

# **ENGINEERING THE INFORMATION FOR A WEB- BASED TRAINING SYSTEM FOR THE USE OF AIRBORNE VEHICLES.**

## **PROGRAMMING PROJECT REPORT.**

**UNIVERSITY OF OSNABRÜCK / UNIVERSITY OF TWENTE.**

Presented by

Jorge A. Martínez Malacara  
Student of the Master in Information Engineering  
Matriculation number 917654

Mecklenburger Str. 2 Zi. 221  
49088 Osnabrück

[jmartine@uni-osnabrueck.de](mailto:jmartine@uni-osnabrueck.de)

Osnabrück, Germany  
May 15<sup>th</sup>, 2004.

# Table of Contents

1. Abstract .....	2
2. Introduction.....	3
3. Description of the system.....	4
4. Description of the problem .....	6
4.1. Collecting the data .....	6
4.2. The information engineering problem .....	8
4.3. Variables selection .....	10
4.4. Flight stages .....	11
5. The Virtual Airline Trainer Black Box program.....	12
6. Future of the project .....	12
7. Bibliography .....	13
8. Annex 1: VAT File Standard (version 0.6.3).....	14
8.1. File Header. ....	15
8.2. Pre-Take-Off Header.....	16
8.3. Pre-Take-Off Terminator. ....	17
8.4. On-Flight Section Header.....	18
8.5. On-Flight Section Terminator .....	19
8.6. Landing Section Header .....	19
8.7. Landing Section Terminator.....	20
8.8. File Trailer.....	20
8.9. Data Elements.....	20
9. Annex 2: Variables sampled from MSFS.....	22
10. Annex 3: Questionnaires used to determine the relevant parameters needed to evaluate a pilot. ....	25
10.1. Questionnaire 1 .....	25
10.2. Questionnaire 2 .....	26
10.3. Questionnaire 3 .....	27

## 1. Abstract

Some systems require large amounts of information to be processed and analyzed in order to find patterns or build abstractions on data which could show unexpected performances revealing faulty designs, set-up errors or improper operation of the system. These systems usually consist of a black box, which monitors and samples data from an array of sensors, and the software which analyzes the data sampled from the black box. This concept can have many applications, ranging from accident investigation, training purposes, simulation or as an alternative to expensive real-time monitoring systems which do not require the information to be collected and processed immediately. Any of these tasks requires engineering the information generated by the system because of space limitations which can occur while sampling data for longer periods of time and/or with high sampling frequencies.

As part of the Programming Project's Seminar from the University of Osnabrück's Master in Information Engineering, such a black box system is being developed along with a knowledge-based system which will analyze and interpret the data produced by a black box connected to Microsoft's Flight Simulator. This report

presents the general characteristics of the system as well as its current development status.

## 2. Introduction

An airplane's flight data recorder (FDR), also known as black box, is a typical example: it stores data collected at certain time intervals from a set of sensors located all around the plane. This information can be analyzed in order to find out the causes of an accident or a serious incident.

The concept of a black box can have many other applications, especially as an alternative to expensive real-time monitoring systems when immediate process of data is not required. The main advantage of this concept is that the information generated by a device can be stored for a certain amount of time, so it can be retrieved and analyzed several times, under different perspectives. This concept can be also very useful for simulation, testing new systems, or training and evaluating personnel.

The black box paradigm is all about storing data obtained over a certain period of time from an array of sensors and actuators. After a certain period of time, the oldest data is usually overwritten, leading to a permanent loss of data whenever it was not properly extracted. In several events, the data stored in the FDR wasn't properly secured for extraction [11], according to the appropriate regulations [13, 14], so its contents were permanently lost before the investigation could begin. [12]

Under this paradigm, the information itself is never analyzed until a malfunction or unexpected performance in the monitored system occurs and is detected. A FDR turns to be a very useful warehouse of clues, which can be used to determine the cause of a problem, but the information itself is not used unless such a failure occurs. The accidents of the USAir flights 585 and 427 were solved mostly because of the information provided by its FDR. In 1995, the National Transportation Safety Board issued an urgent recommendation "regarding the need to increase the number of FDR parameters" [9]. The complete recommendation can be found in [10].

Once the failure occurs, the data must be made available for analysis. Usually this is achieved by providing means to extract or copy the information from the FDR before it is overwritten. Once the information is available, it can be analyzed by external programs which simulate the events leading to the failure, based on the information provided by the FDR. Any discrepancy between the simulation and the actual events recorded in the FDR is the starting point needed to find the cause of the failure.

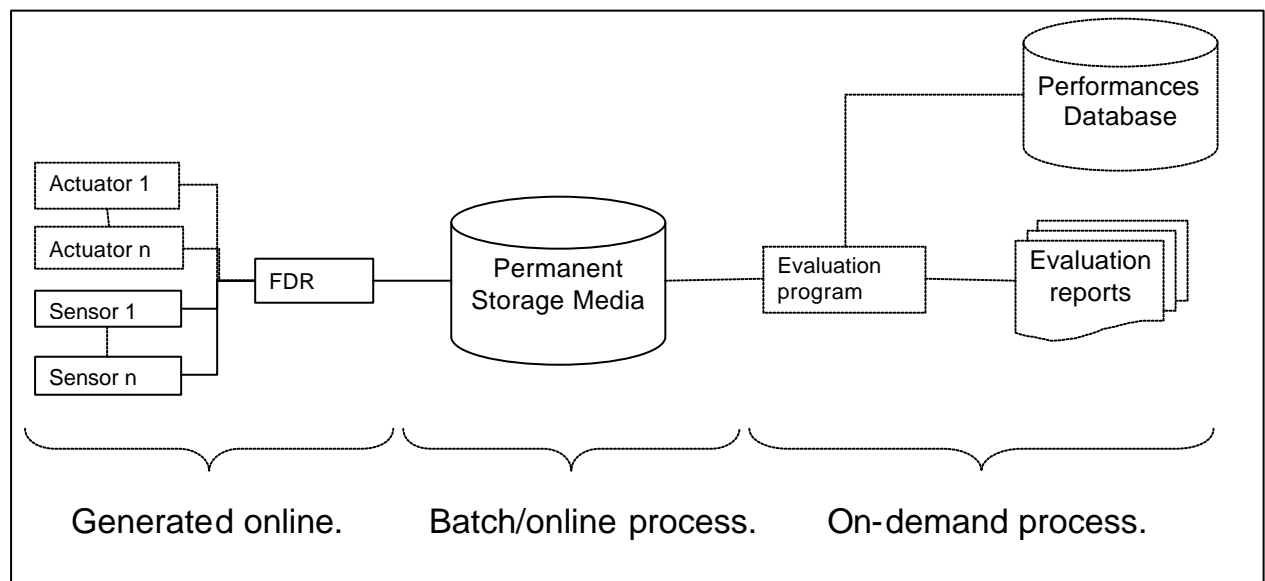
But, what if, instead of waiting for an unexpected performance to become evident, the data stored in the FDR is extracted and analyzed with a software tool which, based on first order logic or any other mean of knowledge representation, would be able to interpret the observed performances. The latter tool should be able also to give a feedback on the system, pinpointing minor unexpected events and/or small failures in the system's operation. This would be helpful preventing major failures, since their causes are detected long before they become a major problem. At the same time, it could be possible to monitor human-operated systems and provide

prompt feedback related to the system's operation, as a training tool. Let's go beyond this point and focus on certain circumstances: what if it's not always possible to have the monitored system and the analysis tool at the same place or at least permanently connected?

The analysis task implies the pre-existence of a knowledge base or a set of sample data which could be used as a reference for comparison. The creation and/or use of that knowledge base make the task even more challenging.

### 3. Description of the system

The existence of a knowledge base or benchmarking information transforms the way the architecture of the whole system. First of all, we can split it in 3 parts. The first would be a FDR as described before. The second will be just the permanent storage media of the information generated by the FDR. Finally, the third major component will be the evaluation program which, comparing the data recorded by the FDR and the expected performances will generate a series of evaluation reports. Figure 1 illustrates the architecture of such a system.



**Figure 1:** Architecture of the proposed FDR-based training system.

As we can see, there are three phases in the information flow. The first stage is generated “online”, with the information from the sensors and actuators feeding the FDR. It is important to state that the information is actually collected at discreet intervals, as it will be described in section 4.2, paragraph 2. The second phase is performed as a batch or could be online, depending on whether there is a permanent communications link between the FDR and the events database. The third phase is the evaluation process itself, when the information contained in the events database is compared with the expected/allowed performances. It could be possible to update the performances database afterwards, given that the nature of the monitored system allows it.

Before going further, it's needed to state that it's not an easy task to find a system with sensors which can be analyzed in such a way that, in just 6 to 8 weeks, both a FDR and its corresponding data analysis software could be even partially implemented. Any data analysis system requires the programmer to get familiar with the system and find ways to represent the knowledge of an expert. In addition to this, full access should be required to such a system and it would be expected that this wouldn't be granted within a company. In order to focus in the information engineering tasks, a previously implemented sensor system had to be found, or at least the software tool which simulates such a system.

Microsoft's Flight Simulator (MSFS) was found to be a very good alternative. This software simulates a real airplane's systems to a very high level of detail and it's possible to read information from the simulated sensors and actuators from external applications. In addition to this, there are some programs which enhance the simulation and data sampling capabilities, increasing the quality of the simulation. Therefore the MSFS was chosen as the source of data for the FDR.

An aeronautical-designed FDR represents an information engineering challenge. The information stored in them must survive almost intact under extreme environments like water or high temperatures caused by fire. These requirements make necessary to use some kind of error-correcting code for the data stored, which necessarily need more bytes than the information itself. In addition to this, large amounts of data have to be sampled at high frequencies, making more complex the storage problem. Most of the requirements of information stored in a black box are listed by the Federal Aviation Administration (FAA) in the Federal Air Regulations (FAR) sections 121.343 and 121.344. According to those regulations, at least 88 parameters have to be sampled (which, for a 4-engine airplane, sum up to 115 variables). [1, 2] Those variables have to be sampled up to 4 times per second [2] and at least 25 hours of data must be saved [3]. Assuming that each variable requires 4 bytes, that no error detecting-code is used, and that there is no underlying data structure, the FDR should be capable of holding at least 165 Mb of information. Because the system proposed doesn't need to survive the environments described above, the only limitation will be the information storage.

Some of FAA's requirements for a FDR, like flight control input forces [1], cannot be simulated within MSFS. At the same time, MSFS can provide information which is not required by FAR 121.343 or 121.344, like the surface type the plane is standing on or radio altimeter's value. One must also consider that those regulations are focused on accident investigation and that's not the goal of this project. Instead, the FDR concept can be used to assess a pilot's performance.

Under this perspective, the program can be designed for MSFS fans or real pilots who want to improve their flying skills at home. Using the architecture proposed in figure 1, I defined the evaluation program to be a series of server-based applets or scripts capable of interpreting the data generated by a black box program for MSFS, working offline and then sending the information to the server once it's generated.

This programming project, as stated in the beginning of the semester, has two main objectives. First of all, the student should get familiar and learn at least a previously unknown programming language, in which at least a significant part of the project

must be developed. On the other hand, the student is expected to deal with and solve an information engineering problem.

Under these constraints, the black box system was programmed using Microsoft's Visual C++ .NET with Microsoft Foundations Classes. The knowledge based system will be programmed using PHP. At the time the proposal for this project was presented, the author of this report was not familiar with any of both languages.

## **4. Description of the problem**

Several problems had to be solved on the early stages of the project. The most obvious are the selection of the method to collect the data from MSFS, the information engineering problem and the selection of variables to be sampled from MSFS.

### **4.1. *Collecting the data***

There are 3 well-known techniques to acquire the data from the MSFS. The first one is provided by Microsoft as part of its Flight Simulator's Software Development Kits (SDKs). Netpipes is an SDK in the form of C++ header files which allows the user to read offsets containing data from MSFS. However, Microsoft has changed the internal offsets of the Flight Simulator which can be read in the latest 2 versions, mostly because of the enhanced simulation capabilities supported by the newest high-end personal computers. Therefore, there exists a version of Netpipes for each of the versions of the Flight Simulator since FS98. Even when 80% of the users use the latest MSFS version (FS9) [7], a significant amount of Flight Simulator's users have older versions of the program (FS98, FS2000, FS2002, in the order they have been released). It would be needed to develop almost a different black box program for each new version of the MSFS, as long as this trend of using different offsets continue, or a more complex program which should be able to recognize the appropriate version and execute the proper algorithms and mappings according to the installed version of MSFS.

A version of MSFS usually has a life cycle of 3 years, before the new version hits the market. That has been the trend since FS95 (released in 1995). The new version is expected to reach the markets in the summer of 2006. There is no way to know in advance what Microsoft will offer in MSFS's future versions but a FDR for MSFS should be able to obtain information from the most used versions. According to AVSIM, more than 85% of Flight Simulation enthusiasts use FS9 (launched in 2003) and FS2002 (launched in 2001) [7]. The same percentage of users claimed they were using MSFS's latest version (FS2002) in December 2002 [8]

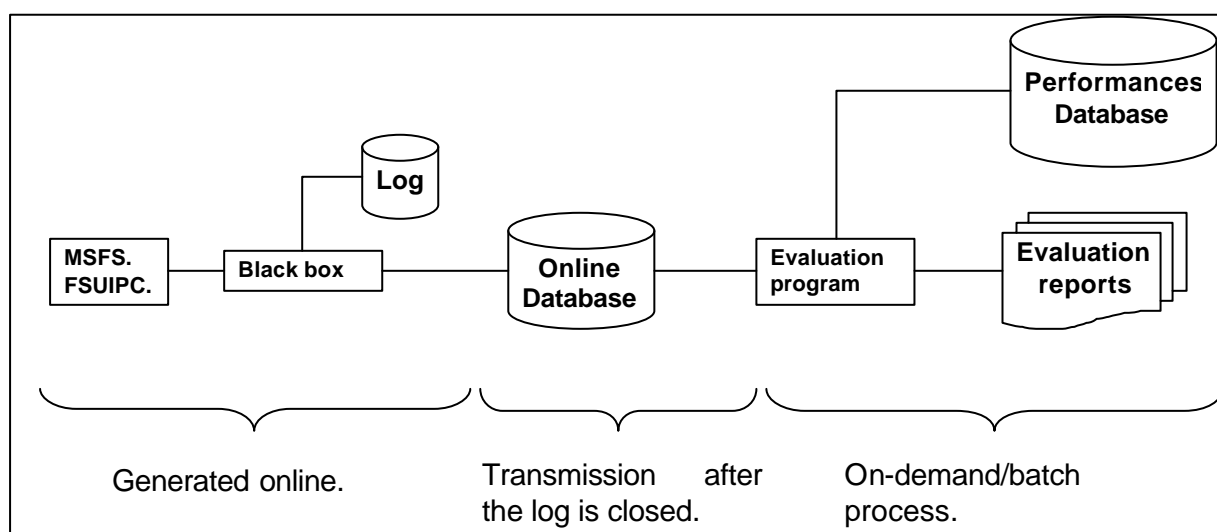
The second alternative is an internal function of MSFS called "Video Flight Recorder". This function is intended to save enough information to reproduce an entire flight (or part of it) later in the same computer or another one. The Software Developer Kits provided at Microsoft's website give a clear description of the file's format, however, the set of sampled variables is extremely limited and, as clearly

stated, “Flight Simulator 2004 provides no mechanism to extend, limit, or filter the data saved in a flight video recording.” [4, page 12]

Maybe the most popular utility among developers for the MSFS is Pete Dowson’s FSUIPC (Flight Simulator Universal Inter Process Communication). It’s an internal Dynamic Link Library (dll) for the MSFS. This program was developed initially for FS98 as a library allowing external programs to communicate with MSFS. Due to its popularity among add-on developers and the new offsets in FS2000 and subsequent versions, Mr. Dowson has updated its library to support the new MSFS versions and even other flight simulators by means of mappings from each simulator’s offset table, to a single table within FSUIPC. These mappings in FSUIPC allow developers to create a single application for use in several MSFS versions, given that the functionality of the application is supported in those versions of MSFS. In addition to this, FSUIPC offers access to information not available via Netpipes and, at the same time, gives a higher degree of control of FS. For example, one can read information from multiple joysticks or any other flight control device. It’s also possible to read weather information from MSFS. The documentation about the library is available for several programming languages and is much more detailed than the one available for Netpipes. Due to its flexibility and extensions, FSUIPC was selected as the method to obtain information from MSFS.

Let’s define log-file as the set of data generated by the black box and thereafter transmitted to the events database. A log-file will contain all the information sampled from MSFS through FSUIPC.

Even when today’s technologies allow decentralizing the analysis task, given the specialized kind of analysis that might be required, the project will be implemented in a website where all logs could be received electronically. Under this perspective, the information engineering problem deepens, since it’s not possible to assume that all users would be able to transmit the log-file over a broadband connection, therefore the file itself has to be kept as small as possible.



**Figure 2:** The black box program, made in Visual C++, reads the information from the MSFS via FSUIPC. The information is transferred to a log file. When the flight is finished, the log is saved. Then the software sends it to the internet to a predetermined online database. The evaluation program is a series of PHP scripts which evaluate the received logs and compare them to a performance database. According to the differences found, an evaluation report is generated for every single log.

Figure 2 shows in more detail the architecture of the system, as described to this point. The application described by this architecture will be called “Virtual Airline Trainer” (VAT).

## **4.2.     *The information engineering problem***

As stated above, the information can be used for personnel training. However, this task requires the information to be engineered. The main challenges are data storage management and information mapping. The sensors usually give the information in a proprietary format with some degree of precision which might not be the one needed. It is, therefore, necessary to map part of the data into a format which could be interpreted and read from the black box.

The first black box systems had storage space constraints due to the technology available. Their design also played a role, because it was focused more on the endurance of the equipment to the extreme conditions which could result from an accident. Even in modern black boxes, a data structure is required in order to maximize the data storage capabilities without compromising the data analysis capabilities.

As mentioned before, a normal FDR is expected to hold at least 165 Mb of information but, for the goals presented for this project, it is an unacceptable amount of information to be transmitted over the internet. Analyzing in detail the Appendixes B and M to FAR 121 [1, 5], it becomes obvious that all sampled data should be connected over a timeline. Since it is possible to assume that the sampled data are “shots” of information taken at a certain time, the data structure has to be based on time frames.

I define a frame as an atomic set of data taken from several variables within MSFS at exactly the same moment. It is possible to take some bytes from a frame, given that one knows the exact position where the needed information is located within the frame. FSUIPC allows this by using the methods Read and Process as described in FSUIPC’s SDK [6]. A proprietary file format is needed in order to define a structure of the information and ease the analysis process, while keeping the size of the file to the minimum possible.

The most critical factor determining the log’s size is the sampling frequency, since they are directly proportional. If the sampling rate is modified in flight, it might be possible to reduce the size of the file without compromising the detail in the information.

Two approaches were taken to achieve this. After a conversation with Flight Captains Jorge S. Holland, Carlos Salicrup and José Luis Ochoa, all experienced pilots each in different Mexican airlines, it was found that, at certain stages of a flight, its critical to monitor the variables at least at 1 Hz. There is also a non-critical stage, known as cruise flight, where the sampling rate could be reduced to 0,20 Hz. Analyzing a worst case scenario, an intercontinental flight with an ultra-long-haul jet would take up to 20 Hrs of flight, and 18 of those would belong to the cruise flight. If the sampling rate over this 18 hrs is 0,20 Hz, the minimum log-file size would be just 34 Mb (under the



same assumptions made on section 3, paragraph 5 of this report). Even when such a file is more manageable than the original 165 Mb, there might be unexpected events which might require switching back to a higher sampling frequency.

A second approach was called “compression by granularity”. Just as some graphic file standards, like JPEG, it could be possible to remove chunks of information which are of an unnecessary level of detail. We can take advantage of the fact that each block of information, or frame, corresponds to information taken from MSFS at the very same time. In certain stages of flight, it is expected that the information in several consecutive frames will be the same or at least will have negligible differences among frames. Whenever this is the case, a single frame suffices to analyze information. For example, if the plane is kept from frame 100 to frame 220 (about 2 minutes) lined up to the runway, without accelerating nor moving, frame 101 to 220 can be deleted from the file and the analysis tool would infer that nothing changed in the plane within those minutes (assuming a frame is taken every second). The only disadvantage of this kind of compression is that it cannot be made efficiently “on the fly” without demanding higher CPU resources, so this compression must be performed always before data transmission to the internet, but always after the flight is completed.

Both approaches are attractive enough to give one up, and, since both are complimentary, it was decided to follow both. A compression by granularity function was added to the requirements of the software, but it will be fully implemented at a later stage.

The structure of the log file is defined in the document called “VAT file description”, attached as an annex of this report. The latest file standard is version 0.6.2 and it considers the assumptions from the following subsections. It is expected that version 1.0 will be the one capable of reflecting some additional information like the relative position of take-off and touchdown from the Runway’s threshold, accuracy in flight related to a flight plan or certain navigation procedures, traffic alerts given by a Traffic Collision Avoidance System (TCAS) and other values which the panel of experts consider relevant and necessary, given that those variables are modelled in Flight Simulator and a method to evaluate them can be implemented.

The file standard is already much more efficient for the purposes of this project than the logs generated by FSUIPC or MSFS. As a way of proving it, a flight was made in MSFS version 2002 under the same circumstances (weather, route, airplane) from Frankfurt am Main to Rome Fiumicino. The flight was logged simultaneously using the FSUIPC’s log, the VAT file standard and the Video Flight Recorder option from MSFS. FSUIPC was configured only to log the basic set of variables, while the VAT File Program logged as data elements the same variables from the Video Flight Recorder (see the definition of Data Element in the VAT File Standard Description). The Video Flight Recorder generated a 22,5 Mb file, FSUIPC a 68,7 Mb file while the VAT log-file’s size was only 885 kb, without granularity compression.

I must make clear that this file standard is subject to change. Additional tests are required in order to benchmark this file standard, in longer flights or changing flight environments but, from the evidence available, the standard creates a file which is small enough without losing the information required for the program. The described test was intended to prove the VAT standard is much more efficient than the other 2

logging possibilities already available for MSFS. The following tests will consist of 3 flights in both FS2002 and FS2004, using 2 different airplanes (Phoenix Simulation's 747 in FS2002 and Eaglesoft's Citation X in FS2004), with duration ranging from 2 hours to 8 hours and with the granularity compression algorithm already implemented.

Even when the size problem is almost solved, the current data architecture seems to be not yet fully space-efficient. If the file would only save those variables which suffered a change since the last sample might be more space efficient and would require no granularity compression, however, MSFS requires lots of computing resources and there would be a trade-off between space required for the log file and MSFS's performance.

### **4.3.     *Variables selection***

Since the objective of the black box program is to analyze a pilot's performance and not to investigate accidents, the set of sampled variables may be different than the set specified in Appendix B and Appendix M to FAR 123 [2, 5]. In order to determine the set of variables which are of more relevance in a pilot's evaluation, a series of questionnaires were prepared and applied to a group of 7 real life pilots. Due to the time constraints, the sample of pilots was not randomly selected, however, this approach is not invalid, since they are considered as experts which will share the knowledge the system must acquire.

There were 3 questionnaires. The first one was intended to determine the level of expertise of the pilot based on their total logged hours, the kind of license they possess and whether they are certified pilot instructors. On the second questionnaire, an abstraction of the flight stages was proposed and the pilots had to provide feedback on whether this abstraction was useful for the purposes of the software. The last questionnaire was intended to obtain a list of representative variables which were useful for the program's objectives.

On the second questionnaire, the flight was divided in 3 stages. Since some variables are relevant only at certain phases of the flight, it makes sense group them and sample only those relevant. For example, it is only relevant to know the kind of surface the plane is standing when it's on the ground, or the vertical speed can be only calculated when the plane is airborne.

The third questionnaire was more focused into giving insight into the relevance of variables in the assessment of a pilot. The main question to be solved is which relation do variables have among them and how can they be used to evaluate a pilot. For example, one can monitor several parameters in a jet-engine, like the percentage of maximum allowed rpm's in the 1<sup>st</sup> and 2<sup>nd</sup> stage compressors (called N1 and N2, respectively), the Exhaust Gas Temperature (EGT), Vibration factor, Torque, Exhaust Power Ratio (EPR), but how to know which parameter is more relevant than other? What kind of relation exists between N1 and N2? Does a pilot have control over the vibration factor? If this value climbs above a certain threshold, what is expected from a pilot to be done? All those questions should be properly answered while modelling an evaluation method with the PHP scripts.

Several other questionnaires might be required. It is needed to determine a way to weight the importance of the events and the criteria for evaluating pilots. For example, it is known that a plane is designed to stand between -1 and 3 G-Forces, within this range, it is also clear that the pilot should try to keep the parameters as close as possible to 1 G. Can one determine ranges in variables like this which one can consider “normal”, “abnormal” and “dangerous”? Which variables are more relevant for safety? Are there some specific rules related to the lighting operation?

These additional questionnaires will be made at later stages of the project and the experts will be chosen more carefully, in order to have at least 9 pilots with Flight Instructor Certificate, 3 of them from the European Joint Aviation Authority, 3 from the FAA and the other 3 from another National Aviation Authority in Asia or Latin America.

The variables selected, based on the questionnaires and some informal communication with the experts as described at the beginning of this section, are described in annex 2 of this report.

#### **4.4. *Flight stages***

As described in section 4.3, one way to reduce the size of the VAT file is to reduce the sampling frequency. This can be done only where the data required for the analysis tool is not compromised. In the questionnaires, a model was proposed where a flight is divided in 3 stages: climb, cruise and descend. Only in the cruise stage, the sampling frequency is reduced to 0,20 Hz. By definition, the cruise stage starts when a plane reaches the cruise altitude, as stated in the flight plan unless a different cruise altitude is assigned by the Air Traffic Controller. However, MSFS doesn't simulate a flight dispatch office which could control the information related to a Flight Plan in such a way that it could be read from external programs.

A solution had to be found in order to use specific situations in a flight, which can be detected by the black box. A first proposal was to use an altitude threshold which should be high enough so the plane won't be flying arrival procedures and, at the same time, low enough so most of the flights cross that altitude. The so-called transition altitude, that is, the altitude where planes adjust their altimeter setting to “standard” conditions, usually at 18,000 ft, was proposed as threshold altitude to the pilots intervening in the questionnaires.

Almost no pilot agreed with this solution but a second one was found. The pilots proposed instead to find a way to determine when the plane has reached an altitude and stays for an arbitrarily fixed amount of time within 100 ft. of that altitude. Whenever the plane stays within that altitude, the Cruise Stage becomes active. Based on the proposal, it was defined a trigger: whenever a plane crosses the transition altitude (even when each country has its own transition altitudes, it was fixed to 19,000 ft), and maintains a vertical speed of at most 100 ft/min for 2 minutes, the Cruise Stage becomes active.

On the other hand, whenever the plane is in the Cruise Stage and the vertical speed becomes higher than 400 ft./min, the Climb or Descent Stage will become active depending on whether the current altitude is higher or lower than the latest read altitude.

In addition to those 3 stages, there are 2 more, namely: Pre Take Off Stage and Post Landing Stage. These stages are defined in the VAT-File Format Standard document.

## **5. The Virtual Airline Trainer Black Box program**

The black box component of the Virtual Airline Trainer is the only working component of the project up to this point. It can already read and manipulate the variables from Annex 2 and save them using the VAT File Standard described in Annex 1.

The program uses 2 classes to manage the information from MSFS. The first one is the CFSUIPC class, as provided by Peter Dowson's FSUIPC SDK. This class contains all necessary methods and functions in order to read the information from MSFS.

Basically, the methods Read and Process are used. The method Read maps an offset from MSFS to a variable, but no operation is committed. The Process method commits all Read and Write (namely, the method which allows writing information into MSFS by mapping a variable to an offset) assignments made since the last time the Process method was executed. More information about the CFSUIPC can be found in [6].

The second class and, by far, the most important in the program, is called Blackbox. This class manages all the black box operation, from the connection to FS, the operations with the CFSUIPC class, it also verifies all special incidences which can trigger a different stage. This class creates and manages the log-file, but the corresponding methods are not yet fully implemented.

## **6. Future of the project**

As described before, several components of the project have still to be developed and/or improved. Basically the black box component is the only working section of the project and it still requires some improvements. The program can, so far, create log files which are 100% compliant with the standard.

The compression functionality of the system has still to be developed and tested. It is expected to be developed after the first successful tests of the PHP scripts are carried out. So far, 5 scripts have been developed and successfully tested. Further 3 scripts are under development and once tested they will be integrated along with the other 5 in order to integrate a data extraction and analysis module.

With its current architecture, the system allows to implement testing scripts. That is, the system can be instructed to retrieve only a certain set of variables from the Flight Simulator and/or generate failures of the plane's systems (like in a real simulator). This wouldn't affect the compression algorithm because the atomicity principle of the frames. However, it is not considered to implement this functionality at this time.

## 7. Bibliography

- [1] Federal Aviation Administration, FEDERAL AVIATION REGULATIONS number 121, section 344, paragraph a, taken on February 26<sup>th</sup>, 2005 from [http://www.faa.gov/regulations\\_policies/faa\\_regulations/](http://www.faa.gov/regulations_policies/faa_regulations/)
- [2] Federal Aviation Administration, FEDERAL AVIATION REGULATIONS number 121, Appendix M, taken on February 26<sup>th</sup>, 2005 from [http://www.faa.gov/regulations\\_policies/faa\\_regulations/](http://www.faa.gov/regulations_policies/faa_regulations/)
- [3] Federal Aviation Administration, FEDERAL AVIATION REGULATIONS number 121, section 344, paragraph h, taken on February 26<sup>th</sup>, 2005 from [http://www.faa.gov/regulations\\_policies/faa\\_regulations/](http://www.faa.gov/regulations_policies/faa_regulations/)
- [4] Microsoft Corporation. NETPIPES: DATA INPUT / OUTPUT. From Microsoft's Flight Simulator 2004 Software Development's Kit taken on March 1<sup>st</sup>, 2005 from [http://www.microsoft.com/games/flightsimulator/fs2004\\_downloads\\_sdk.asp](http://www.microsoft.com/games/flightsimulator/fs2004_downloads_sdk.asp)
- [5] Federal Aviation Administration, FEDERAL AVIATION REGULATIONS number 121, Appendix B, taken on February 26<sup>th</sup>, 2005 from [http://www.faa.gov/regulations\\_policies/faa\\_regulations/](http://www.faa.gov/regulations_policies/faa_regulations/),
- [6] Peter Dowson. FSUIPC FOR PROGRAMMERS. From FSUIPC SDK 24<sup>th</sup> release. Taken on March 1<sup>st</sup>, 2005 from <http://www.schiratti.com/dowson.html>
- [7] AVSIM, THE 2004 FLIGHT SIM DEMOGRAPHIC SURVEY, taken on May 17<sup>th</sup>, 2005 from <http://www.avsim.com/nabopoll/result.php?surv=5>.
- [8] AVSIM, THE 2003 FLIGHT SIM DEMOGRAPHIC SURVEY, taken on May 17<sup>th</sup>, 2005 from <http://www.avsim.com/nabopoll/result.php?surv=5>.
- [9] National Transportation Safety Board. AIRCRAFT ACCIDENT REPORT. UNCONTROLLED DESCENT AND COLLISION WITH TERRAIN. USAIR FLIGHT 427. Executive summary, page ix. Report number PB99-910401
- [10] National Transportation Safety Board. AIRCRAFT ACCIDENT REPORT. UNCONTROLLED DESCENT AND COLLISION WITH TERRAIN. USAIR FLIGHT 427. page 297. Report number PB99-910401
- [11] Federal Aviation Administration, FEDERAL AVIATION REGULATIONS number 121, Appendix M, taken on February 26<sup>th</sup>, 2005 from [http://www.faa.gov/regulations\\_policies/faa\\_regulations/](http://www.faa.gov/regulations_policies/faa_regulations/)
- [12] National Transportation Safety Board. FACTUAL REPORT AVIATION. NTSB ID CHI97LA078, page 1.
- [13] Federal Aviation Administration, FEDERAL AVIATION REGULATIONS number 125, section 226, paragraph i, taken on May 26<sup>th</sup>, 2005 from [http://www.faa.gov/regulations\\_policies/faa\\_regulations/](http://www.faa.gov/regulations_policies/faa_regulations/)
- [14] Federal Aviation Administration, FEDERAL AVIATION REGULATIONS number 121, section 343, paragraph i, taken on May 26<sup>th</sup>, 2005 from [http://www.faa.gov/regulations\\_policies/faa\\_regulations/](http://www.faa.gov/regulations_policies/faa_regulations/)

## 8. Annex 1: VAT File Standard (version 0.6.3)

This document describes the structure of the Virtual Airline Trainer's files. These are created by the Virtual Airline Trainer (VAT) before being sent to the Virtual Airline's website.

There exist indeed 2 file-types which comply with the same standard. Those files with extension .vat are created "on-the-fly" by Virtual Airline Trainer. Once the flight is finished, VAT write some pending fields in the .vat file (which can be used to test whether the file was corrupted or is incomplete).

The second file type has an extension .vtc. This file is derived from the .vat file when the program executes the "compression" function. The compression only eliminates redounding data from the frames (frames of variables which have not changed its value). It also eliminates unnecessary records corresponding to the Cruise flight stage.

The .vat and .vtc files are binary documents which contain a file header, a pre-take-off data section, an in-flight data section, a post-landing data section and a file trailer. The structure looks like follows:

```
<VAT-File's Header>
  <Pre-Take-Off Header>
    <Pre-Take-Off Data Element 1>
    ...
    <Pre-Take-Off Data Element k>
    [<Engines-On Information Marker>]
    <Pre-Take-Off Data Element k + 1>
    ...
    <Pre-Take-Off Data Element n>
  <Pre-Take-Off Terminator>
  <On-Flight Section Header >
    <Climb-Phase Data Element 1>
    ...
    <Climb-Phase Data Element n>
    <CRuise Data Element 1>
    ...
    <CRuise Data Element n>
    <DeScent Data Element 1>
    ...
    <DeScent Data Element n>
  <On-Flight Section Terminator>
  <After Landing Section Header>
    <Post-Landing Data Element 1>
    ...
    <Post-Landing Data Element n>
  <After Landing Section Terminator>
<VAT-File's Trailer>
```

## **8.1. File Header.**

It's a 129-byte long section. It is written as soon as the user selects an option to start recording a new flight. It contains the basic information related to the Flight Simulator, FSUIPC, the Virtual Airline Trainer system (VAT) and the user's data. Those 32 bytes are composed of the following fields:

- A 4 character string. It identifies the VAT file and it is always "VATH".
- A 2 bytes hex-word containing the version of the VAT system. The version is codified as follows: FFFF. The leftmost hex-bit refers to the major version and the following positions refer to sub-versions. In this case, version 3.9.7.13 would be codified as 0x397D.
- A 4 character string. It identifies the FSUIPC field. It is always "UIPC". In case new software is provided as interface between FS and external applications, these 4 characters would identify such utility.
- A 4 containing the version of FSUIPC. It is coded the same way VAT system's version number.
- A 6-character string referring to the Flight Simulator and its version. Only "FS2002" and "FS2004" are supported.
- A 6-byte string containing to pilot's number. This information is taken from the AirlineT.ini file.
- A 4-byte hex-word timestamp from the moment the file was created. The number stored is the number of seconds elapsed since midnight of January 1<sup>st</sup>, 1970.
- A 4 byte string containing the letters "SPS0". This field validates that the following 4 bytes correspond to the beginning position of the Pre-Take-Off Header.
- A 4 byte word containing the byte-position in the file where the Pre-Take-Off Header starts.
- A 4 byte string containing the letters "SPS1". This field validates that the following 4 bytes correspond to the beginning position of the Pre-Take-Off Data section.
- A 4 byte word containing the byte-position in the file where the Pre-Take-Off Data section starts.
- A 4 byte string containing the letters "SPS2". This field validates that the following 4 bytes correspond to the beginning position of the Pre-Take-Off Section Terminator.
- A 4 byte word containing the byte-position in the file where the Pre-Take-Off Section Terminator starts.
- A 4 byte string containing the letters "SPS3". This field validates that the following 4 bytes correspond to the beginning position of the On Flight Section Header.
- A 4 byte word containing the byte-position in the file where the On Flight Section Header starts.
- A 4 byte string containing the letters "SPS4". This field validates that the following 4 bytes correspond to the beginning position of On Flight Data Section.
- A 4 byte word containing the byte-position in the file where the On Flight Data section starts.

- A 4 byte string containing the letters “SPS5”. This field validates that the following 4 bytes correspond to the beginning position of the On Flight Section Terminator.
- A 4 byte word containing the byte-position in the file where the On Flight Section Terminator starts.
- A 4 byte string containing the letters “SPS6”. This field validates that the following 4 bytes correspond to the beginning position of the Landing Section Header
- A 4 byte word containing the byte-position in the file where the Landing Section Header starts.
- A 4 byte string containing the letters “SPS7”. This field validates that the following 4 bytes correspond to the beginning position of the Landing Data section.
- A 4 byte word containing the byte-position in the file where the Landing Data section starts.
- A 4 byte string containing the letters “SPS8”. This field validates that the following 4 bytes correspond to the beginning position of the Landing Section Terminator
- A 4 byte word containing the byte-position in the file where the Landing Section Terminator starts.
- A 4 byte string containing the letters “SPS9”. This field validates that the following 4 bytes correspond to the beginning position of the File Trailer.
- A 4 byte word containing the byte-position in the file where the File Trailer starts.
- A 6 byte string containing the letters “TVATFS” separating the section containing the position of each file’s section from the rest of the File Header.
- A 4 byte word containing the VAT file’s size.
- A 3-byte word containing the total number of frames recorded. This field is written once the flight has been completed and the document will be closed. The total amount of frames is given by the sum of all frames with identifiers “PTDE”, “CPDE”, “CRDE”, “DSDE” and “LDGS” (see following sections).
- A 5-character string terminator. It is always “VATHT” indicating “Virtual Airline Trainer Header Terminator”.

## **8.2.     *Pre-Take-Off Header***

This area is written until the application verifies that the Flight Simulator (FS) is ready to receive information (view offsets 0x05DC, 0x0628, 0x0760,, 0x3364, 0x3365 and 0x11D4). Once one of the offsets indicates that FS is ready, this header is written. It contains the following fields:

- A 4-character string indicating the Pre-Take-Off Header starts. It is always “PTOH”.
- A 4-character string indicating the ICAO of the departing airport. This will remain as “XXXX” for the time being. FSUIPC provides no method to read the current airport’s ICAO-code, therefore a method to determine the location will be implemented in the future.



- A 256-char string containing the path of the Flight Simulator installation from offset 0x3E00. Whenever the string is shorter, it is filled with 0s.
- A 256-char string containing the pathname of the current AIR file, excluding the FS main path. Taken from offset 0x3C00. Whenever the string is shorter, it is filled with 0s.
- A 256-char string containing the name of the current aircraft (from the “title” parameter in the aircraft.cfg file). Taken from offset 0x3D00.
- A 24-char string containing the ATC Aircraft type as declared in aircraft.cfg. It’s taken from offset 0x3160. Whenever the string is shorter, it is filled with 0s.
- A 24-char string containing the ATC Airline name as declared in aircraft.cfg. It’s taken from offset 0x3148. Whenever the string is shorter, it is filled with 0s.
- A 12-char string containing the ATC Aircraft identifier (tail number) as declared in aircraft.cfg. It’s taken from offset 0x313C. Whenever the string is shorter, it is filled with 0s.
- A 12-char string containing the ATC flight number as declared in aircraft.cfg. It’s taken from offset 0x3130. Whenever the string is shorter, it is filled with 0s.
- A single byte, containing the percentage of realism set for FS plus whether the auto rudder option is activated. Since the realism is given in percentage, 128 will be added to the percentage read from FS (offset 0x0372) in case the auto rudder option is activated (offset 0x0278).
- A single byte indicating whether the crash detection is enabled (offset 0x0830).
- A single byte hex-word containing the number and type of engines of the plane. The left character corresponds to the number of engines (offset 0x0AEC) and the right character corresponds to the type of engine (offset 0x0609).
- A 2-byte number containing the mask of the most important characteristics of the plane. The bit mask is given by the following order: [0]-Has autopilot, [1]-Has flaps, [2]-Has Stall alarm, [3]-Has Mixture, [4]-Has Spoilers, [5]-Has Strobes, [6]-Has NAV1, [7]-Has NAV2, [8]-Is Tail dragger, [9]-Has Carb Heat control, [10]-Has toe breaks, [11]-Has prop pitch system. The offsets are 0x0764, 0x0778, 0x077C, 0x0780, 0x078C, 0x0794, 0x07A0, 0x07A4, 0x0790, 0x0784, 0x079C and 0x0AF0 respectively.
- A 3-byte word containing the total number of frames recorded in this stage. This field is written once the phase has been. The amount of frames is given by the sum of all frames with identifier “PTDE” (see following sections).
- A 5-char string used as section terminator. It is always “PTOHT”

**NOTE:** *The 3 fields written on italics will be only used in the development versions of the software.*

### **8.3. Pre-Take-Off Terminator.**

The terminator for the Pre-Take-Off section will be a 16-byte string containing the following fields:

- A 4-byte string used as section-terminator identifier. It will be always “PTOT”.

- A 3-byte word containing the number of frames recorded in the pre-take-off section.
- A byte indicating the number of bounces detected on take-off. A bounce occurs when the plane is airborne for a few moments and touches the ground again.
- A 3-byte reserved word. Its value will be set to 0.
- A 5-byte string terminator. It will be always "PTOTT".

## **8.4. On-Flight Section Header**

This section is written whenever the value of the offset 0x0366 changes from 1 to 0, that is, whenever the Flight Simulator detects the plane becomes airborne. The standard for this file establishes that, if the offset 0x0366 changes again to 1 after the plane became airborne, the "On-flight" section is closed. However, it could be possible that the plane gets airborne and, due to several reasons, it could touch the ground instants later. In order to avoid such a scenario, the program will have a "protection" mode, such that it will close the "On-flight" section only if the "on-the-ground" offset is set to 1 at any moment after 20 seconds after take-off.

The header contains the weather, weight and fuel information on take-off. It won't consider a full METAR report in order to save space. This header consists of the following fields:

- A 4-byte string used as header identifier. It is always "OFSH".
- A 4-byte word indicating the ICAO code of the nearest reporting weather station. In case the system is connected to FS2002, the string will be set to "GLOB".
- A 2-byte word indicating the wind direction as taken from offset 0x0EF2.
- A byte indicating the wind speed.
- A byte indicating the visibility at the current position.
- A byte indicating the presence of wind gusts.
- A byte to indicate the turbulence at the surface.
- A byte indicating the current temperature.
- A byte indicating the dew point temperature.
- A byte indicating the precipitation rate and type. The lower hex part indicates the rate (0-5) and the highest indicates the type (0-2).
- A 2-byte word indicating the barometric pressure (in milibars multiplied by 100 in order to eliminate the decimal point). MSFS uses milibars as standard measure and converting it into hectopascals might reduce precision.
- A 2-byte word indicating the Indicated airspeed of the plane.
- A 2-byte word indicating the heading of the plane.
- A 2-byte word indicating the Altimeter setting for the plane.
- A 3-byte word indicating the Fuel Weight of the plane.
- A 3-byte word indicating the Zero-Fuel Weight of the plane.
- A 8-byte word as timestamp. It is the value given by offset 0x0310.
- A 5 byte terminator for the header. It is always "OFSHT"

## **8.5.     *On-Flight Section Terminator***

The terminator for the On-Flight section will be a 16-byte string. It is written in the same operation where the Landing Section Header is created. It contains the following fields:

- A 4-byte string used as section-terminator identifier. It will be always “OFST”.
- A 3-byte word containing the number of frames recorded in the On-Flight section.
- A 4-byte reserved word. Its value will be set to 0.
- A 5-byte string terminator. It will be always “OFSTT”.

## **8.6.     *Landing Section Header***

This section is written whenever the value of the offset 0x0366 changes from 0 to 1, that is, whenever the Flight Simulator detects the plane lands. It is important to state that this section won't start unless two conditions are fulfilled: the plane was airborne and at least 20 seconds after that event, the offset 0x0366 was 1.

The header contains the weather, weight and fuel information on take-off. It won't consider a full METAR report in order to save space. This header consists of the following fields:

- A 4-byte string used as header identifier. It is always “ALSH”.
- A 4-byte word indicating the ICAO code of the nearest reporting weather station. In case the system is connected to FS2002, the string will be set to “GLOB”.
- A 2-byte word indicating the wind direction as taken from offset 0x0EF2.
- A byte indicating the wind speed.
- A byte indicating the visibility at the current position.
- A byte indicating the presence of wind gusts.
- A byte to indicate the turbulence at the surface.
- A byte indicating the current temperature.
- A byte indicating the dew point temperature.
- A byte indicating the precipitation rate and type. The lower hex part indicates the rate (0-5) and the highest indicates the type (0-2).
- A 2-byte word indicating the barometric pressure (in Mb).
- A 2-byte word indicating the Indicated airspeed of the plane.
- A 2-byte word indicating the heading of the plane.
- A 2-byte word indicating the Altimeter setting for the plane.
- A 3-byte word indicating the Fuel Weight of the plane.
- A byte indicating the surface where the plane landed, according to offset 0x31E8.
- A byte indicating the surface condition where the plane landed, according to offset 0x31EC.
- A 2-byte word indicating the vertical speed of the plane at the moment of landing.

- An 8-byte word as timestamp. It is the value given by offset 0x0310.
- A 5 byte terminator for the header. It is always “ALSHT”

### **8.7.     *Landing Section Terminator***

The terminator for the On-Flight section will be a 17-byte string containing the following fields:

- A 4-byte string used as section-terminator identifier. It will be always “ALST”.
- A 3-byte word containing the number of frames recorded in the After Landing section.
- A byte containing the number of bounces at landing.
- A 4-byte reserved word. Its value will be set to 0.
- A 5-byte string terminator. It will be always “ALSTT”.

### **8.8.     *File Trailer***

Once all engines are shut down, the landing section is closed, and this section is recorded. It contains the control information from the file. The fields in the File Trailer are:

- A 4-char string indicating the file trailer. It will be always “VATT”.
- An 8-character string indicating the ICAO of the departing airport and landing airport. This will remain as “XXXXXXXX” for the time being. FSUIPC provides no method to read the current airport’s ICAO-code, therefore a method to determine the location will be implemented in the future.
- A 3-byte word indicating the number of frames written in the file. The number should be the same contained in the file header and the last data element recorded.
- A 4-byte timestamp, stating the moment the trailer was written.
- A 5-byte file terminator. It will be always “VATTT”.

### **8.9.     *Data Elements***

Between the Pre-Take-Off Header and the After Landing Terminator, there are several frames of information, which contain all the information recorded by the program. The frames are classified in 5 groups, according to the stage of flight when they occurred. The length of each frame is variable, but the structure is always the same. After 13 bytes of frame header, blocks of 5 bytes containing information are added.

<b>Stage of flight</b>	<b>Identity string</b>	<b>Block size</b>	<b>Activation</b>
Inactive status	INAC	4	
Pre-Take-Off Data Element	PTDE	26	These frames are always at the beginning of the file. In case the program starts recording when the plane is already airborne, there is at least one frame of this kind.
Climb Data Elements	CLDE	41	These frames appear once the plane becomes airborne.
Cruise Data Elements	CRDE	41	Once the plane remains 5 minutes at the same altitude, +/- 300 feet. A few seconds protection applies before switching to the "Descent mode" or back to "Climb mode".
Descent Data Elements	DSDE	41	Once the plane initiates a descent which lasts at least 2 minutes. In case the plane stabilizes at a lower Flight Level for at least 5 minutes, the cruise stage becomes active again.
After-Landing Data Elements	ALDE	25	When the offset 0x0366 becomes positive again.

The frame itself looks as follows:

- A 4-byte string containing the element describer (see table above).
- A 3-byte word indicating the consecutive frame number.
- A 6-byte word, containing the timestamp at the time the variable was written.
- A block of n-bytes, according to the "block-size" column in the above table.
- N-series of 32-bytes, where n is the number of engine the plane has.

It is important to mention that there is a set of variables to be read and analyzed per each of the above described groups. For example, it is irrelevant, for training issues, the altitude, vertical speed, mach speed or bank and pitch levels when the plane is on the ground. Those variables are read every second except for the cruise stage, when they are read every 5 seconds

## 9. Annex 2: Variables sampled from MSFS

Nr.	Stage	Variable name in the program	Data Type in MSFS	Offset	Byte length in MSFS	Data Type in VAT standard	Byte length in VAT standard	Units	Values / Formula applied obtain the real value.
0	SPECIAL	ADDR_SLEW_MODE	SmallInt	0x05DC	2	SmallInt	1	Bool	0 = False, 1 = True
1	SPECIAL	ADDR_CRASHED	SmallInt	0x0840	2	SmallInt	1	Bool	0 = False, 1 = True
2	SPECIAL	ADDR_PAUSE_FLAG	SmallInt	0x0264	2	SmallInt	1	Bool	0 = False, 1 = True
3	ALL	ADDR_CONTROL_TIMER_1_FS2002_SEE_0368	Float64	0x0310	8	Double	8	Seconds	
4	ALL	ADDR_LIGHTS_FS2000	SmallInt	0x0D0C	2	Unsigned Short	2	Bitwise	From bit-0: Nav, Bcn Land, Taxi, Strobe, Panel, Recognition, Wing, Logo, Cabin
5	ALL	ADDR_GFORCE	SmallInt	0x11BA	2	Float	4	G-Forces	#/625
6	ALL	ADDR_SIMULATION_RATE	SmallInt	0x0C1A	2	Float	4	Scalar	#/256
7	ALL	ADDR_TURN_COORDINATOR_BALL	ShortInt	0x036E	1	Char	1	Scalar	-128 = Full Left, 0 = Centered, +127= Full right
8	ALL	ADDR_AP_YAW_DAMPER	LongWord	0x0808	4	Char	1	Bool	0 = False, 1 = True
9	ALL	ADDR_INDICATED_AIR_SPEED	LongInt	0x02BC	4	Float	4	Knots	#/128
10	PTDE, ALDE	ADDR_PANEL_AUTOBRAKE_SWITCH	Byte	0x2F80	1	Unsigned Char	1	Set	0=RTO, 1=off, 2=brake1, 3=brake2, 4=brake3, 5=max
11	PTDE, ALDE	ADDR_SURFACE_TYPE_FS2002	LongWord	0x31E8	4	Unsigned Char	1	Set	25 different types of surfaces given by a number index.
12	PTDE, ALDE	ADDR_SURFACE_CONDITION_FS2002	LongWord	0x31EC	4	Unsigned Char	1	Set	From 0: Normal, Wet lcy, Snow on a non-snow surface
13	PTDE, ALDE	ADDR_GROUND_ELEVATION	LongInt	0x0020	4	Float	4	Feet	#*3.28084/256
14	PTDE, ALDE	ADDR_GROUND_SPEED	LongInt	0x02B4	4	Int	2	Knots	#*3600/65536/1852
15	PTDE	ADDR_PUSHBACK_STATE_FS2002	LongWord	0x31F0	4	Unsigned Char	1	Set	3=off, 0=pushing back, 1=pushing back, tail to left, 2=pushing back, tail to right

Nr.	Stage	Variable name in the program	Data Type in MSFS	Offset	Byte length in MSFS	Data Type in VAT standard	Byte length in VAT standard	Units	Values / Formula applied obtain the real value.
16	CLDE, CRDE, DSDE	ADDR_STALL_WARNING	Byte	0x036C	1	Unsigned Char	1	Bool	0 = False, 1 = True
17	CLDE, CRDE, DSDE	ADDR_OVERSPEED_WARNING	Byte	0x036D	1	Unsigned Char	1	Bool	0 = False, 1 = True
18	CLDE, CRDE, DSDE	ADDR_ALTIMETER_SETTING_MB	Word	0x0330	2	ShortInt	2	mmHg * 100	#*2.953/16
19	CLDE, CRDE, DSDE	ADDR_PRESSURE_QNH	Word	0x0EC6	2	ShortInt	2	mmHg * 100	#*2.953/16
20	CLDE, CRDE, DSDE	ADDR_ALT_HI	LongInt	0x0574	4	Float	4	Feet	#*3.28084
21	CLDE, CRDE, DSDE	ADDR_PITCH	LongInt	0x0578	4	ShortInt	2	Degrees	#*360/(65536*65536 up<0, dn>0, 0 = level)
22	CLDE, CRDE, DSDE	ADDR_BANK	LongInt	0x057C	4	ShortInt	2	Degrees	#*360/(65536*65536 Left>0, right<0)
23	CLDE, CRDE, DSDE	ADDR_HEADING	LongWord	0x0580	4	Unsigned ShortInt	2	Degrees	#*360/(65536*65536)
24	CLDE, CRDE, DSDE	ADDR_AP_MASTER_SWITCH	LongWord	0x07BC	4	Unsigned Char	1	Bool	0 = False, 1 = True
25	CLDE, CRDE, DSDE	ADDR_VERTICAL_SPEED	SmallInt	0x0842	2	ShortInt	2	ft/min	#*-3.28084, upwards<0
26	CLDE, CRDE, DSDE	ADDR_SPOILER_ARM	LongInt	0x0BCC	4	Unsigned Char	1	Bool	0 = False, 1 = True
27	CLDE, CRDE, DSDE	ADDR_FLAPS_CONTROL	LongInt	0x0BDC	4	Unsigned Char	1	Scalar	0=Retracted, 16383=fully extended. (inc = 16383/(no. of position-1))
28	CLDE, CRDE, DSDE	ADDR_GEAR_COMMANDED	LongInt	0x0BE8	4	Unsigned Char	1	Scalar	0=Gear up, 16383=Gear down
29	CLDE, CRDE, DSDE	ADDR_AOA_ANGLE_OF_ATTACK	SmallInt	0x11BE	2	Unsigned Char	1	% of a-max	100-(100*#/32767)
30	CLDE, CRDE, DSDE	ADDR_MACH_SPEED	Word	0x11C6	2	Unsigned ShortInt	2	Mach * 100	#/204,80
31	ALL	ADDR_PROP_RPM	Float64	Several	8	Int	4	RPM	#
32	ALL	ADDR_ENG_N1	SmallInt	Several	2	Int	4	Percent	#*100/16384

Nr.	Stage	Variable name in the program	Data Type in MSFS	Offset	Byte length in MSFS	Data Type in VAT standard	Byte length in VAT standard	Units	Values / Formula applied obtain the real value.
33	ALL	ADDR_ENG_N2	SmallInt	Several	2	Int	4	Percent	#*100/16384
34	ALL	ADDR_ENG_PRESSURE_RATIO_EPR	SmallInt	Several	2	Float	4	Scalar	#*1.6/16384
35	ALL	ADDR_ENG_EGT	SmallInt	Several	2	Float	4	Celsius deg	#*860/16384
36	ALL	ADDR_ENG_FUEL_FLOW_PPH	Float64	Several	8	Int	4	lb/h	#
37	ALL	ADDR_TURB_ENG_ITT	Float64	Several	8	Float	4	Celsius deg	#
38	ALL	ADDR_ENG_COMBUSTION_FLAG	SmallInt	Several	2	Int	4	Bool	0 = Engine off, 1 = Engine On

Note: the Variables 31 to 38 are sampled from 4 different MSFS variables. Each one contains the information relevant to 1 engine. Therefore, if the airplane has 3 engines, each of variables 31 to 38 are sampled 3 times, each for each engine. It is important to remark that the values sampled represent indeed independent values for each engine.



## 10. Annex 3: Questionnaires used to determine the relevant parameters needed to evaluate a pilot.

The following questionnaires were applied to several commercial pilot license holders issued by both the FAA and the Mexico's DGAC (Mexican Civil Aviation Authority). Those individuals listed in section 4.2 of this report were among those who answered. The questionnaires were meant to be only a reference for the purposes of this report. More detailed questionnaires and a more exhaustive methodology will be used in the upcoming days in order to document not only the relevant variables, but also the basic evaluation's criteria the system should implement, as well as the weight of each evaluation's parameter.

### 10.1. Questionnaire 1

This questionnaire tries to establish background of the expert and it will be used in order to weigh his/her answers accordingly. The expertise reflected in the system will be the weighted average (whenever it is statistically possible) from the answers in the other questionnaires. The answers given in this questionnaire will be strictly confidential and won't appear in the final report of the project.

1. Do you possess/possessed an airplane pilot's license?

Yes

No

2. How many hours approximately do you have in your flight log?

Under 50    50 – 200

200 – 500

500 – 1000

1000 – 5000

Over 5000

3. How would you distribute (in percentage) the amount of hours you have flown?

Single engine piston:

\_\_\_\_\_

Multi-engine piston:

\_\_\_\_\_

Turboprop:

\_\_\_\_\_

Jet :

\_\_\_\_\_

100 %

4. Which of the following capacities do you possess? (Mark with an X to the left of the answer)

[    ]    Instrument rating

[    ]    Multi-engine rating

[    ]    Commercial pilot

[    ]    Airline Transport Pilot (ATP)

[    ]    Flight Instructor

5. In case you are Flight Instructor, in which of the capacities listed above have you been training pilots over the past 24 months? In case you gave another kind of flight instruction, please give details.

---

- Yes No

- Yes No

When the MSFS determines that the plane is not airborne anymore, the Post-Landing phase starts. From this moment on, the same variables from the Pre-Take Off stage are monitored. The Post-landing finishes if the plane becomes airborne again (aborted landing or touch and go) or the engines are shut-down. Another rule comes to play here: if the plane lands, it shouldn't take off after 45 seconds. The flight is considered over once the plane shuts down its engines.

- Yes No

No

- Yes      No      Proposed event:

No

- Yes      No      Proposed event: \_\_\_\_\_

No

- Yes No

No

- Yes ☐ No ☐

No

### 10.3. Questionnaire 3

This questionnaire tries to determine which variables available in the Flight Data Recorder proposed for this study are relevant to evaluate a pilot. Some evaluation criteria which might be relevant but cannot be obtained from a FDR should not be considered.

It is already known that a pilot should be able to follow certain tracks (like flying a straight line between 2 nav aids, or following a STAR). The system will not evaluate that at the moment (the algorithms required for that are quite complex and a large database with all nav aids, airways and fixes is also required).

At this point it is only relevant to determine which variables should be analyzed. The pilot is expected to follow certain procedures (setting parking breaks and turning on the beacon before turning on the engines, for example). These procedures should be independent of the airplane flown. Even when a 747 will have more procedures than a Cessna 150, some principles remain unchanged, like setting flaps on take-off, limiting the bank angle, not

exceeding certain g-forces, not over speeding, etc. The expected values and tolerances for each variable might be airplane dependant and, therefore, won't be considered at this point.

It is already clear that some variables are relevant only at certain moments of the flight, so they will be analyzed only at some specific moments. For example, the altimeter setting becomes relevant at the moment the airplane becomes airborne. The wind direction and speed are relevant when the pilot chooses the wrong runway. By calculating the airplane's weight it is possible to determine whether the pilot tried to take-off above the designed maximum weights. The first 2 questions of these questionnaire deal with those situations.

1. At the moment of take-off and landing some variables will be read and registered. In order to determine their relevance during take off or landing, give them a value from 1 to 5, where 1 is unimportant and 5 is extremely important.

Indicated airspeed		Vertical speed		Heading	
Surface type		Surface Condition		Altimeter setting	
Latest METAR		Zero fuel weight		Fuel weight	
Yaw damper		Lights configuration		Flaps position	
Trim position		Aileron trim position		Rudder trim position	

2. Do you think some variables are missing from the previous list? If so, list them and describe the relevance they have.

Yes                      No.              If you answered YES, detail the variables in the following lines:

---



---

3. Let's focus only in the "Pre-Take off" and "Post-Landing" stages. Lights settings, weather related and engine data should not be considered at this point. Use the values from 1 to 5 as you did in question 1.

Parking breaks		Autospoilers-arm		Prop / Engine de-ice	
Pitot heat		Flaps		Structural de-ice	
Ground speed		G-Forces		Surface type (grass, asphalt, etc).	
Turn coordinator ball		Spoilers		Surface condition	
Trim position		Aileron trim position		Rudder trim position	
Carb Heat		Autobreak			

4. Now let's consider the "Climb" and "Descend" stage. The values have to be set in the same way as in the previous question. Use the values from 1 to 5 as you did in question 1.

Altitude		Autospoilers-arm		Prop / Engine de-ice	
Pitot heat		Flaps		Structural de-ice	
Indicated Airspeed		G-Forces		Surface type (grass, asphalt, etc).	
Turn coordinator ball		Spoilers		Surface condition	
Trim position		Aileron trim position		Rudder trim position	
Vertical speed		Pitch		Bank	
Carb Heat		Radar altimeter		Heading	
Autobreak		Autopilot (switch and mode)		Flight director	

5. Now let's consider the "Cruise" stage. The values have to be set in the same way as in the previous question. Use the values from 1 to 5 as you did in question 1.

Altitude		Autospoilers-arm		Prop / Engine de-ice	
Pitot heat		Flaps		Structural de-ice	
Indicated Airspeed		G-Forces		Surface type (grass, asphalt, etc).	
Turn coordinator ball		Spoilers		Surface condition	
Trim position		Aileron trim position		Rudder trim position	
Vertical speed		Pitch		Bank	
Carb Heat		Radar altimeter		Heading	
Autobreak		Autopilot (switch and mode)		Flight director	

6. There are also engine-related variables which the system can monitor. We are aware that the standard parameters differ between planes, but the variables to be analyzed are basically the same for pistons, turboprops or jets. From the following list make a classification of the variables which could be relevant. Give values from 1 to 5, where 5 is very important and 1 irrelevant.

<i>Piston</i>		<i>Turboprops</i>		<i>Jets</i>	
RPM		N1		N1	
Manifold		N2		N2	
Magnetos		Torque		EPR	
		EPR		EGT	
		ITT		ITT	

7. The frequency used to read the variables is also very important. All variables are read at the same time, so every "shot" or set of records at a certain moment reflects the state of the plane. By taking a series of "shots" it is possible to recreate the entire flight for its analysis. It becomes clear that the more shots taken per second, the more detailed information we get, but the space and memory restrictions make such an approach prohibitive. What time intervals do you consider adequate for taking "shots"?

Every ¼ of a second.      Every ½ second      Every second      Every 2 seconds.

8. As detailed before, on the "Cruise" stage it might not be appropriate to take shots with the same frequency. Which relation between the shots per second in this stage versus those in all other stages you think is more convenient? Think only as an instructor.

1:1      1:2      1:4      1:5      1:10  
(Cruise : Other stages)

9. In the following table, mark with the proper letter the lights which should be already on at the time the event described in the first row occurs. Mark with an "X" those which are required by ICAO conventions or universally accepted standards. Use a "C" when the requirement is given by your operator's standards. An "N" indicates whenever it is a national/regional regulator's requirement (FAA, JAL, DGAC, etc.). See the example in next page.

<b>Light system as given by MSFS</b>	<b>Engine start</b>	<b>Taxi start</b>	<b>Take-off run</b>	<b>Pos. rate of climb</b>	<b>10,000 ft are reached</b>	<b>Trans. altitude is reached on climb</b>
Beacon						
Navigation						
Landing						
Taxi						
Strobes						
Wing lights						
Recognition						
Logo						
Runway turn-off						

<b>Light system as given by MSFS</b>	<b>Trans. altitude is reached on desc.</b>	<b>10,000 ft. are reached</b>	<b>RWY in sight / ILS active</b>	<b>Final approach</b>	<b>Landing</b>	<b>Taxi to platform.</b>
Beacon						
Navigation						
Landing						
Taxi						
Strobes						
Wing lights						
Recognition						
Logo						
Runway turn-off						

*Example:*

<b>Light system as given by MSFS</b>	<b>Trans. altitude is reached on desc.</b>	<b>10,000 ft. are reached</b>	<b>RWY in sight / ILS active</b>	<b>Final approach</b>	<b>Landing</b>	<b>Taxi to platform</b>
Light 1		X	X	X	X	
Light 2		C	N	X	X	

*This can be interpreted as follows: Light 1 must be on all the time between 10,000 ft. and when the plane leaves the active runway while the company you flight for requires Light 2 to be on once 10,000 ft are reached, even when the national regulator requires it until the plane has the runway in sight. The universal standard or ICAO requires this light to be on once one is in short final and until the plane leaves the active runway.*