 43. It describes the necessary information the simulation requires to define a traffic in one point flying a Point Merge approach.

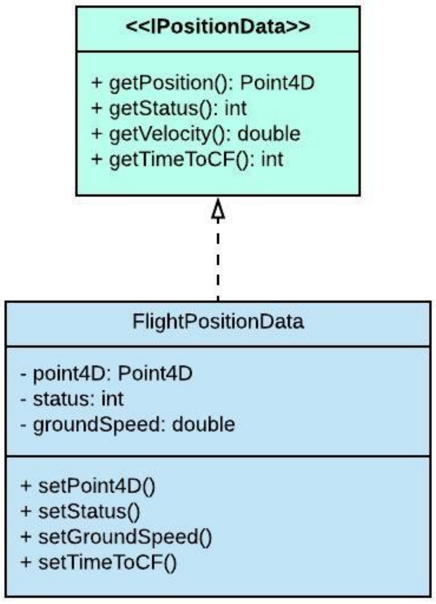


Figure 43. IPositionData.

18.3. IFlight

The interface IFlight is implemented by the class TrafficFlight, figure 44. It describes the aircraft position at the specified time.

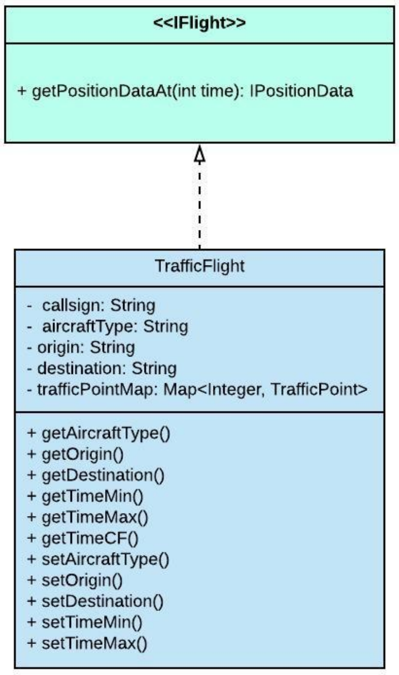
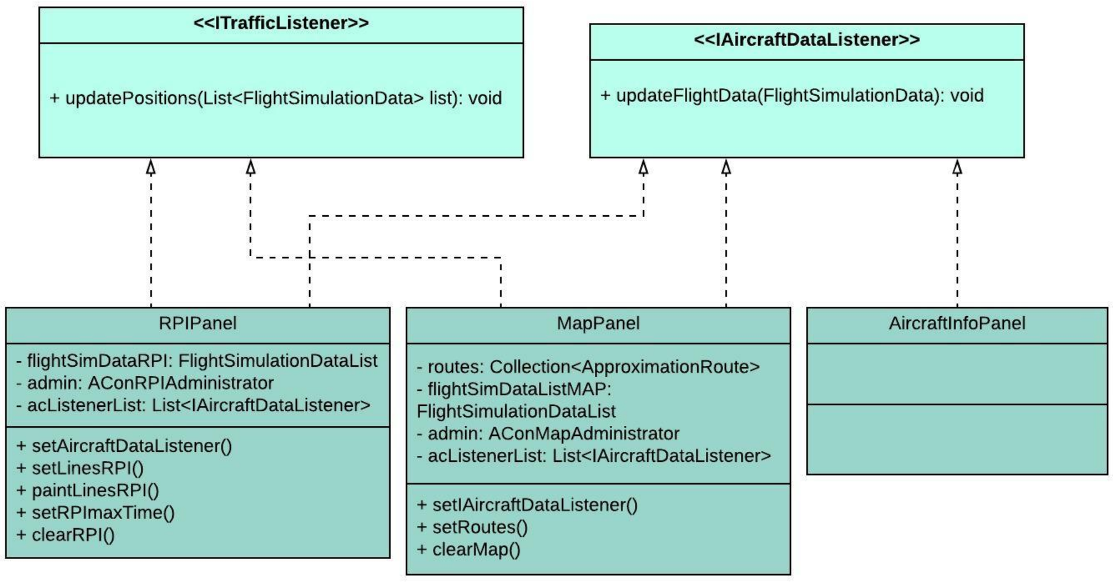


Figure 44. IFlight.

18.4. ITrafficListener

The RPIPanel and the MapPanel implement the interface ITrafficListener. The implemented method, to listen and receive the actual List with all the FlightSimulationData, figure 45.



<<ITrafficListener>>
+ updatePositions(List<FlightSimulationData> list): void
<<IAircraftDataListener>>
+ updateFlightData(FlightSimulationData): void
RPIPanel
- flightSimDataRPI: FlightSimulationDataList
- admin: AConRPIAdministrator
- acListenerList: List<IAircraftDataListener>
+ setAircraftDataListener()
+ setLinesRPI()
+ paintLinesRPI()
+ setRPImaxTime()
+ clearRPI()
MapPanel
- routes: Collection<ApproximationRoute>
- flightSimDataListMAP: FlightSimulationDataList
- admin: AConMapAdministrator
- acListenerList: List<IAircraftDataListener>
+ setIAircraftDataListener()
+ setRoutes()
+ clearMap()
AircraftInfoPanel

Figure 45. ITrafficListener and IAircraftDataListener.

18.5. IAircraftDataListener

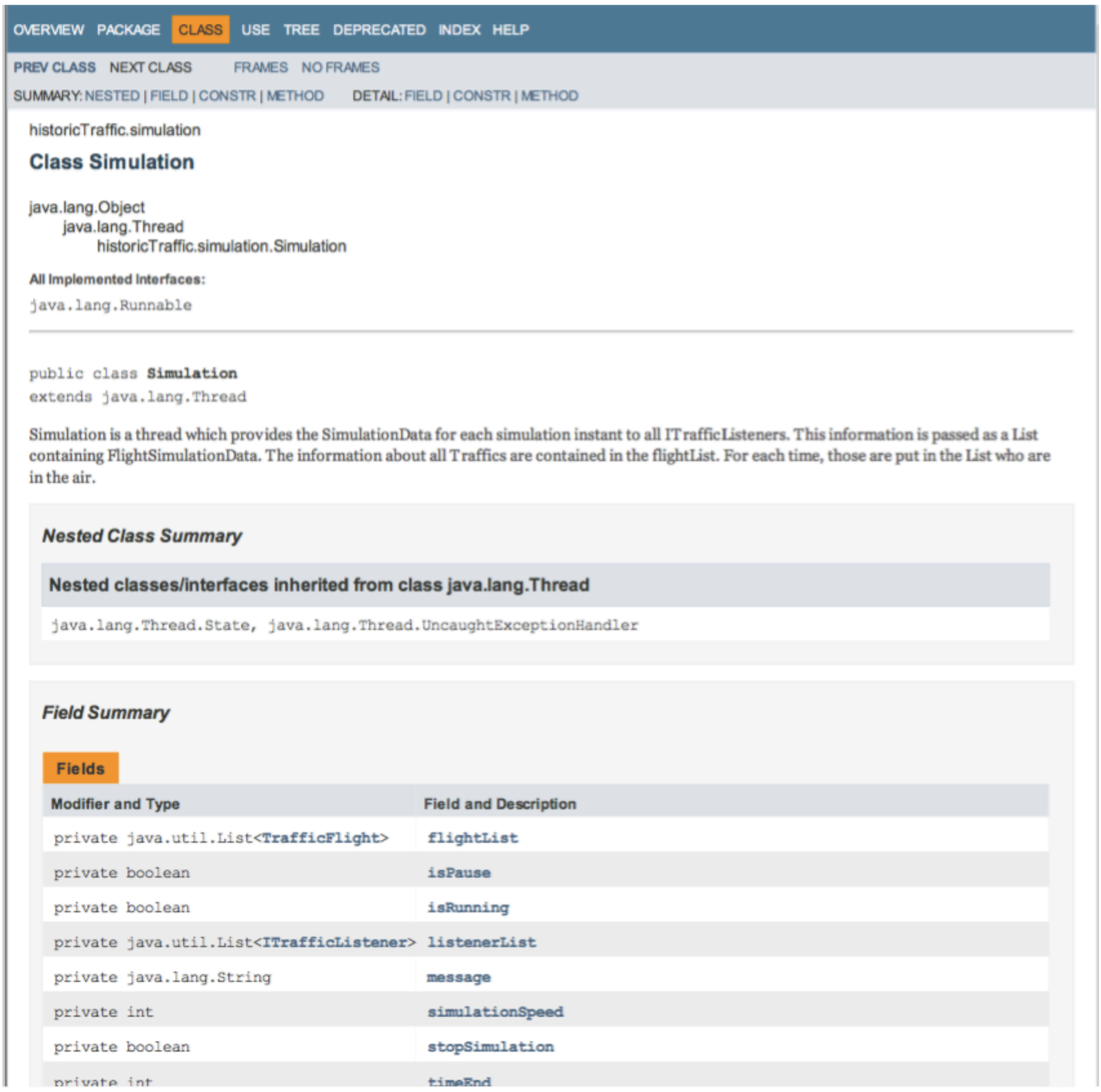
The interface IAircraftDataListener is implemented by the three classes RPIPanel, MapPanel and AircraftInfoPanel. The implemented method is used to listen and receive the updated information of one aircraft.

In this program, the interface is used to update all classes implementing this interface about the chosen aircraft, so it can be represented in yellow or it can show the aircraft information.

19. Java Class Documentation

In this section, the javadocs for some of the most important classes is documented. The documentation for all classes can be accessed through the following link: Traffic Simulation Tool/javadoc/index.html .

19.1. Simulation


OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

historicTraffic.simulation

Class Simulation

```
java.lang.Object
    java.lang.Thread
        historicTraffic.simulation.Simulation
```

All Implemented Interfaces:

`java.lang.Runnable`

```
public class Simulation
extends java.lang.Thread
```

Simulation is a thread which provides the SimulationData for each simulation instant to all ITrafficListeners. This information is passed as a List containing FlightSimulationData. The information about all Traffics are contained in the flightList. For each time, those are put in the List who are in the air.

Nested Class Summary

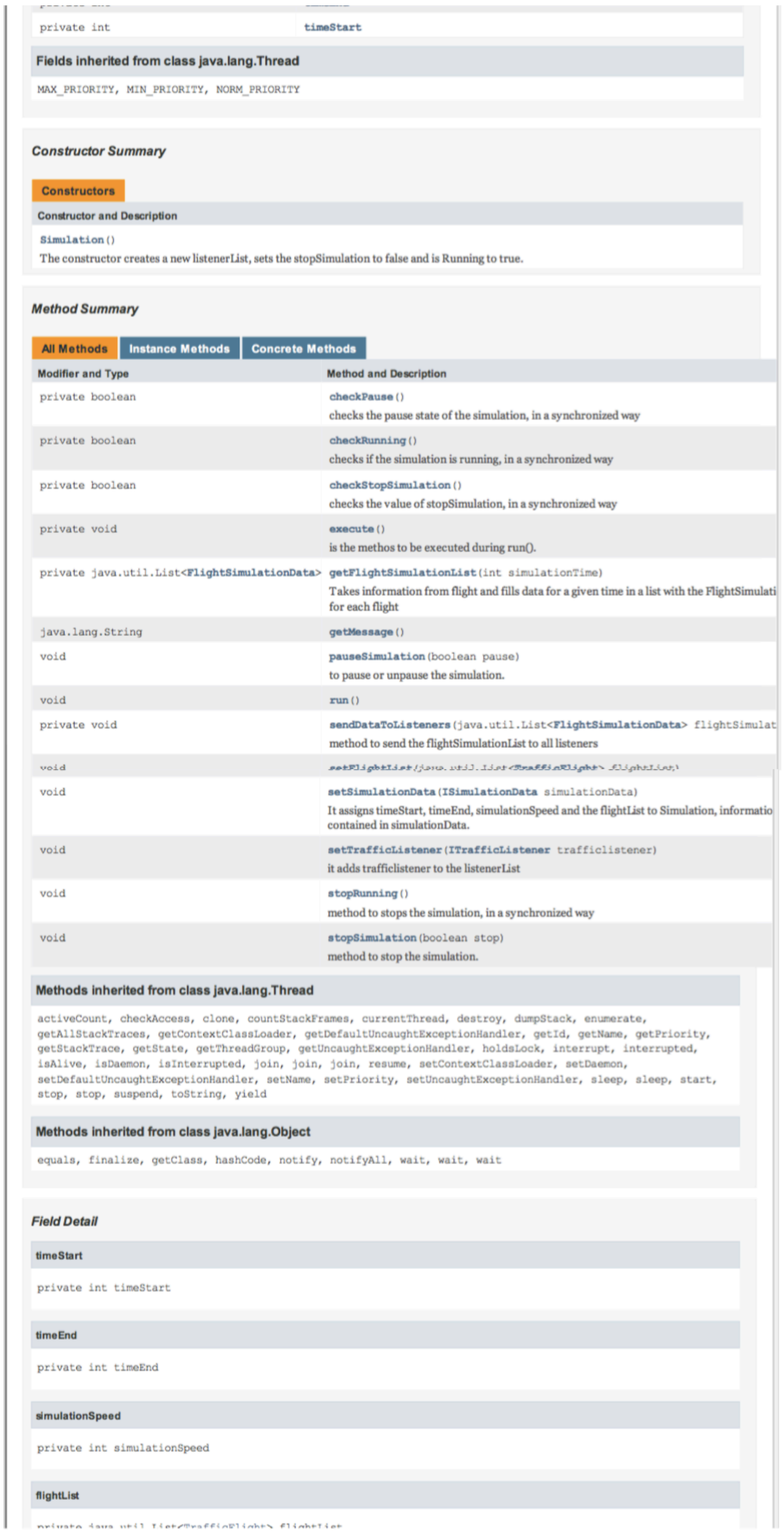
Nested classes/interfaces inherited from class java.lang.Thread

`java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler`

Field Summary

Fields

Modifier and Type	Field and Description
private java.util.List<TrafficFlight>	flightList
private boolean	isPause
private boolean	isRunning
private java.util.List<ITrafficListener>	listenerList
private java.lang.String	message
private int	simulationSpeed
private boolean	stopSimulation
private int	timeEnd



private int	**timeStart**

Fields inherited from class java.lang.Thread

`MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY`

Constructor Summary

Constructors

Constructor and Description
Simulation()   The constructor creates a new listenerList, sets the stopSimulation to false and is Running to true.

Method Summary

All Methods **Instance Methods** **Concrete Methods**

Modifier and Type	Method and Description
private boolean	**checkPause**() checks the pause state of the simulation, in a synchronized way
private boolean	**checkRunning**() checks if the simulation is running, in a synchronized way
private boolean	**checkStopSimulation**() checks the value of stopSimulation, in a synchronized way
private void	**execute**() is the methos to be executed during run().
private java.util.List<**FlightSimulationData**>	**getFlightSimulationList**(int simulationTime) Takes information from flight and fills data for a given time in a list with the FlightSimulati for each flight
java.lang.String	**getMessage**()
void	**pauseSimulation**(boolean pause) to pause or unpause the simulation.
void	**run**()
private void	**sendDataToListeners**(java.util.List<**FlightSimulationData**> flightSimulat method to send the flightSimulationList to all listeners
void	**setFlightList**(java.util.List<**TrafficFlight**> flightList)
void	**setSimulationData**(**ISimulationData** simulationData) It assigns timeStart, timeEnd, simulationSpeed and the flightList to Simulation, informatio contained in simulationData.
void	**setTrafficListener**(**ITrafficListener** trafficlistener) it adds trafficlistener to the listenerList
void	**stopRunning**() method to stops the simulation, in a synchronized way
void	**stopSimulation**(boolean stop) method to stop the simulation.

Methods inherited from class java.lang.Thread

```
activeCount, checkAccess, clone, countStackFrames, currentThread, destroy, dumpStack, enumerate,
getAllStackTraces, getContextClassLoader, getDefaultUncaughtExceptionHandler, getId, getName, getPriority,
getStackTrace, getState, getThreadGroup, getUncaughtExceptionHandler, holdsLock, interrupt, interrupted,
isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon,
setDefaultUncaughtExceptionHandler, setName, setPriority, setUncaughtExceptionHandler, sleep, sleep, start,
stop, stop, suspend, toString, yield
```

Methods inherited from class java.lang.Object

```
equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait
```

Field Detail

timeStart

```
private int timeStart
```

timeEnd

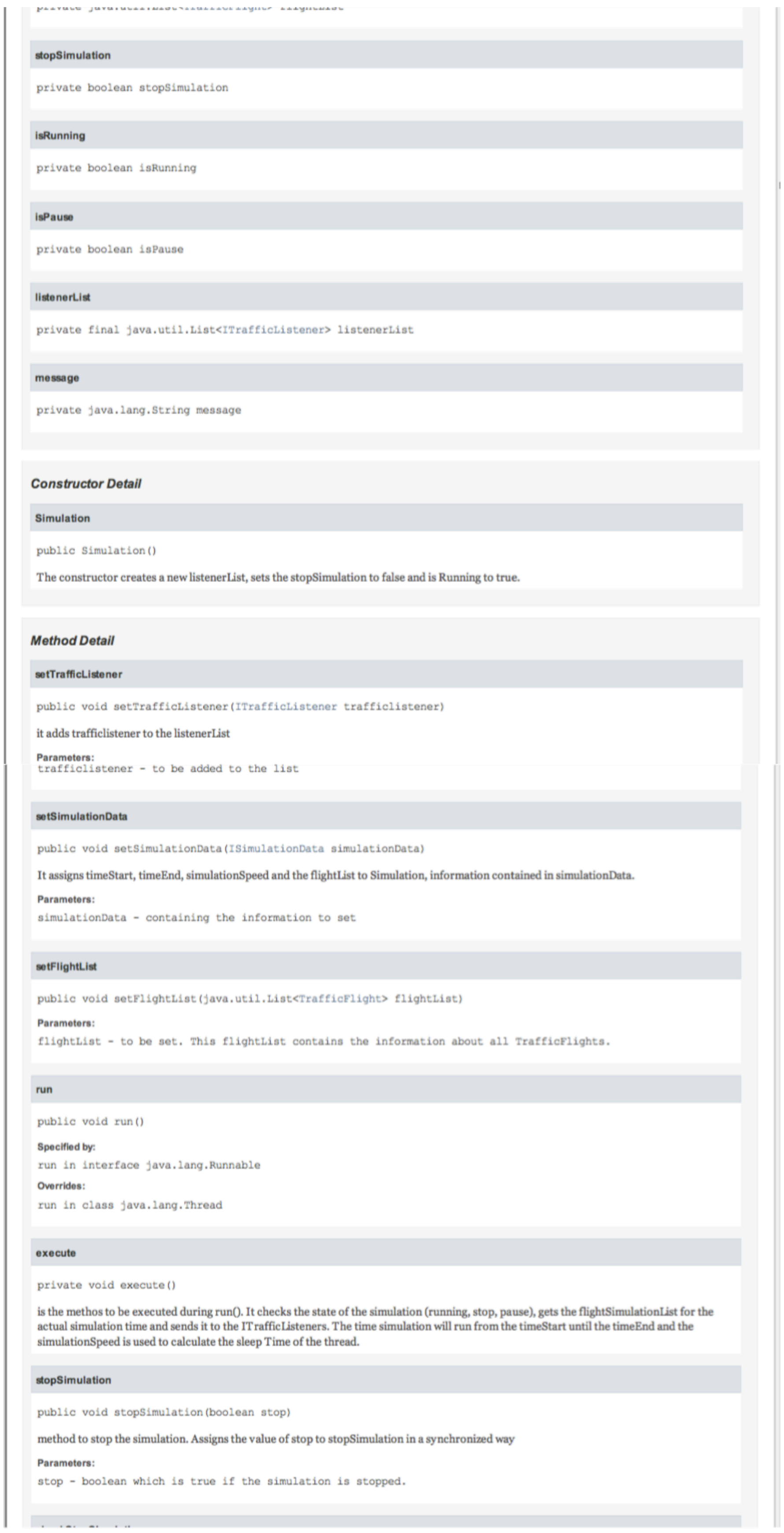
```
private int timeEnd
```

simulationSpeed

```
private int simulationSpeed
```

flightList

```
private java.util.List<TrafficFlight> flightList
```



private java.util.List<TrafficFlight> flightList

stopSimulation

```
private boolean stopSimulation
```

isRunning

```
private boolean isRunning
```

isPause

```
private boolean isPause
```

listenerList

```
private final java.util.List<ITrafficListener> listenerList
```

message

```
private java.lang.String message
```

Constructor Detail

Simulation

```
public Simulation()
```

The constructor creates a new listenerList, sets the stopSimulation to false and is Running to true.

Method Detail

setTrafficListener

```
public void setTrafficListener(ITrafficListener trafficlistener)
```

it adds trafficlistener to the listenerList

Parameters:
`trafficlistener - to be added to the list`

setSimulationData

```
public void setSimulationData(ISimulationData simulationData)
```

It assigns timeStart, timeEnd, simulationSpeed and the flightList to Simulation, information contained in simulationData.

Parameters:
`simulationData - containing the information to set`

setFlightList

```
public void setFlightList(java.util.List<TrafficFlight> flightList)
```

Parameters:
`flightList - to be set. This flightList contains the information about all TrafficFlights.`

run

```
public void run()
```

Specified by:
`run in interface java.lang.Runnable`

Overrides:
`run in class java.lang.Thread`

execute

```
private void execute()
```

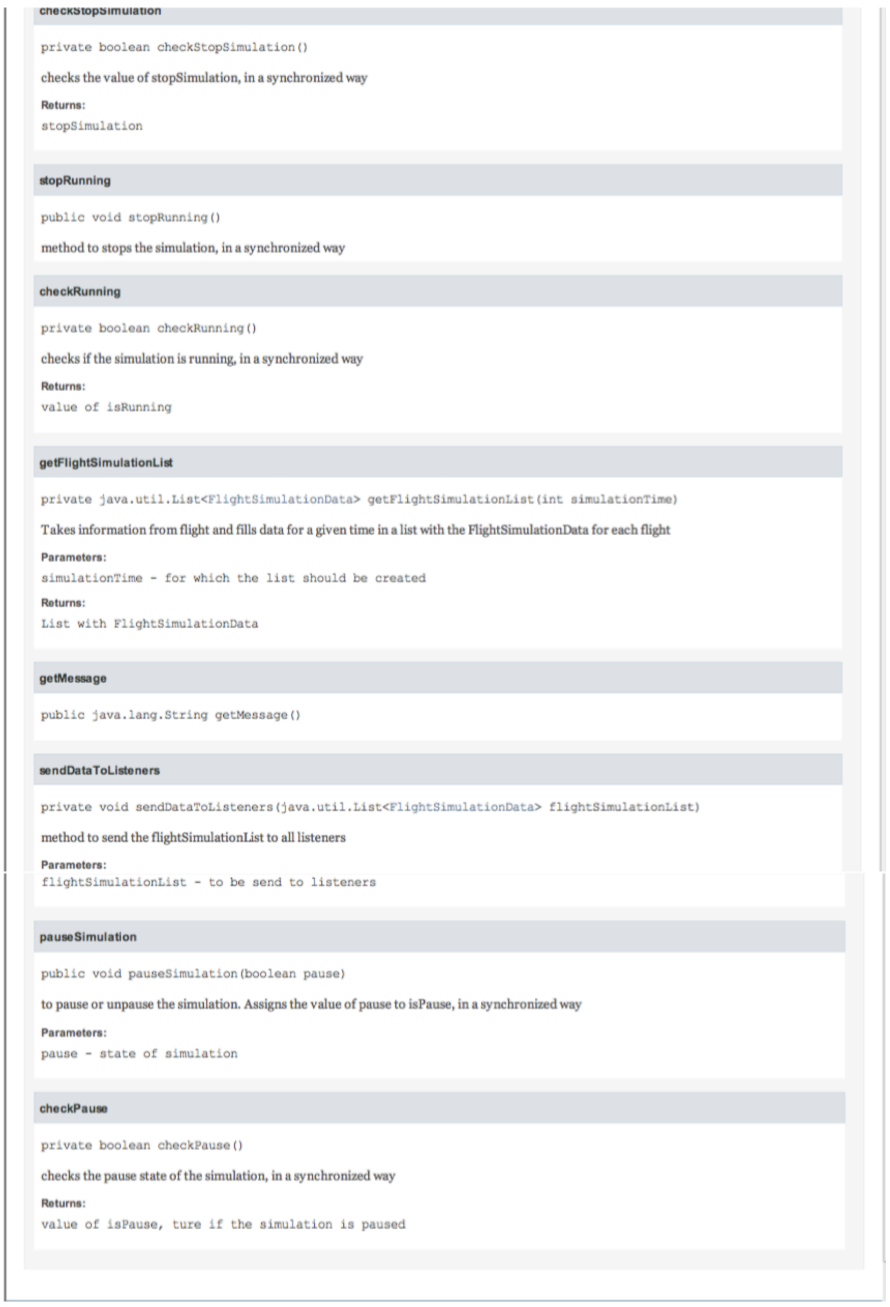
is the methos to be executed during run(). It checks the state of the simulation (running, stop, pause), gets the flightSimulationList for the actual simulation time and sends it to the ITrafficListeners. The time simulation will run from the timeStart until the timeEnd and the simulationSpeed is used to calculate the sleep Time of the thread.

stopSimulation

```
public void stopSimulation(boolean stop)
```

method to stop the simulation. Assigns the value of stop to stopSimulation in a synchronized way

Parameters:
`stop - boolean which is true if the simulation is stopped.`



checkStopSimulation

```
private boolean checkStopSimulation()
```

checks the value of stopSimulation, in a synchronized way

Returns:
`stopSimulation`

stopRunning

```
public void stopRunning()
```

method to stops the simulation, in a synchronized way

checkRunning

```
private boolean checkRunning()
```

checks if the simulation is running, in a synchronized way

Returns:
`value of isRunning`

getFlightSimulationList

```
private java.util.List<FlightSimulationData> getFlightSimulationList(int simulationTime)
```

Takes information from flight and fills data for a given time in a list with the FlightSimulationData for each flight

Parameters:
`simulationTime - for which the list should be created`

Returns:
`List with FlightSimulationData`

getMessage

```
public java.lang.String getMessage()
```

sendDataToListeners

```
private void sendDataToListeners(java.util.List<FlightSimulationData> flightSimulationList)
```

method to send the flightSimulationList to all listeners

Parameters:
`flightSimulationList - to be send to listeners`

pauseSimulation

```
public void pauseSimulation(boolean pause)
```

to pause or unpause the simulation. Assigns the value of pause to isPause, in a synchronized way

Parameters:
`pause - state of simulation`

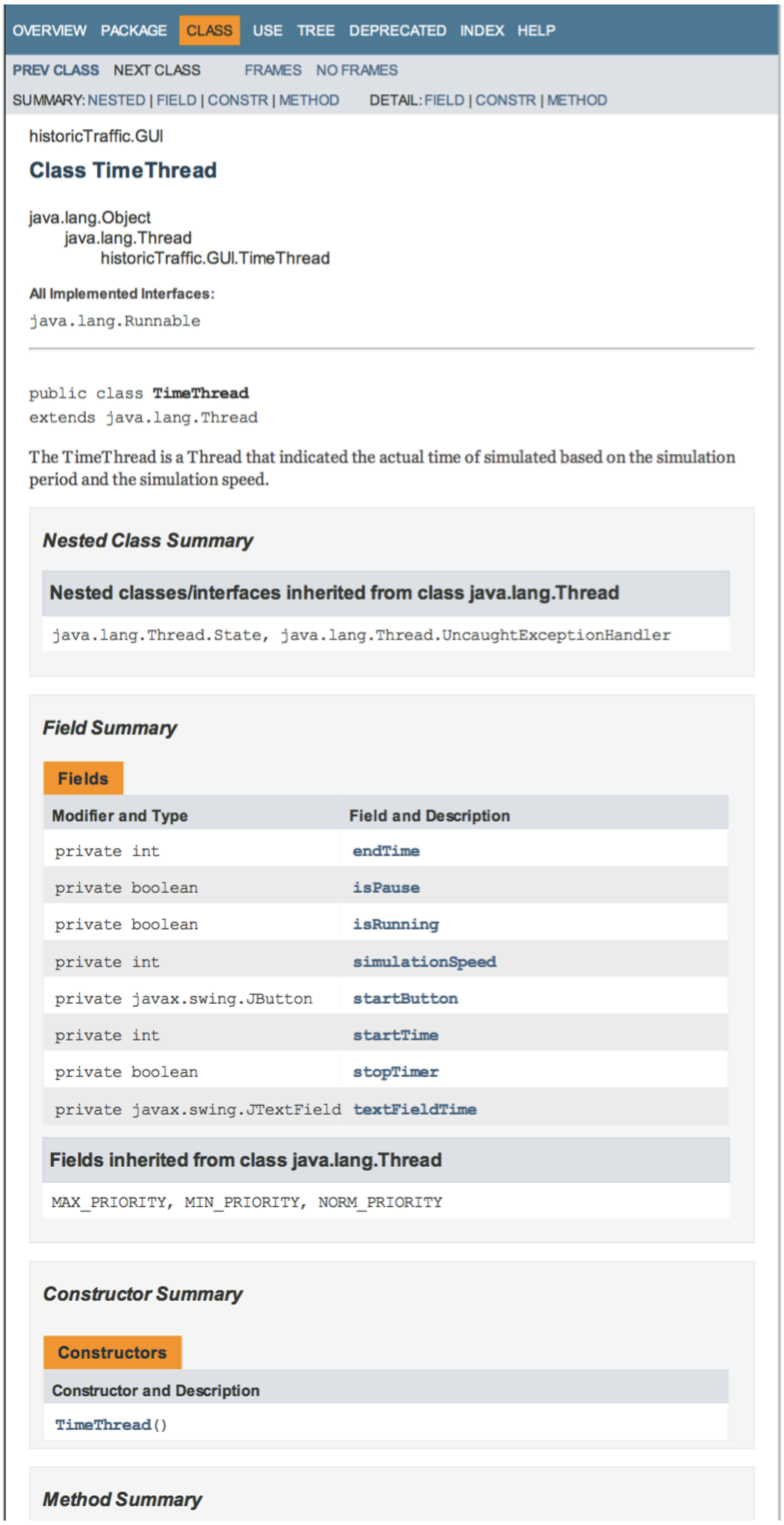
checkPause

```
private boolean checkPause()
```

checks the pause state of the simulation, in a synchronized way

Returns:
`value of isPause, ture if the simulation is paused`

19.2. TimeThread



OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

historicTraffic.GUI

Class TimeThread

```
java.lang.Object
    java.lang.Thread
        historicTraffic.GUI.TimeThread
```

All Implemented Interfaces:

`java.lang.Runnable`

```
public class TimeThread
extends java.lang.Thread
```

The TimeThread is a Thread that indicated the actual time of simulated based on the simulation period and the simulation speed.

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread

`java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler`

Field Summary

Fields

Modifier and Type	Field and Description
`private int`	**endTime**
`private boolean`	**isPause**
`private boolean`	**isRunning**
`private int`	**simulationSpeed**
`private javax.swing.JButton`	**startButton**
`private int`	**startTime**
`private boolean`	**stopTimer**
`private javax.swing.JTextField`	**textFieldTime**

Fields inherited from class java.lang.Thread

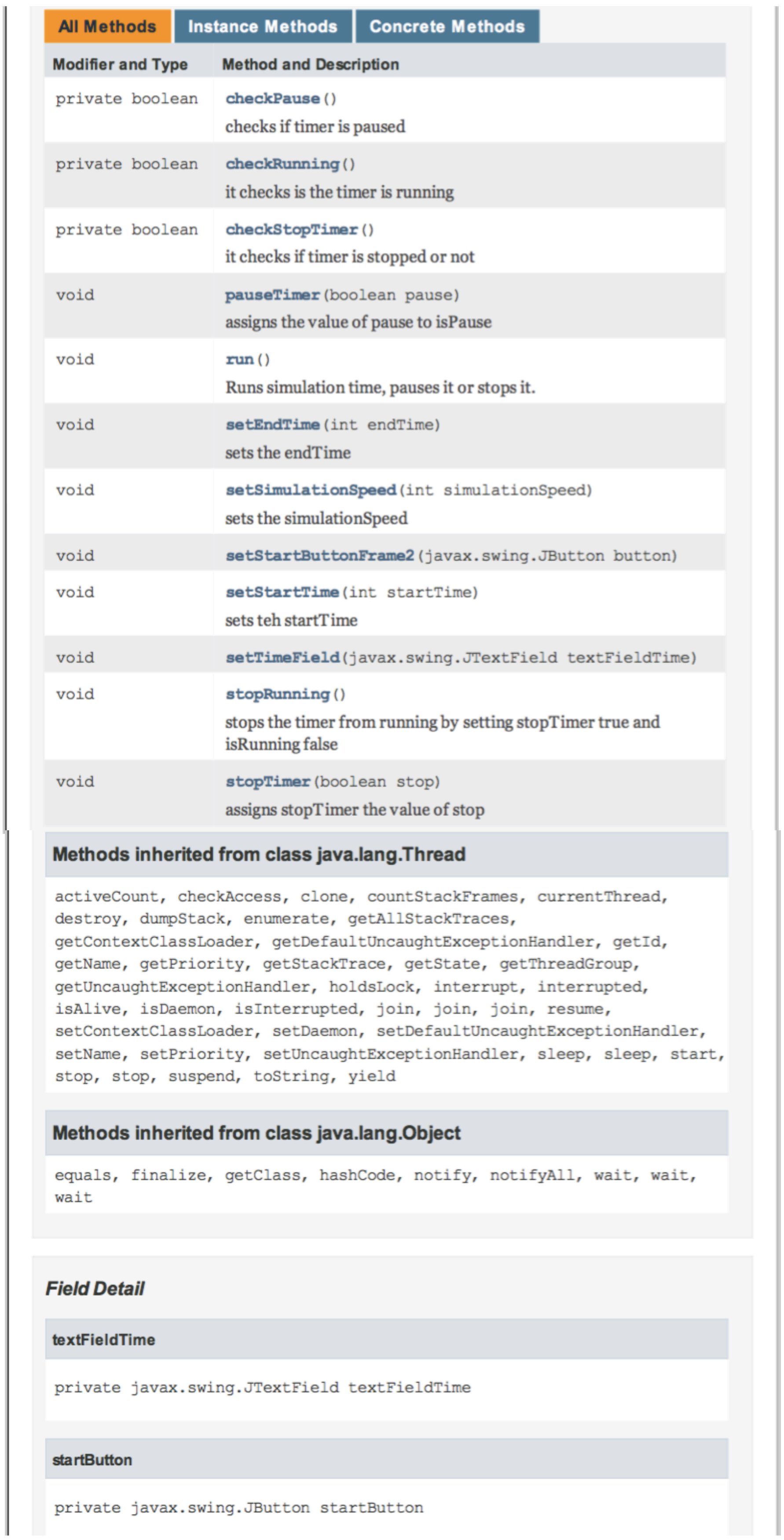
`MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY`

Constructor Summary

Constructors

Constructor and Description
`TimeThread()`

Method Summary



Modifier and Type	Method and Description
`private boolean`	`checkPause()` checks if timer is paused
`private boolean`	`checkRunning()` it checks is the timer is running
`private boolean`	`checkStopTimer()` it checks if timer is stopped or not
`void`	`pauseTimer(boolean pause)` assigns the value of pause to isPause
`void`	`run()` Runs simulation time, pauses it or stops it.
`void`	`setEndTime(int endTime)` sets the endTime
`void`	`setSimulationSpeed(int simulationSpeed)` sets the simulationSpeed
`void`	`setStartButtonFrame2(javax.swing.JButton button)`
`void`	`setStartTime(int startTime)` sets teh startTime
`void`	`setTimeField(javax.swing.JTextField textFieldTime)`
`void`	`stopRunning()` stops the timer from running by setting stopTimer true and isRunning false
`void`	`stopTimer(boolean stop)` assigns stopTimer the value of stop

Methods inherited from class java.lang.Thread

```
activeCount, checkAccess, clone, countStackFrames, currentThread,
destroy, dumpStack, enumerate, getAllStackTraces,
getContextClassLoader, getDefaultUncaughtExceptionHandler, getId,
getName, getPriority, getStackTrace, getState, getThreadGroup,
getUncaughtExceptionHandler, holdsLock, interrupt, interrupted,
isAlive, isDaemon, isInterrupted, join, join, join, resume,
setContextClassLoader, setDaemon, setDefaultUncaughtExceptionHandler,
setName, setPriority, setUncaughtExceptionHandler, sleep, sleep, start,
stop, stop, suspend, toString, yield
```

Methods inherited from class java.lang.Object

```
equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait,
wait
```

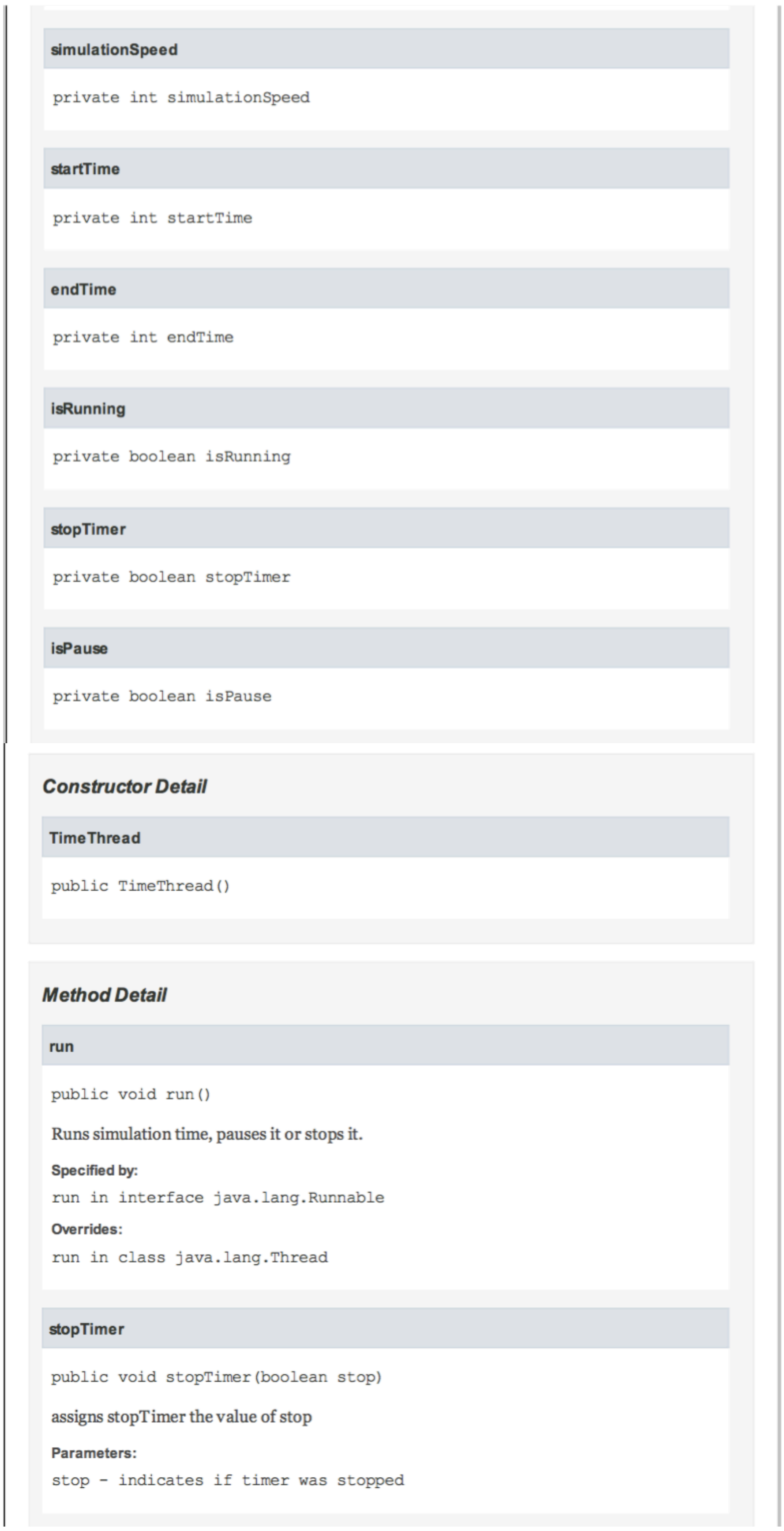
Field Detail

textFieldTime

```
private javax.swing.JTextField textFieldTime
```

startButton

```
private javax.swing.JButton startButton
```



simulationSpeed

```
private int simulationSpeed
```

startTime

```
private int startTime
```

endTime

```
private int endTime
```

isRunning

```
private boolean isRunning
```

stopTimer

```
private boolean stopTimer
```

isPause

```
private boolean isPause
```

Constructor Detail

TimeThread

```
public TimeThread()
```

Method Detail

run

```
public void run()
```

Runs simulation time, pauses it or stops it.

Specified by:

`run in interface java.lang.Runnable`

Overrides:

`run in class java.lang.Thread`

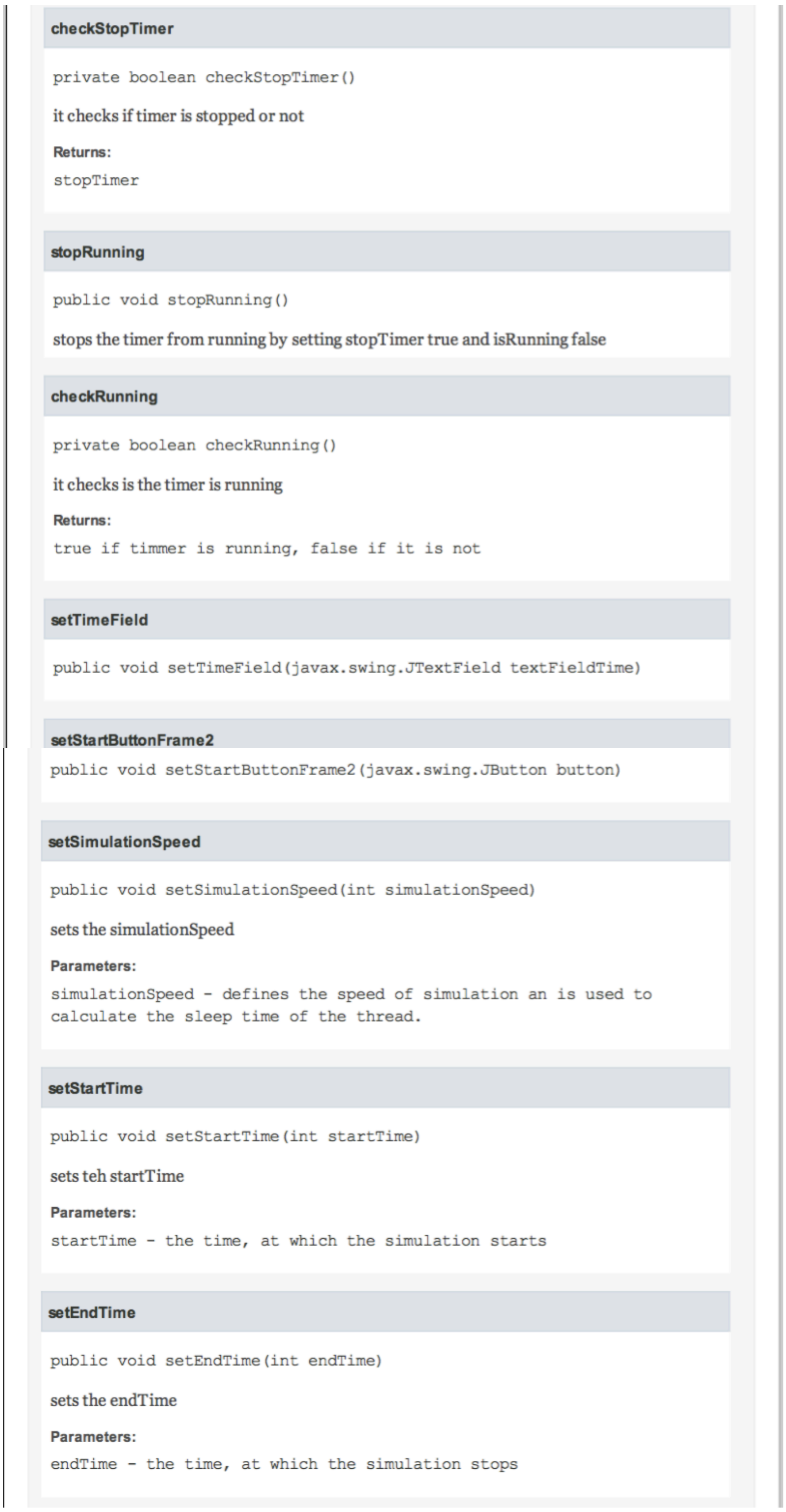
stopTimer

```
public void stopTimer(boolean stop)
```

assigns stopTimer the value of stop

Parameters:

`stop - indicates if timer was stopped`



checkStopTimer

```
private boolean checkStopTimer()
```

it checks if timer is stopped or not

Returns:

```
stopTimer
```

stopRunning

```
public void stopRunning()
```

stops the timer from running by setting stopTimer true and isRunning false

checkRunning

```
private boolean checkRunning()
```

it checks is the timer is running

Returns:

```
true if timmer is running, false if it is not
```

setTimeField

```
public void setTimeField(javax.swing.JTextField textFieldTime)
```

setStartButtonFrame2

```
public void setStartButtonFrame2(javax.swing.JButton button)
```

setSimulationSpeed

```
public void setSimulationSpeed(int simulationSpeed)
```

sets the simulationSpeed

Parameters:

```
simulationSpeed - defines the speed of simulation an is used to
calculate the sleep time of the thread.
```

setStartTime

```
public void setStartTime(int startTime)
```

sets teh startTime

Parameters:

```
startTime - the time, at which the simulation starts
```

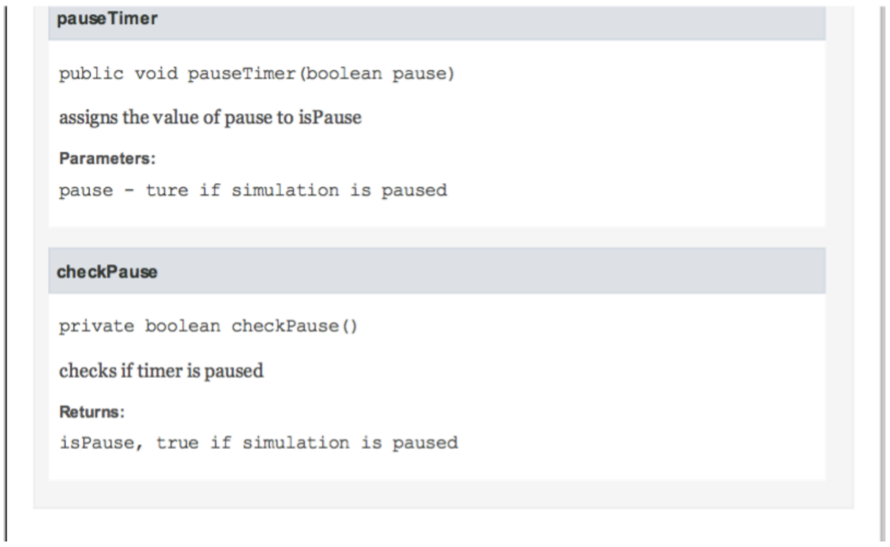
setEndTime

```
public void setEndTime(int endTime)
```

sets the endTime

Parameters:

```
endTime - the time, at which the simulation stops
```



pauseTimer

```
public void pauseTimer(boolean pause)
```

assigns the value of pause to isPause

Parameters:

```
pause - ture if simulation is paused
```

checkPause

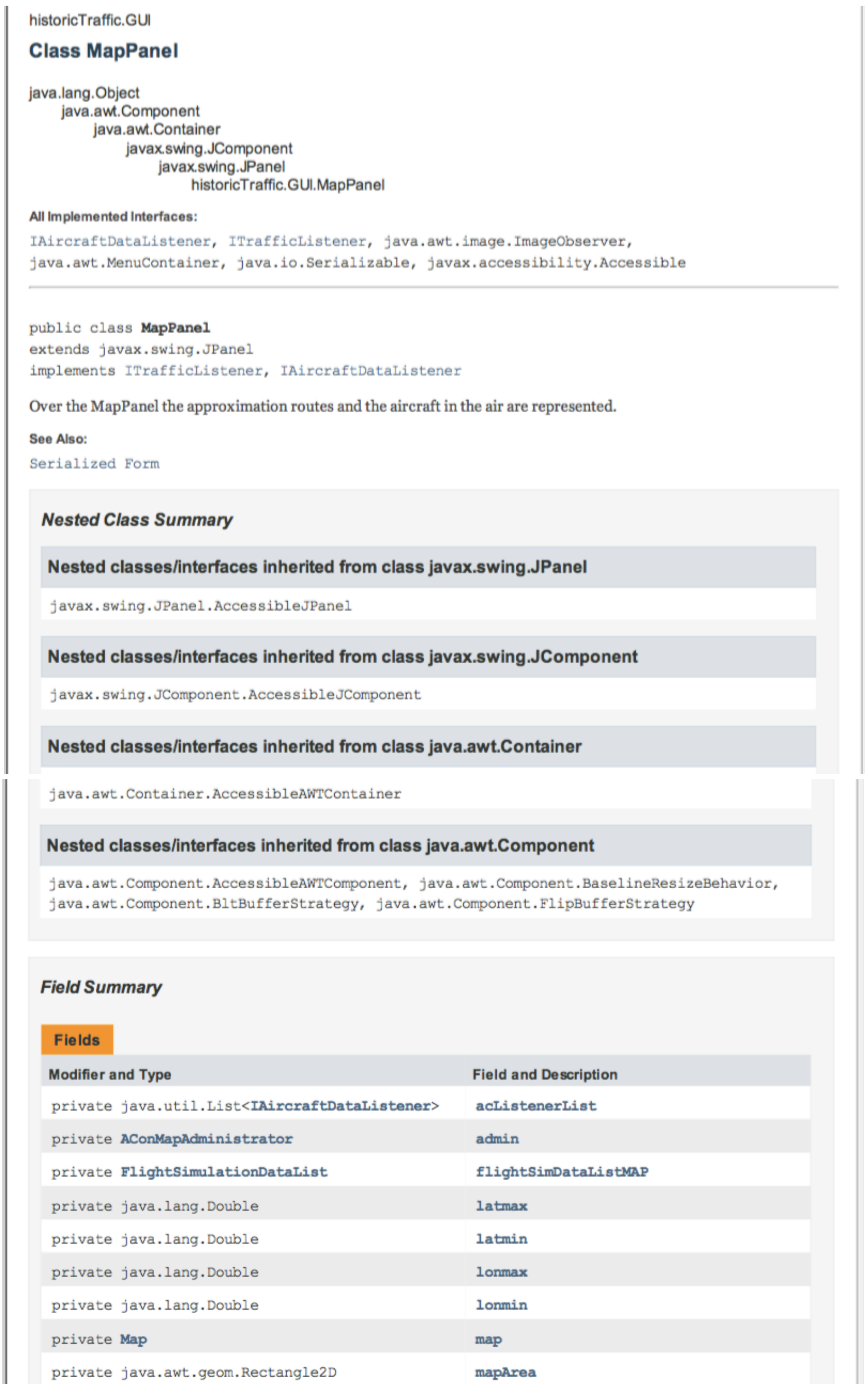
```
private boolean checkPause()
```

checks if timer is paused

Returns:

```
isPause, true if simulation is paused
```

19.3. MapPanel



historicTraffic.GUI

Class MapPanel

```
java.lang.Object
    java.awt.Component
        java.awt.Container
            javax.swing.JComponent
                javax.swing.JPanel
                    historicTraffic.GUI.MapPanel
```

All Implemented Interfaces:

`IAircraftDataListener, ITrafficListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible`

```
public class MapPanel
extends javax.swing.JPanel
implements ITrafficListener, IAircraftDataListener
```

Over the MapPanel the approximation routes and the aircraft in the air are represented.

See Also:

`Serialized Form`

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JPanel

`javax.swing.JPanel.AccessibleJPanel`

Nested classes/interfaces inherited from class javax.swing.JComponent

`javax.swing.JComponent.AccessibleJComponent`

Nested classes/interfaces inherited from class java.awt.Container

`java.awt.Container.AccessibleAWTContainer`

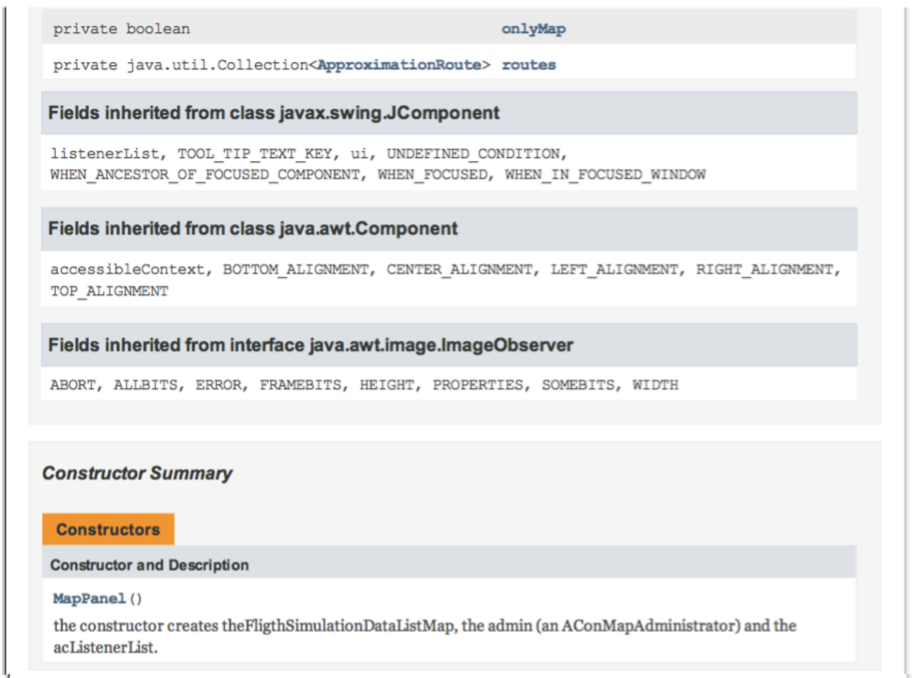
Nested classes/interfaces inherited from class java.awt.Component

`java.awt.Component.AccessibleAWTComponent, java.awt.Component.BaselineResizeBehavior, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy`

Field Summary

Fields

Modifier and Type	Field and Description
`private java.util.List<IAircraftDataListener>`	**acListenerList**
`private AConMapAdministrator`	**admin**
`private FlightSimulationDataList`	**flightSimDataListMAP**
`private java.lang.Double`	**latmax**
`private java.lang.Double`	**latmin**
`private java.lang.Double`	**lonmax**
`private java.lang.Double`	**lonmin**
`private Map`	**map**
`private java.awt.geom.Rectangle2D`	**mapArea**



`private boolean`	**onlyMap**
`private java.util.Collection<`**ApproximationRoute**`>`	**routes**

Fields inherited from class javax.swing.JComponent

`listenerList, TOOL_TIP_TEXT_KEY, ui, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW`

Fields inherited from class java.awt.Component

`accessibleContext, BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT`

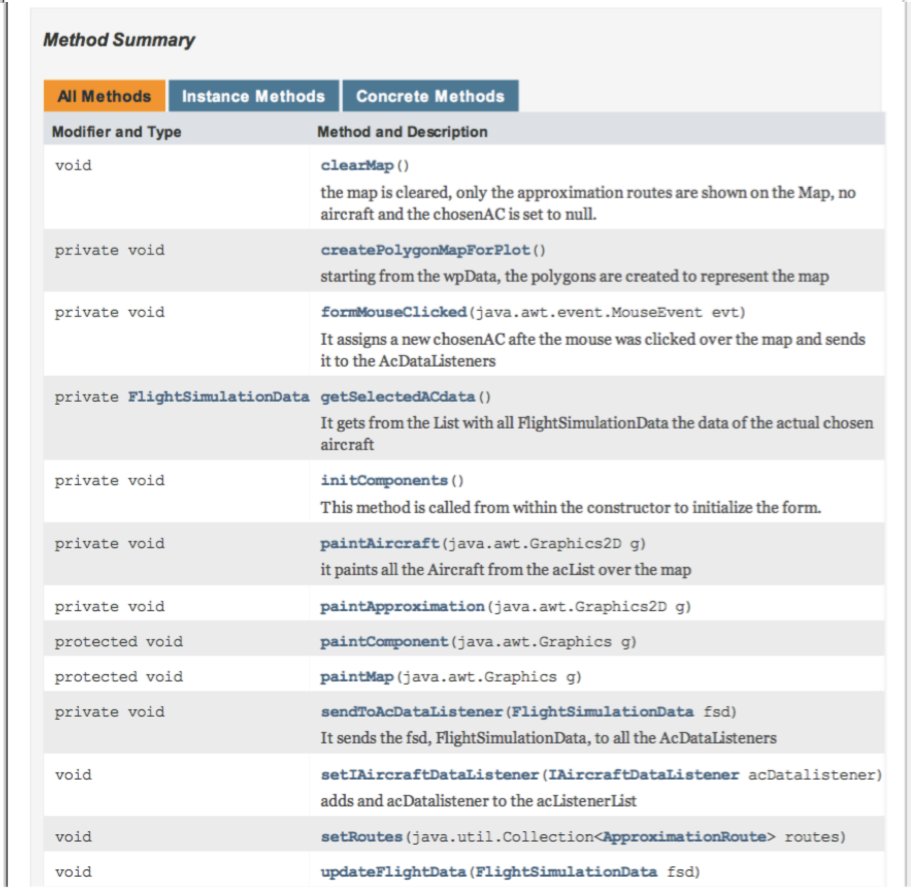
Fields inherited from interface java.awt.image.ImageObserver

`ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH`

Constructor Summary

Constructors

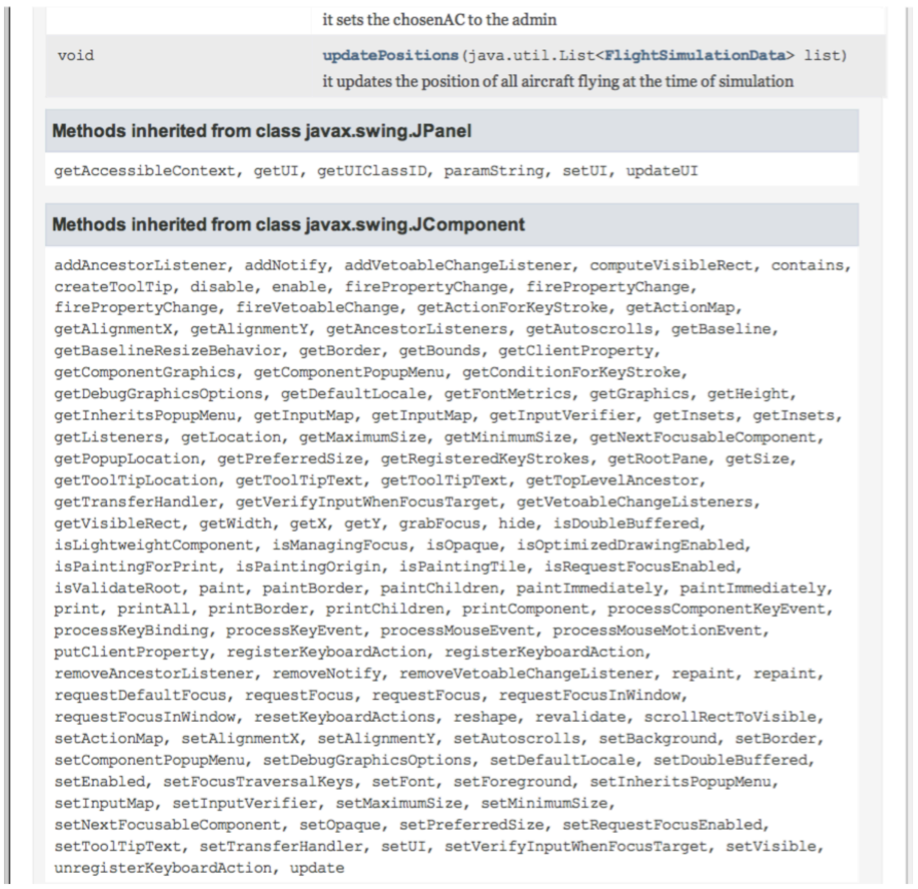
Constructor and Description
MapPanel`()` the constructor creates theFligthSimulationDataListMap, the admin (an AConMapAdministrator) and the acListenerList.



Method Summary

All Methods **Instance Methods** **Concrete Methods**

Modifier and Type	Method and Description
`void`	**clearMap**`()` the map is cleared, only the approximation routes are shown on the Map, no aircraft and the chosenAC is set to null.
`private void`	**createPolygonMapForPlot**`()` starting from the wpData, the polygons are created to represent the map
`private void`	**formMouseClicked**`(java.awt.event.MouseEvent evt)` It assigns a new chosenAC afte the mouse was clicked over the map and sends it to the AcDataListeners
`private` **FlightSimulationData**	**getSelectedACdata**`()` It gets from the List with all FlightSimulationData the data of the actual chosen aircraft
`private void`	**initComponents**`()` This method is called from within the constructor to initialize the form.
`private void`	**paintAircraft**`(java.awt.Graphics2D g)` it paints all the Aircraft from the acList over the map
`private void`	**paintApproximation**`(java.awt.Graphics2D g)`
`protected void`	**paintComponent**`(java.awt.Graphics g)`
`protected void`	**paintMap**`(java.awt.Graphics g)`
`private void`	**sendToAcDataListener**`(`**FlightSimulationData**` fsd)` It sends the fsd, FlightSimulationData, to all the AcDataListeners
`void`	**setIAircraftDataListener**`(`**IAircraftDataListener**` acDatalistener)` adds and acDatalistener to the acListenerList
`void`	**setRoutes**`(java.util.Collection<`**ApproximationRoute**`> routes)`
`void`	**updateFlightData**`(`**FlightSimulationData**` fsd)`



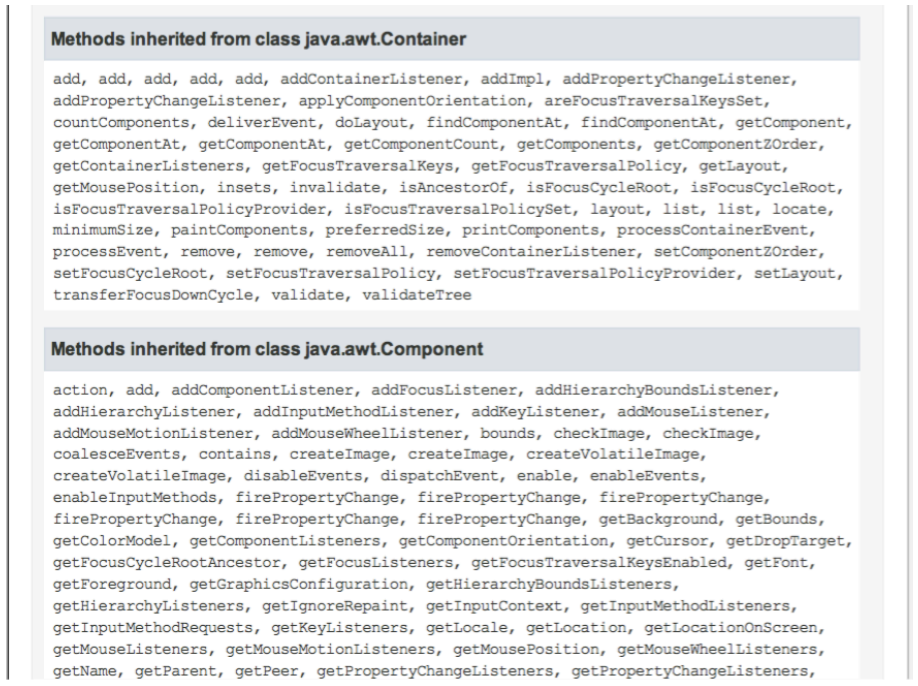
	it sets the chosenAC to the admin
void	**updatePositions**(java.util.List<**FlightSimulationData**> list) it updates the position of all aircraft flying at the time of simulation

Methods inherited from class javax.swing.JPanel

```
getAccessibleContext, getUI, getUIClassID, paramString, setUI, updateUI
```

Methods inherited from class javax.swing.JComponent

```
addAncestorListener, addNotify, addVetoableChangeListener, computeVisibleRect, contains,
createToolTip, disable, enable, firePropertyChange, firePropertyChange,
firePropertyChange, fireVetoableChange, getActionForKeyStroke, getActionMap,
getAlignmentX, getAlignmentY, getAncestorListeners, getAutoscrolls, getBaseline,
getBaselineResizeBehavior, getBorder, getBounds, getClientProperty,
getComponentGraphics, getComponentPopupMenu, getConditionForKeyStroke,
getDebugGraphicsOptions, getDefaultLocale, getFontMetrics, getGraphics, getHeight,
getInheritsPopupMenu, getInputMap, getInputMap, getInputVerifier, getInsets, getInsets,
getListeners, getLocation, getMaximumSize, getMinimumSize, getNextFocusableComponent,
getPopupLocation, getPreferredSize, getRegisteredKeyStrokes, getRootPane, getSize,
getToolTipLocation, getToolTipText, getToolTipText, getTopLevelAncestor,
getTransferHandler, getVerifyInputWhenFocusTarget, getVetoableChangeListeners,
getVisibleRect, getWidth, getX, getY, grabFocus, hide, isDoubleBuffered,
isLightweightComponent, isManagingFocus, isOpaque, isOptimizedDrawingEnabled,
isPaintingForPrint, isPaintingOrigin, isPaintingTile, isRequestFocusEnabled,
isValidateRoot, paint, paintBorder, paintChildren, paintImmediately, paintImmediately,
print, printAll, printBorder, printChildren, printComponent, processComponentKeyEvent,
processKeyBinding, processKeyEvent, processMouseEvent, processMouseMotionEvent,
putClientProperty, registerKeyboardAction, registerKeyboardAction,
removeAncestorListener, removeNotify, removeVetoableChangeListener, repaint, repaint,
requestDefaultFocus, requestFocus, requestFocus, requestFocusInWindow,
requestFocusInWindow, resetKeyboardActions, reshape, revalidate, scrollRectToVisible,
setActionMap, setAlignmentX, setAlignmentY, setAutoscrolls, setBackground, setBorder,
setComponentPopupMenu, setDebugGraphicsOptions, setDefaultLocale, setDoubleBuffered,
setEnabled, setFocusTraversalKeys, setFont, setForeground, setInheritsPopupMenu,
setInputMap, setInputVerifier, setMaximumSize, setMinimumSize,
setNextFocusableComponent, setOpaque, setPreferredSize, setRequestFocusEnabled,
setToolTipText, setTransferHandler, setUI, setVerifyInputWhenFocusTarget, setVisible,
unregisterKeyboardAction, update
```

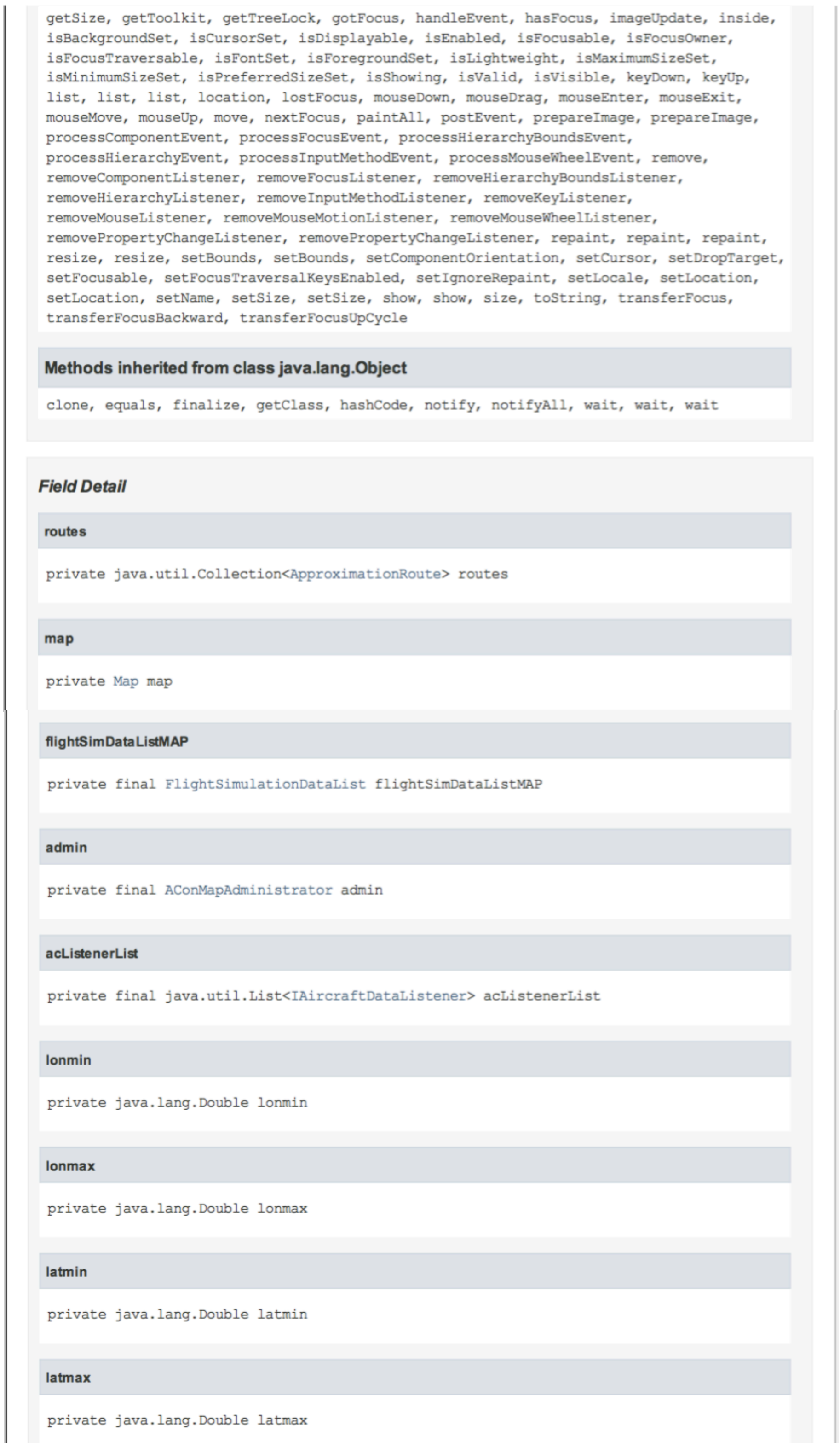


Methods inherited from class java.awt.Container

```
add, add, add, add, add, addContainerListener, addImpl, addPropertyChangeListener,
addPropertyChangeListener, applyComponentOrientation, areFocusTraversalKeysSet,
countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getComponent,
getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder,
getContainerListeners, getFocusTraversalKeys, getFocusTraversalPolicy, getLayout,
getMousePosition, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusCycleRoot,
isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate,
minimumSize, paintComponents, preferredSize, printComponents, processContainerEvent,
processEvent, remove, remove, removeAll, removeContainerListener, setComponentZOrder,
setFocusCycleRoot, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setLayout,
transferFocusDownCycle, validate, validateTree
```

Methods inherited from class java.awt.Component

```
action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener,
addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener,
addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage,
coalesceEvents, contains, createImage, createImage, createVolatileImage,
createVolatileImage, disableEvents, dispatchEvent, enable, enableEvents,
enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange,
firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds,
getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget,
getFocusCycleRootAncestor, getFocusListeners, getFocusTraversalKeysEnabled, getFont,
getForeground, getGraphicsConfiguration, getHierarchyBoundsListeners,
getHierarchyListeners, getIgnoreRepaint, getInputContext, getInputMethodListeners,
getInputMethodRequests, getKeyListeners, getLocale, getLocation, getLocationOnScreen,
getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners,
getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners,
```



```
getSize, getToolkit, getTreeLock, gotFocus, handleEvent, hasFocus, imageUpdate, inside,
isBackgroundSet, isCursorSet, isDisplayable, isEnabled, isFocusable, isFocusOwner,
isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet,
isMinimumSizeSet, isPreferredSizeSet, isShowing, isValid, isVisible, keyDown, keyUp,
list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit,
mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage,
processComponentEvent, processFocusEvent, processHierarchyBoundsEvent,
processHierarchyEvent, processInputMethodEvent, processMouseWheelEvent, remove,
removeComponentListener, removeFocusListener, removeHierarchyBoundsListener,
removeHierarchyListener, removeInputMethodListener, removeKeyListener,
removeMouseListener, removeMouseMotionListener, removeMouseWheelListener,
removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint,
resize, resize, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget,
setFocusable, setFocusTraversalKeysEnabled, setIgnoreRepaint, setLocale, setLocation,
setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus,
transferFocusBackward, transferFocusUpCycle
```

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait
```

Field Detail

routes

```
private java.util.Collection<ApproximationRoute> routes
```

map

```
private Map map
```

flightSimDataListMAP

```
private final FlightSimulationDataList flightSimDataListMAP
```

admin

```
private final AConMapAdministrator admin
```

acListenerList

```
private final java.util.List<IAircraftDataListener> acListenerList
```

lonmin

```
private java.lang.Double lonmin
```

lonmax

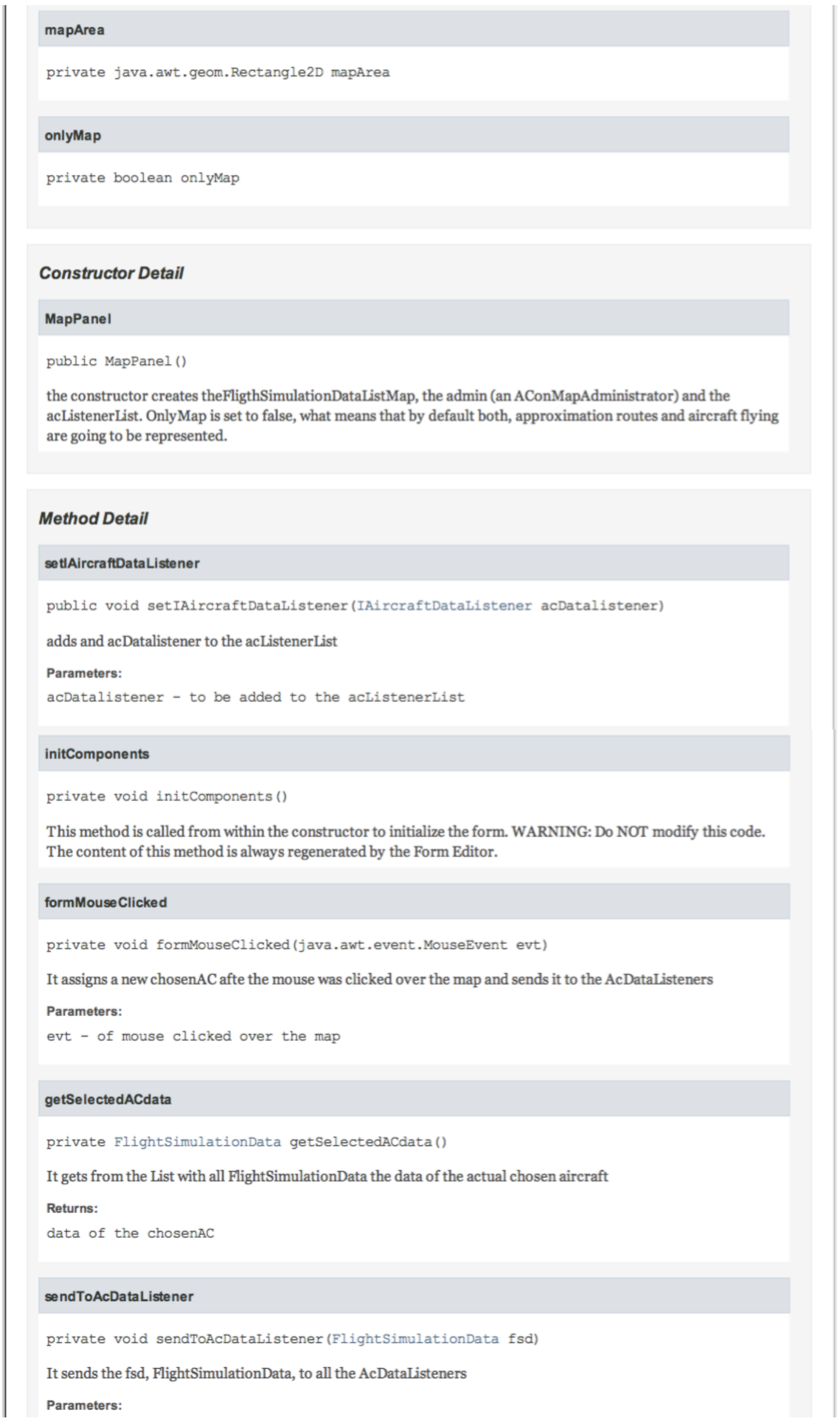
```
private java.lang.Double lonmax
```

latmin

```
private java.lang.Double latmin
```

latmax

```
private java.lang.Double latmax
```



mapArea

```
private java.awt.geom.Rectangle2D mapArea
```

onlyMap

```
private boolean onlyMap
```

Constructor Detail

MapPanel

```
public MapPanel()
```

the constructor creates theFligthSimulationDataListMap, the admin (an AConMapAdministrator) and the acListenerList. OnlyMap is set to false, what means that by default both, approximation routes and aircraft flying are going to be represented.

Method Detail

setIAircraftDataListener

```
public void setIAircraftDataListener(IAircraftDataListener acDatalistener)
```

adds and acDatalistener to the acListenerList

Parameters:
`acDatalistener - to be added to the acListenerList`

initComponents

```
private void initComponents()
```

This method is called from within the constructor to initialize the form. WARNING: Do NOT modify this code. The content of this method is always regenerated by the Form Editor.

formMouseClicked

```
private void formMouseClicked(java.awt.event.MouseEvent evt)
```

It assigns a new chosenAC afte the mouse was clicked over the map and sends it to the AcDataListeners

Parameters:
`evt - of mouse clicked over the map`

getSelectedACdata

```
private FlightSimulationData getSelectedACdata()
```

It gets from the List with all FlightSimulationData the data of the actual chosen aircraft

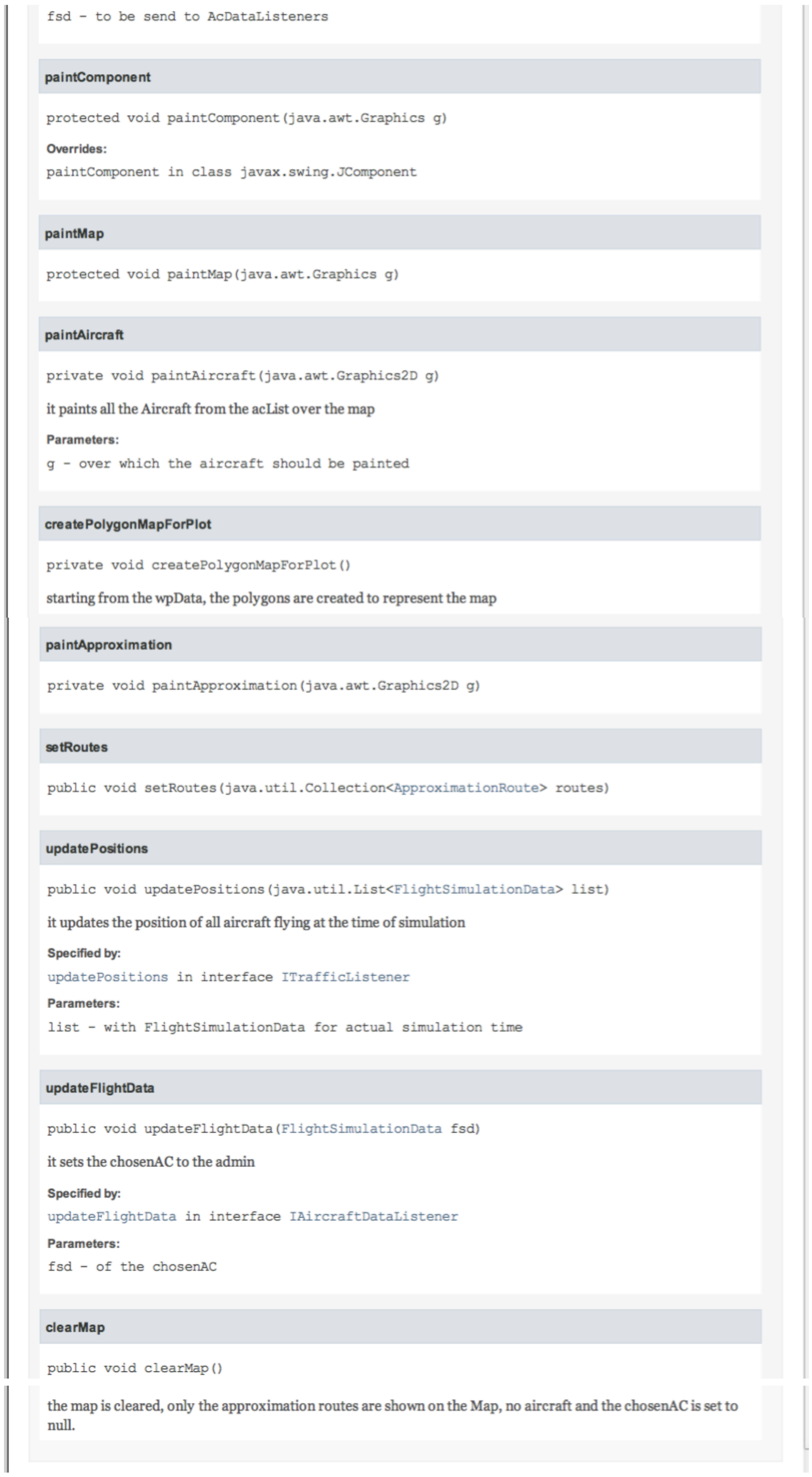
Returns:
`data of the chosenAC`

sendToAcDataListener

```
private void sendToAcDataListener(FlightSimulationData fsd)
```

It sends the fsd, FlightSimulationData, to all the AcDataListeners

Parameters:



`fsd - to be send to AcDataListeners`

paintComponent

```
protected void paintComponent(java.awt.Graphics g)
```

Overrides:

`paintComponent in class javax.swing.JComponent`

paintMap

```
protected void paintMap(java.awt.Graphics g)
```

paintAircraft

```
private void paintAircraft(java.awt.Graphics2D g)
```

it paints all the Aircraft from the acList over the map

Parameters:

`g - over which the aircraft should be painted`

createPolygonMapForPlot

```
private void createPolygonMapForPlot()
```

starting from the wpData, the polygons are created to represent the map

paintApproximation

```
private void paintApproximation(java.awt.Graphics2D g)
```

setRoutes

```
public void setRoutes(java.util.Collection<ApproximationRoute> routes)
```

updatePositions

```
public void updatePositions(java.util.List<FlightSimulationData> list)
```

it updates the position of all aircraft flying at the time of simulation

Specified by:

`updatePositions in interface ITrafficListener`

Parameters:

`list - with FlightSimulationData for actual simulation time`

updateFlightData

```
public void updateFlightData(FlightSimulationData fsd)
```

it sets the chosenAC to the admin

Specified by:

`updateFlightData in interface IAircraftDataListener`

Parameters:

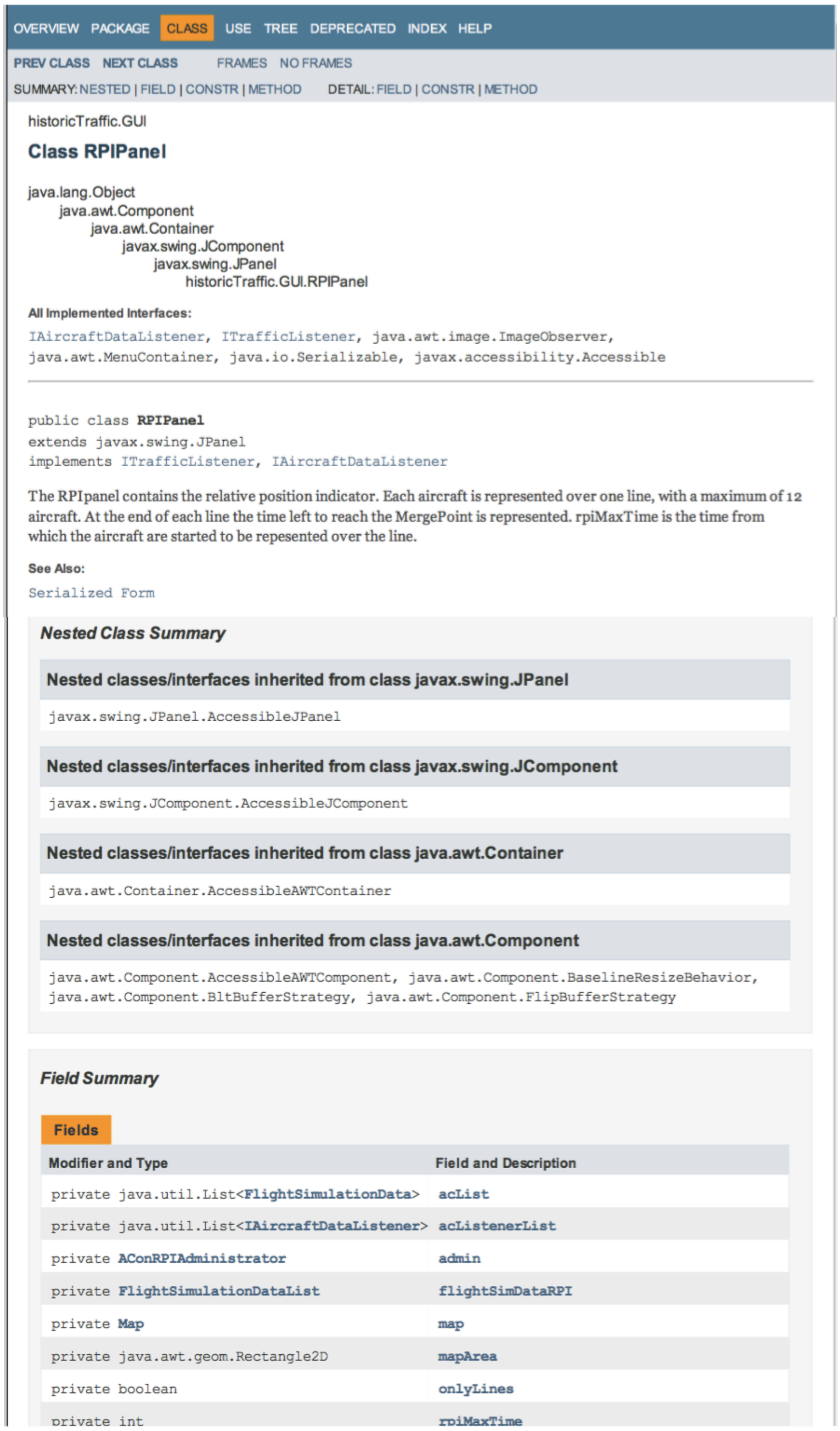
`fsd - of the chosenAC`

clearMap

```
public void clearMap()
```

the map is cleared, only the approximation routes are shown on the Map, no aircraft and the chosenAC is set to null.

19.4. RPIPanel


OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

historicTraffic.GUI

Class RPIPanel

```
java.lang.Object
    java.awt.Component
        java.awt.Container
            javax.swing.JComponent
                javax.swing.JPanel
                    historicTraffic.GUI.RPIPanel
```

All Implemented Interfaces:

`IAircraftDataListener, ITrafficListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible`

```
public class RPIPanel
extends javax.swing.JPanel
implements ITrafficListener, IAircraftDataListener
```

The RPIpanel contains the relative position indicator. Each aircraft is represented over one line, with a maximum of 12 aircraft. At the end of each line the time left to reach the MergePoint is represented. rpiMaxTime is the time from which the aircraft are started to be repesented over the line.

See Also:

`Serialized Form`

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JPanel

`javax.swing.JPanel.AccessibleJPanel`

Nested classes/interfaces inherited from class javax.swing.JComponent

`javax.swing.JComponent.AccessibleJComponent`

Nested classes/interfaces inherited from class java.awt.Container

`java.awt.Container.AccessibleAWTContainer`

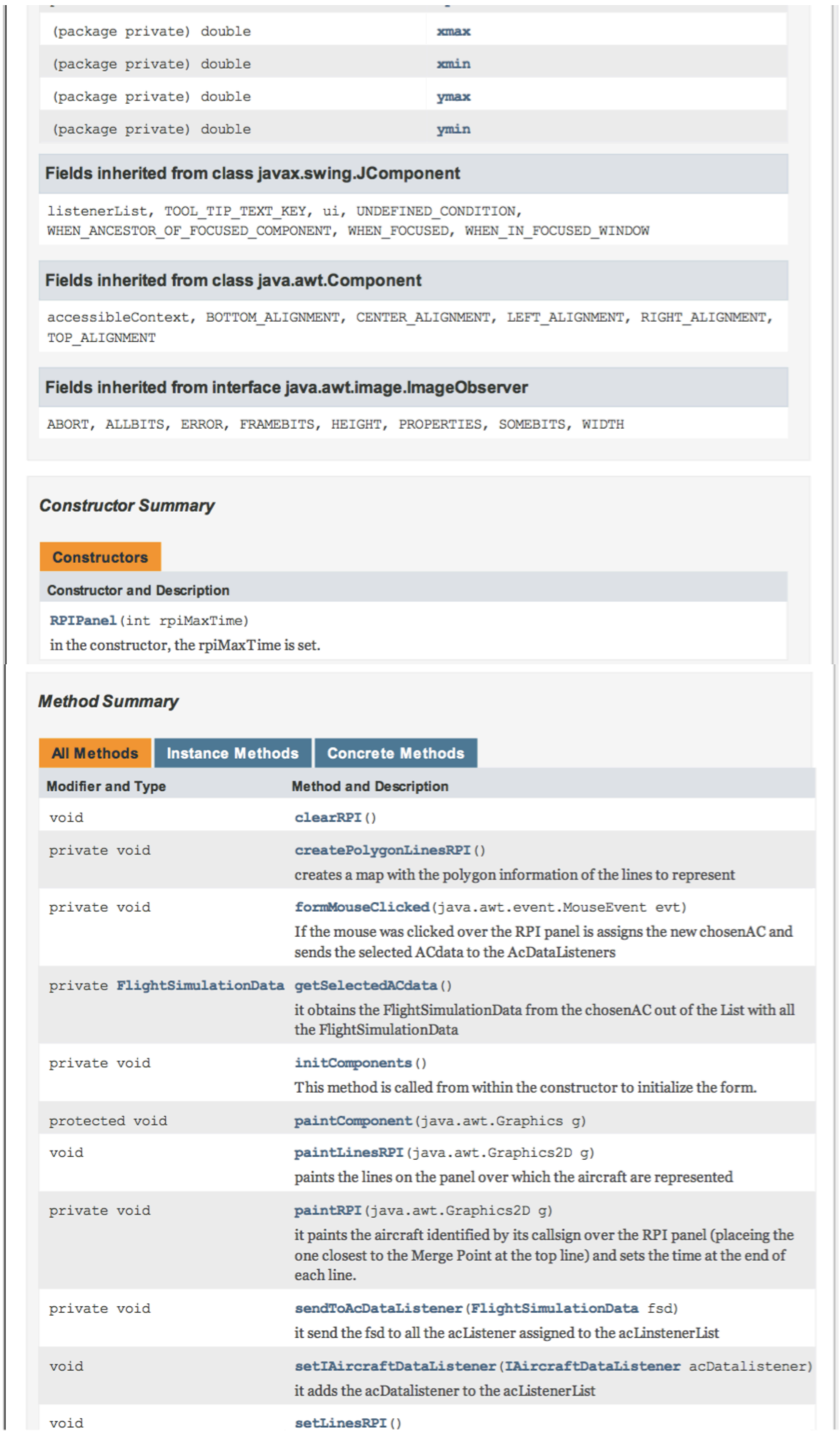
Nested classes/interfaces inherited from class java.awt.Component

`java.awt.Component.AccessibleAWTComponent, java.awt.Component.BaselineResizeBehavior, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy`

Field Summary

Fields

Modifier and Type	Field and Description
private java.util.List<**FlightSimulationData**>	**acList**
private java.util.List<**IAircraftDataListener**>	**acListenerList**
private **AConRPIAdministrator**	**admin**
private **FlightSimulationDataList**	**flightSimDataRPI**
private **Map**	**map**
private java.awt.geom.Rectangle2D	**mapArea**
private boolean	**onlyLines**
private int	**rpiMaxTime**



(package private) double	xmax
(package private) double	xmin
(package private) double	ymax
(package private) double	ymin

Fields inherited from class javax.swing.JComponent

listenerList, TOOL_TIP_TEXT_KEY, ui, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW

Fields inherited from class java.awt.Component

accessibleContext, BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

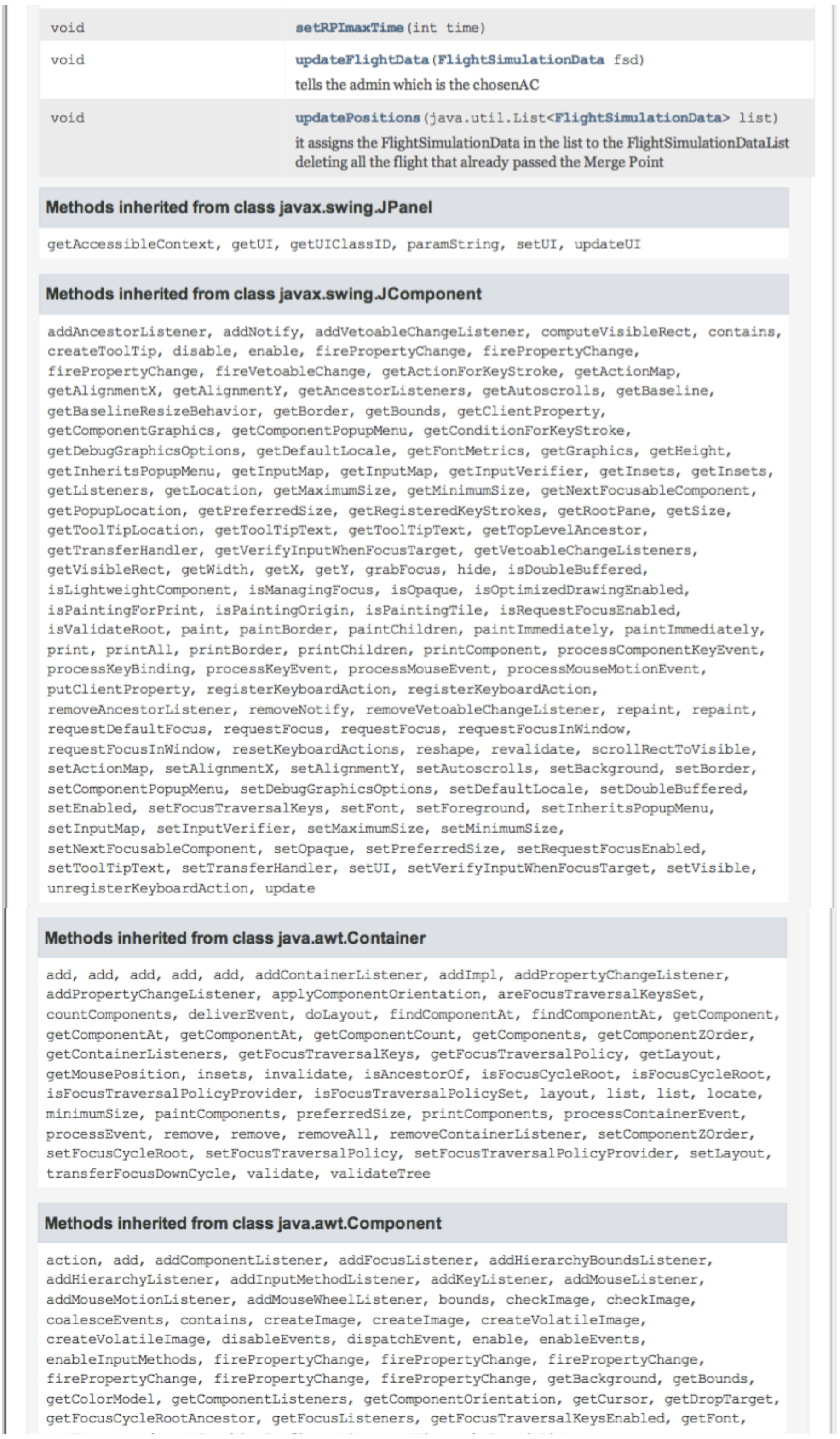
Constructors

Constructor and Description
RPIPanel(int rpiMaxTime) in the constructor, the rpiMaxTime is set.

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
void	clearRPI()
private void	createPolygonLinesRPI() creates a map with the polygon information of the lines to represent
private void	formMouseClicked(java.awt.event.MouseEvent evt) If the mouse was clicked over the RPI panel is assigns the new chosenAC and sends the selected ACdata to the AcDataListeners
private FlightSimulationData	getSelectedACdata() it obtains the FlightSimulationData from the chosenAC out of the List with all the FlightSimulationData
private void	initComponents() This method is called from within the constructor to initialize the form.
protected void	paintComponent(java.awt.Graphics g)
void	paintLinesRPI(java.awt.Graphics2D g) paints the lines on the panel over which the aircraft are represented
private void	paintRPI(java.awt.Graphics2D g) it paints the aircraft identified by its callsign over the RPI panel (placeing the one closest to the Merge Point at the top line) and sets the time at the end of each line.
private void	sendToAcDataListener(FlightSimulationData fsd) it send the fsd to all the acListener assigned to the acLinstenerList
void	setIAircraftDataListener(IAircraftDataListener acDatalistener) it adds the acDatalistener to the acListenerList
void	setLinesRPI()



void	**setRPImaxTime**(int time)
void	**updateFlightData**(**FlightSimulationData** fsd) tells the admin which is the chosenAC
void	**updatePositions**(java.util.List<**FlightSimulationData**> list) it assigns the FlightSimulationData in the list to the FlightSimulationDataList deleting all the flight that already passed the Merge Point

Methods inherited from class javax.swing.JPanel

```
getAccessibleContext, getUI, getUIClassID, paramString, setUI, updateUI
```

Methods inherited from class javax.swing.JComponent

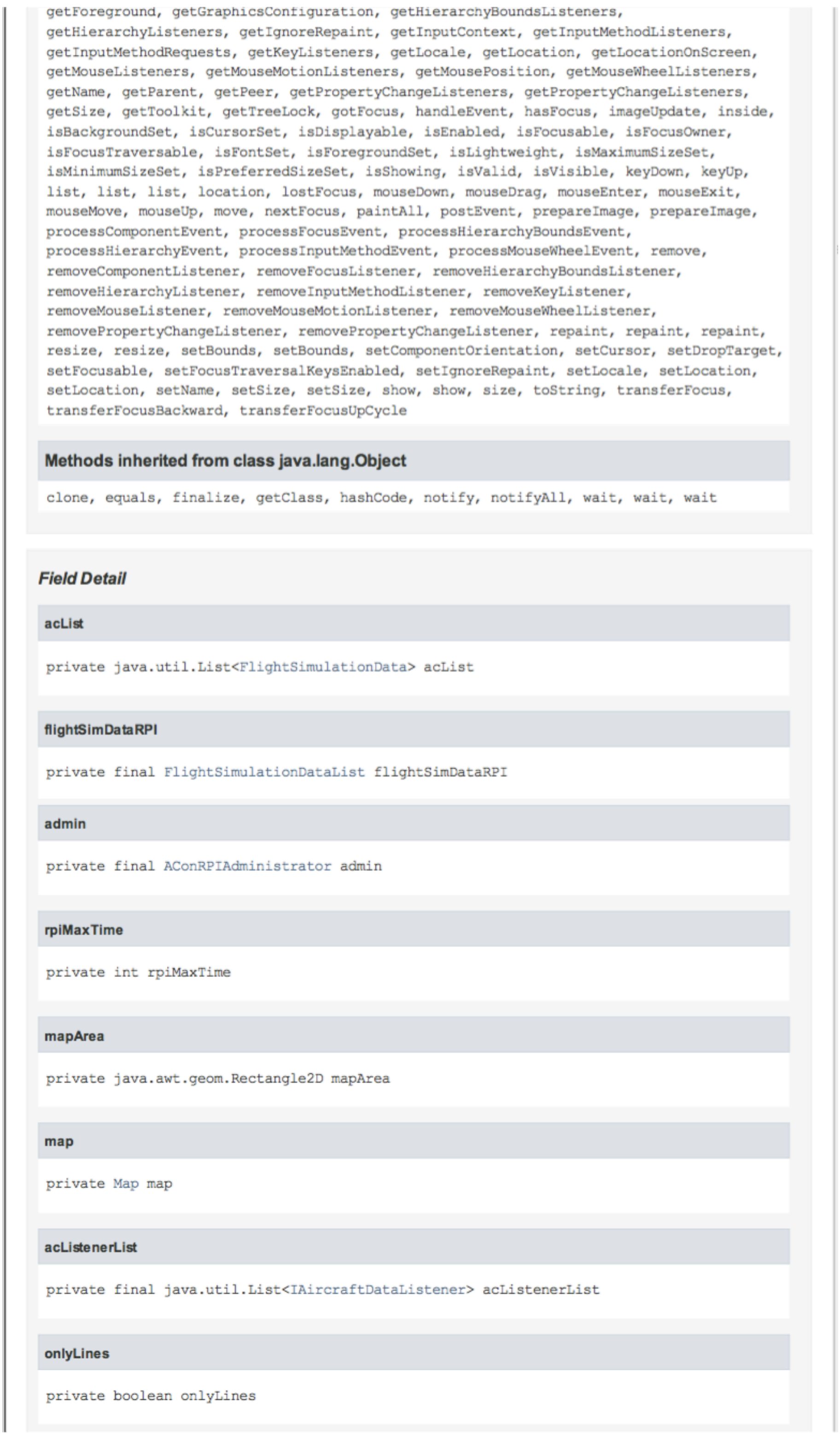
```
addAncestorListener, addNotify, addVetoableChangeListener, computeVisibleRect, contains,
createToolTip, disable, enable, firePropertyChange, firePropertyChange,
firePropertyChange, fireVetoableChange, getActionForKeyStroke, getActionMap,
getAlignmentX, getAlignmentY, getAncestorListeners, getAutoscrolls, getBaseline,
getBaselineResizeBehavior, getBorder, getBounds, getClientProperty,
getComponentGraphics, getComponentPopupMenu, getConditionForKeyStroke,
getDebugGraphicsOptions, getDefaultLocale, getFontMetrics, getGraphics, getHeight,
getInheritsPopupMenu, getInputMap, getInputMap, getInputVerifier, getInsets, getInsets,
getListeners, getLocation, getMaximumSize, getMinimumSize, getNextFocusableComponent,
getPopupLocation, getPreferredSize, getRegisteredKeyStrokes, getRootPane, getSize,
getToolTipLocation, getToolTipText, getToolTipText, getTopLevelAncestor,
getTransferHandler, getVerifyInputWhenFocusTarget, getVetoableChangeListeners,
getVisibleRect, getWidth, getX, getY, grabFocus, hide, isDoubleBuffered,
isLightweightComponent, isManagingFocus, isOpaque, isOptimizedDrawingEnabled,
isPaintingForPrint, isPaintingOrigin, isPaintingTile, isRequestFocusEnabled,
isValidateRoot, paint, paintBorder, paintChildren, paintImmediately, paintImmediately,
print, printAll, printBorder, printChildren, printComponent, processComponentKeyEvent,
processKeyBinding, processKeyEvent, processMouseEvent, processMouseMotionEvent,
putClientProperty, registerKeyboardAction, registerKeyboardAction,
removeAncestorListener, removeNotify, removeVetoableChangeListener, repaint, repaint,
requestDefaultFocus, requestFocus, requestFocus, requestFocusInWindow,
requestFocusInWindow, resetKeyboardActions, reshape, revalidate, scrollRectToVisible,
setActionMap, setAlignmentX, setAlignmentY, setAutoscrolls, setBackground, setBorder,
setComponentPopupMenu, setDebugGraphicsOptions, setDefaultLocale, setDoubleBuffered,
setEnabled, setFocusTraversalKeys, setFont, setForeground, setInheritsPopupMenu,
setInputMap, setInputVerifier, setMaximumSize, setMinimumSize,
setNextFocusableComponent, setOpaque, setPreferredSize, setRequestFocusEnabled,
setToolTipText, setTransferHandler, setUI, setVerifyInputWhenFocusTarget, setVisible,
unregisterKeyboardAction, update
```

Methods inherited from class java.awt.Container

```
add, add, add, add, add, addContainerListener, addImpl, addPropertyChangeListener,
addPropertyChangeListener, applyComponentOrientation, areFocusTraversalKeysSet,
countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getComponent,
getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder,
getContainerListeners, getFocusTraversalKeys, getFocusTraversalPolicy, getLayout,
getMousePosition, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusCycleRoot,
isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate,
minimumSize, paintComponents, preferredSize, printComponents, processContainerEvent,
processEvent, remove, remove, removeAll, removeContainerListener, setComponentZOrder,
setFocusCycleRoot, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setLayout,
transferFocusDownCycle, validate, validateTree
```

Methods inherited from class java.awt.Component

```
action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener,
addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener,
addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage,
coalesceEvents, contains, createImage, createImage, createVolatileImage,
createVolatileImage, disableEvents, dispatchEvent, enable, enableEvents,
enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange,
firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds,
getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget,
getFocusCycleRootAncestor, getFocusListeners, getFocusTraversalKeysEnabled, getFont,
```



```
getForeground, getGraphicsConfiguration, getHierarchyBoundsListeners,
getHierarchyListeners, getIgnoreRepaint, getInputContext, getInputMethodListeners,
getInputMethodRequests, getKeyListeners, getLocale, getLocation, getLocationOnScreen,
getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners,
getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners,
getSize, getToolkit, getTreeLock, gotFocus, handleEvent, hasFocus, imageUpdate, inside,
isBackgroundSet, isCursorSet, isDisplayable, isEnabled, isFocusable, isFocusOwner,
isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet,
isMinimumSizeSet, isPreferredSizeSet, isShowing, isValid, isVisible, keyDown, keyUp,
list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit,
mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage,
processComponentEvent, processFocusEvent, processHierarchyBoundsEvent,
processHierarchyEvent, processInputMethodEvent, processMouseWheelEvent, remove,
removeComponentListener, removeFocusListener, removeHierarchyBoundsListener,
removeHierarchyListener, removeInputMethodListener, removeKeyListener,
removeMouseListener, removeMouseMotionListener, removeMouseWheelListener,
removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint,
resize, resize, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget,
setFocusable, setFocusTraversalKeysEnabled, setIgnoreRepaint, setLocale, setLocation,
setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus,
transferFocusBackward, transferFocusUpCycle
```

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait
```

Field Detail

acList

```
private java.util.List<FlightSimulationData> acList
```

flightSimDataRPI

```
private final FlightSimulationDataList flightSimDataRPI
```

admin

```
private final AConRPIAdministrator admin
```

rpiMaxTime

```
private int rpiMaxTime
```

mapArea

```
private java.awt.geom.Rectangle2D mapArea
```

map

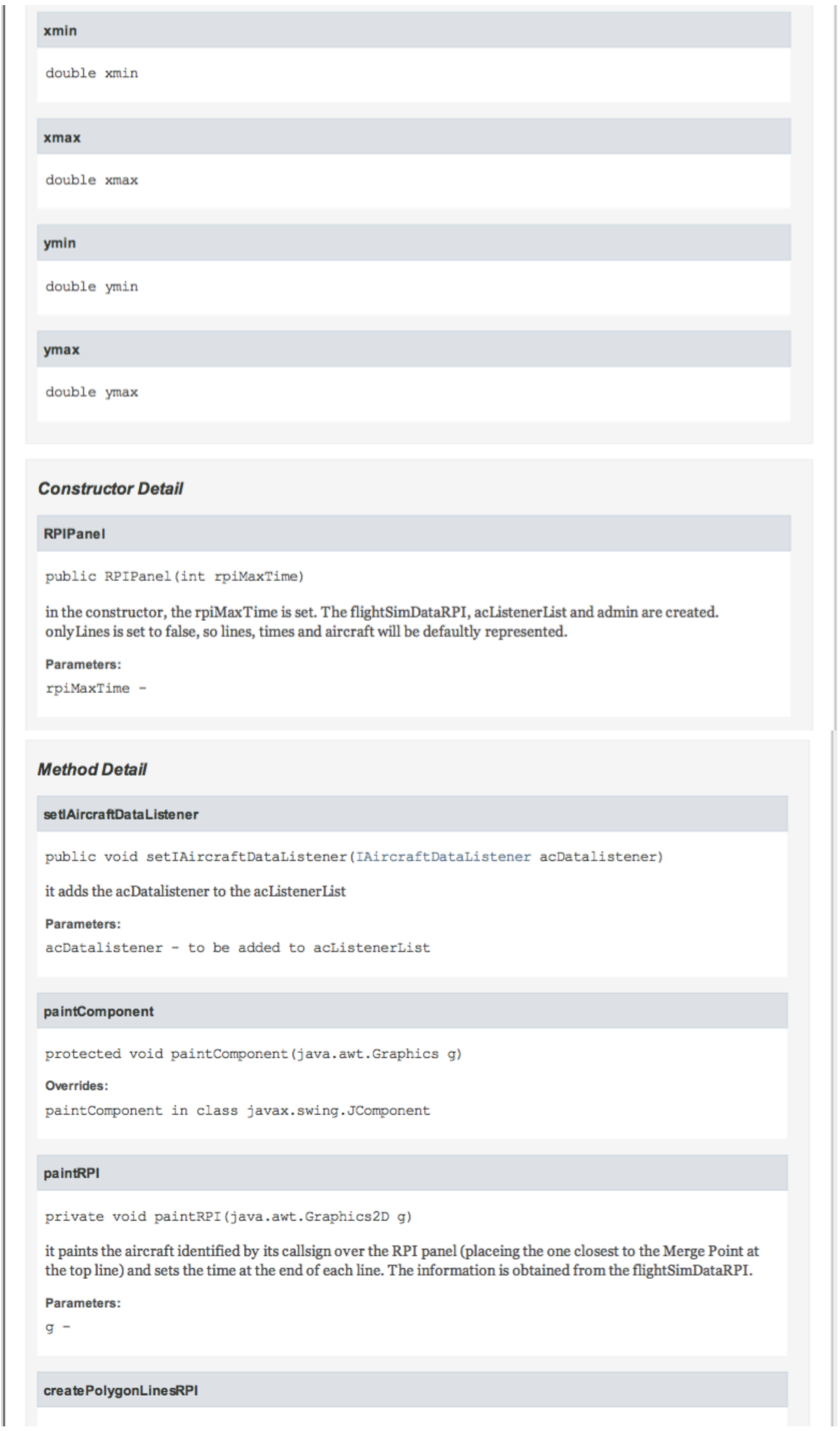
```
private Map map
```

acListenerList

```
private final java.util.List<IAircraftDataListener> acListenerList
```

onlyLines

```
private boolean onlyLines
```



xmin

`double xmin`

xmax

`double xmax`

ymin

`double ymin`

ymax

`double ymax`

Constructor Detail

RPIPanel

`public RPIPanel(int rpiMaxTime)`

in the constructor, the rpiMaxTime is set. The flightSimDataRPI, acListenerList and admin are created. onlyLines is set to false, so lines, times and aircraft will be defaultly represented.

Parameters:
`rpiMaxTime -`

Method Detail

setIAircraftDataListener

`public void setIAircraftDataListener(IAircraftDataListener acDatalistener)`

it adds the acDatalistener to the acListenerList

Parameters:
`acDatalistener - to be added to acListenerList`

paintComponent

`protected void paintComponent(java.awt.Graphics g)`

Overrides:
`paintComponent in class javax.swing.JComponent`

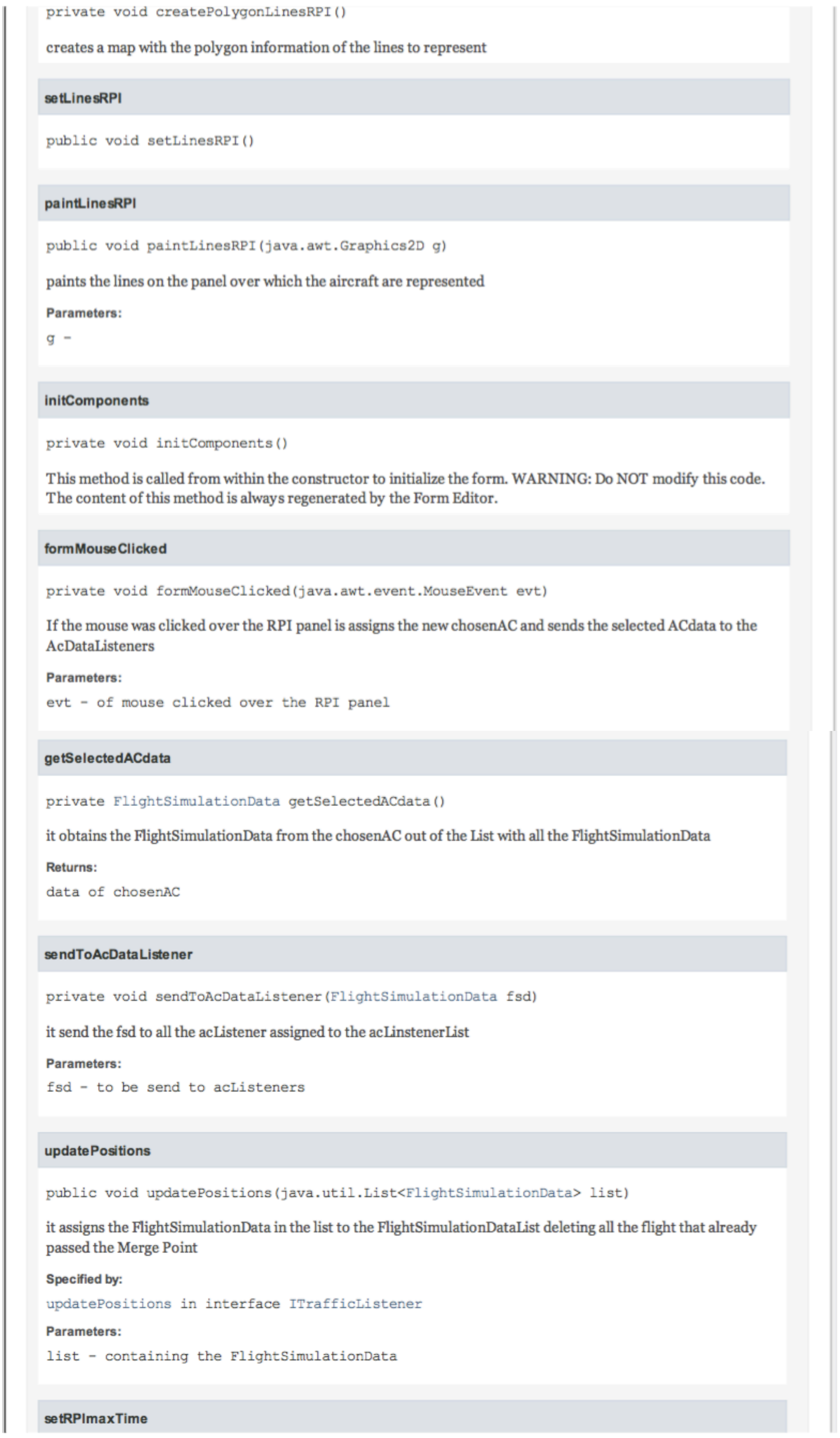
paintRPI

`private void paintRPI(java.awt.Graphics2D g)`

it paints the aircraft identified by its callsign over the RPI panel (placeing the one closest to the Merge Point at the top line) and sets the time at the end of each line. The information is obtained from the flightSimDataRPI.

Parameters:
`g -`

createPolygonLinesRPI



```
private void createPolygonLinesRPI()
```

creates a map with the polygon information of the lines to represent

setLinesRPI

```
public void setLinesRPI()
```

paintLinesRPI

```
public void paintLinesRPI(java.awt.Graphics2D g)
```

paints the lines on the panel over which the aircraft are represented

Parameters:

`g -`

initComponents

```
private void initComponents()
```

This method is called from within the constructor to initialize the form. WARNING: Do NOT modify this code. The content of this method is always regenerated by the Form Editor.

formMouseClicked

```
private void formMouseClicked(java.awt.event.MouseEvent evt)
```

If the mouse was clicked over the RPI panel is assigns the new chosenAC and sends the selected ACdata to the AcDataListeners

Parameters:

`evt - of mouse clicked over the RPI panel`

getSelectedACdata

```
private FlightSimulationData getSelectedACdata()
```

it obtains the FlightSimulationData from the chosenAC out of the List with all the FlightSimulationData

Returns:

`data of chosenAC`

sendToAcDataListener

```
private void sendToAcDataListener(FlightSimulationData fsd)
```

it send the fsd to all the acListener assigned to the acLinstenerList

Parameters:

`fsd - to be send to acListeners`

updatePositions

```
public void updatePositions(java.util.List<FlightSimulationData> list)
```

it assigns the FlightSimulationData in the list to the FlightSimulationDataList deleting all the flight that already passed the Merge Point

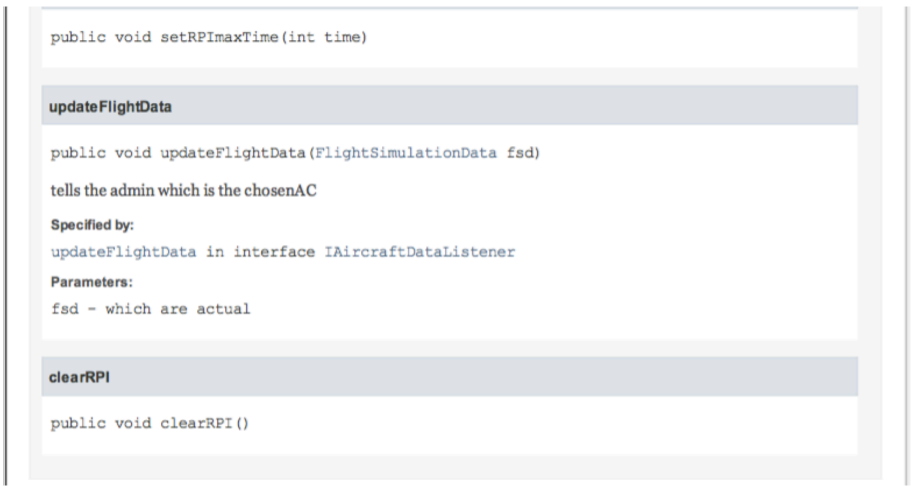
Specified by:

`updatePositions in interface ITrafficListener`

Parameters:

`list - containing the FlightSimulationData`

setRPImaxTime



```
public void setRPImaxTime(int time)
```

updateFlightData

```
public void updateFlightData(FlightSimulationData fsd)
```

tells the admin which is the chosenAC

Specified by:

`updateFlightData in interface IAircraftDataListener`

Parameters:

`fsd - which are actual`

clearRPI

```
public void clearRPI()
```

PART 6 – PROJECT FINALIZATION

20. Budget calculation

The budget for the presented Project is composed of three main blocks, labour costs, material and tools costs and indirect costs. The budget calculation for each block is listed in detail and the resulting final budget can be consulted at the end.

Labour costs

This project was realized by two engineers. One, who is the main developer (Engineer1) and the second, who was giving support and supervision. The engineer's salary is set at 50 €/hour. The project was realized from the end of April until the end of July, a total of 12 weeks. The total labour costs are calculated as follows:

	Hours/week	*Total weeks*	*Total hours*	*€/hours*	*Total €*
Engineer1	30	12	360	50	18.000
Engineer2	3	12	36	50	1.800
Total labour cost					19.800€

Material and Tools costs

This block contains the costs for all the material and programs used during the realization of the project.

The software used for the realization of the project and the costs for their licenses are the following:

- Netbeans 8.2 → free
- NEST → free
- Microsoft Office → 4,28 € / month
- Lucidchart → 9,95 € / month
- Zotero → free

Regarding the material cost, the project was mainly developed with a MacBook 7,2. Due to incompatibilities of the NEST software with the MacBook operating system, a Lenovo 80UD (windows) was also used.

	Total durability (months)	*Original price (€)*	*Usage during project (%)*	*Usage period (months)*	*Total cost €*
MacBook	48	1177,33	95%	3	70
Lenovo	36	600	3%	3	1,5
Microsoft Office	-	-	-	3	12,84
Lucidchart	-	-	-	3	29,85
Total material cost					114,19€

Indirect costs

The indirect costs are related to transport, electricity, administration and office costs. As they are difficult to calculate, an estimation of these costs based on the direct costs will be given. The indirect costs are estimated to be 20% of the direct costs.

	Part of direct cost	*Total direct cost*	*Total €*
Indirect costs	20%	19.914,19	3.982,84

The sum of these three blocks makes the total budget of the project; so the resulting final budget amounts to 23.897,03 €.

	Costs (€)
Labour	19.800
Material and tools	114,19
Indirect	3.982,84
Total budget	23.897,03 €

21. Conclusion

The objective of this work was introducing the theoretical concept of Point Merge operations, establishing all the relevant concepts related and analysing the advantages and opportunities Point Merge operations provide. The main part of this work is the development of a graphical tool for traffic simulation. Revising the program requirements with the final version of the developed tool, you arrive at the conclusion that all the requirements could be fulfilled. The implementation of a use case was an important part of the project in order to arrive at this conclusion.

RPI is a promising aid to improve the advantages of Point Merge identified in the theoretical part. The program developed constitutes a starting point to further improvement of RPI representation and offers an approach to integrate RPI as a standard tool for air traffic controllers monitoring Point Merge approaches.

Inside the program structure, the RPI panel can be easily replaced by another panel. If at any moment, a new tool is designed to be placed on the controller's monitor, it can be integrated in the present tool for testing, improvement and demonstration

21.1. Future work

As the developed tool - especially the inclusion of RPI - is only a first approach to a new monitor representation and reduced to historic traffic, future developments are necessary in order to make it a powerful tool adding new value to the controllers' work environment.

Regarding the design and starting from the presented use case, the clients and final users of the tool should be given the opportunity to comment any change requests. These changes could include the arrangement of information over the monitor, colour or dimension adjustments and most important, modifications in the RPI representation.

As for the historic traffic, the information about the route flown is known from the beginning and no calculations of higher complexity need to be done. The program for simulated and real traffics should include an algorithm to calculate prospectively. It should be based on the actual intended route and the time left to reach the Merge Point. For these calculations, aspects like turns, descent and speed variations need to be taken into account.

Another difference concerning simulated and real traffic compared to historic traffic, is the chance that the route to fly can be modified. This feature will have to be added to the code for these traffics.

The way information about the different traffics is obtain also varies between the three traffics to simulate. While for historic traffic all the necessary information is obtained in the .so6 file or can be easily calculated from it, in the simulated traffic all the information needs to be defined previously, manually changed or calculated. In case of real traffic, information is taken directly from the flying aircraft via radar.

21.2. Personal conclusion

The realization of this project allowed me to study and understand in detail the ATM Merge Point procedure. During my investigation, I could amplify my knowledge about Eurocontrol and learn about the tools and services they offer. The development of the program with NetBeans IDE using Java language and the GUI swing tool posed the challenge of learning about object oriented programming and a new programming language. By way of conclusion, it should be said that this work not only meant to apply knowledge acquired during my studies, but also demanded the transfer of this knowledge and the use of the developed skills to acquire the domination of a previously unknown tool. And this definitely is an utmost relevant skill engineers are frequently faced with during their professional career.

22. BIBLIOGRAPHY

[1] tagesschau.de, ‘Trotz Nachtflugverbots: So viele Nachtflüge wie noch nie’, *tagesschau.de*. [Online]. Available: https://www.tagesschau.de/inland/nachtfluege-101.html. [Accessed: 11-Jul-2018].

[2] ‘Chapter6.website.v3.en.pdf’. .

[3] ‘Air traffic management (ATM) explained | Eurocontrol’. [Online]. Available: http://www.eurocontrol.int/articles/air-traffic-management-atm-explained. [Accessed: 18-May-2018].

[4] ICAO, ‘WHAT IS ATM’, presented at the ASIA/PACIFIC MET/ATM Seminar, Fukuoka, Japan, 24-Jan-2011.

[5] ‘airspace management’, *TheFreeDictionary.com*. [Online]. Available: https://www.thefreedictionary.com/airspace+management. [Accessed: 19-May-2018].

[6] ‘Airspace Management (ASM)’, *BULATSA*, 26-Mar-2017. [Online]. Available: http://www.bulatsa.com/en/activities/air-navigation-services/air-traffic-management/airspace-management-asm. [Accessed: 19-May-2018].

[7] ICAO, ‘Air Navigation Report’, 2014.

[8] ‘NOP Network Operations Portal’. [Online]. Available: https://www.public.nm.eurocontrol.int/PUBPORTAL/gateway/spec/. [Accessed: 19-May-2018].

[9] J. Vila Carbó, ‘Air Traffic Services & Airspace organization’, 13-Oct-2015.

[10] ‘Airspace - EUROCONTROL ATM Lexicon’. [Online]. Available: https://ext.eurocontrol.int/lexicon/index.php/Airspace. [Accessed: 22-May-2018].

[11] ICAO, ‘Directives to Regional Air Navigation Meetings and Rules of Procedure for their Conduct’. 1987.

[12] ‘Lower airspace - EUROCONTROL ATM Lexicon’. [Online]. Available: https://ext.eurocontrol.int/lexicon/index.php/Lower_airspace.

[Accessed: 22-May-2018].

[13] ‘Upper airspace - EUROCONTROL ATM Lexicon’. [Online]. Available: https://ext.eurocontrol.int/lexicon/index.php/Upper_airspace. [Accessed: 22-May-2018].

[14] ‘Introduction to Airspace’, *NATS*. [Online]. Available: https://www.nats.aero/ae-home/introduction-to-airspace/. [Accessed: 25-Jun-2018].

[15] ‘Air Traffic Services. Air Traffic Control Service, Flight Information Service, Alerting Service.’ International Civil Aviation Organization, Jul-2001.

[16] ICAO, ‘Doc 8168, Procedimientos para los servicios de navegación aérea — Operación de aeronaves, Volumen II, Construcción de procedimientos de vuelo visual y por instrumentos’. 2014.

[17] ‘Is this the end of stack holding?’, *NATS*. [Online]. Available: https://www.nats.aero/apac/september-enewsletter/end-stack-holding/. [Accessed: 29-May-2018].

[18] EUROCONTROL, ‘RNAV Approaches’. Dec-2012.

[19] ‘RNAV Avionic System Basics | RNAV advantages,disadvantages’. [Online]. Available: http://www.rfwireless-world.com/Terminology/RNAV-Random-or-Area-Navigation.html. [Accessed: 01-Jun-2018].

[20] B. Favennec, T. Symmans, D. Houlihan, F. Vergne, and K. Zeghal, ‘Point Merge Integration of Arrival Flows Enabling Extensive RNAV Application and Continuous Descent - Operational Services and Environment Definition’. Eurocontrol, 19-Jul-2010.

[21] P. V. MacWilliams, A. P. Smith, and D. T. A. Becher, ‘RNP RNAV ARRIVAL ROUTE COORDINATION’, 2006, p. 12.

[22] ‘EUROCONTROL - Point Merge for Oslo, Dublin and Rome?’ [Online]. Available: https://www.eurocontrol.int/eec/public/standard_page/EEC_News_2008_3_PM.html#dublin. [Accessed: 08-Jun-2018].

[23] L. Boursier, B. Favennec, E. Hoffman, A. Trzmiel, F. Vergne, and K. Zeghal, 'MERGING ARRIVAL FLOWS WITHOUT HEADING INSTRUCTIONS', presented at the 7 th USA/Europe Air Traffic Management R&D Seminar, Barcelona, Spain, 2007.

[24] 'Real Time Simulation Dublin TMA2012 Phase 2. Implementation of a Point Merge System in Dublin TMA.', EUROCONTROL EXPERIMENTAL CENTRE, Nov. 2010.

[25] B. Favennec, L. Rognin, A. Trzmiel, F. Vergne, and K. Zeghal, 'POINT MERGE IN EXTENDED TERMINAL AREA (PMS-TE 2009-2010)', EUROCONTROL EXPERIMENTAL CENTRE, Oct. 2011.

[26] T. Symmans, 'Point Merge. A more efficient way of sequencing arrivals.', presented at the Eurocontrol Experimental Center.

[27] 'NATS and SESAR – working together to deliver a Single European Sky'. [Online]. Available: http://www.nats.aero/static/sesar/. [Accessed: 30-Jun-2018].

[28] 'Point Merge: improving and harmonising arrival operations | Eurocontrol'. [Online]. Available: http://www.eurocontrol.int/services/point-merge-concept. [Accessed: 08-Jun-2018].

[29] 'About the Network Manager | Eurocontrol'. [Online]. Available: http://www.eurocontrol.int/articles/about-network-manager. [Accessed: 12-May-2018].

[30] 'Airspace modelling | Eurocontrol'. [Online]. Available: http://www.eurocontrol.int/articles/airspace-modelling. [Accessed: 12-May-2018].

[31] 'NEST modelling tool | Eurocontrol'. [Online]. Available: http://www.eurocontrol.int/services/nest-modelling-tool. [Accessed: 12-May-2018].

[32] 'Demand Data Repository (DDR) | Eurocontrol'. [Online]. Available: http://www.eurocontrol.int/ddr. [Accessed: 14-May-2018].

[33] 'DDR2 Reference Manual for General Users 2.9.5'. 13-Feb-2018.

[34] 'DDR2 - Web Portal | Eurocontrol'. [Online]. Available: http://www.eurocontrol.int/articles/ddr2-web-portal. [Accessed: 14-May-2018].

[35] D. Poo, D. Kiong, and S. Ashok, *Object-Oriented Programming and Java*, Second Edition. Springer, 2008.

[36] 'Introduction to object oriented programming - Java Tutorial - Java With Us'. [Online]. Available: http://www.javawithus.com/tutorial/introduction-to-object-oriented-programming. [Accessed: 11-Jul-2018].

[37] 'Processes and Threads (The Java^{TM} Tutorials > Essential Classes > Concurrency)'. [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html. [Accessed: 13-Jul-2018].

[38] 'NetBeans IDE - Overview'. [Online]. Available: https://netbeans.org/features/index.html. [Accessed: 11-Jul-2018].

[39] 'NetBeans', *Wikipedia*. 05-Jun-2018.

[40] 'Point Merge successfully implemented in Dublin | Eurocontrol'. [Online]. Available: https://www.eurocontrol.int/news/point-merge-successfully-implemented-dublin. [Accessed: 08-Jun-2018].

[41] 'Dublin Point Merge Irish Aviation Authority - Point Merge Conference March 2015 Oslo', presented at the Point Merge Conference, Oslo, Mar-2015.

APPENDIX

Appendix 1. AIP IRELAND EIDW AD 2.24-17.1

AIP IRELAND

EIDW AD 2.24-17.1

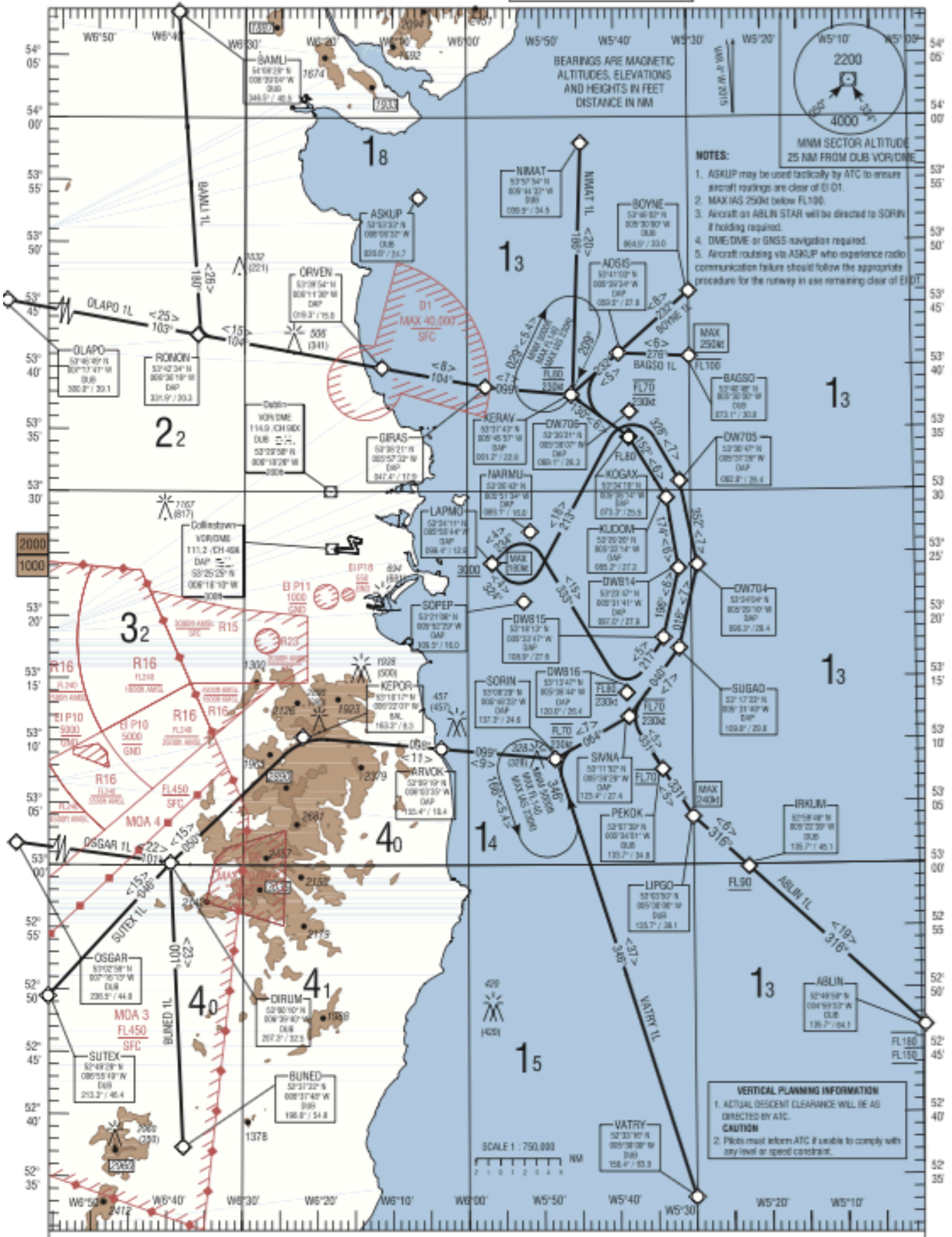
RNAV STANDARD ARRIVAL CHART
INSTRUMENT (STAR) - ICAO

TRANS ALT 5000ft
TRANS LEVEL by ATC

ATIS	124.525
APP	121.100
DUBLIN ACC UPPER NORTH	129.175
DUBLIN ACC UPPER SOUTH	124.650
DUBLIN ACC LOWER NORTH	132.575
DUBLIN ACC LOWER SOUTH	126.250

DUBLIN RWY 28

NIMAT 1L, BOYNE 1L, BAGSO 1L, ABLIN 1L, VATRY 1L,
BUNED 1L, SUTEX 1L, OSGAR 1L, OLAPO 1L, BAMLI 1L.



NOTES:

1. ASKUP may be used tactically by ATC to ensure aircraft routings are clear of EI D1.
2. MAX IAS 250kt below FL100.
3. Aircraft on ABLIN STAR will be directed to SORIN if holding required.
4. DME/DME or GNSS navigation required.
5. Aircraft routeing via ASKUP who experience radio communication failure should follow the appropriate procedure for the runway in use remaining clear of EI D1.

VERTICAL PLANNING INFORMATION

1. ACTUAL DESCENT CLEARANCE WILL BE AS DIRECTED BY ATC.

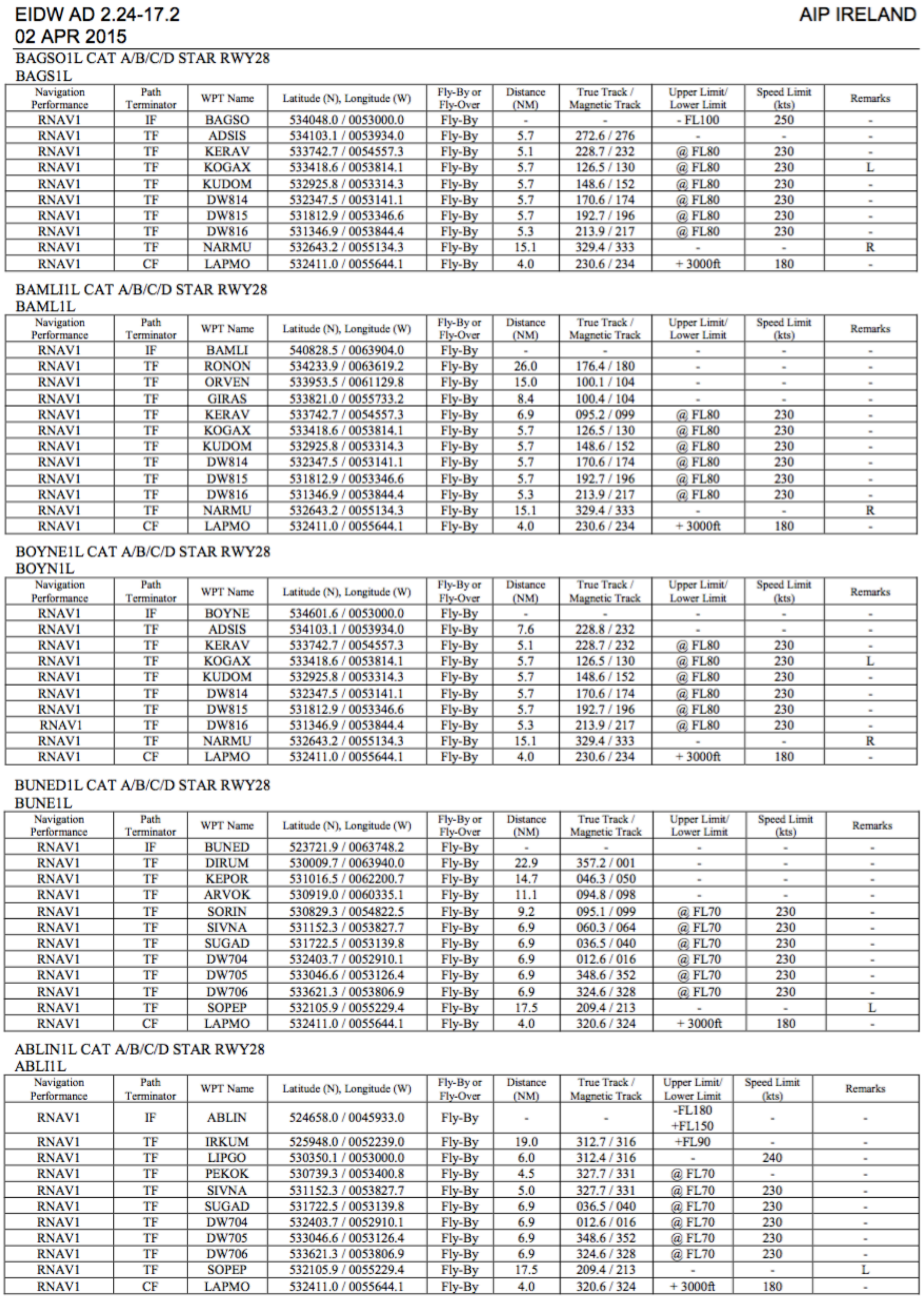
CAUTION

2. Pilots must inform ATC if unable to comply with any level or speed constraint.

CHANGE: LIPGO 1L removed and replaced with ABLIN 1L. MAG VAR and Bearings. Revised Note 3.

Irish Aviation Authority

AERONAUTICAL INFORMATION 02 APR 15



EIDW AD 2.24-17.2 AIP IRELAND
02 APR 2015

BAGSO1L CAT A/B/C/D STAR RWY28
BAGS1L

Navigation Performance	Path Terminator	WPT Name	Latitude (N), Longitude (W)	Fly-By or Fly-Over	Distance (NM)	True Track / Magnetic Track	Upper Limit/ Lower Limit	Speed Limit (kts)	Remarks
RNAV1	IF	BAGSO	534048.0 / 0053000.0	Fly-By	-	-	- FL100	250	-
RNAV1	TF	ADSIS	534103.1 / 0053934.0	Fly-By	5.7	272.6 / 276	-	-	-
RNAV1	TF	KERAV	533742.7 / 0054557.3	Fly-By	5.1	228.7 / 232	@ FL80	230	-
RNAV1	TF	KOGAX	533418.6 / 0053814.1	Fly-By	5.7	126.5 / 130	@ FL80	230	L
RNAV1	TF	KUDOM	532925.8 / 0053314.3	Fly-By	5.7	148.6 / 152	@ FL80	230	-
RNAV1	TF	DW814	532347.5 / 0053141.1	Fly-By	5.7	170.6 / 174	@ FL80	230	-
RNAV1	TF	DW815	531812.9 / 0053346.6	Fly-By	5.7	192.7 / 196	@ FL80	230	-
RNAV1	TF	DW816	531346.9 / 0053844.4	Fly-By	5.3	213.9 / 217	@ FL80	230	-
RNAV1	TF	NARMU	532643.2 / 0055134.3	Fly-By	15.1	329.4 / 333	-	-	R
RNAV1	CF	LAPMO	532411.0 / 0055644.1	Fly-By	4.0	230.6 / 234	+ 3000ft	180	-

BAMLI1L CAT A/B/C/D STAR RWY28
BAML1L

Navigation Performance	Path Terminator	WPT Name	Latitude (N), Longitude (W)	Fly-By or Fly-Over	Distance (NM)	True Track / Magnetic Track	Upper Limit/ Lower Limit	Speed Limit (kts)	Remarks
RNAV1	IF	BAMLI	540828.5 / 0063904.0	Fly-By	-	-	-	-	-
RNAV1	TF	RONON	534233.9 / 0063619.2	Fly-By	26.0	176.4 / 180	-	-	-
RNAV1	TF	ORVEN	533953.5 / 0061129.8	Fly-By	15.0	100.1 / 104	-	-	-
RNAV1	TF	GIRAS	533821.0 / 0055733.2	Fly-By	8.4	100.4 / 104	-	-	-
RNAV1	TF	KERAV	533742.7 / 0054557.3	Fly-By	6.9	095.2 / 099	@ FL80	230	-
RNAV1	TF	KOGAX	533418.6 / 0053814.1	Fly-By	5.7	126.5 / 130	@ FL80	230	-
RNAV1	TF	KUDOM	532925.8 / 0053314.3	Fly-By	5.7	148.6 / 152	@ FL80	230	-
RNAV1	TF	DW814	532347.5 / 0053141.1	Fly-By	5.7	170.6 / 174	@ FL80	230	-
RNAV1	TF	DW815	531812.9 / 0053346.6	Fly-By	5.7	192.7 / 196	@ FL80	230	-
RNAV1	TF	DW816	531346.9 / 0053844.4	Fly-By	5.3	213.9 / 217	@ FL80	230	-
RNAV1	TF	NARMU	532643.2 / 0055134.3	Fly-By	15.1	329.4 / 333	-	-	R
RNAV1	CF	LAPMO	532411.0 / 0055644.1	Fly-By	4.0	230.6 / 234	+ 3000ft	180	-

BOYNE1L CAT A/B/C/D STAR RWY28
BOYN1L

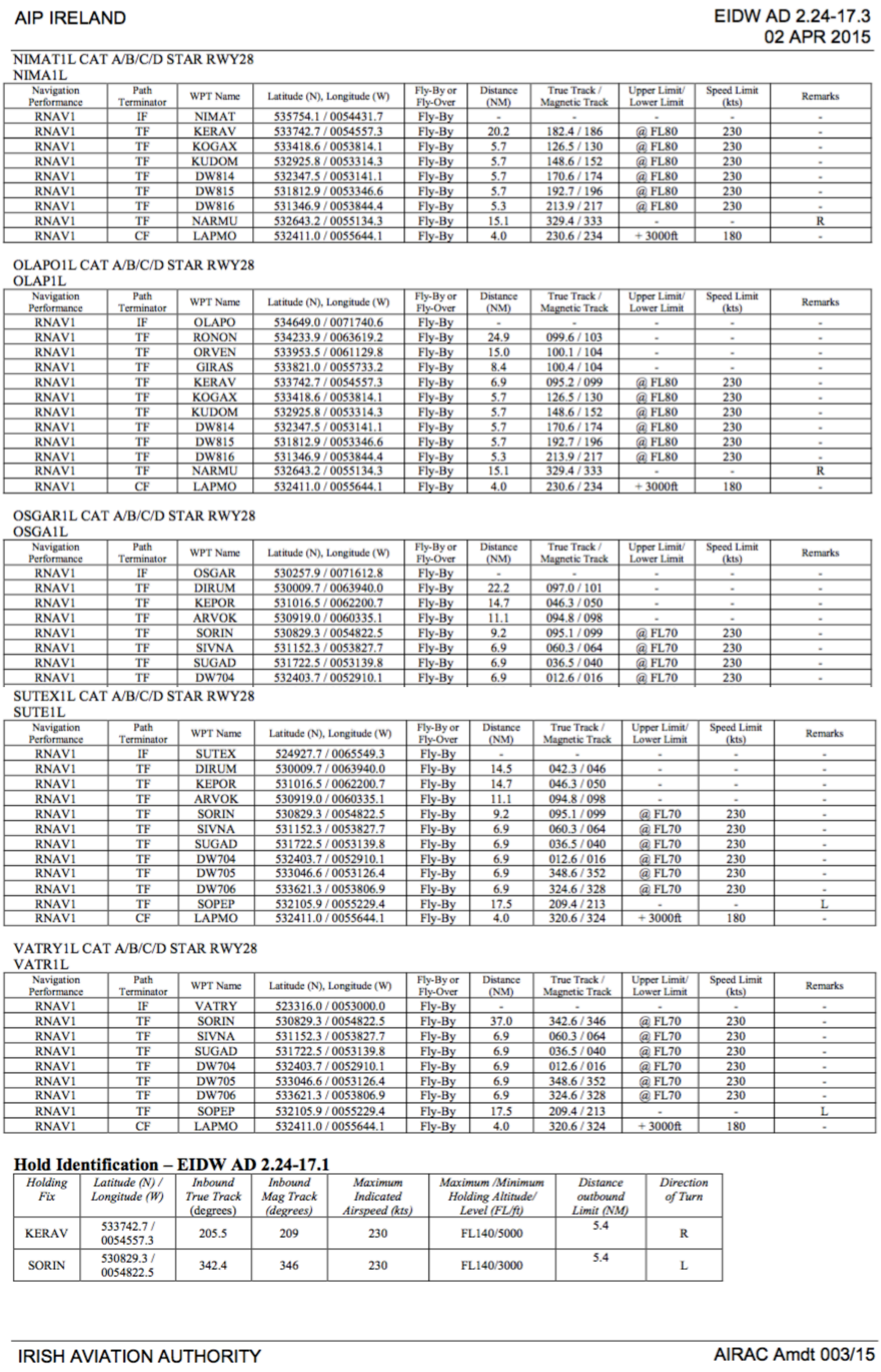
Navigation Performance	Path Terminator	WPT Name	Latitude (N), Longitude (W)	Fly-By or Fly-Over	Distance (NM)	True Track / Magnetic Track	Upper Limit/ Lower Limit	Speed Limit (kts)	Remarks
RNAV1	IF	BOYNE	534601.6 / 0053000.0	Fly-By	-	-	-	-	-
RNAV1	TF	ADSIS	534103.1 / 0053934.0	Fly-By	7.6	228.8 / 232	-	-	-
RNAV1	TF	KERAV	533742.7 / 0054557.3	Fly-By	5.1	228.7 / 232	@ FL80	230	-
RNAV1	TF	KOGAX	533418.6 / 0053814.1	Fly-By	5.7	126.5 / 130	@ FL80	230	L
RNAV1	TF	KUDOM	532925.8 / 0053314.3	Fly-By	5.7	148.6 / 152	@ FL80	230	-
RNAV1	TF	DW814	532347.5 / 0053141.1	Fly-By	5.7	170.6 / 174	@ FL80	230	-
RNAV1	TF	DW815	531812.9 / 0053346.6	Fly-By	5.7	192.7 / 196	@ FL80	230	-
RNAV1	TF	DW816	531346.9 / 0053844.4	Fly-By	5.3	213.9 / 217	@ FL80	230	-
RNAV1	TF	NARMU	532643.2 / 0055134.3	Fly-By	15.1	329.4 / 333	-	-	R
RNAV1	CF	LAPMO	532411.0 / 0055644.1	Fly-By	4.0	230.6 / 234	+ 3000ft	180	-

BUNED1L CAT A/B/C/D STAR RWY28
BUNE1L

Navigation Performance	Path Terminator	WPT Name	Latitude (N), Longitude (W)	Fly-By or Fly-Over	Distance (NM)	True Track / Magnetic Track	Upper Limit/ Lower Limit	Speed Limit (kts)	Remarks
RNAV1	IF	BUNED	523721.9 / 0063748.2	Fly-By	-	-	-	-	-
RNAV1	TF	DIRUM	530009.7 / 0063940.0	Fly-By	22.9	357.2 / 001	-	-	-
RNAV1	TF	KEPOR	531016.5 / 0062200.7	Fly-By	14.7	046.3 / 050	-	-	-
RNAV1	TF	ARVOK	530919.0 / 0060335.1	Fly-By	11.1	094.8 / 098	-	-	-
RNAV1	TF	SORIN	530829.3 / 0054822.5	Fly-By	9.2	095.1 / 099	@ FL70	230	-
RNAV1	TF	SIVNA	531152.3 / 0053827.7	Fly-By	6.9	060.3 / 064	@ FL70	230	-
RNAV1	TF	SUGAD	531722.5 / 0053139.8	Fly-By	6.9	036.5 / 040	@ FL70	230	-
RNAV1	TF	DW704	532403.7 / 0052910.1	Fly-By	6.9	012.6 / 016	@ FL70	230	-
RNAV1	TF	DW705	533046.6 / 0053126.4	Fly-By	6.9	348.6 / 352	@ FL70	230	-
RNAV1	TF	DW706	533621.3 / 0053806.9	Fly-By	6.9	324.6 / 328	@ FL70	230	-
RNAV1	TF	SOPEP	532105.9 / 0055229.4	Fly-By	17.5	209.4 / 213	-	-	L
RNAV1	CF	LAPMO	532411.0 / 0055644.1	Fly-By	4.0	320.6 / 324	+ 3000ft	180	-

ABLIN1L CAT A/B/C/D STAR RWY28
ABLI1L

Navigation Performance	Path Terminator	WPT Name	Latitude (N), Longitude (W)	Fly-By or Fly-Over	Distance (NM)	True Track / Magnetic Track	Upper Limit/ Lower Limit	Speed Limit (kts)	Remarks
RNAV1	IF	ABLIN	524658.0 / 0045933.0	Fly-By	-	-	-FL180 +FL150	-	-
RNAV1	TF	IRKUM	525948.0 / 0052239.0	Fly-By	19.0	312.7 / 316	+FL90	-	-
RNAV1	TF	LIPGO	530350.1 / 0053000.0	Fly-By	6.0	312.4 / 316	-	240	-
RNAV1	TF	PEKOK	530739.3 / 0053400.8	Fly-By	4.5	327.7 / 331	@ FL70	-	-
RNAV1	TF	SIVNA	531152.3 / 0053827.7	Fly-By	5.0	327.7 / 331	@ FL70	230	-
RNAV1	TF	SUGAD	531722.5 / 0053139.8	Fly-By	6.9	036.5 / 040	@ FL70	230	-
RNAV1	TF	DW704	532403.7 / 0052910.1	Fly-By	6.9	012.6 / 016	@ FL70	230	-
RNAV1	TF	DW705	533046.6 / 0053126.4	Fly-By	6.9	348.6 / 352	@ FL70	230	-
RNAV1	TF	DW706	533621.3 / 0053806.9	Fly-By	6.9	324.6 / 328	@ FL70	230	-
RNAV1	TF	SOPEP	532105.9 / 0055229.4	Fly-By	17.5	209.4 / 213	-	-	L
RNAV1	CF	LAPMO	532411.0 / 0055644.1	Fly-By	4.0	320.6 / 324	+ 3000ft	180	-



AIP IRELAND

EIDW AD 2.24-17.3
02 APR 2015

NIMAT1L CAT A/B/C/D STAR RWY28
NIMA1L

Navigation Performance	Path Terminator	WPT Name	Latitude (N), Longitude (W)	Fly-By or Fly-Over	Distance (NM)	True Track / Magnetic Track	Upper Limit/ Lower Limit	Speed Limit (kts)	Remarks
RNAV1	IF	NIMAT	535754.1 / 0054431.7	Fly-By	-	-	-	-	-
RNAV1	TF	KERAV	533742.7 / 0054557.3	Fly-By	20.2	182.4 / 186	@ FL80	230	-
RNAV1	TF	KOGAX	533418.6 / 0053814.1	Fly-By	5.7	126.5 / 130	@ FL80	230	-
RNAV1	TF	KUDOM	532925.8 / 0053314.3	Fly-By	5.7	148.6 / 152	@ FL80	230	-
RNAV1	TF	DW814	532347.5 / 0053141.1	Fly-By	5.7	170.6 / 174	@ FL80	230	-
RNAV1	TF	DW815	531812.9 / 0053346.6	Fly-By	5.7	192.7 / 196	@ FL80	230	-
RNAV1	TF	DW816	531346.9 / 0053844.4	Fly-By	5.3	213.9 / 217	@ FL80	230	-
RNAV1	TF	NARMU	532643.2 / 0055134.3	Fly-By	15.1	329.4 / 333	-	-	R
RNAV1	CF	LAPMO	532411.0 / 0055644.1	Fly-By	4.0	230.6 / 234	+ 3000ft	180	-

OLAPO1L CAT A/B/C/D STAR RWY28
OLAP1L

Navigation Performance	Path Terminator	WPT Name	Latitude (N), Longitude (W)	Fly-By or Fly-Over	Distance (NM)	True Track / Magnetic Track	Upper Limit/ Lower Limit	Speed Limit (kts)	Remarks
RNAV1	IF	OLAPO	534649.0 / 0071740.6	Fly-By	-	-	-	-	-
RNAV1	TF	RONON	534233.9 / 0063619.2	Fly-By	24.9	099.6 / 103	-	-	-
RNAV1	TF	ORVEN	533953.5 / 0061129.8	Fly-By	15.0	100.1 / 104	-	-	-
RNAV1	TF	GIRAS	533821.0 / 0055733.2	Fly-By	8.4	100.4 / 104	-	-	-
RNAV1	TF	KERAV	533742.7 / 0054557.3	Fly-By	6.9	095.2 / 099	@ FL80	230	-
RNAV1	TF	KOGAX	533418.6 / 0053814.1	Fly-By	5.7	126.5 / 130	@ FL80	230	-
RNAV1	TF	KUDOM	532925.8 / 0053314.3	Fly-By	5.7	148.6 / 152	@ FL80	230	-
RNAV1	TF	DW814	532347.5 / 0053141.1	Fly-By	5.7	170.6 / 174	@ FL80	230	-
RNAV1	TF	DW815	531812.9 / 0053346.6	Fly-By	5.7	192.7 / 196	@ FL80	230	-
RNAV1	TF	DW816	531346.9 / 0053844.4	Fly-By	5.3	213.9 / 217	@ FL80	230	-
RNAV1	TF	NARMU	532643.2 / 0055134.3	Fly-By	15.1	329.4 / 333	-	-	R
RNAV1	CF	LAPMO	532411.0 / 0055644.1	Fly-By	4.0	230.6 / 234	+ 3000ft	180	-

OSGAR1L CAT A/B/C/D STAR RWY28
OSGA1L

Navigation Performance	Path Terminator	WPT Name	Latitude (N), Longitude (W)	Fly-By or Fly-Over	Distance (NM)	True Track / Magnetic Track	Upper Limit/ Lower Limit	Speed Limit (kts)	Remarks
RNAV1	IF	OSGAR	530257.9 / 0071612.8	Fly-By	-	-	-	-	-
RNAV1	TF	DIRUM	530009.7 / 0063940.0	Fly-By	22.2	097.0 / 101	-	-	-
RNAV1	TF	KEPOR	531016.5 / 0062200.7	Fly-By	14.7	046.3 / 050	-	-	-
RNAV1	TF	ARVOK	530919.0 / 0060335.1	Fly-By	11.1	094.8 / 098	-	-	-
RNAV1	TF	SORIN	530829.3 / 0054822.5	Fly-By	9.2	095.1 / 099	@ FL70	230	-
RNAV1	TF	SIVNA	531152.3 / 0053827.7	Fly-By	6.9	060.3 / 064	@ FL70	230	-
RNAV1	TF	SUGAD	531722.5 / 0053139.8	Fly-By	6.9	036.5 / 040	@ FL70	230	-
RNAV1	TF	DW704	532403.7 / 0052910.1	Fly-By	6.9	012.6 / 016	@ FL70	230	-

SUTEX1L CAT A/B/C/D STAR RWY28
SUTE1L

Navigation Performance	Path Terminator	WPT Name	Latitude (N), Longitude (W)	Fly-By or Fly-Over	Distance (NM)	True Track / Magnetic Track	Upper Limit/ Lower Limit	Speed Limit (kts)	Remarks
RNAV1	IF	SUTEX	524927.7 / 0065549.3	Fly-By	-	-	-	-	-
RNAV1	TF	DIRUM	530009.7 / 0063940.0	Fly-By	14.5	042.3 / 046	-	-	-
RNAV1	TF	KEPOR	531016.5 / 0062200.7	Fly-By	14.7	046.3 / 050	-	-	-
RNAV1	TF	ARVOK	530919.0 / 0060335.1	Fly-By	11.1	094.8 / 098	-	-	-
RNAV1	TF	SORIN	530829.3 / 0054822.5	Fly-By	9.2	095.1 / 099	@ FL70	230	-
RNAV1	TF	SIVNA	531152.3 / 0053827.7	Fly-By	6.9	060.3 / 064	@ FL70	230	-
RNAV1	TF	SUGAD	531722.5 / 0053139.8	Fly-By	6.9	036.5 / 040	@ FL70	230	-
RNAV1	TF	DW704	532403.7 / 0052910.1	Fly-By	6.9	012.6 / 016	@ FL70	230	-
RNAV1	TF	DW705	533046.6 / 0053126.4	Fly-By	6.9	348.6 / 352	@ FL70	230	-
RNAV1	TF	DW706	533621.3 / 0053806.9	Fly-By	6.9	324.6 / 328	@ FL70	230	-
RNAV1	TF	SOPEP	532105.9 / 0055229.4	Fly-By	17.5	209.4 / 213	-	-	L
RNAV1	CF	LAPMO	532411.0 / 0055644.1	Fly-By	4.0	320.6 / 324	+ 3000ft	180	-

VATRY1L CAT A/B/C/D STAR RWY28
VATR1L

Navigation Performance	Path Terminator	WPT Name	Latitude (N), Longitude (W)	Fly-By or Fly-Over	Distance (NM)	True Track / Magnetic Track	Upper Limit/ Lower Limit	Speed Limit (kts)	Remarks
RNAV1	IF	VATRY	523316.0 / 0053000.0	Fly-By	-	-	-	-	-
RNAV1	TF	SORIN	530829.3 / 0054822.5	Fly-By	37.0	342.6 / 346	@ FL70	230	-
RNAV1	TF	SIVNA	531152.3 / 0053827.7	Fly-By	6.9	060.3 / 064	@ FL70	230	-
RNAV1	TF	SUGAD	531722.5 / 0053139.8	Fly-By	6.9	036.5 / 040	@ FL70	230	-
RNAV1	TF	DW704	532403.7 / 0052910.1	Fly-By	6.9	012.6 / 016	@ FL70	230	-
RNAV1	TF	DW705	533046.6 / 0053126.4	Fly-By	6.9	348.6 / 352	@ FL70	230	-
RNAV1	TF	DW706	533621.3 / 0053806.9	Fly-By	6.9	324.6 / 328	@ FL70	230	-
RNAV1	TF	SOPEP	532105.9 / 0055229.4	Fly-By	17.5	209.4 / 213	-	-	L
RNAV1	CF	LAPMO	532411.0 / 0055644.1	Fly-By	4.0	320.6 / 324	+ 3000ft	180	-

Hold Identification – EIDW AD 2.24-17.1

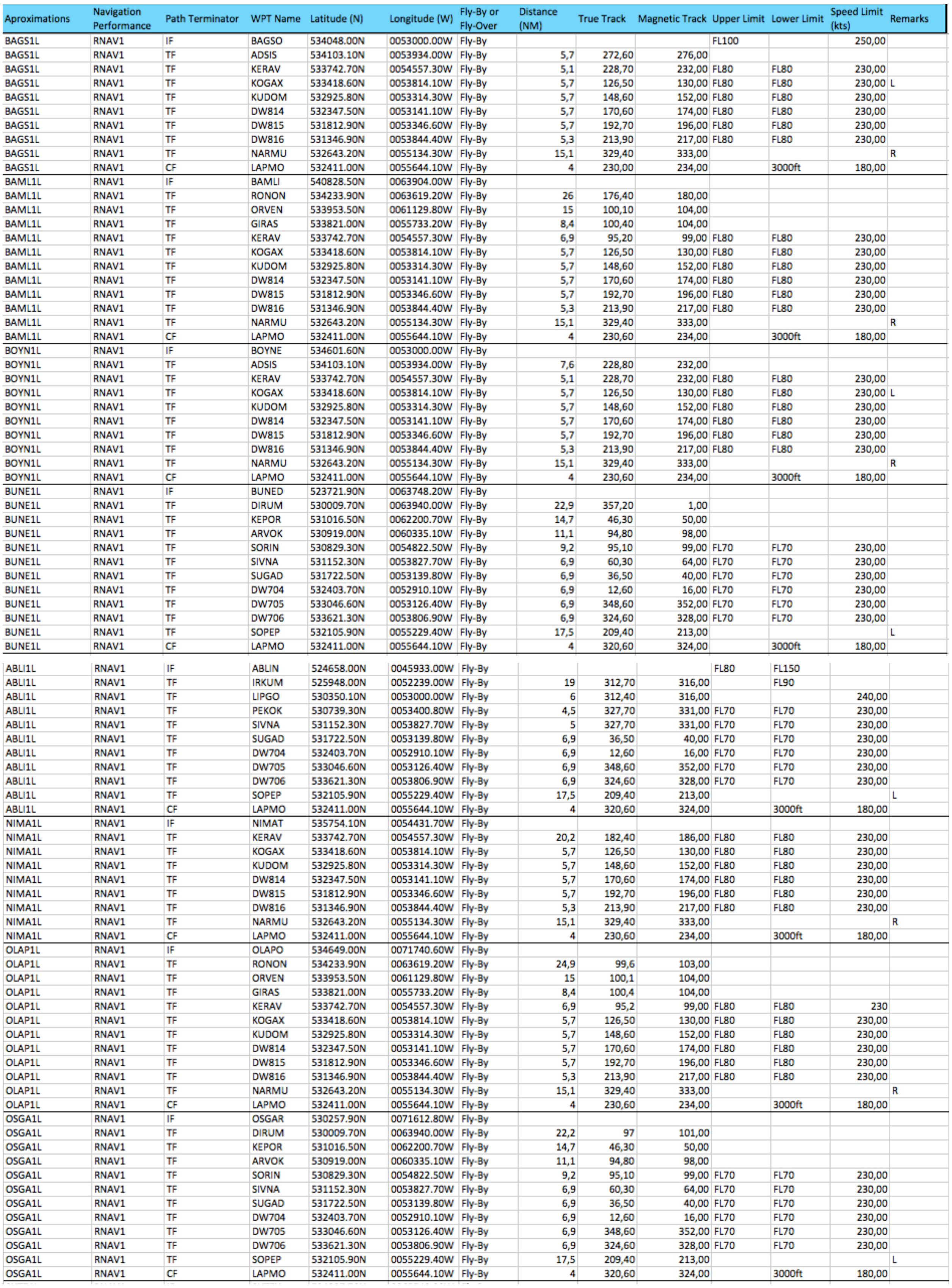
Holding Fix	*Latitude (N) / Longitude (W)*	*Inbound True Track* (degrees)	*Inbound Mag Track (degrees)*	*Maximum Indicated Airspeed (kts)*	*Maximum /Minimum Holding Altitude/ Level (FL/ft)*	*Distance outbound Limit (NM)*	*Direction of Turn*
KERAV	533742.7 / 0054557.3	205.5	209	230	FL140/5000	5.4	R
SORIN	530829.3 / 0054822.5	342.4	346	230	FL140/3000	5.4	L

IRISH AVIATION AUTHORITY

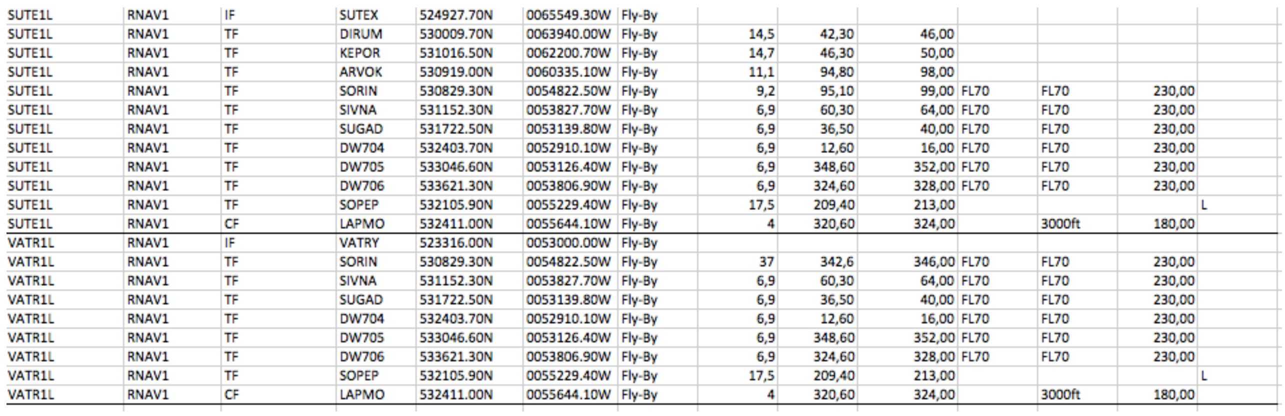
AIRAC Amdt 003/15

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix 2. Excel File – Routes for Approach Dublin Runway 28



Aproximations	Navigation Performance	Path Terminator	WPT Name	Latitude (N)	Longitude (W)	Fly-By or Fly-Over	Distance (NM)	True Track	Magnetic Track	Upper Limit	Lower Limit	Speed Limit (kts)	Remarks
BAGS1L	RNAV1	IF	BAGSO	534048.00N	0053000.00W	Fly-By				FL100		250,00	
BAGS1L	RNAV1	TF	ADSIS	534103.10N	0053934.00W	Fly-By	5,7	272,60	276,00				
BAGS1L	RNAV1	TF	KERAV	533742.70N	0054557.30W	Fly-By	5,1	228,70	232,00	FL80	FL80	230,00	
BAGS1L	RNAV1	TF	KOGAX	533418.60N	0053814.10W	Fly-By	5,7	126,50	130,00	FL80	FL80	230,00	L
BAGS1L	RNAV1	TF	KUDOM	532925.80N	0053314.30W	Fly-By	5,7	148,60	152,00	FL80	FL80	230,00	
BAGS1L	RNAV1	TF	DW814	532347.50N	0053141.10W	Fly-By	5,7	170,60	174,00	FL80	FL80	230,00	
BAGS1L	RNAV1	TF	DW815	531812.90N	0053346.60W	Fly-By	5,7	192,70	196,00	FL80	FL80	230,00	
BAGS1L	RNAV1	TF	DW816	531346.90N	0053844.40W	Fly-By	5,3	213,90	217,00	FL80	FL80	230,00	
BAGS1L	RNAV1	TF	NARMU	532643.20N	0055134.30W	Fly-By	15,1	329,40	333,00				R
BAGS1L	RNAV1	CF	LAPMO	532411.00N	0055644.10W	Fly-By	4	230,00	234,00		3000ft	180,00	
BAML1L	RNAV1	IF	BAMLI	540828.50N	0063904.00W	Fly-By							
BAML1L	RNAV1	TF	RONON	534233.90N	0063619.20W	Fly-By	26	176,40	180,00				
BAML1L	RNAV1	TF	ORVEN	533953.50N	0061129.80W	Fly-By	15	100,10	104,00				
BAML1L	RNAV1	TF	GIRAS	533821.00N	0055733.20W	Fly-By	8,4	100,40	104,00				
BAML1L	RNAV1	TF	KERAV	533742.70N	0054557.30W	Fly-By	6,9	95,20	99,00	FL80	FL80	230,00	
BAML1L	RNAV1	TF	KOGAX	533418.60N	0053814.10W	Fly-By	5,7	126,50	130,00	FL80	FL80	230,00	
BAML1L	RNAV1	TF	KUDOM	532925.80N	0053314.30W	Fly-By	5,7	148,60	152,00	FL80	FL80	230,00	
BAML1L	RNAV1	TF	DW814	532347.50N	0053141.10W	Fly-By	5,7	170,60	174,00	FL80	FL80	230,00	
BAML1L	RNAV1	TF	DW815	531812.90N	0053346.60W	Fly-By	5,7	192,70	196,00	FL80	FL80	230,00	
BAML1L	RNAV1	TF	DW816	531346.90N	0053844.40W	Fly-By	5,3	213,90	217,00	FL80	FL80	230,00	
BAML1L	RNAV1	TF	NARMU	532643.20N	0055134.30W	Fly-By	15,1	329,40	333,00				R
BAML1L	RNAV1	CF	LAPMO	532411.00N	0055644.10W	Fly-By	4	230,60	234,00		3000ft	180,00	
BOYN1L	RNAV1	IF	BOYNE	534601.60N	0053000.00W	Fly-By							
BOYN1L	RNAV1	TF	ADSIS	534103.10N	0053934.00W	Fly-By	7,6	228,80	232,00				
BOYN1L	RNAV1	TF	KERAV	533742.70N	0054557.30W	Fly-By	5,1	228,70	232,00	FL80	FL80	230,00	
BOYN1L	RNAV1	TF	KOGAX	533418.60N	0053814.10W	Fly-By	5,7	126,50	130,00	FL80	FL80	230,00	L
BOYN1L	RNAV1	TF	KUDOM	532925.80N	0053314.30W	Fly-By	5,7	148,60	152,00	FL80	FL80	230,00	
BOYN1L	RNAV1	TF	DW814	532347.50N	0053141.10W	Fly-By	5,7	170,60	174,00	FL80	FL80	230,00	
BOYN1L	RNAV1	TF	DW815	531812.90N	0053346.60W	Fly-By	5,7	192,70	196,00	FL80	FL80	230,00	
BOYN1L	RNAV1	TF	DW816	531346.90N	0053844.40W	Fly-By	5,3	213,90	217,00	FL80	FL80	230,00	
BOYN1L	RNAV1	TF	NARMU	532643.20N	0055134.30W	Fly-By	15,1	329,40	333,00				R
BOYN1L	RNAV1	CF	LAPMO	532411.00N	0055644.10W	Fly-By	4	230,60	234,00		3000ft	180,00	
BUNE1L	RNAV1	IF	BUNED	523721.90N	0063748.20W	Fly-By							
BUNE1L	RNAV1	TF	DIRUM	530009.70N	0063940.00W	Fly-By	22,9	357,20	1,00				
BUNE1L	RNAV1	TF	KEPOR	531016.50N	0062200.70W	Fly-By	14,7	46,30	50,00				
BUNE1L	RNAV1	TF	ARVOK	530919.00N	0060335.10W	Fly-By	11,1	94,80	98,00				
BUNE1L	RNAV1	TF	SORIN	530829.30N	0054822.50W	Fly-By	9,2	95,10	99,00	FL70	FL70	230,00	
BUNE1L	RNAV1	TF	SIVNA	531152.30N	0053827.70W	Fly-By	6,9	60,30	64,00	FL70	FL70	230,00	
BUNE1L	RNAV1	TF	SUGAD	531722.50N	0053139.80W	Fly-By	6,9	36,50	40,00	FL70	FL70	230,00	
BUNE1L	RNAV1	TF	DW704	532403.70N	0052910.10W	Fly-By	6,9	12,60	16,00	FL70	FL70	230,00	
BUNE1L	RNAV1	TF	DW705	533046.60N	0053126.40W	Fly-By	6,9	348,60	352,00	FL70	FL70	230,00	
BUNE1L	RNAV1	TF	DW706	533621.30N	0053806.90W	Fly-By	6,9	324,60	328,00	FL70	FL70	230,00	
BUNE1L	RNAV1	TF	SOPEP	532105.90N	0055229.40W	Fly-By	17,5	209,40	213,00				L
BUNE1L	RNAV1	CF	LAPMO	532411.00N	0055644.10W	Fly-By	4	320,60	324,00		3000ft	180,00	
ABLI1L	RNAV1	IF	ABLIN	524658.00N	0045933.00W	Fly-By				FL80	FL150		
ABLI1L	RNAV1	TF	IRKUM	525948.00N	0052239.00W	Fly-By	19	312,70	316,00		FL90		
ABLI1L	RNAV1	TF	LIPGO	530350.10N	0053000.00W	Fly-By	6	312,40	316,00			240,00	
ABLI1L	RNAV1	TF	PEKOK	530739.30N	0053400.80W	Fly-By	4,5	327,70	331,00	FL70	FL70	230,00	
ABLI1L	RNAV1	TF	SIVNA	531152.30N	0053827.70W	Fly-By	5	327,70	331,00	FL70	FL70	230,00	
ABLI1L	RNAV1	TF	SUGAD	531722.50N	0053139.80W	Fly-By	6,9	36,50	40,00	FL70	FL70	230,00	
ABLI1L	RNAV1	TF	DW704	532403.70N	0052910.10W	Fly-By	6,9	12,60	16,00	FL70	FL70	230,00	
ABLI1L	RNAV1	TF	DW705	533046.60N	0053126.40W	Fly-By	6,9	348,60	352,00	FL70	FL70	230,00	
ABLI1L	RNAV1	TF	DW706	533621.30N	0053806.90W	Fly-By	6,9	324,60	328,00	FL70	FL70	230,00	
ABLI1L	RNAV1	TF	SOPEP	532105.90N	0055229.40W	Fly-By	17,5	209,40	213,00				L
ABLI1L	RNAV1	CF	LAPMO	532411.00N	0055644.10W	Fly-By	4	320,60	324,00		3000ft	180,00	
NIMA1L	RNAV1	IF	NIMAT	535754.10N	0054431.70W	Fly-By							
NIMA1L	RNAV1	TF	KERAV	533742.70N	0054557.30W	Fly-By	20,2	182,40	186,00	FL80	FL80	230,00	
NIMA1L	RNAV1	TF	KOGAX	533418.60N	0053814.10W	Fly-By	5,7	126,50	130,00	FL80	FL80	230,00	
NIMA1L	RNAV1	TF	KUDOM	532925.80N	0053314.30W	Fly-By	5,7	148,60	152,00	FL80	FL80	230,00	
NIMA1L	RNAV1	TF	DW814	532347.50N	0053141.10W	Fly-By	5,7	170,60	174,00	FL80	FL80	230,00	
NIMA1L	RNAV1	TF	DW815	531812.90N	0053346.60W	Fly-By	5,7	192,70	196,00	FL80	FL80	230,00	
NIMA1L	RNAV1	TF	DW816	531346.90N	0053844.40W	Fly-By	5,3	213,90	217,00	FL80	FL80	230,00	
NIMA1L	RNAV1	TF	NARMU	532643.20N	0055134.30W	Fly-By	15,1	329,40	333,00				R
NIMA1L	RNAV1	CF	LAPMO	532411.00N	0055644.10W	Fly-By	4	230,60	234,00		3000ft	180,00	
OLAP1L	RNAV1	IF	OLAPO	534649.00N	0071740.60W	Fly-By							
OLAP1L	RNAV1	TF	RONON	534233.90N	0063619.20W	Fly-By	24,9	99,6	103,00				
OLAP1L	RNAV1	TF	ORVEN	533953.50N	0061129.80W	Fly-By	15	100,1	104,00				
OLAP1L	RNAV1	TF	GIRAS	533821.00N	0055733.20W	Fly-By	8,4	100,4	104,00				
OLAP1L	RNAV1	TF	KERAV	533742.70N	0054557.30W	Fly-By	6,9	95,2	99,00	FL80	FL80	230	
OLAP1L	RNAV1	TF	KOGAX	533418.60N	0053814.10W	Fly-By	5,7	126,50	130,00	FL80	FL80	230,00	
OLAP1L	RNAV1	TF	KUDOM	532925.80N	0053314.30W	Fly-By	5,7	148,60	152,00	FL80	FL80	230,00	
OLAP1L	RNAV1	TF	DW814	532347.50N	0053141.10W	Fly-By	5,7	170,60	174,00	FL80	FL80	230,00	
OLAP1L	RNAV1	TF	DW815	531812.90N	0053346.60W	Fly-By	5,7	192,70	196,00	FL80	FL80	230,00	
OLAP1L	RNAV1	TF	DW816	531346.90N	0053844.40W	Fly-By	5,3	213,90	217,00	FL80	FL80	230,00	
OLAP1L	RNAV1	TF	NARMU	532643.20N	0055134.30W	Fly-By	15,1	329,40	333,00				R
OLAP1L	RNAV1	CF	LAPMO	532411.00N	0055644.10W	Fly-By	4	230,60	234,00		3000ft	180,00	
OSGA1L	RNAV1	IF	OSGAR	530257.90N	0071612.80W	Fly-By							
OSGA1L	RNAV1	TF	DIRUM	530009.70N	0063940.00W	Fly-By	22,2	97	101,00				
OSGA1L	RNAV1	TF	KEPOR	531016.50N	0062200.70W	Fly-By	14,7	46,30	50,00				
OSGA1L	RNAV1	TF	ARVOK	530919.00N	0060335.10W	Fly-By	11,1	94,80	98,00				
OSGA1L	RNAV1	TF	SORIN	530829.30N	0054822.50W	Fly-By	9,2	95,10	99,00	FL70	FL70	230,00	
OSGA1L	RNAV1	TF	SIVNA	531152.30N	0053827.70W	Fly-By	6,9	60,30	64,00	FL70	FL70	230,00	
OSGA1L	RNAV1	TF	SUGAD	531722.50N	0053139.80W	Fly-By	6,9	36,50	40,00	FL70	FL70	230,00	
OSGA1L	RNAV1	TF	DW704	532403.70N	0052910.10W	Fly-By	6,9	12,60	16,00	FL70	FL70	230,00	
OSGA1L	RNAV1	TF	DW705	533046.60N	0053126.40W	Fly-By	6,9	348,60	352,00	FL70	FL70	230,00	
OSGA1L	RNAV1	TF	DW706	533621.30N	0053806.90W	Fly-By	6,9	324,60	328,00	FL70	FL70	230,00	
OSGA1L	RNAV1	TF	SOPEP	532105.90N	0055229.40W	Fly-By	17,5	209,40	213,00				L
OSGA1L	RNAV1	CF	LAPMO	532411.00N	0055644.10W	Fly-By	4	320,60	324,00		3000ft	180,00	



SUTE1L	RNAV1	IF	SUTEX	524927.70N	0065549.30W	Fly-By							
SUTE1L	RNAV1	TF	DIRUM	530009.70N	0063940.00W	Fly-By	14,5	42,30	46,00				
SUTE1L	RNAV1	TF	KEPOR	531016.50N	0062200.70W	Fly-By	14,7	46,30	50,00				
SUTE1L	RNAV1	TF	ARVOK	530919.00N	0060335.10W	Fly-By	11,1	94,80	98,00				
SUTE1L	RNAV1	TF	SORIN	530829.30N	0054822.50W	Fly-By	9,2	95,10	99,00	FL70	FL70	230,00	
SUTE1L	RNAV1	TF	SIVNA	531152.30N	0053827.70W	Fly-By	6,9	60,30	64,00	FL70	FL70	230,00	
SUTE1L	RNAV1	TF	SUGAD	531722.50N	0053139.80W	Fly-By	6,9	36,50	40,00	FL70	FL70	230,00	
SUTE1L	RNAV1	TF	DW704	532403.70N	0052910.10W	Fly-By	6,9	12,60	16,00	FL70	FL70	230,00	
SUTE1L	RNAV1	TF	DW705	533046.60N	0053126.40W	Fly-By	6,9	348,60	352,00	FL70	FL70	230,00	
SUTE1L	RNAV1	TF	DW706	533621.30N	0053806.90W	Fly-By	6,9	324,60	328,00	FL70	FL70	230,00	
SUTE1L	RNAV1	TF	SOPEP	532105.90N	0055229.40W	Fly-By	17,5	209,40	213,00				L
SUTE1L	RNAV1	CF	LAPMO	532411.00N	0055644.10W	Fly-By	4	320,60	324,00		3000ft	180,00	
VATR1L	RNAV1	IF	VATRY	523316.00N	0053000.00W	Fly-By							
VATR1L	RNAV1	TF	SORIN	530829.30N	0054822.50W	Fly-By	37	342,6	346,00	FL70	FL70	230,00	
VATR1L	RNAV1	TF	SIVNA	531152.30N	0053827.70W	Fly-By	6,9	60,30	64,00	FL70	FL70	230,00	
VATR1L	RNAV1	TF	SUGAD	531722.50N	0053139.80W	Fly-By	6,9	36,50	40,00	FL70	FL70	230,00	
VATR1L	RNAV1	TF	DW704	532403.70N	0052910.10W	Fly-By	6,9	12,60	16,00	FL70	FL70	230,00	
VATR1L	RNAV1	TF	DW705	533046.60N	0053126.40W	Fly-By	6,9	348,60	352,00	FL70	FL70	230,00	
VATR1L	RNAV1	TF	DW706	533621.30N	0053806.90W	Fly-By	6,9	324,60	328,00	FL70	FL70	230,00	
VATR1L	RNAV1	TF	SOPEP	532105.90N	0055229.40W	Fly-By	17,5	209,40	213,00				L
VATR1L	RNAV1	CF	LAPMO	532411.00N	0055644.10W	Fly-By	4	320,60	324,00		3000ft	180,00	