Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TÍTOL DEL TFG: CPDLC digital communication implementation between an ATC and RPAS

TITULACIÓ: Grau en Enginyeria d'Aeronavegació

AUTOR: Joel Herrero Montolio

DIRECTOR: Marc Pérez-Batlle

DATA: 11 de maig de 2015

Title: CPDLC digital communication implementation between an ATC and RPAS

Author: Joel Herrero Montolio

Director: Marc Pérez-Batlle

Date: May 11th, 2015

Overview

This project has been developed with the principal aim of simulating CPDLC(Controller-Pilot Data Link Communications) between ATC (Air Traffic Controller) and RPAS (Remote Piloted Aircraft Systems). Communication between ATC and conventional aircraft is possible too.

The uprising number of users in airspace compels the aerial communications to reinvent themselves for not collapsing the assigned frequency band.
As a solution it is proposed CPDLC communications, an application that allows the direct exchange of text-based messages between a controller and a pilot.

The most direct benefits in utilization of CPDLC are:

1) Air traffic management is more efficient.
2) Increase of airspace capacity.
3) Communication, surveillance and route conformance monitoring improvement.
4) Voice channel congestion reduction.

The CPDLC code developed is integrated with ATM(Air Traffic Management) simulation program “eDEP”, provided by Eurocontrol and it is also integrated with Icarus group research simulations.

The project is divided in two main parts:

a) Encoding library messages. It has been done the effort of collecting all the possible messages that can be exchanged between ATC and pilot. Messages have been encoded to byte level and it has been prepared to be sent through the net.
b) Program graphic development. It has been developed a friendly and understandable graphical environment to be able to interact with a computer for sending the messages in a correct way.

As a long term objective, it is expected that CPDLC program can be evaluated by real air traffic controllers in ATM simulations where RPAS and conventional aviation is mixed in non-segregated airspace. TFG will be useful for extracting conclusions about benefits provided by CPDLC and to know the real workload that CPDLC system provides to the air traffic controller.

Títol: Implementació de comunicacions digitals tipus CPDLC entre un RPAS i ATC.

Autor: Joel Herrero Montolio

Director: Marc Pérez-Batlle

Data: 11 de maig de 2015

Resum

Aquest treball s'ha desenvolupat amb l'objectiu principal de poder simular comunicacions tipus CPDLC (Controller-Pilot Data Link Communications) entre ATC (Air Traffic Controller) i RPAS (Remote Piloted Aircraft Systems). També és possible la comunicació entre ATC i avions convencionals.

El creixent número d'usuaris al espai aeri obliga a les comunicacions aèries a reinventar-se per tal de no saturar la banda de freqüències assignada.
Com a solució es proposen les comunicacions tipus CPDLC, una aplicació data link que permet l'intercanvi directe de missatges basats en text.

Els beneficis més directes en la utilització del CPDLC són els següents:

1) Direcció del tràfic aeri més eficient.
2) Increment de la capacitat del espai aeri.
3) Millora de comunicació, vigilància i seguiment de l'aeronau.
4) Reducció de la congestió del canal de veu.

El codi CPDLC desenvolupat esta integrat amb el programa de simulació ATM(Air Traffic Management) "eDEP", proveït per Eurocontrol i amb les simulacions a temps real portades pel grup d'investigació ICARUS.

El projecte esta dividit en dues parts importants:

a) Codificació de la biblioteca de missatges. S'ha fet l'esforç de recopilar tots els missatges possibles que poden ser intercanviats entre ATC i pilot. Els missatges han sigut codificats a nivell de byte i s'han preparat per ser enviats a través de la xarxa.
b) Desenvolupament gràfic del programa. S'ha preparat un ambient gràfic amigable i comprensible per poder interactuar amb la màquina per tal de poder enviar els missatges correctament.

Com a objectiu final a llarg termini, s'espera que el programa CPDLC pugui ser avaluat per controladors reals en simulacions ATM on es barregen RPAS i aviació convencional en espai aeri no segregat. El TFG servirà per treure conclusions en quant als beneficis aportats pel CPDLC i carga de treball real que aporta al controlador aeri.

CONTENTS

List of figures

List of tables

Chapter 1. Introduction

1.1. Background

This work is about how CPDLC technology impacts as a new communication method between ATC and pilots.

To put us in context we will start by defining briefly what CPDLC is, why it is necessary and which are its main advantages over the current radio communication system. We will also see its current situation and how future is expected for CPDLC in civil aviation. References [1] and [2] have been used for developing this subsection.

CPDLC stands for control pilot data link communications. Hence, CPDLC is a data link communication that allows the direct exchange of text-based messages between controller and pilot. CPDLC was introduced in the nineties to complement the traditional voice communication medium.

The need of a new communication system comes from aeronautic radio channel congestion. There are more and more flights every day and radio voice channel it is beginning to become a little place for that huge number of users.

CPDLC adds a great number of benefits to the air traffic system such as:

- Enhanced safety, it offers an unambiguous communication channel with no risk of misunderstanding, since crews and air traffic controllers can actually read the messages.
- Additional communication channel, it offers a second independent communication channel, reducing the strain on busy sector frequencies. Voice communication will remain available in case of emergency or anomalies.
- Improves communication capabilities in oceanic areas, especially in situations where controllers and pilots have previously had to rely on a Third Party HF communications.
- Allows the flight crew to print messages, if an onboard printer is available.
- Allows the auto load of specific uplink messages into the Flight Management System (FMS), reducing crew-input errors.
- Allows the crew to downlink a complex route clearance request, which the controller can re-send when approved without having to type a long string of coordinates.
- Specific uplink messages arm the FMS to automatically downlink a report when an event, such as crossing a waypoint, occurs. This automation assists with workload management for the flight crew and the controller.
- Specific downlink messages, and the response to some uplink messages will automatically update the Flight Data Record in some ground systems.

Figure 1.1: Datalink communication

Currently, CPDLC is being tested and developed by important organizations such as Eurocontrol or the federal aviation administration (FAA).

In Eurocontrol case, the Maastricht Upper Area Control Centre (MUAC) has been pioneering the use of CPDLC for over a decade, and in 2012 close to 105,000 logons by some 77 different airlines were recorded, exchanging an average of 670 messages with MUAC every day. The proportion of flights resorting to CPDLC has been regularly increasing in recent years.

In February 2014, a second Initial 4D (I-4D) flight trial was successfully conducted exchanging trajectories between air and ground.
The I-4D procedure uses the next-generation ATN data link standard to clear a lateral flight path, vertical maneuvers as well as a time constraint and receive an updated plan from the aircraft's flight management system. The standard used was the subject of a joint EUROCAE/RTCA development, the initial version (known as ATN Baseline 2) of which was published in March 2014.

Boarding CPDLC future issue, MUAC's commitment to developing data link and providing an ever increasing service to the controllers and pilots testifies to a firm belief in the advantages of this technology and the benefits that can be gained. MUAC will continue to be involved in SESAR2020 validations on the use of the downlinked Extended Projected Profile (EPP) in order to further improve ground based tools with this airborne data.

1.2. Motivation

Since aviation controlling is known, all communications between controller and pilot has been done via radio voice. Air traffic is growing more and more every year and available voice communication frequencies are becoming congested.
Controller-Pilot data link communication (CPDLC) offers an alternative communication.

It is known that Eurocontrol and Federal Aviation Administration (FAA) are testing CPDLC in civil aviation for being implanted in the years to come. Although, is CPDLC useful for controlling civil unmanned aerial system (UAS) in a non-segregated airspace?

Let us take an example. A civil UAS mission has been approved to monitor Catalonian coast. Depending on altitude and flight plan segment being flown, UAS can intercede in civil airways. Controller has to contact the pilot who is controlling the UAS to modify its flight path in order to avoid a crash.
In other way, UAS may contact sector controller in order to report its intentions for achieving its mission. Would be possible this communication using CPDLC? Would CPDLC help controller? Or would it increase controller workload?

Main motivation, in this bachelor's final degree work, is to make a reliable report capable of answering key questions about CPDLC integration in a traffic controlling environment. Report will be based in simulation results using a CPDLC application developed from zero. It will be used in Icarus simulations where real air traffic controllers are involved.
Once application has been totally developed and accepted by the users; it will be a really useful tool to get important study data of CPDLC communication.

Figure 1.2: Global Hawk UAS

1.3. Objectives

In this final degree work, objectives are split in three parts.

Information gathering, understanding and adaptation. In this first part of the work main objectives are:

- Reading and understanding of global operational data link communications [3] (GOLD) document published by ICAO.
- CPDLC message list elaboration from GOLD document.
- Thinking of a strategy to integrate CPDLC application into Icarus simulations environment.

CPDLC application developing. Secondly, CPDLC application development basically pursues next objectives:

- Developing an optimal code capable to prepare a CPDLC message to be sent through the net.
- Developing an optimal code capable to receive and understand CPLDC messages sent into the net.
- Developing an understandable human-machine interface (HMI).

CPDLC application testing. Finally, CPDLC testing is meant to follow next objectives:

- CPDLC application HMI user feedback.
- CPDLC application improving.
- CPDLC application bugs search.
- Study data extraction from Icarus simulations where can be discussed issues like controller workload, controller adaptation to the new communication tool, etc.

Collecting all information from several simulations, it can be done a reliable report which studies CPDLC impact in ATC-UAS communication in a non-segregated airspace as explained in section 1.2.

Chapter 2. Understanding CPDLC

2.1. Global Operational Data Link Document (GOLD)

Global data operational data link document (GOLD) is a document done by ICAO.

The GOLD addresses data link service provision, operator readiness, controller and flight crew procedures, performance-based specifications and post-implementation monitoring and analysis.
The GOLD also provides guidance and information concerning data link operations and is intended to facilitate the uniform application of Standards and Recommended Practices. GOLD material is intended to improve safety and maximize operational benefits by promoting seamless and interoperable data link operations throughout the world.

As CPDLC is a data link communication, before starting application design, it was studied the GOLD document for leaning how CPDLC really works in a real aeronautic communication.

This subsection pretends to summarize, in a really compacted way, how CPDLC connection management is performed.

2.1.1. Active and inactive CPDLC connections

An active CPDLC connection allows an Air Traffic Service Unit (ATSU) and the aircraft to exchange CPDLC messages. The ATSU with which an aircraft has an active CPDLC connection is referred to as the Current Data Authority (CDA).

An inactive CPDLC connection does not allow the exchange of messages between the ATSU and the aircraft. This connection is referred as the next data authority (NDA).

2.1.2. Establishing CPDLC connection

The ATSU can only initiate a CPDLC connection request after successfully correlating an aircraft with the associated flight plan.
The ATSU initiates a CPDLC connection by sending a CPDLC connection request.

The aircraft system:

a) Accepts the connection request
b) Establishes this CPDLC connection as the active connection
c) Responds with a CPDLC connection confirm

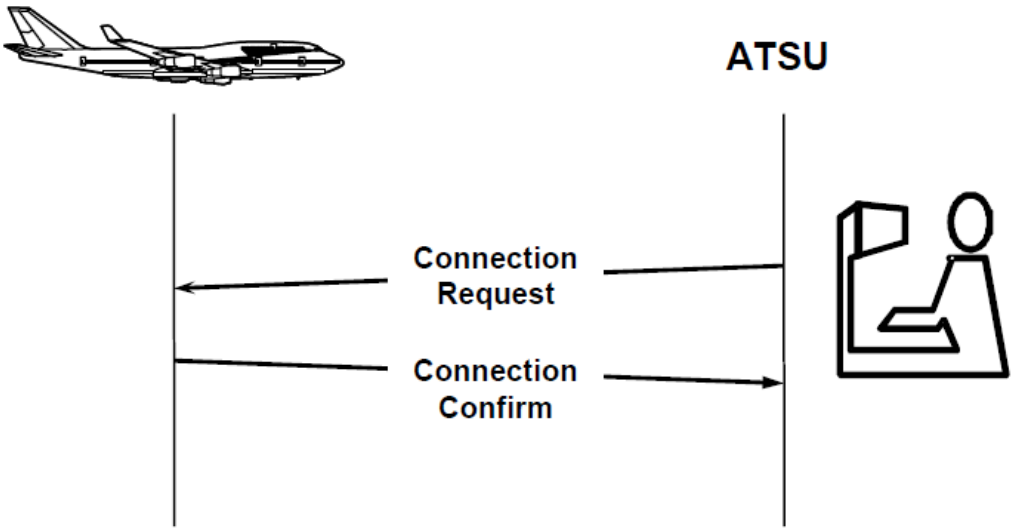


Figure 2.1: CPDLC connection sequence

If there is an existing CPDLC connection when CPDLC connection request is received, the aircraft system verifies that the ATSU sending the connection request is the next data authority. The aircraft system:

a) Accepts the CPDLC connection request
b) Establishes the connection
c) Responds with a CPDLC connection confirm

Otherwise, the aircraft system rejects the CPDLC connection request by sending a connection rejection message.

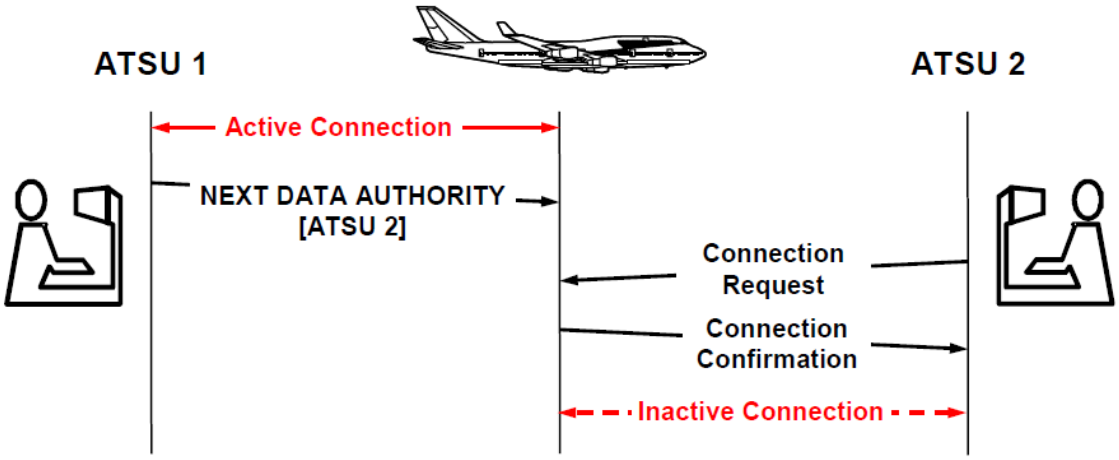


Figure 2.2: Successful attempt to establish a CPDLC connection (inactive)

2.1.3. Terminating CPDLC connection

The CDA initiates the termination of the CPDLC connection by sending a termination request message.

On receipt of a termination request message the aircraft system will downlink a CPDLC termination confirmation message. The aircraft system will consider the aircraft to be disconnected as soon as the termination confirmation message has been sent.

On receipt of a termination request message containing a CONTACT or MONITOR message element, the aircraft system will:

a) Display the message contained in the termination request message for flight crew
b) If the flight crew responds DM 0 WILCO, send a CPDLC message DM 0 and then consider the aircraft to be disconnected
c) In case the flight crew sends a DM 1 UNABLE response to the message, the aircraft maintains the CPDLC connection with the CDA.

If the next data authority attempts to uplink a termination request message to the aircraft, the aircraft system will maintain the inactive CPDLC connection and send a termination rejection message including DM 63 NOT CURRENT DATA AUTHORITY.

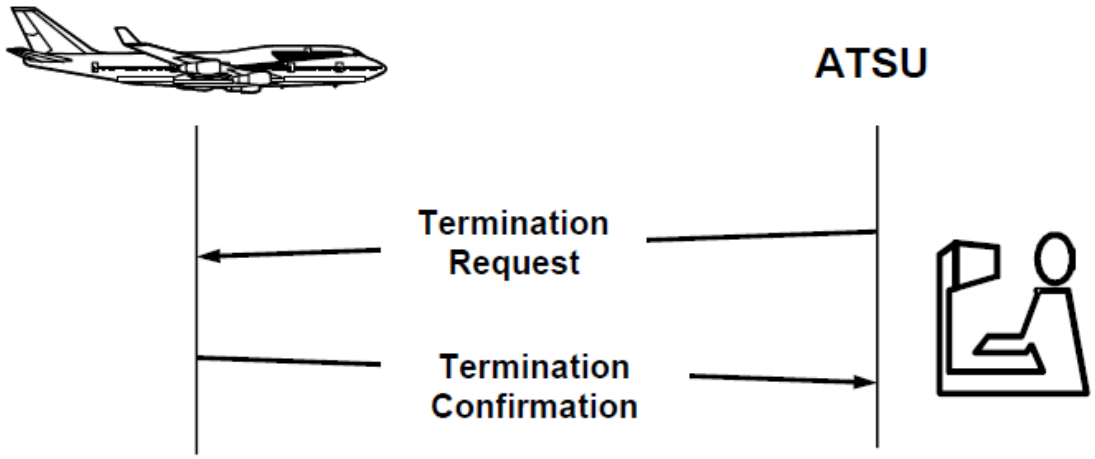


Figure 2.3: CPDLC connection termination

2.1.4. Transferring CPDLC connection

The CDA will initiate a CPDLC transfer to an adjacent ATSU as the aircraft transits from the current ATSU to another CPDLC-capable ATSU. These transfers are normally automatic, without flight crew action.

The CDA performs the following steps to transfer a CPDLC connection to the next ATSU:

a) Sends a NDA message to notify the aircraft of the identity of the next ATSU permitted to establish a CPDLC connection.
b) Initiates address forwarding with the next ATSU.
c) Sends a CPDLC termination request message when the aircraft is near the boundary with the next ATSU.

Only the CDA can specify the NDA by including the four-character ICAO identifier.

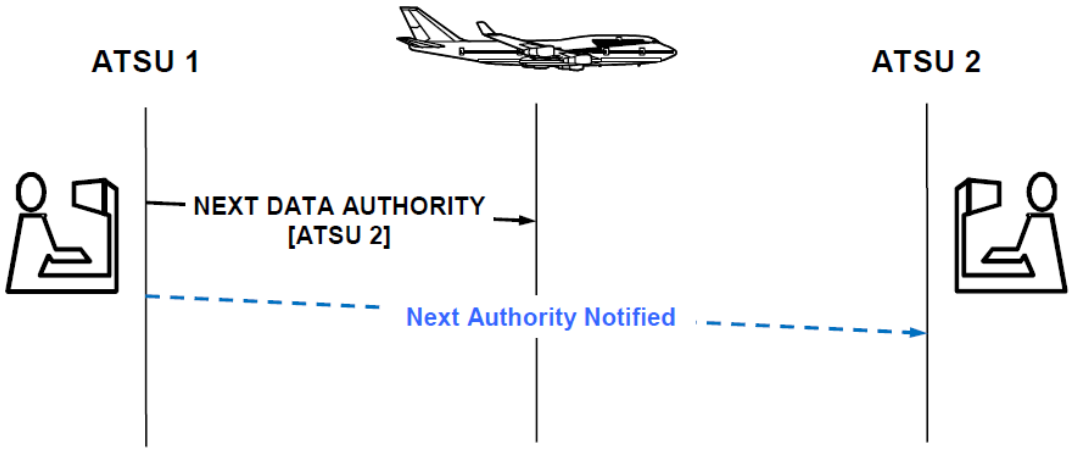


Figure 2.4: Next data authority notification

When the active CPDLC connection is terminated, the NDA becomes the CDA and is now able to exchange CPDLC messages with the aircraft.

ATSU 1 may use the connection forwarding message to provide notification to the ATSU 2 for advising that that ATSU 1 has terminated CPDLC connection.

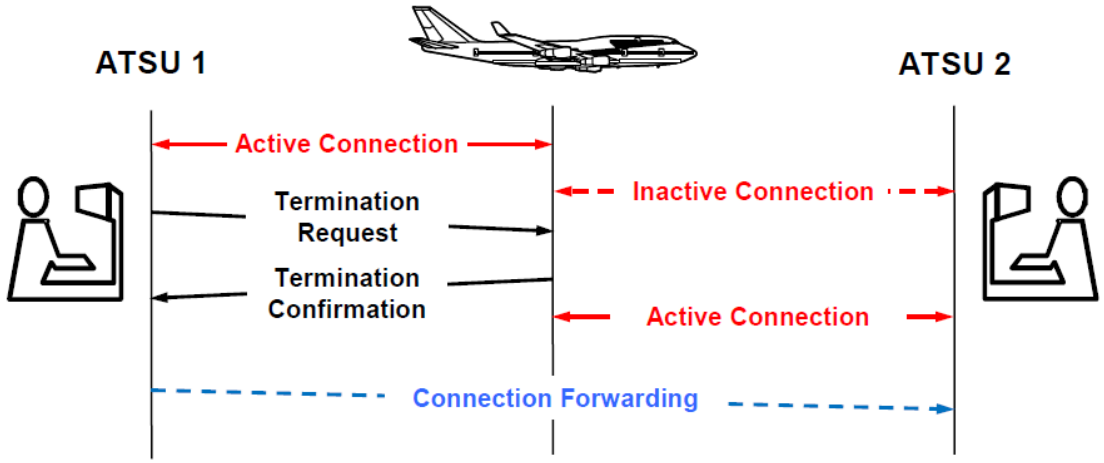


Figure 2.5: Connection forwarding

If the aircraft is entering to an airspace where data link services are not provided, no NDA message is sent, nor is the address forwarding process performed.
When the active CPDLC connection is terminated, the aircraft will no longer have CPDLC connection.

2.1.5. CPDLC connection sequence

As the aircraft transits from one CPDLC-capable ATSU to another, the same CPDLC transfer process is repeated. The algorithm is showed in the next figure:

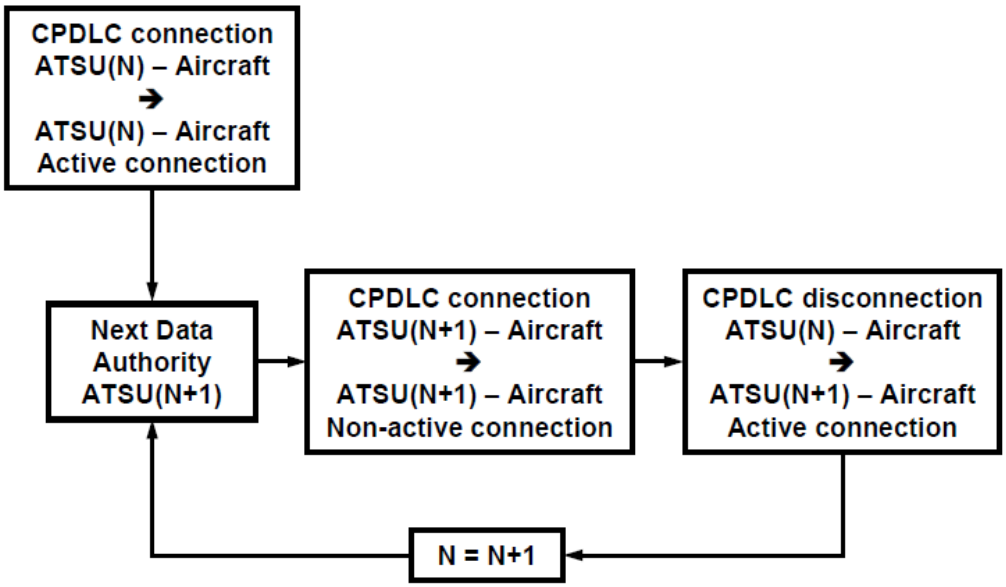


Figure 2.6: Life cycle of the CPDLC connection process

The sequence of messages from the logon request to the completion of the CPDLC transfer when using air-ground address forwarding is showed in the next figure:

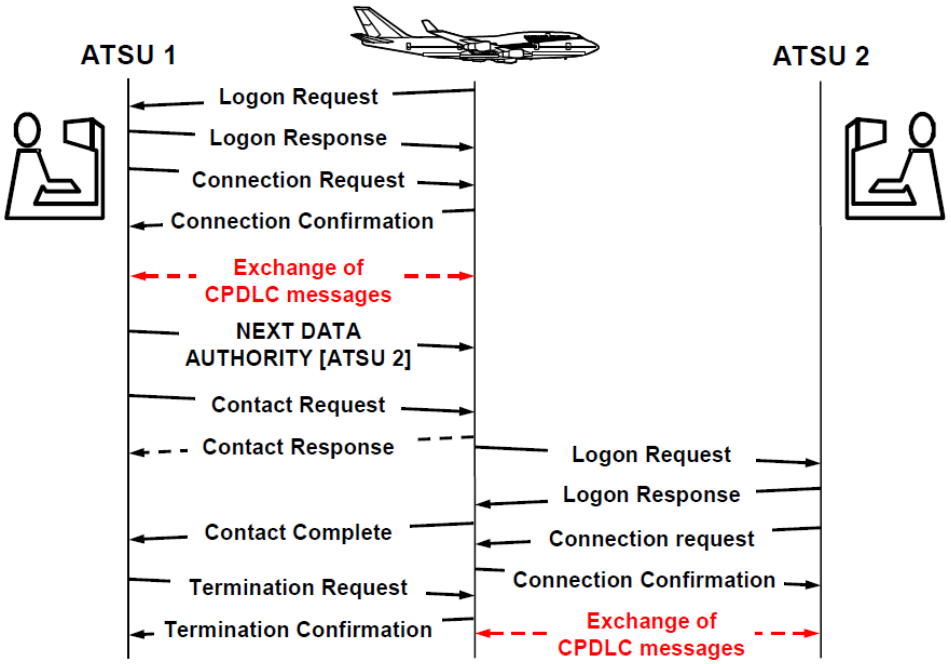


Figure 2.7: Nominal sequence for initial CPDLC connection establishment and transfer of CPDLC connection using air-ground address forwarding

The sequence of messages from the logon request to the completion of the CPDLC transfer when using ground-ground address forwarding is showed in the next figure:

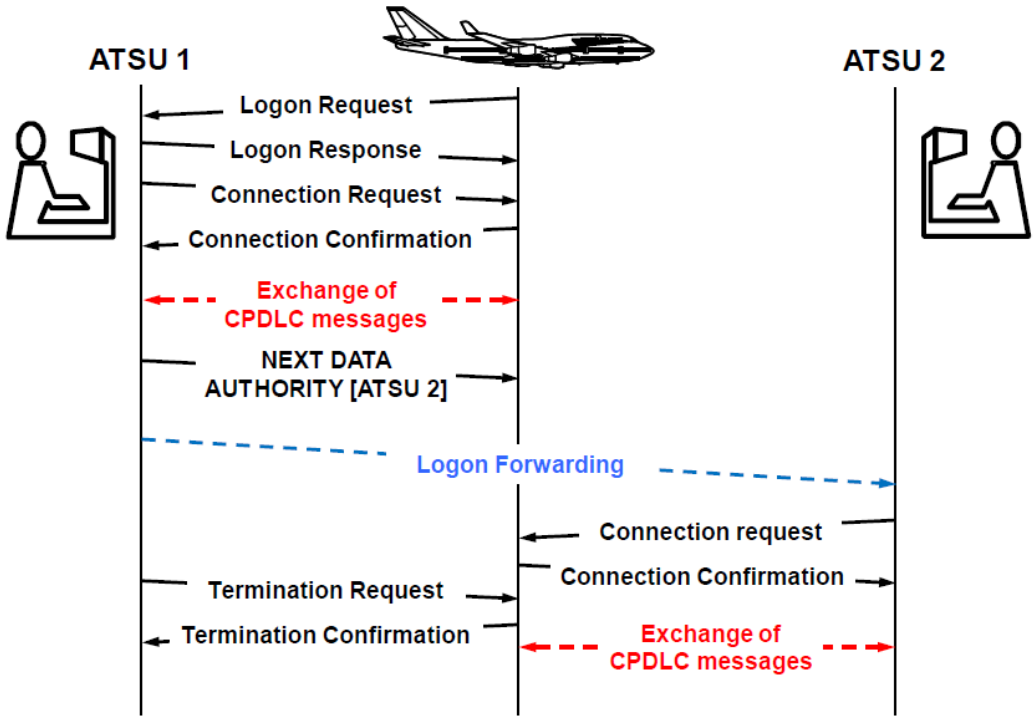


Figure 2.8: Nominal sequence for initial CPDLC connection establishment and transfer of CPDLC connection using ground-ground address forwarding.

2.1.6. CPDLC message set

The CPDLC message set consists of a set of message elements, most of which correspond to a radiotelephony phraseology.

CPDLC message elements are referred as:

a) Uplinks (message elements that are sent to an aircraft).
b) Downlinks (message elements that are sent by the aircraft).

Each message element has a number of attributes associated to it, including:

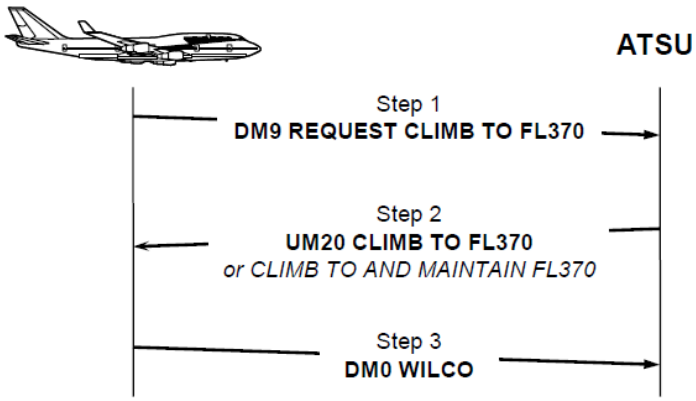
a) A message number that uniquely identifies each type of message element.
b) A response attribute that defines whether or not a response is required for a message element.

2.1.7. CPDLC dialogues

Messages that are related constitute a CPDLC dialogue.

A CPDLC dialogue is open if any of the CPDLC messages in the dialogue are open.
A CPDLC dialogue is closed if all CPDLC messages in the dialogue are closed.

Figure 2.9 provides an example of the individual message and dialogue status for a CPDLC request and clearance exchange.



Step	CPDLC Message	Response Attribute	UM Status	DM Status	Dialogue Status
1	Downlink request	Y	N/A	Open	Open
2	Uplink response	W/U	Open	Closed	Open
3	Downlink response	N	Closed	Closed	Closed

Figure 2.9: Message/dialogue status for CPDLC request and clearance exchange.

2.2. CPDLC simulations adaptation

In section 2.1 we have seen how CPDLC connection management works nowadays. However, in this bachelor's final degree work we will try to adapt connection management in a simpler way, since the work pursues to study CPDLC tool vs the current communication tool. We are not trying to perform a 100% realistic scenario. We just need to perform a realistic CPDLC app in order to see if CPDLC tool works in simulations and it becomes useful.

2.2.1. Current ICARUS group simulation concept

Nowadays, Icarus group has its own simulation scenario to test and get data about UAS integration in a non-segregated airspace.

The simulation has always three parts involved:

a) ATC user: Real air traffic controllers come to play this role. Lately, a survey it is done to know their stress level in each part of simulation and to know which their workload was approximately in every moment.
b) Pseudopilot user: Icarus team members play this role. A pseudopilot is understood as a user that controls all civil aircrafts flying the same sector. Thus, communication is done one to one. The same ATC will be communicated always with the same pseudopilot since aircraft crosses the sector border.
c) UAS user: Icarus team members play this role. The UAS has a different mission each time that a simulation is done. Mission intercedes with airways being used by civil aircraft.

This way, ATC must take into account civil aircrafts and UAS mission at the same time and try to avoid collisions and make traffic smooth in an optimal way.

Nowadays, communication between ATC-Pseudopilot and ATC-UAS is done simulating radiofrequency using TeamSpeak program. TeamSpeak is set depending on the role being played in each computer.
CPDLC app will substitute TeamSpeak program.

2.2.2. Main CPDLC adaptations

In order to establish a solid base to start, it was decided to adapt CPDLC app to current needs of Icarus simulations. It was also thought that, this way, CPDLC app users can evolve with the application. Different versions can be introduced along the time, each version more faithful to GOLD[3] document. But, for this work, following points have not been taken into account as GOLD says:

- Establishing CPDLC connection.
- Terminating CPDLC connection.
- Transferring CPDLC connection.

In GOLD, it exists a single message, a way to proceed and a complex casuistic of what to do in each fact presented above.

These three points will not be as explained in sections 2.1.2 to 2.1.4 because in this version of CPDLC app we will not correlate all aircrafts with their flight plan, thus we will not be able to know which next authority data is for each aircraft. Besides, the fact of being a simulation based on a pseudopilot makes difficult to think of a simple interface for pseudopilot in order to have under control all messages related with connection management.

In the current version, connection management is done by comparing current position sent by aircraft ADS-B with sector border limits. This way we can compare and demonstrate if CPDLC communication belongs to one sector or to another. It has to be taken into account that for establishing, transferring or terminating connection, it will not be needed a specific sequence of events and will not exist messages related with connection management since it is done automatically under code.

Chapter 3. CPDLC application development

3.1. CPDLC application architecture

3.1.1. CPDLC architecture diagram

In this subsection will be explained how CPDLC application works through the net in order to exchange information between different CPDLC users.

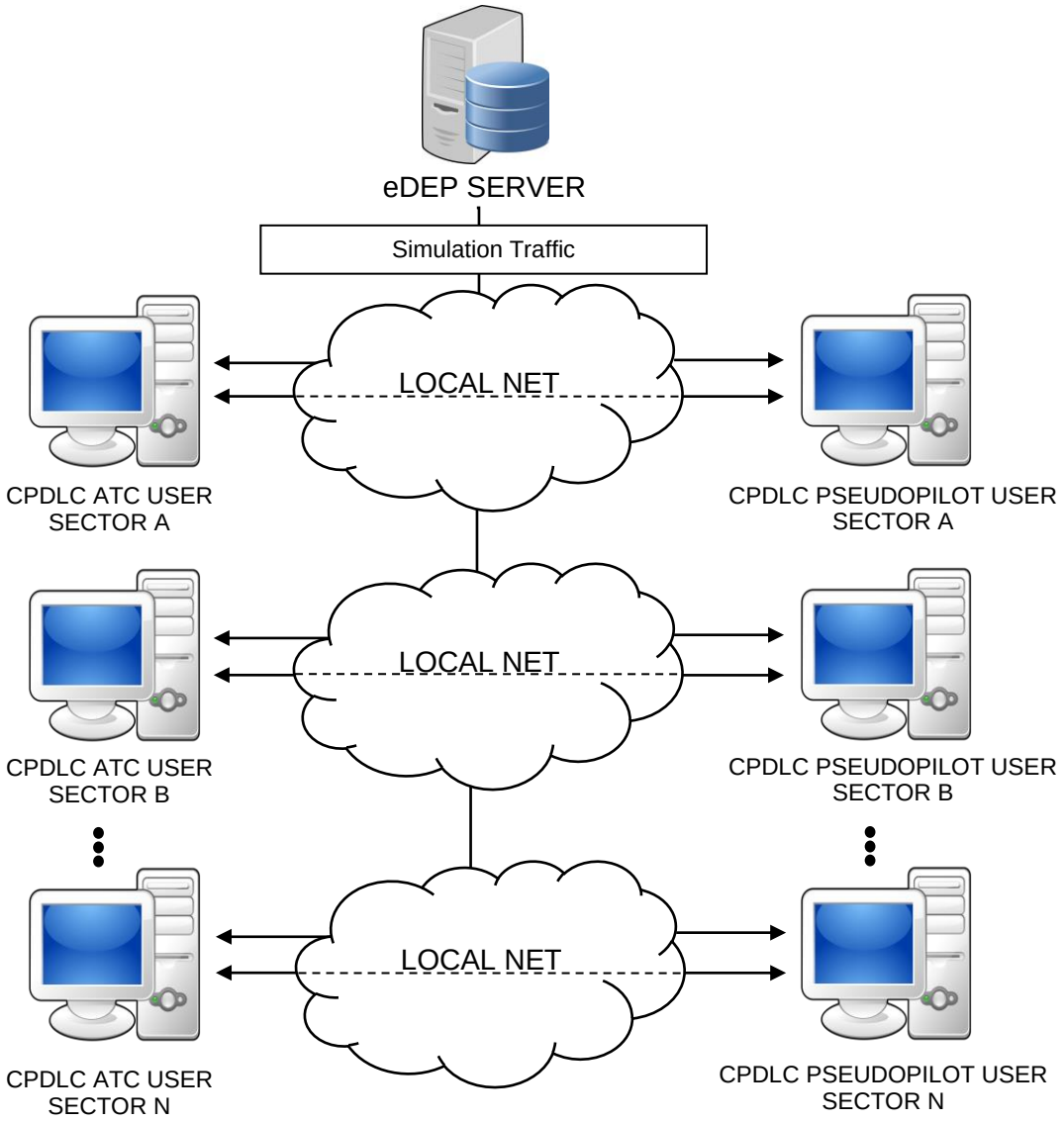


Figure 3.1: CPDLC architecture diagram

As seen in figure 3.1, it exists two kind of connections for CPDLC communications. Both are configured to go through local net and both are user data protocol (UDP) connections.

UDP connection is a protocol in transport level. This protocol allows sending data without a previous handshaking, header has enough addressing information. Nor have confirmation or flow control, so packets can preempt each other; nor is it known whether it has arrived properly, as there is no confirmation of delivery or receipt. UDP is suitable for purposes where error checking and correction is either not necessary or is performed in the application, avoiding the overhead of such processing at the network interface level.

In following subsections we will see the objective of each connection and what information is sent.

3.1.2. CPDLC application – eDEP communication

The purpose is to obtain traffic information by sector from eDEP simulation for being displayed in CPDLC application.

Connection is one to many, in other words, eDEP will be throwing traffic information into the local net. Then, CPDLC application is able to get this information and select the proper traffic depending on the sector selected.

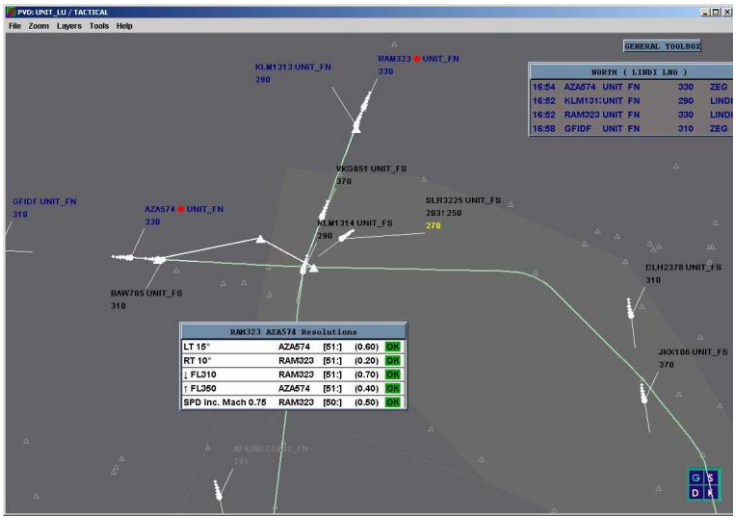


Figure 3.2: eDEP simulation example

3.1.3. CPDLC ATC user – CPDLC pseudopilot user communication

The purpose is to achieve the communication between CPDLC users by sending CPDLC messages through the application.

Connection is one to one. An ATC is always connected to the same pseudopilot and both must be controlling, in ATC case, or flying, in pseudopilot case, the same sector. It is done this way in order to follow ICARUS simulations philosophy (see subsection 2.2.1.).
As local net connection is one to one, a previous agreement must be done in order to set up connections for each CPDLC application (IP, port numbers…).
CPDLC users cannot communicate CPDLC messages to other sectors.

3.2. CPDLC messages organization

3.2.1. Objective

The vast majority of messages were extracted from GOLD [3]. However, the organization provided by GOLD document was too generalist and it did not organize the messages in an optimal way.

The principal aim in this part of the project was to find an optimal way to arrange all the possible messages that could be used in a conversation between ATC – Pilot (UpLink) or Pilot – ATC (DownLink).

Once all the messages were arranged in a proper way, it allowed a faster way to recognize messages. In other words, you are able to navigate in a better way to the message that you are looking for.

As a result, it has been won a better perspective with all messages packaged in a determinate group instead of having all of them stacked in a huge list.

3.2.2. Organizational diagram

Next figures show how messages are arranged in CPDLC application. Messages were split in UpLinks and DownLinks. Following we will find both organizations in figure 3.3. and figure 3.4.

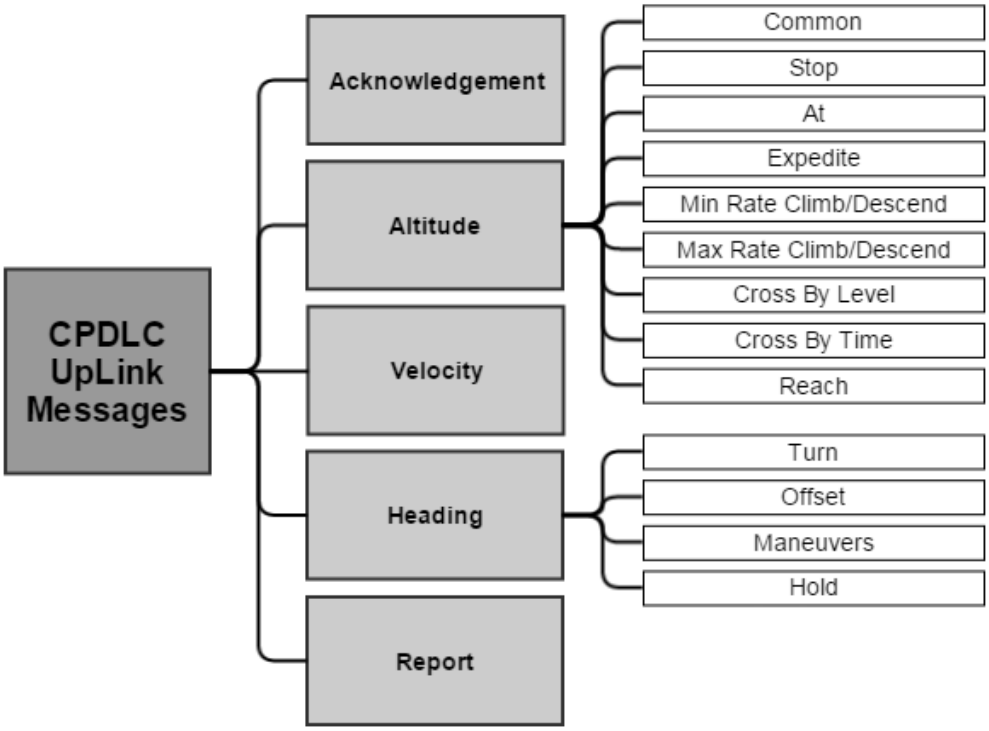


Figure 3.3: CPDLC UpLink Messages Diagram

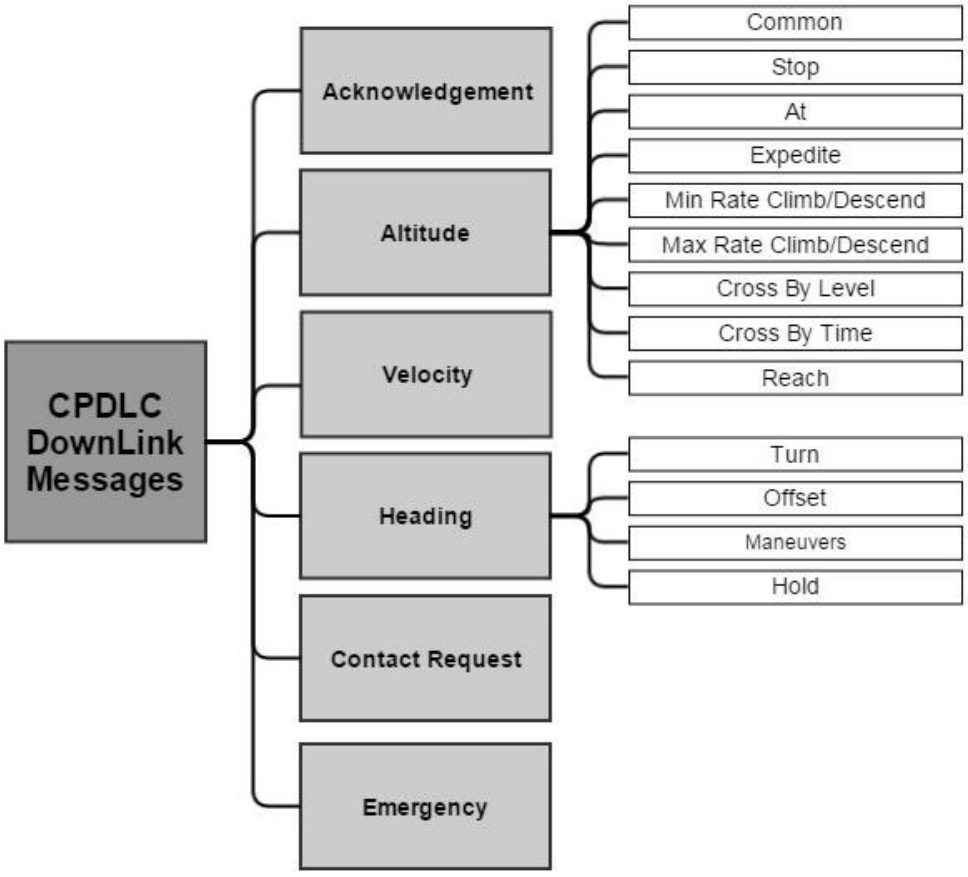


Figure 3.4: CPDLC DownLink Messages Diagram

As we can see both types of messages has been divided in a similar way. There are seven different divisions and the same subdivisions for altitude and heading messages. The principal reason for subdividing these divisions is that it exists a large quantity of messages for being just in a single division. Consequently it was needed another level of detail.

Next table 3.1 provides an accurate description of divisions.

Table 3.1. Division descriptions

Division	Description	Example
Acknowledgement	Brief messages which purpose is to accomplish with the communication protocol	UM 3: ROGER
Altitude	Messages related with flight level changes	UM 19: MAINTAIN [level]
Velocity	Messages related with velocity changes	UM 111: INCREASE SPEED TO [speed]
Heading	Messages related with heading changes	UM 94 : TURN [direction] HEADING [degrees]
Report*	Messages which purpose is to corroborate an aircraft action or data which is not completely clear for the ATC	UM 127 : REPORT BACK ON ROUTE
Contact Request**	Messages which purpose is to provide another way of communication	DM 21: REQUEST VOICE CONTACT [frequency]
Emergency**	Messages related with emergencies	DM 56 : MAYDAY MAYDAY MAYDAY

*Only used in UpLinks
**Only used in DownLinks

3.2.3. Organizational diagram translated to byte level

A translation to byte level is required for sending CPDLC messages through the net in an optimal way. All messages have a header at the begging when translated. The header has the purpose of Identify which is the message being communicated.
Header have always the same size, one byte.

In order to assign the header byte, the organization explained in previous subsection *3.2.2.* was used. All CPDLC message headers follow the structure presented in table 3.2.

Table 3.2. CPDLC message header structure

	Header structure							
CPDLC Messages	8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
UM/DM	-	-	-	-	-	-	-	-

- 8-bit: Identifier for uplink or downlink message. (0 uplink, 1 downlink)
- 7/5-bit: Division identifier
- 4/1-bit: Identifiers for concrete uplink/downlink message.

This way, we are able to have 8 divisions (2^3) per UpLink/DownLink messages and 16 messages per division (2^4).

Altitude division has been subdivided in three subdivisions because there are more than 16 messages.

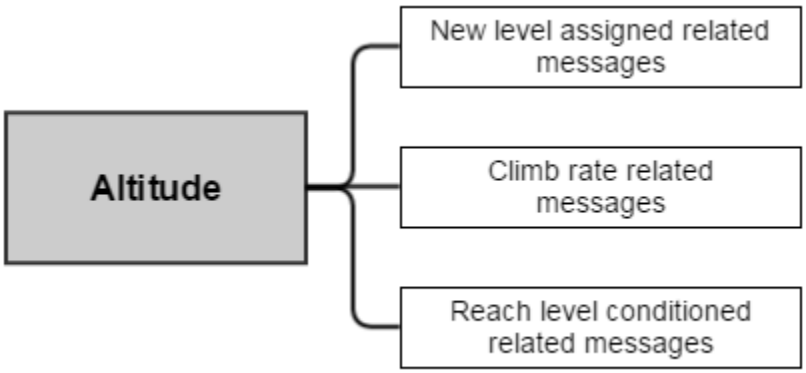


Figure 3.5: Altitude field subdivisions

In next table 3.3. is shown the relation between division – byte.

Table 3.3. Byte – division translation

Division	Subdivision	(7 bit - 6 bit - 5 bit) assigned
Acknowledgement	-	0-0-0
Altitude	New level assigned	0-0-1
	Climb Rate	0-1-0
	Reach level conditioned	0-1-1
Heading	-	1-0-0
Velocity	-	1-0-1
Report*	-	1-1-0
Contact Request**	-	1-1-0
Emergency**	-	1-1-1

*Only used in UpLinks
**Only used in DownLinks

3.2.4. Message example

In this subsection we will take two messages (an UL and a DL) and it will be shown all organizational concepts seen before.

3.2.4.1. UM 46 - CROSS [position] AT [level]

UM 46 is classified in altitude division, cross by level subdivision.
Byte translation is shown in table 3.4.

Table 3.4. Uplink header structure example

CPDLC Messages	8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
UM 46	0	0	1	1	0	1	0	0

3.2.4.2. DM 17 - AT [time] REQUEST OFFSET [specified distance] [direction] OF ROUTE

DM 17 is classified in heading division, offset subdivision.
Byte translation is shown in table 3.5.

Table 3.5. Downlink header structure example

CPDLC Messages	8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
DM 17	1	1	0	0	1	1	0	0

3.3. CPDLC fields codification

3.3.1. Objective

Fields in CPDLC application is understood as data linked to CPDLC messages. Messages can be accompanied by fields depending on the message being communicated. These fields are filled by ATC or pseudopilot users in the application. This subsection will explain how fields are organized, which properties has each field and how are translated into byte level.

3.3.2. Field diagram

In next table 3.6. we can see all fields being used in CPDLC application and their main characteristics.

Table 3.6. Field characteristics

Field	Byte Size	Range	Last Significant Byte
Level	2 bytes	-10 ≤ FL ≤ 900	1 FL
Position	6 bytes	-90 ≤ latitude ≤ 90 -180 ≤ longitude ≤ 180	$180/2^{23}$ deg
Time	3 bytes	UTC time	1/128 s
Vertical Rate	2 bytes	-5000 < VR < 5000	0.25 ft./min
Velocity	2 bytes	0 < Kts < 900	2^{-17} NM/s
Direction	2 bytes	0 ≤ Direction ≤ 360	$360/2^{16}$ deg
Degrees	2 bytes	-180 ≤ Degrees ≤ 180	$360/2^{16}$ deg
Specified Distance	2 bytes	0 < Distance ≤ 1000	$1000/2^{16}$ NM
Frequency	2 bytes	117.975 ≤ Freq ≤ 137	0.025 MHz
Leg Type	1 byte	21 leg types	-

Field's last significant byte has been taken from Asterix[4] as a trusted reference. Asterix is an ADS-B document published by Eurocontrol which also translates these fields into a byte level to be sent.
Consequently, a range of values it has been defined for each field depending on possible values taken by the field.
Finally it has been calculated the necessary byte size.

3.3.3. Byte translation

In this subsection will be shown the byte structure for each field. Each field has been thought to be as optimum as possible. As example it has been put a few field examples. All field structures are shown in appendix A.

Table 3.7. Level field structure

16-bit	15-bit	14-bit	13-bit	12-bit	11-bit	10-bit	9-bit
Level							
8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
Level		Reserved					

Table 3.8. Position field structure

48-bit	47-bit	46-bit	45-bit	44-bit	43-bit	42-bit	41-bit
Latitude[degrees]							
40-bit	39-bit	38-bit	37-bit	36-bit	35-bit	34-bit	33-bit

Latitude[degrees]							
32-bit	31-bit	30-bit	29-bit	28-bit	27-bit	26-bit	25-bit
Latitude[degrees]							
24-bit	23-bit	22-bit	21-bit	20-bit	19-bit	18-bit	17-bit
Longitude[degrees]							
16-bit	15-bit	14-bit	13-bit	12-bit	11-bit	10-bit	9-bit
Longitude[degrees]							
8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
Longitude[degrees]							

Table 3.9. Time field structure

24-bit	23-bit	22-bit	21-bit	20-bit	19-bit	18-bit	17-bit
Time[seconds]							
16-bit	15-bit	14-bit	13-bit	12-bit	11-bit	10-bit	9-bit
Time[seconds]							
8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
Time[seconds]							

3.3.4. Field example

It is going to be taken a data input for level field in order to show the translation procedure for fields.

- Level data input: FL 300
- LSB: 1FL
- Range: -10 ≤ FL ≤ 900
- Total allocations: 910
- Real allocations: 2^{10} → 1024
- Current allocation for data input example: 310

Table 3.10. Level field byte structure example

1	0	0	1	1	0	1	1
Level							
0	-	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
Level		Reserved					

3.4. CPDLC message library

3.4.1. Objective

The objective of CPDLC library is to create an algorithm capable of encoding and decoding all CPDLC messages organized in previous subsections.
Message structure is shown in next figure.

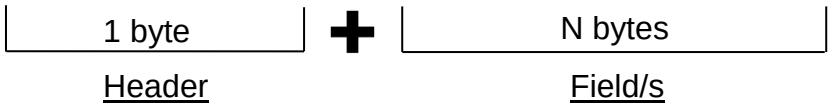


Figure 3.6: CPDLC message structure

Basically CPDLC message library pursues two main objectives:

- Code CPDLC message to byte level
- Decode CPDLC message to string

In order to achieve these main objectives the algorithm must be able to:

- Recognize all CPDLC headers
- Distinguish between UpLinks and DownLinks
- Know CPDLC message attributes
- Link each CPDLC message with its pertinent field/s
- Know how many fields has each message
- Link concrete field attributes to pertinent field
- Be able to split correctly the byte chain in order to extract the correct data

Next subsections will explain how code works and how it is organized for achieving all objectives.

3.4.2. Key algorithm concepts

To understand completely the algorithm, before starting with its flow diagram, it is shown the key concepts whom are used all along the library.

On the one hand, we must know that each CPDLC message it is encoded as an *Imessage*.

Imessage is an interface class which has the following attributes shown in table 3.11.

Table 3.11. Imessage attributes and usages

Imessage Attribute	Use
Boolean recent	Boolean used to know if the message has been read or not
string Mtime	String representing the exact UTC time which the message has been sent with
string Callsign	String representing the aircraft callsign which the message is to be sent or it is to be received
MessageId Id	Enum which relates each header with its concrete unique byte defined in the CPDLC message organization
string ToString	String that englobes the message once it has been translated from byte level.

All CPDLC classes inherit from Imessage interface, thus all CPDLC classes apply the same attributes.

On the other hand, fields are composed by *FieldAttributes* which is a class that inherits from Attribute. Attribute represents the base class for custom attributes. Inside FieldAttributes there is a constructor which defines all field attributes shown in table 3.12.

Table 3.12. FieldAttributes attributes and usages

FieldAttributes Attribute	Use
Int nBits	Int representing number of bits used for field encoding
Int offset	Int representing if it exists any offset on field encoding
Double res	Double representing the last significant byte from each field
Boolean isTwosComplement	Boolean which represents if a field is negative
FieldType Type	Fieldtype is an enum which defines which kind of field it is.
Int totalBytes	Int representing total bytes used for field encoding/decoding

3.4.3. Algorithm flow diagram

The following flow diagrams were designed for achieving the two main goals presented in subsection 3.4.1.
Now we will see the logic that follows CPDLC library in its two main methods.

- Serialize or encoding flow diagram:

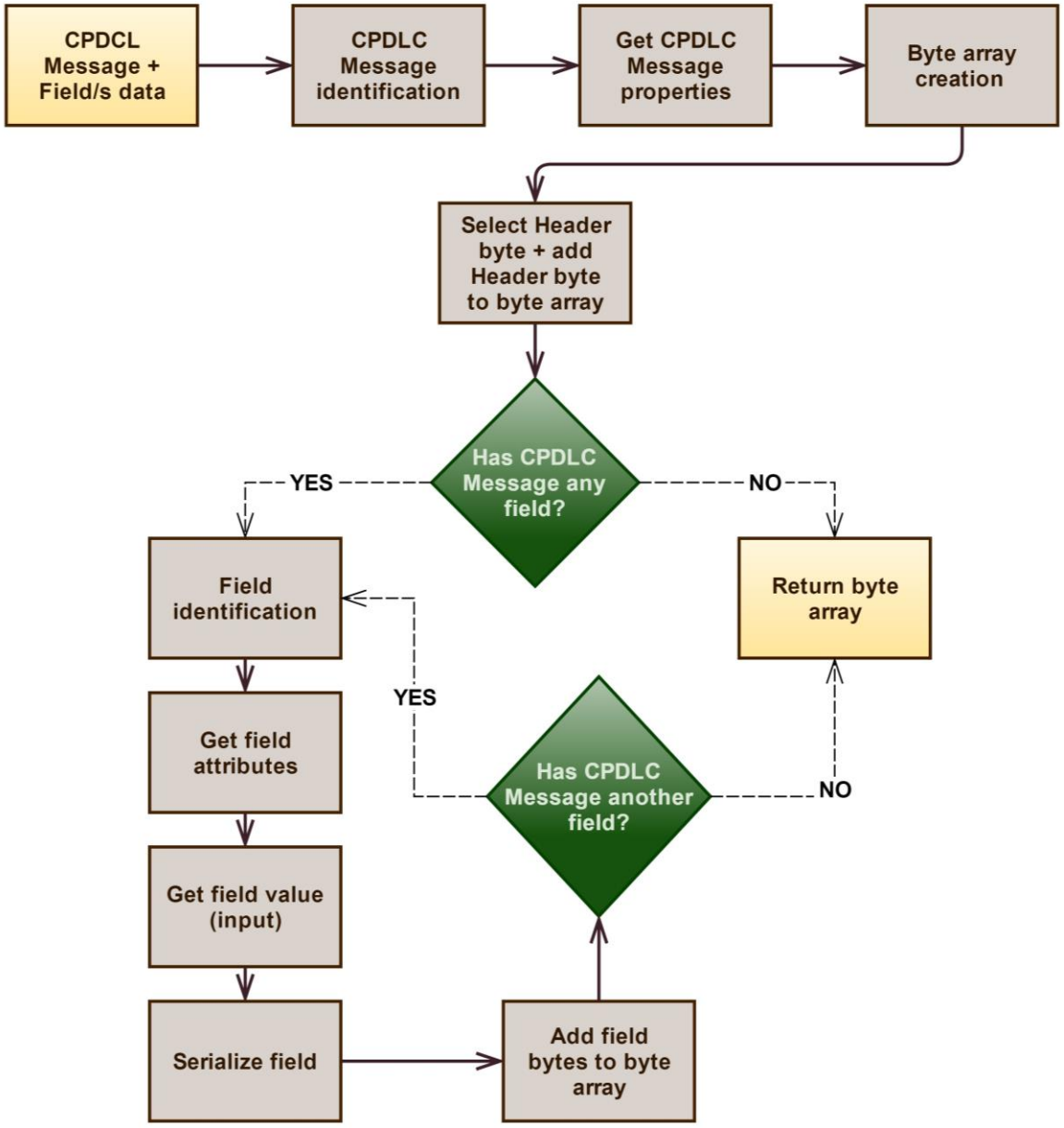


Figure 3.7: Serialize or encoding diagram

- Deserialize or decoding flow diagram:

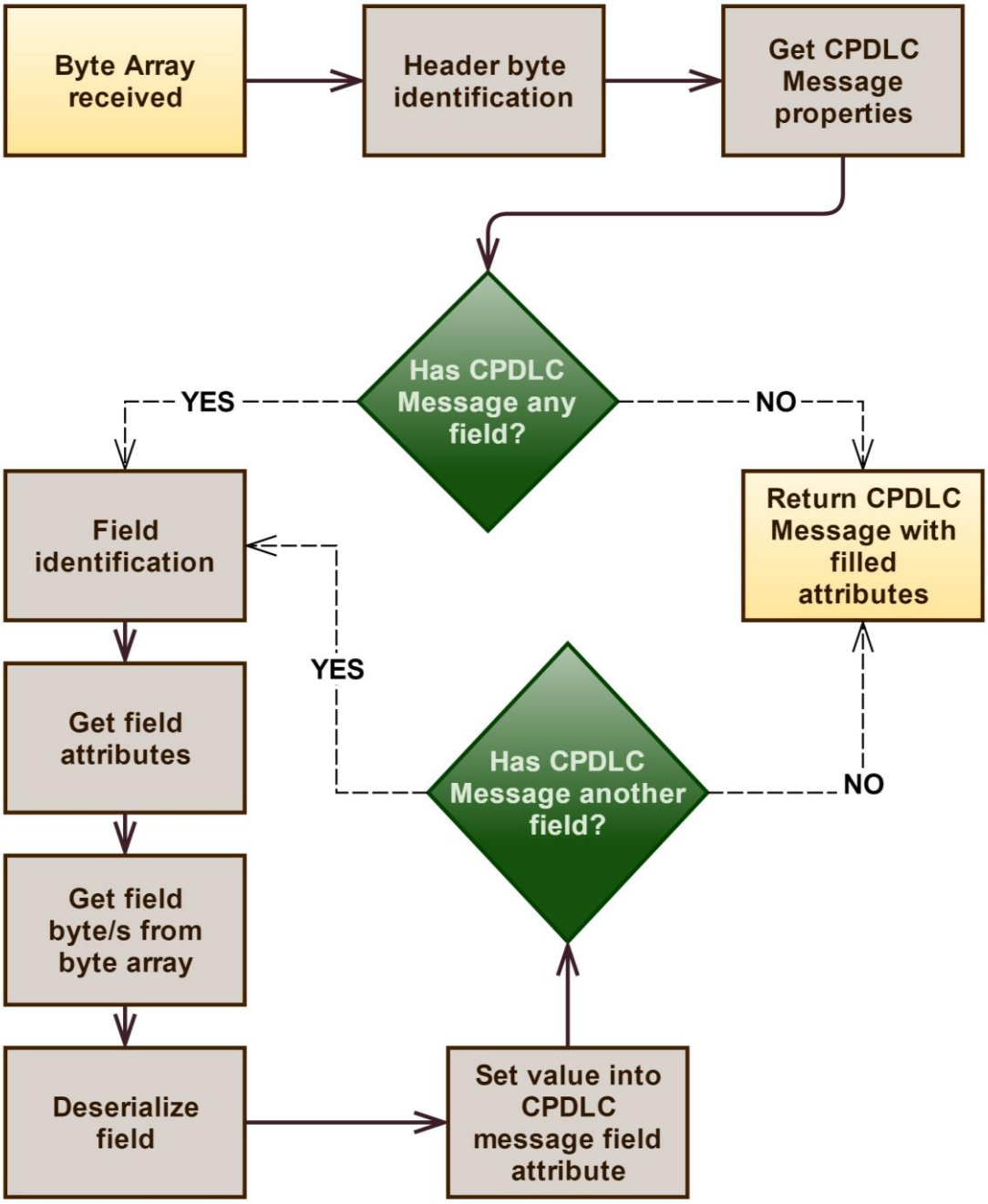


Figure 3.8: Deserialize or decoding flow diagram

3.4.4. Encode and decode methods

It is going to be explained the methods which make possible the main algorithms. We will see which method calls the main algorithm and which methods implement the main algorithm itself.
To clarify this fact, methods are split in two groups:

- Outer methods: Method which uses main algorithm to serialize or deserialize the CPDLC message.
- Inner methods: Methods used in the main algorithm to achieve its goal.

Outer methods:

- GetImessage: method that receives the whole byte array list as input and select the CPDLC Message being communicated using the header bytes and it triggers deserialize method.

*In order to trigger serialize algorithm it is used the user interface, when user selects the message is already known which message it is going to be communicated. Therefore there is no need a method for selecting the CPDLC message. The process will be clarified in section 3.5.

Inner methods:

- type.GetProperties: Method used to get the Imessage properties. Using this method it can be known which CPDLC message is being treated.
- GetCustomAttributesData: Method used to get the field attributes. It can be used because FieldAttributes class inherits from Attribute system class.
- GetValue: Method used to obtain the value related with the field in situ.
- BitHandling.GetBytes: Method used to translate the field value to byte level. The inputs for this method are the field attributes.
- BitHandling.ProcessOctet: Method used to translate the field bytes to the field values. It also uses the field attributes as inputs.

3.4.5. Message path in CPDLC message library

Now to completely understand the CPDLC message library we will see the process step by step for UM19: MAINTAIN [level].

The itinerary would be:

- Serialize:

1) Field input: User will fill the field with the value to be communicated.

UM19: MAINTAIN FL 220.

2) UM 19 properties are:

Table 3.13. UpLink CPDLC message properties example

UM 19 properties	Values
Boolean recent	true
string Mtime	Current local time
string Callsign	Current aircraft callsign
MessageId Id	UM19
string ToString	MAINTAIN FL 220

3) The header byte taking into account the inner data would be:

Table 3.14. UpLink CPDLC message header serialize example

Header	Decimal byte	Bit level
UM19	[16]	[00010000]

4) By knowing the messageId id we are able to know which fields go with UM19. In this case a [level] field. Its attributes are:

Table 3.15. Altitude field properties

UM 19 fields	Field properties	Values
Altitude	Int nBits	10
	Int offset	6
	Double res	1
	Boolean isTwosComplement	True
	FieldType Type	Altitude
	Int totalBytes	2

5) Now, with input value (FL 220) and field properties, we are able to translate the field value to byte level:

Table 3.16. Altitude field input serialize example

Field	Input	Decimal byte	Bit level
Altitude	FL 220	[55] [0]	[00110111] [00000000]

6) Finally the byte array to be communicated will be:

[16] [55] [0] - [00010000] [00110111] [00000000]

- Deserialize:

1) Message received: Once CPDLC message is received, the algorithm starts working with the byte array communicated.

[16] [55] [0] - [00010000] [00110111] [00000000]

2) The algorithm splits the header byte in order to recognize which CPDLC message is being communicated. Inverse logic from serialize, 3rd step.

3) UM19 properties are the same as seen in serialize, 2nd step. In a same way field attributes are the same seen in step 4th.

4) With field attributes and taking the correct bytes corresponding to the field [level] we can deserialize the field value.

Table 3.17. Altitude field input deserialize example

Field	Input Decimal byte	Input Bit level	Output
Altitude	[55] [0]	[00110111] [00000000]	FL 220

5) Finally we can trigger to string method which takes the field value deserialized and it builds the CPDLC message for being finally communicated:

UM19: MAINTAIN FL 220.

3.5. CPDLC application graphic development

3.5.1. Objective

Human Machine Interface (HMI) is defined to be the connection between human and machine. CPDLC HMI pursuits the goal of creating a friendly, clear and fast environment at the same time in order to understand the communication flow and make it smooth.

CPDLC HMI must fulfill next requirements:

- Understandable: If someone uses the interface for first time, some elements should be familiar.
- Responsiveness: The user must know what is going on and if their inputs are being successful. It must have a visual feedback.
- Clarity: Interface must avoid ambiguity.
- Efficiency: Interface must be able to be faster by using shortcuts and good design.

Next subsections will explain how CPDLC HMI is designed and how it works.

3.5.2. CPDLC Human-Machine interface design

In this subsection we will see all steps that take place in CPDLC HMI.

3.5.2.1. CPDLC application introduction

As first step we see an introduction screen, the user will select his role.

Figure 3.9: Introduction window CPDLC HMI

3.5.2.2. Login

Once the user has selected his role, next screen is a login. The purpose of this login is to know which sector will be controlled. As we have seen in subsection 3.1.3. communication is one to one, and pseudopilot and airspace controller will be controlling/flying same airspace.

Thus, login screen is the same for ATC and pseudopilot. Moreover, from this step on, we will just focus on ATC path, because both paths are programmed in a similar way. The only thing that differes is CPDLC messages used in each way, uplinks for ATC users and downlinks for pseudopilot users.

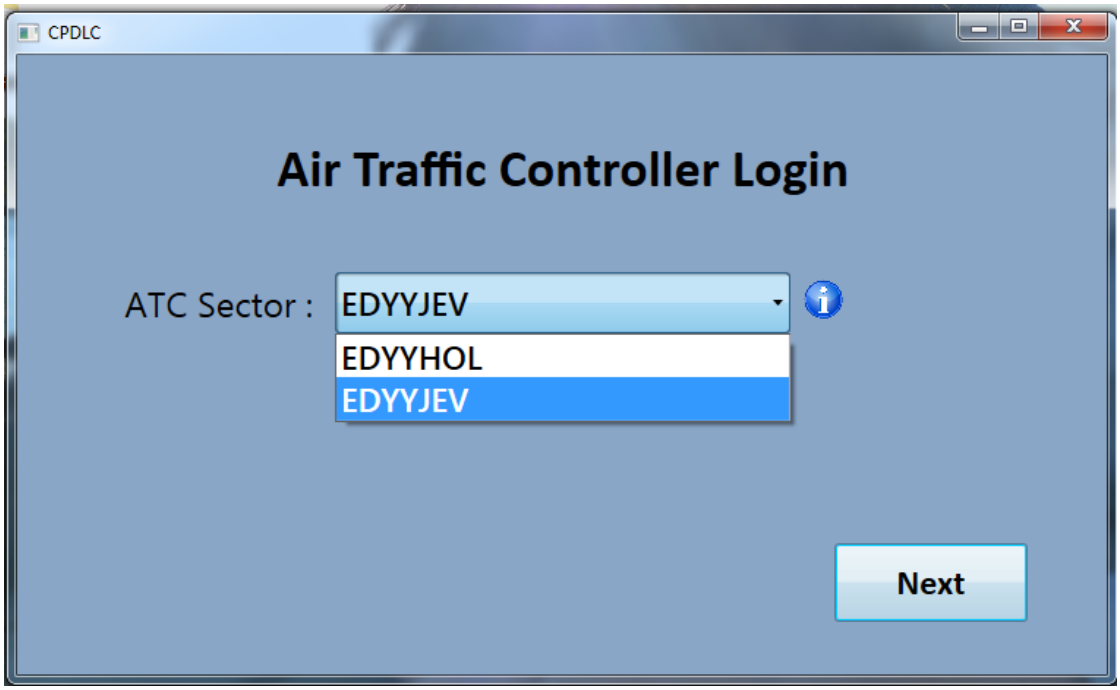


Figure 3.10: Login window

3.5.2.3. Chat Window

This window is the most important in CPDLC app. It will be operative the most time of simulation.

As we can see, there are three kind of chats:

- Global chat: It englobes all CPDLC messages incoming from all pilot users present in the sector. Controller will always see a continuously flow of incoming messages in this chat.

- Acknowledgement chat: It englobes just acknowledgement CPDLC messages from all pilot users present in the sector. The purpose of it is to have a better agility to look for a concrete answer when ATC is expecting a message like “WILCO” or “UNABLE”.

- ATC-Pilot / Pilot-ATC chat: It englobes all CPDLC message incoming from a selected aircraft callsign. The purpose of this chat is to give to ATC user a tool for focusing in a concrete chat that may be a priority over the rest conversations.

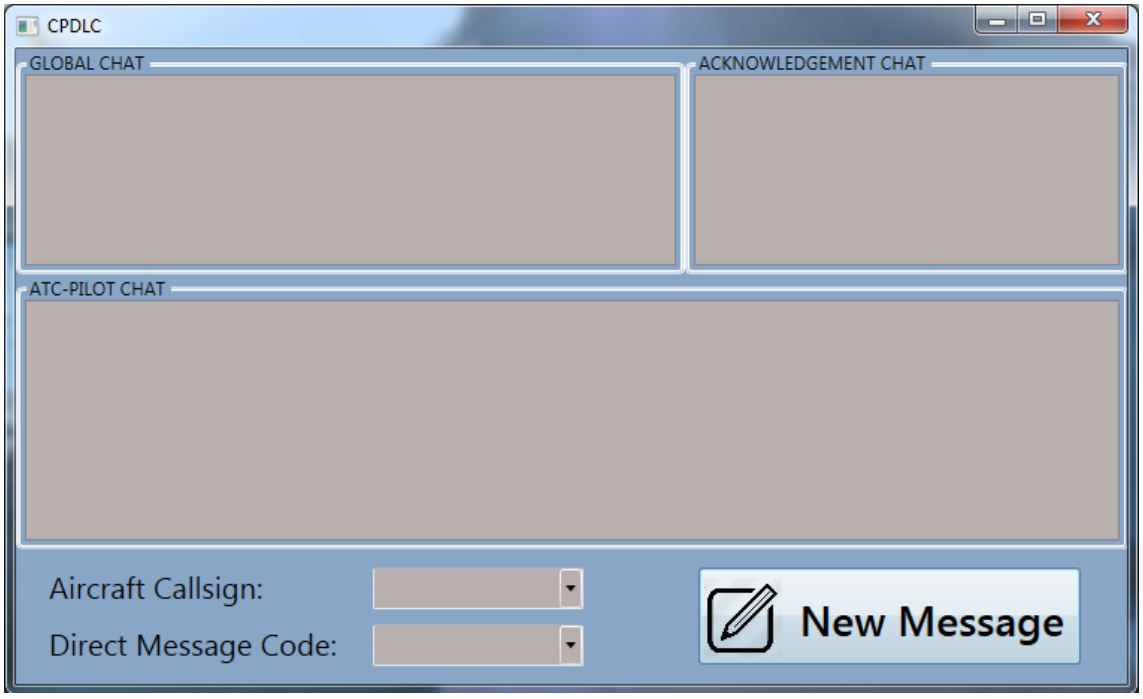


Figure 3.11: Chat window

To start sending a message exists two different ways:

- Common way: ATC user must fill aircraft callsign control. Then he just has to click in new message button.

- Direct Message Code way: When ATC user gets a better control on CPDLC Message codes (UM19, UM0…). He will be able to get to the final screen (point 3.5.2.7) in a faster way. The user must to fill the aircraft callsign control too. Then he has to type or search CPDLC message code in direct message code control. Finally he has to click new message button.

If direct message code control remains empty, when clicking new message, application will always go through common way.

3.5.2.4. Message type selection

Next window represents first level of CPDLC uplink messages diagram seen in figure 3.3.

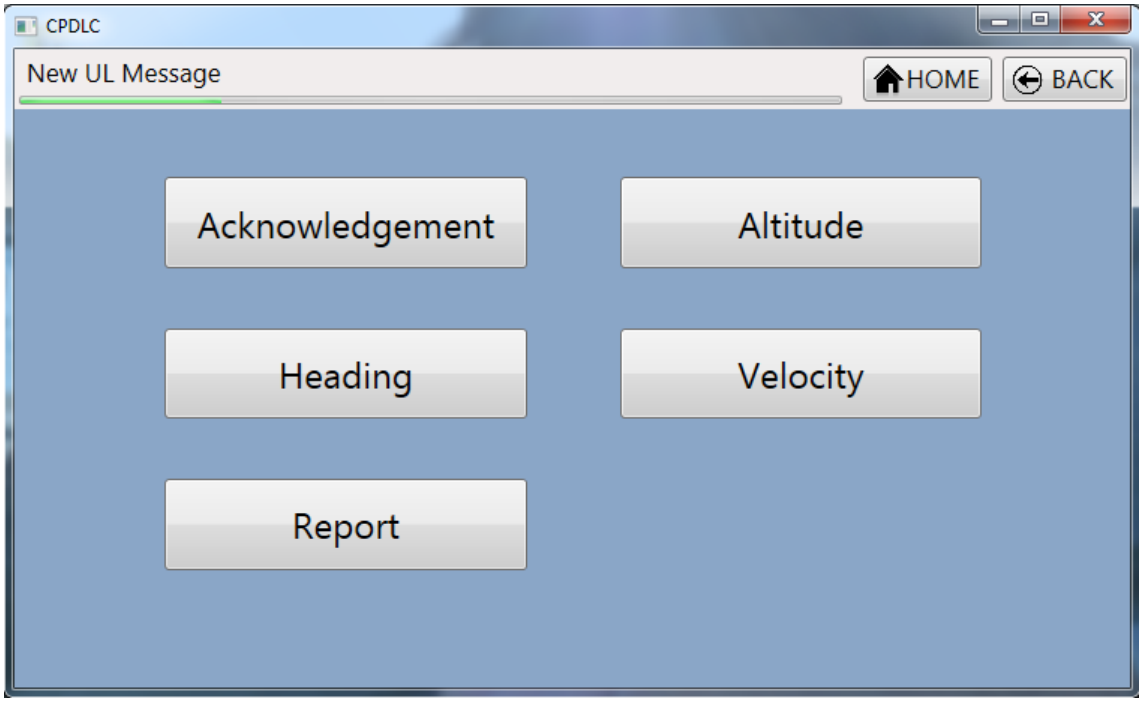


Figure 3.12: Message type window

As we can see, in the top of the application window, it has appeared a feedback indicator. This feedback will give information about how much is left to send CPDLC message and it provides in which part of the application flow you are in each moment.

It has also appeared home and back buttons. These buttons exist for granting a way back in the application:

- Home button will send back ATC user to chat window, which is considered as the most important.
- Back button will send back ATC user to the previous window.

To keep with the flow, ATC user must select which type of message he wants to send.

3.5.2.5. Message type selection II

In the case in that ATC user has selected altitude or heading a second message type window will appear. This happens for the reason described in subsection 3.2.2.

It works in the same way as previous message type window.

We can appreciate that feedback indicator evolves as we progress through the application.

Figure 3.13: Message type window II (altitude)

Figure 3.14: Message type window II (heading)

3.5.2.6. CPDLC message choice

In this window ATC user will see displayed a message list in relation of the message type selected in previous steps.

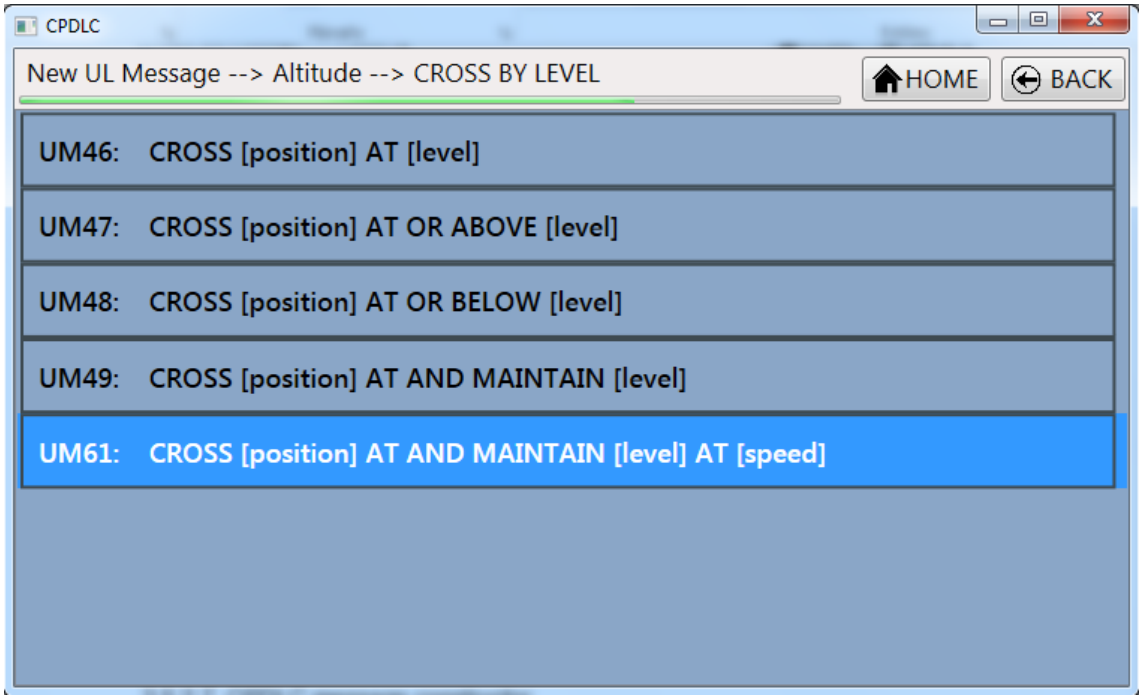


Figure 3.15: Message choice window

To keep the flow, ATC user must double click in one of the messages displayed.

3.5.2.7. CPDLC message constructor window

Finally as last step for sending the message through the net, ATC user will have to fill fields in a proper way.

If we take a look into the feedback indicator, we will see that it is already completed.

Message field values are provided directly. ATC user just has to click in field value control and a list with all possible values will be displayed. This way we ensure a logic value and we avoid incorrect or misleading values.

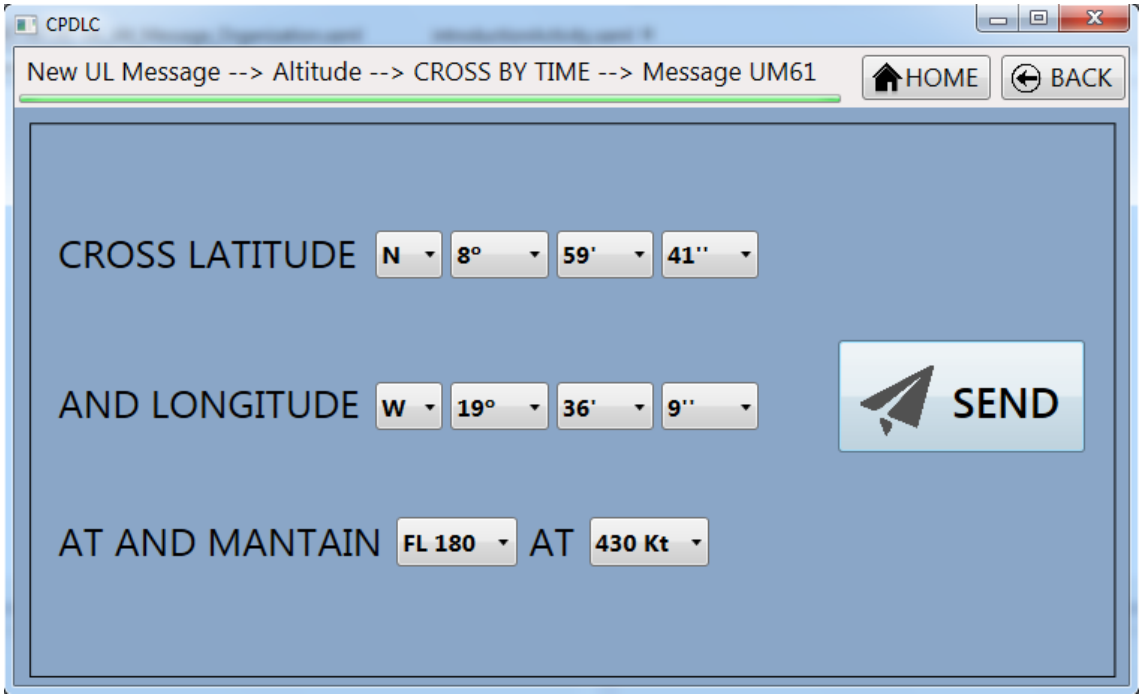


Figure 3.16: Message constructor window

Once ATC user has filled the field values, next step is to click into send button. Right after, a message confirmation window with all message data will pop up to really ensure that the message being showed is the correct one and that is being sent to the proper aircraft callsign.

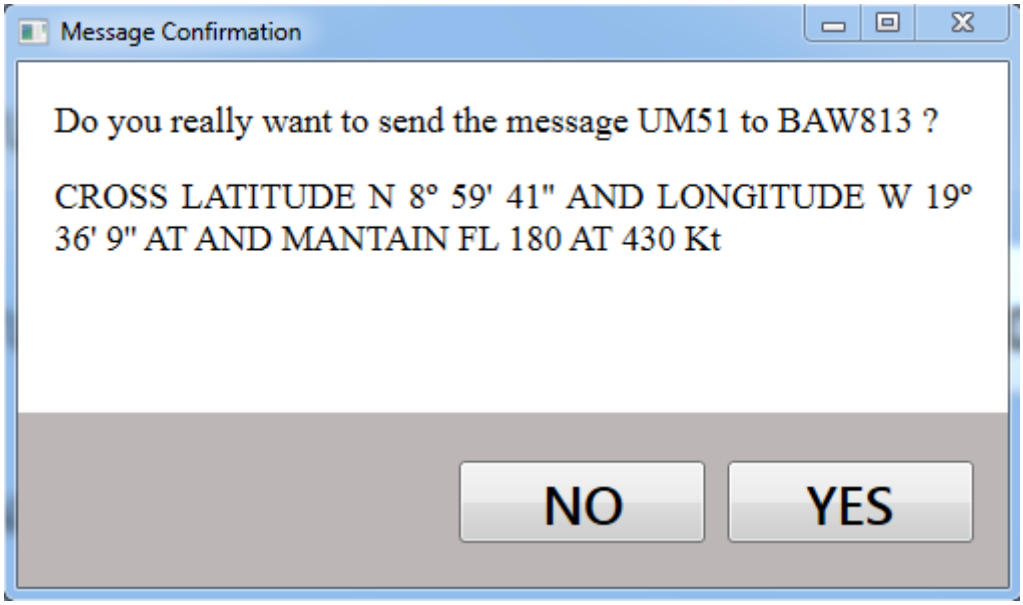


Figure 3.17: Message confirmation window

3.5.2.8. Chat Window with messages

Next figure shows how CPDLC messages sent looks like in chat window.

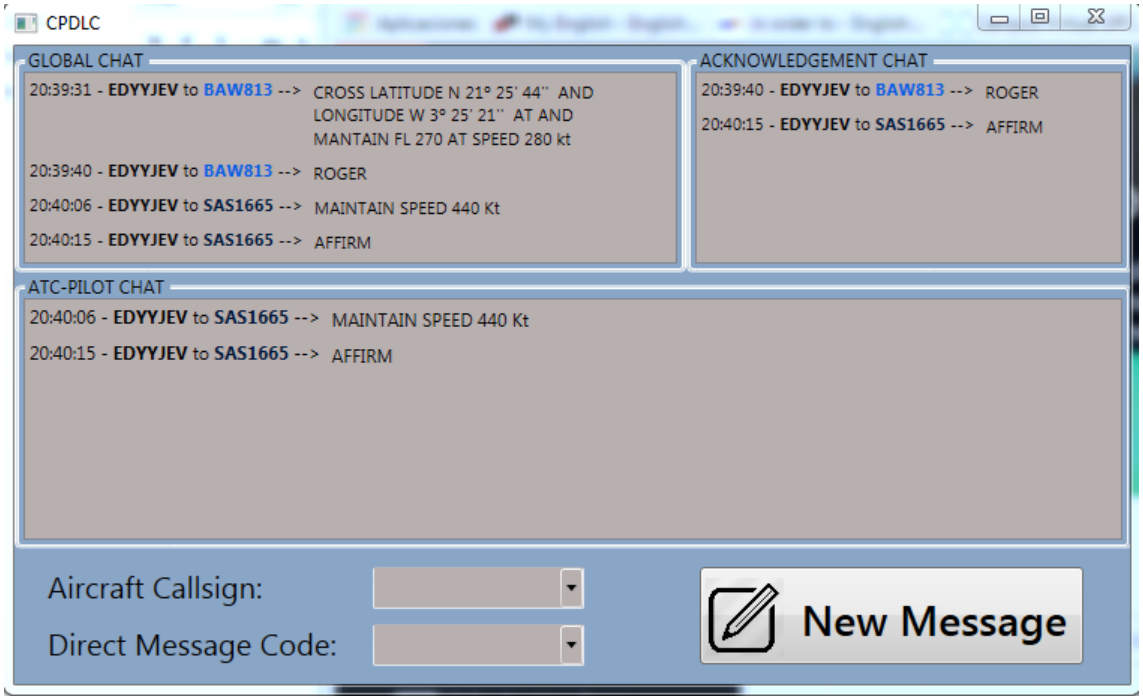


Figure 3.18: Chat window with messages

As we see, callsign is identified by different colors. Color selection is a random function. The algorithm always ensures different colors for current aircrafts being controlled. It has to be said that algorithm also avoids red range colors which are reserved for incoming emergency messages. When it is about an emergency message, the text containing the information is highlighted in red too.

In addition when ATC user receives a new message it is painted in green color to indicate that the message is a new one. When user clicks on it, the message passes to have the ordinary black color as a signal of read.

3.5.3. Implementation diagram

In this subsection it will be explained how CPDLC HMI relevant data flows through different windows and classes. We will be able to see what information is required in each part of the application and where is created.

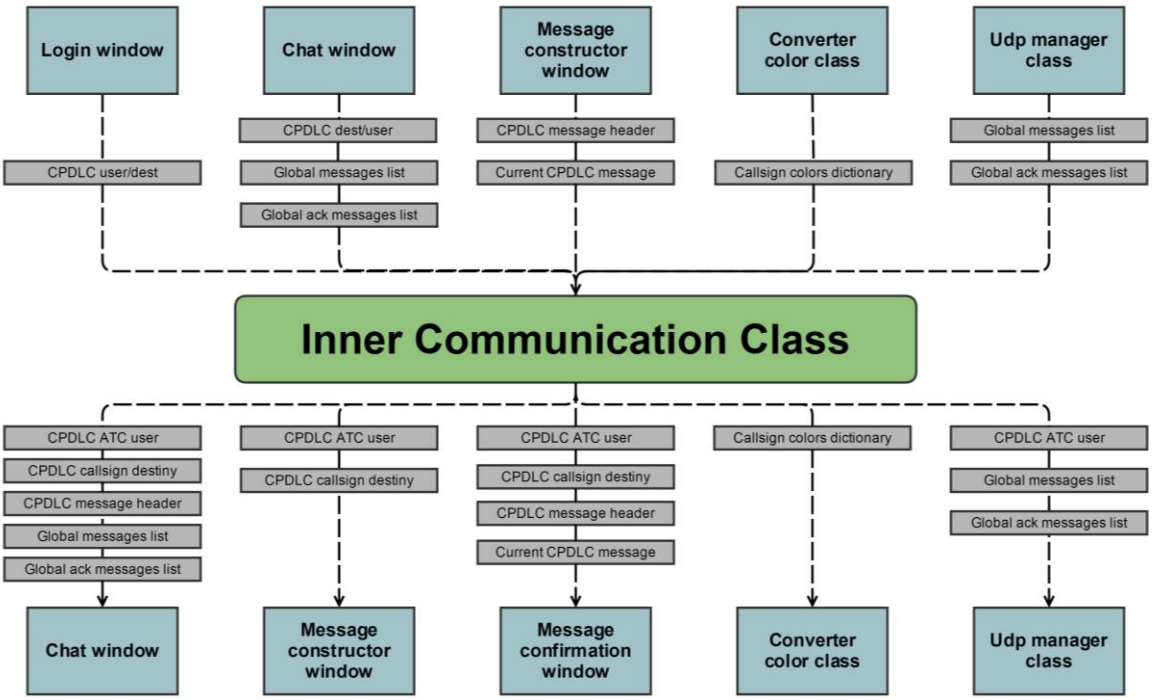


Figure 3.19: CPDLC HMI implementation diagram

As connection to set and get data it has been implemented a singleton method in inner communication class. A singleton is a design pattern that restricts the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system. This way it is ensured that correct data will be stored.

In figure 3.19 it can be seen all connections done through inner communication class. In the top there are windows and classes that set data. In the bottom there are the windows and classes fed by the inner communication class. Grey boxes is the data being set or get in each case.

Furthermore, CPDLC HMI application stores individual conversations between ATC-Pilot and vice versa. Conversations are stored in a txt file. A daily directory is created each time that CPDLC application is used.

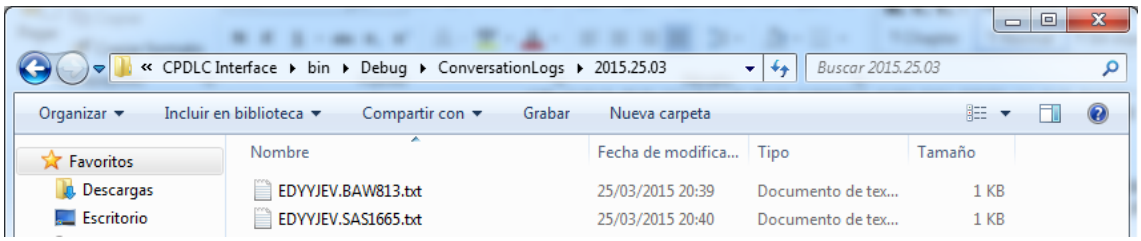


Figure 3.20: Conversations stored from 03/25/2015

This way it is ensured having a historical from all conversations made through the application.

3.5.4. CDPLC HMI & CPDLC library connection

This subsection will explain when CPDLC HMI uses CPDLC library in order to encode/decode a CPDLC message selected/received with its proper fields. This way it will be clarified the usage of CDPLC library.

For encoding, in last step of CPDLC HMI, when user clicks into button yes in message confirmation window (fig. 3.17), code gathers all data information and calls CPDLC library method serialize. To put an example we will see how CPDLC message UM98 works.

```
if (DB.ShowDialog() == true)
{
    //Get which is the current user/dest
    String User = Singleton.getInstance().CurrentUser;
    String Dest = Singleton.getInstance().CurrentDest;

    //Get callsign byte array
    callsignCode CS = new callsignCode(Dest);
    byte[] nb = CS.callsignEncoding();

    //Gather information introduced by the user and uses CPDLC library to encode it.
    UM98_code m1 = new UM98_code();
    m1.Direction = Convert.ToDouble(dirCB.Text.Substring(0, dirCB.Text.Length - 1));
    m1.Degrees = Convert.ToDouble(degCB.Text.Substring(0, degCB.Text.Length - 1));
    byte[] b = MessageConverter.Serialize(m1);

    //Concatenates callsign bytes array and cpdlc message bytes array
    byte[] total = new byte[nb.Length + b.Length];
    System.Buffer.BlockCopy(nb, 0, total, 0, nb.Length);
    System.Buffer.BlockCopy(b, 0, total, nb.Length, b.Length);

    //Method to send the message through the net
    UdpManager.GetInstance().SendToSelected("PWP", total);
```

Figure 3.21: Code used to connect CPDLC HMI with CPDLC library for encoding CPDLC Message

As we can see, now total byte being sent through the net is composed by next structure:

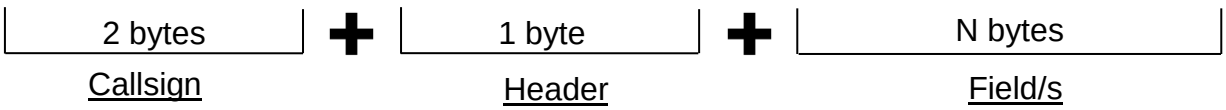


Figure 3.22: Final byte array structure for CPDLC messages

Callsing bytes are encoded by using callsignEncoding method (see subsection 3.5.5). CPDLC message bytes are encoded by MessageConverter.Serialize which is called from CPDLC library.

Finally total byte array structure is sent through the net using UdpManager library which provides an udp type connection.

For decoding, once the code has detected an incoming message, the first step is split the array in its three parts.

```
//Array split
byte[] callsignBytes = new byte[2];
byte[] messageBytes = new byte[bytes.Length - 2];
Array.Copy(bytes, callsignBytes, 2);
Array.ConstrainedCopy(bytes, 2, messageBytes, 0, bytes.Length - 2);
```

Figure 3.23: Splitting incoming byte array code

Then, each part split is used in different methods to obtain current callsign and the Imessage received. To get current Imessage, CPDLC library method CPDLC.Utils.getImessage(bytes[]) is used.
Next step is to set Imessage attributes, and finally save the incoming message into different chat lists to be displayed in chat window (Fig. 3.11).

```
//Callsign and deserialize methods
callsignCode CS = new callsignCode(callsignBytes);
IMessage incMessage = CPDLC.Utils.getImessage(messageBytes);

//Imessage attributes
incMessage.Mtime = DateTime.Now.ToString("HH:mm:ss");
incMessage.Recent = true; //always true for inc messages
incMessage.Callsign = CS.callsignDecoding();

//Inc message being saved in a txt file
String txtpath = path + "\\" + user + "." + incMessage.Callsign + ".txt";
TextWriter tw = new StreamWriter(txtpath, true); //this method checks if the .txt has been created.

String beforeMessage = incMessage.Mtime + " - " + incMessage.Callsign + " to " + user + " --> ";
tw.WriteLine(beforeMessage + incMessage.ToString());
tw.Close();

//Inc message add to global chat
var GlobalMsg = Singleton.getInstance().GlobalMessages;
GlobalMsg.Add(incMessage);
Singleton.getInstance().GlobalMessages = GlobalMsg;
//Inc message add to ack chat if it is an ack message
String currentHeader = incMessage.Id.ToString();
if (currentHeader == "UM0" || currentHeader == "UM1" || currentHeader == "UM2" || currentHeader == "UM3"
|| currentHeader == "UM4" || currentHeader == "UM5")
{
    var GlobalAckMsg = Singleton.getInstance().GlobalAckMessages;
    GlobalAckMsg.Add(incMessage);
    Singleton.getInstance().GlobalAckMessages = GlobalAckMsg;
}
```

Figure 3.24: Code used to connect CPDLC HMI with CPDLC library for decoding CPDLC Message

3.5.5. CPDLC HMI classes and functionalities

Now we will focus on the classes used throughout CPDLC HMI. As any code, it exists classes used to obtain certain information. Each class has some functions that has to carry out with specific data proportioned by the user or by

the proper code. Thus, now it will be explained what classes are used, what functionalities has each one and what are the inputs and outputs.

3.5.4.1. TrafficGenerator

Trafficgenerator is a console executable that generates traffic code automatically. This class was created in order to have a tool to generate proper code for each simulation depending on the traffic being used.

Input is a txt with all simulation data extracted from eDep.
Output is the code that will be implemented on callsignCode class.

3.5.4.2. CallsignCode

CallsignCode is the class that relates callsign codes with a unique byte array to be used into the final byte array structure (fig. 3.22) and vice versa. The code inside this class is generated by trafficGenerator as explained before.

Input for this class can be a string callsign or a byte array representing the callsign.
Output is the opposed pair of the input.

3.5.4.3. InputValues

InputValues is a class that has all the data for filling field information in message constructor window (fig. 3.16.). Field data is constant, there is no need to instantiate the method. Therefore, all methods used in this class are statics.

There is no need for an input value in this class.
Output is a string array containing all the values used by the field.

```
public static String[] FLvalues =
    {
            "FL 0","FL 10","FL 20","FL 30","FL 40","FL 50","FL 60","FL 70","FL 80",
            "FL 90","FL 100","FL 110","FL 120","FL 130","FL 140","FL 150","FL 160",
            "FL 170","FL 180","FL 190","FL 200","FL 210","FL 220","FL 230","FL 240",
            "FL 250","FL 260","FL 270","FL 280","FL 290","FL 300"
    };
```

Figure 3.25: Static method example of inputValues class

3.5.4.4. UplinkMessageInfo

UplinkMessageInfo class has the same philosophy of inputValues class. It has all uplink messages split by divisions. This class is used for filling the list on message choice window (fig.3.15.)

Input value is not needed.
Output value is a string array containing all CPDLC messaged belonging to a certain division.

3.5.4.5. DownlinkMessageInfo

DownlinkMessageInfo is the same class as uplinkMessageInfo class but used for downlink CPDLC messages.

3.5.4.6. Converter

Converter class is used to have a dictionary that relates a callsign with a unique color. This dictionary is stored each time that is changed in a singleton as explained in subsection 3.5.3.

Input is a string callsign.
Output is the color from dictionary being used later to paint callsign strings shown in chat window.

3.5.4.7. EmergencyConverter

Emergency converter is used for giving color to the messages shown in chat window too. However, emergencyConverter class is just used to paint in red color emergency messages in order to highlight them.

Input is a string representing all the message.
Output is a font message change if emergency condition is true.

3.5.4.8. Singleton

Singleton method has been explained in subsection 3.5.3. The objective is to share information when needed throughout all CPDLC application.

Input is the value to be shared.
Output is the value needed.

3.5.4.9. UdpManager

UdpManager class manages local net connection to make possible communication between eDep user – CPDLC user and CPDLC user – CPDLC user as explained in section 3.1.

Input value is the byte array to be communicated through the net.

Output value is the byte array communicated in receiver’s side.

Chapter 4. Conclusions

4.1. Objectives achieved

Taking as a reference three main aims presented in section 1.3 it can be said that there are two of three aims accomplished.

It has been understood the GOLD[3] document and essential information about CPDLC data link technology has been extracted. It has been possible to design an application following real methodology providing of the GOLD document.

It has been done a CPDLC application starting from zero. It was designed, thought and elaborated from the very beginning. CPDLC application developed is a beta version and has to be tested and evaluated. Therefore, it accomplishes all the requirements put at the beginning.

Regrettably, there was no time to test the CPDLC's Beta version in Icarus simulations. CPDLC application developing goal took more time from expected at the beginning.

4.2. Future development

Setting aside all objectives achieved presented in section 5.1. there is a lot of work to do with the CPDLC project. At the end of the day, in this project it has been put just the basis for starting the study of the CPDLC impact over ICARUS simulations.

To improve CPDLC performance there are some direct areas that should be enhanced:

- HMI improvement
- CPDLC communication management improvement
- Local net set up improvement

4.2.1. HMI improvement

First task after finishing any HMI is to test it with users. In this case it has to be done simulations for testing CPDLC tool for communicating. It has to be done a survey in order to know what users have liked and what have disliked about HMI. Questions about requirements that must fulfill an interface can be asked. It is important to know if users are satisfied and comfortable with the CPDLC tool.

This way, we can have a really important feedback and a direct improvement could be done directly from the customer's approval.

Besides, taking into account that CPDLC app has been developed as a beta version, CPDLC HMI design has to be improved a lot for making it more attractive and easier to understand and use.

4.2.2. CPDLC communication management improvement

As explained in section 2.2, CPDLC application has been adapted to Icarus simulations and there are some procedures which do not follow the GOLD document.

In order to make a more realistic tool, it can be introduced updates including new procedures, new messages and new features in CPDLC interface.

4.2.3. Local net set up improvement

Nowadays, CPDLC local net set up has to be done “manually” by changing IP field in txt files which are read by CPDLC app to get data for the communication. It does not exist any button in the interface capable of setting up this file.

It would be a great upgrade to manage CPDLC connection files from the application in a clear way. It will give more robustness to the application.

4.3. Personal evaluation

This bachelor’s degree final work has been, personally, really satisfactory due to the learning I reached both theoretically and practically.

In the one hand, theoretically, I delved into the data link concept knowledge and above all I learnt how CPDLC is managed under ICAO rules.
Moreover, I have learnt to manipulate complex and technical information for getting what was really important for the last degree work.

In the other hand, practically, I learnt to develop an application since the very beginning. Covering since design phase, passing through intern algorithm developing, until making an operative application with a user interface.

Besides, I have found out new programming techniques really useful whom I haven’t heard about before. We can put hereditary classes or singleton using as examples.
Also noteworthy is that I learnt a lot about the handling of byte level information and how to treat it.

Developing an application from zero takes a lot of work. It seems unbelievable how something really simple seen from outside, it becomes a really hard task with many hours of working and designing. This final degree work taught me to appreciate all the effort involved in any project creation from zero.

To conclude, I would like to dedicate a few words to CPDLC technology, since although the CPDLC technology is under phase of development, aeronautic world is really slow, but very robust at the same time. It cannot happen anything that has not been thought previously. For that reason, all studies related to new technologies applicable to aeronautics and all data collected about it are very important when verifying and implementing these new technologies.

I like to think that with the use of this application, when it passes testing phases and it is used in Icarus simulations, somehow, when data collected and studies made about CPDLC will be sent to Eurocontrol, I can become a tiny fraction of the many others existing who has contributed in the CPDLC technology implementing in aeronautics' world.

References

[1] – ATC Data Link News Copyright © 1999, Craig. J. Roberts - Site Updated: March 18, 2015: http://members.optusnet.com.au/~cjr/index.html

[2] – Leading technological innovation: Controller Pilot Data Link Communications – January 2013: https://www.eurocontrol.int/sites/default/files/publication/files/2013-cpdlc.pdf

[3] – GOLD, Global Data Link Document. ICAO, April 2013.

[4] – ASTERIX, All Purpose Structured Eurocontrol Surveillance Information Exchange. Eurocontrol, May 2011.

Appendix A- CPDLC fields structure

Table A.1. Level field structure

16-bit	15-bit	14-bit	13-bit	12-bit	11-bit	10-bit	9-bit
Level							
8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
Level		Reserved					

Table A.2. Position field structure

48-bit	47-bit	46-bit	45-bit	44-bit	43-bit	42-bit	41-bit
Latitude[degrees]							
40-bit	39-bit	38-bit	37-bit	36-bit	35-bit	34-bit	33-bit
Latitude[degrees]							
32-bit	31-bit	30-bit	29-bit	28-bit	27-bit	26-bit	25-bit
Latitude[degrees]							
24-bit	23-bit	22-bit	21-bit	20-bit	19-bit	18-bit	17-bit
Longitude[degrees]							
16-bit	15-bit	14-bit	13-bit	12-bit	11-bit	10-bit	9-bit
Longitude[degrees]							
8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
Longitude[degrees]							

Table A.3. Time field structure

24-bit	23-bit	22-bit	21-bit	20-bit	19-bit	18-bit	17-bit
Time[seconds]							
16-bit	15-bit	14-bit	13-bit	12-bit	11-bit	10-bit	9-bit
Time[seconds]							
8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
Time[seconds]							

Table A.4. Vertical Rate field structure

16-bit	15-bit	14-bit	13-bit	12-bit	11-bit	10-bit	9-bit
Vertical Rate							
8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
Vertical Rate							

Table A.5. Vertical Rate field structure

16-bit	15-bit	14-bit	13-bit	12-bit	11-bit	10-bit	9-bit
IM	Velocity						
8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
Velocity							

Table A.6. Direction field structure

16-bit	15-bit	14-bit	13-bit	12-bit	11-bit	10-bit	9-bit
Direction							
8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
Direction							

Table A.7. Degrees field structure

16-bit	15-bit	14-bit	13-bit	12-bit	11-bit	10-bit	9-bit
Degrees							
8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
Degrees							

Table A.8. Distance field structure

16-bit	15-bit	14-bit	13-bit	12-bit	11-bit	10-bit	9-bit
Distance							
8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
Distance							

Table A.9. Frequency field structure

16-bit	15-bit	14-bit	13-bit	12-bit	11-bit	10-bit	9-bit
Frequency							
8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
Frequency		Reserved					

Table A.10. Leg type field structure

8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
Leg Type					Reserved		