CONTRACT N° : G4RD-2000-00228

PROJECT N° : GRD1-1999-10516

ACRONYM : MA-AFAS

THE MORE AUTONOMOUS - AIRCRAFT IN THE FUTURE AIR TRAFFIC MANAGEMENT SYSTEM

D57 – AGP Software Detailed Design

AUTHOR: SKYSOFT PORTUGAL

PROJECT CO-ORDINATOR: BAE SYSTEMS

PRINCIPAL CONTRACTORS:
Airtel ATN Ltd (Ireland) Qinetiq (UK)
ETG (Germany) EUROCONTROL (France)
NLR (Netherlands)

ASSISTANT CONTRACTORS:
Airsys ATM (France) Alenia Difesa (Italy)
AMS (Italy)DLR (Germany)
FRQ (Austria)Indra Sistemas (Spain)
NATS (UK)SCAA (Sweden)
SC-TT (Sweden)Skysoft (Portugal)
SOFREAVIA (France)Stasys Limited (UK)

Report Number :TBD
Project Reference number :MA-AFAS – WP2.4 – D57
Date of issue of this report : 06-February-03
Issue No:DRAFT A
PROJECT START DATE :1/3/2000
DURATION:36 Months

Project funded by the European Community under the ‘Competitive and Sustainable Growth’ Programme (1998-2002)

All enquiries related to this publication should be referred to:

αβχδ
AVIONIC SYSTEMS

Airport Works, Rochester, Kent. ME1 2XX
England
Tel. 01634 844400 Fax. 01634 816721

D57 – AGP Software Detailed Design

Document No:

Issue DRAFT A

Contains 180 pages total

Comprising:

8	pages front matter
172	pages text and figures

Compiled by:		Title:	Skysoft Software Engineer
	Ricardo Pereira	Date:	
Approved by:		Title:	Skysoft Project Manager
	José Neves	Date:	
Authorised by:		Title:	MA-AFAS Project Manager
	Alfie Hanna	Date:	

LIST OF EFFECTIVE PAGES AND CHANGE HISTORY

Insert latest changed pages. Destroy superseded pages

TOTAL NUMBER OF PAGES IN THIS PUBLICATION IS 180 CONSISTING OF THE FOLLOWING							
Page No.	Date	Issue	DCR	Page No.	Date	Issue	DCR
All							

DISTRIBUTION LIST

This Document is distributed as below.

Additional copies held by unnamed recipients will not be updated.

Paper Copy No.	Name	Address
1	MA-AFAS Library	Avionic Systems
MASTER	Library	BAE SYSTEMS, Rochester
Electronic Copy	**Name**	**Address**
1	European Commission	EC, Brussels
2-20	MA-AFAS Consortium	maafas@bluecoat.org

TABLE OF CONTENTS

1.INTRODUCTION

1.1.Scope

This Software Detailed Design document applies to the MA-AFAS Project AOC Ground Platform (AGP).

1.2.Identification

AGP Software Detailed Design (D57) has been produced by Skysoft Portugal and AMS-UK for the MA-AFAS programme on behalf of BAE Systems.

The objective of this document is to provide a detailed description of MA-AFAS AGP design.

1.3.System Overview

The “More Autonomous Aircraft in the Future ATM System” (MA-AFAS) program is focused on developing CNS-based avionics components that will provide aircraft greater flexibility within the ATM system. The program includes development of an operational concept; specification and implementation of avionics packages, ground systems and infrastructure to demonstrate the operational concept; and trials and further work towards implementation of the concept.

One of the themes of the programme is Airline Operational Centre (AOC). The AOC Ground Platform (AGP) is a trials system that simulates the functions of a ground end system. The AGP supports trials of the MA-AFAS avionics package AOC functions.

1.4.Design Method

AGP high-level design was develop using AxiomSys. Besides providing an overview of the system, the AGP AxiomSys model was also used to define Skysoft and AMS scope of work and work shares for the platform development.

AGP detailed design was implemented using UML (Unified Modeling Language). UML is a set of standard models used to design a object-oriented project, which allows the visualization, specification, construction and documentation of the artifacts of a software intensive system

The detailied design of the AGP application was done in two stages. The first stage of the AGP detailed design is intended to design the central structure and external communications of the AGP. In the second stage, it is intended to complete the detailed design of the AGP core and include the final man-machine interface. This document reflects the detailed design of the overall AGP application and consequently it encompasses both stages of the detailed design phase[1].

[1] There is a document available only applicable for the first stage of AGP design (MA-AFAS AGP Software Design Document, Version 1.1 [AGP SDD]).

2.AGP ARCHITECTURE

The following diagrams provide the high level functional model of the AGP system. Figure 1 presents the AGP context diagram. The AGP context diagram is an abstraction of the AGP and all entities that interact with it. These entities are described below:

- The AOC Operator will be a user that provides inputs to the AGP, and uses the visual data presented on the AGP displays.
- The Aircrafts entity correspond to the set of real aircrafts exchanging messages/data with the AGP.
- The Fleet Simulator is a simulator of a certain number of aircraft and respective flights.

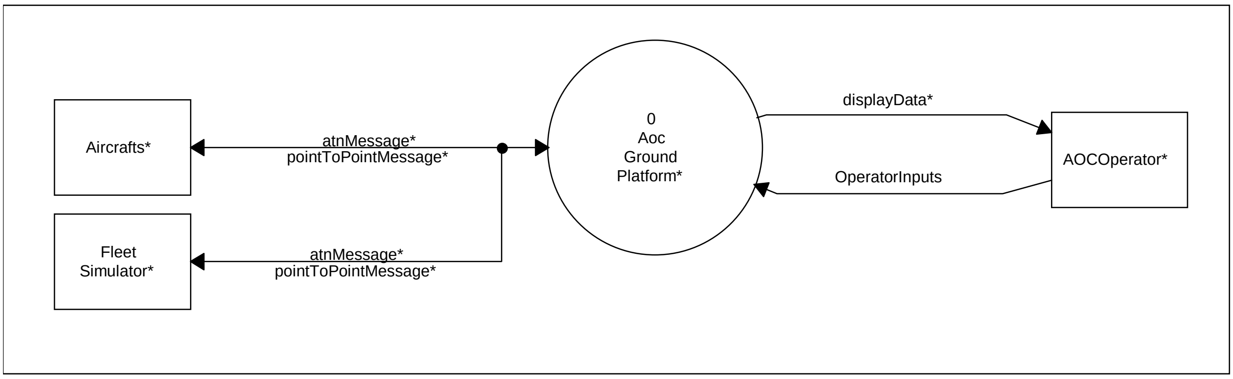


Figure 1 – AGP Context Diagram.

The diagram below (Figure 2) provides the functional breakdown of the AGP system into its three main modules: Communications, AGP Functions and HMI.

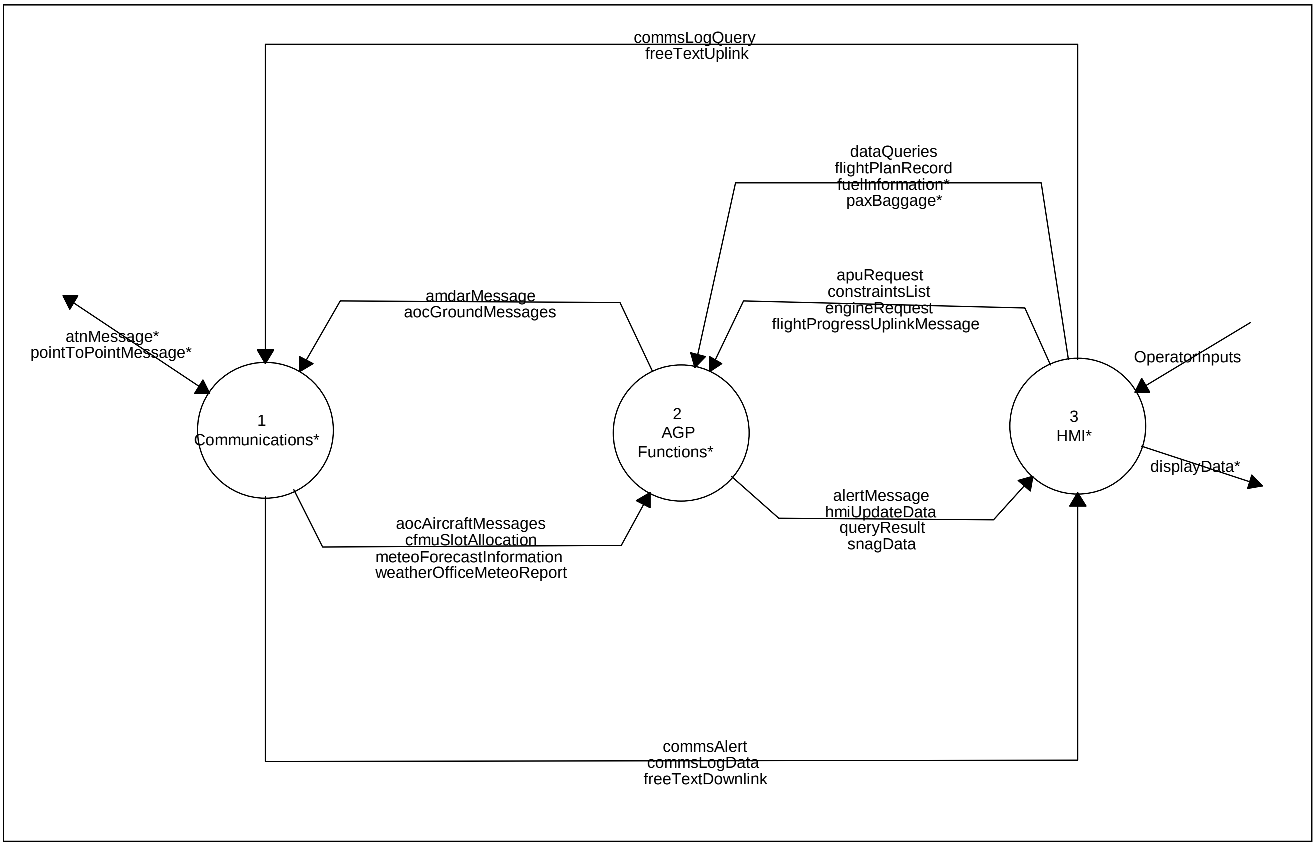
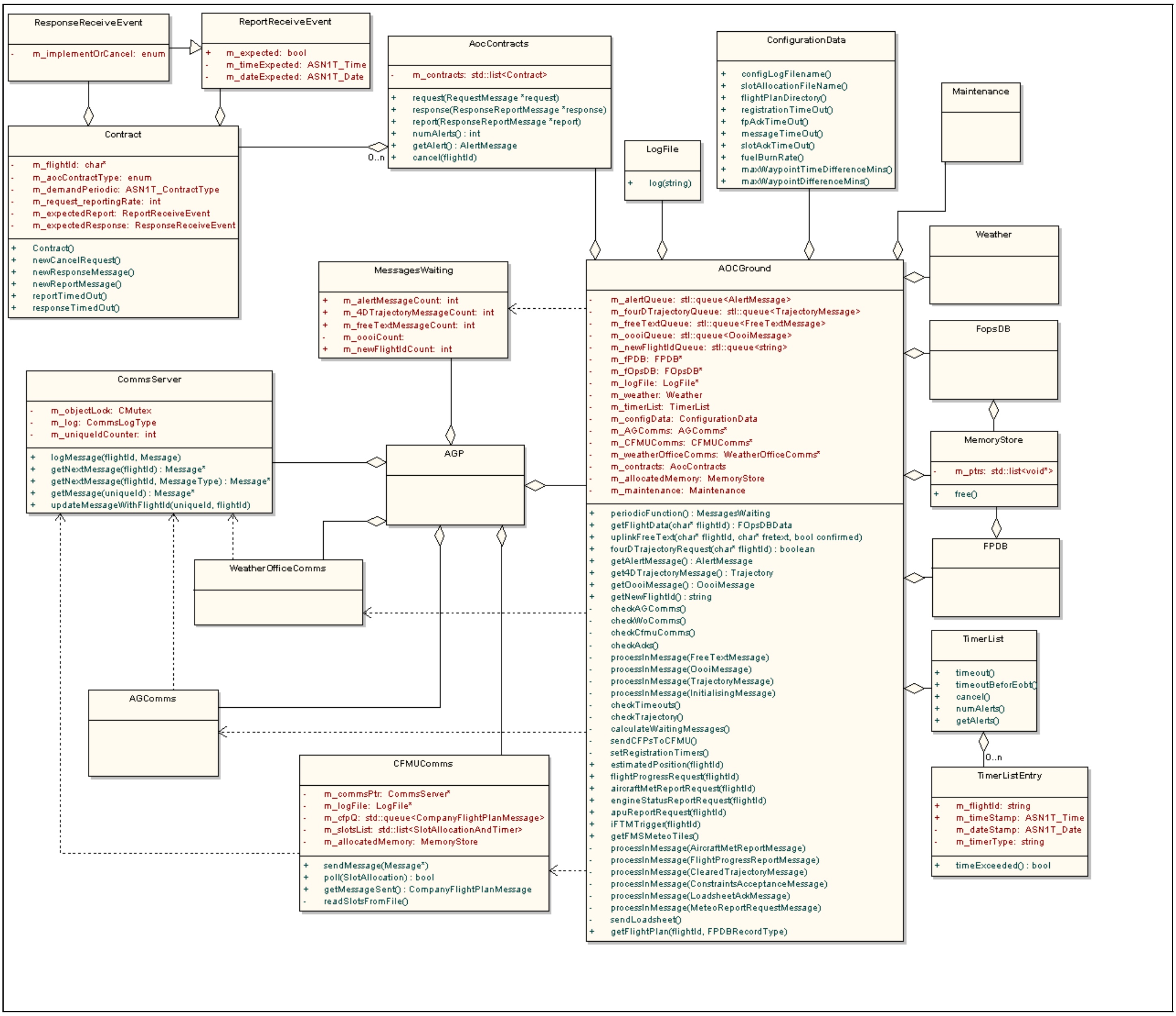


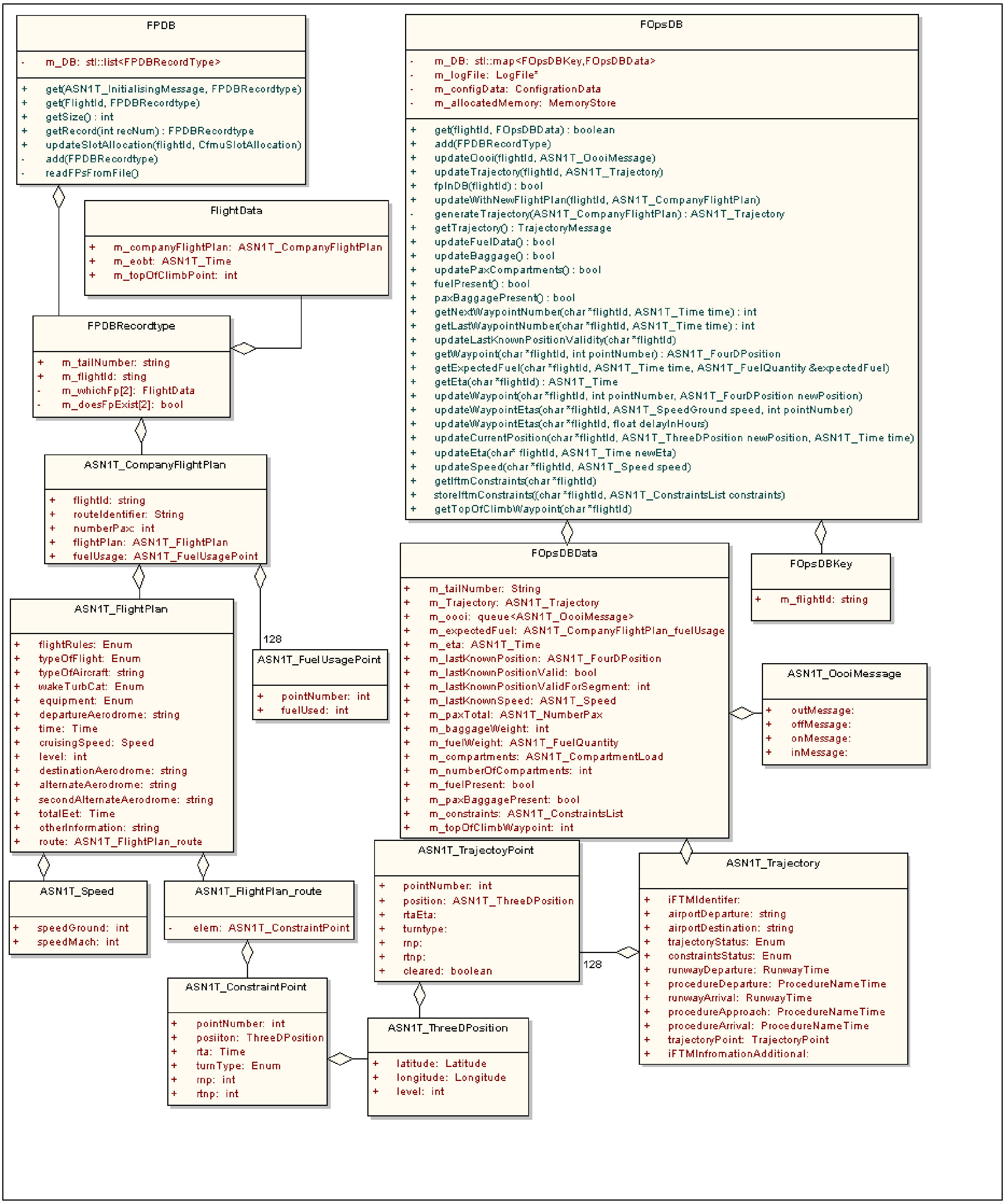
Figure 2 – AGP Functional Breakdown

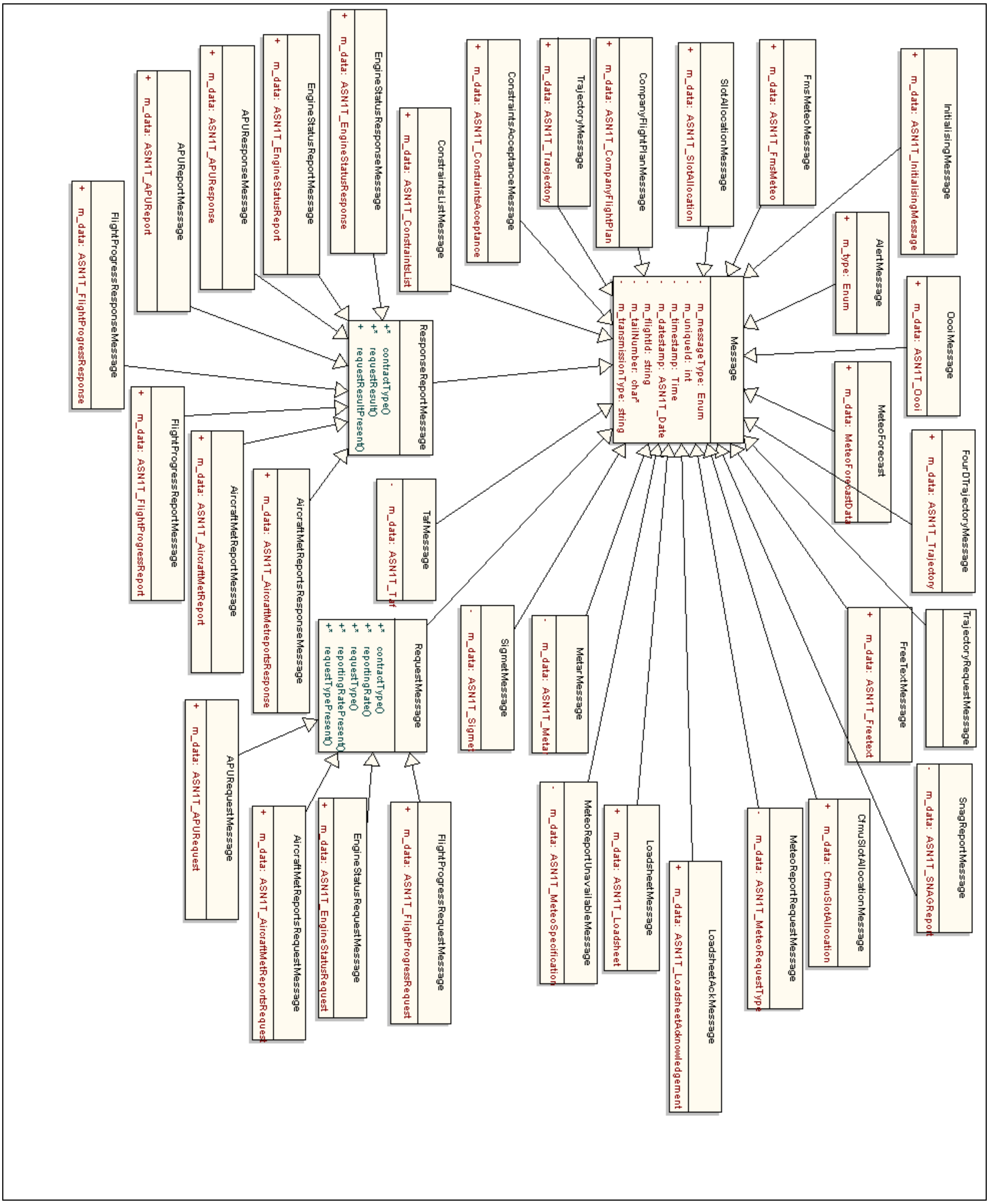
2.1. AGP Class Diagram

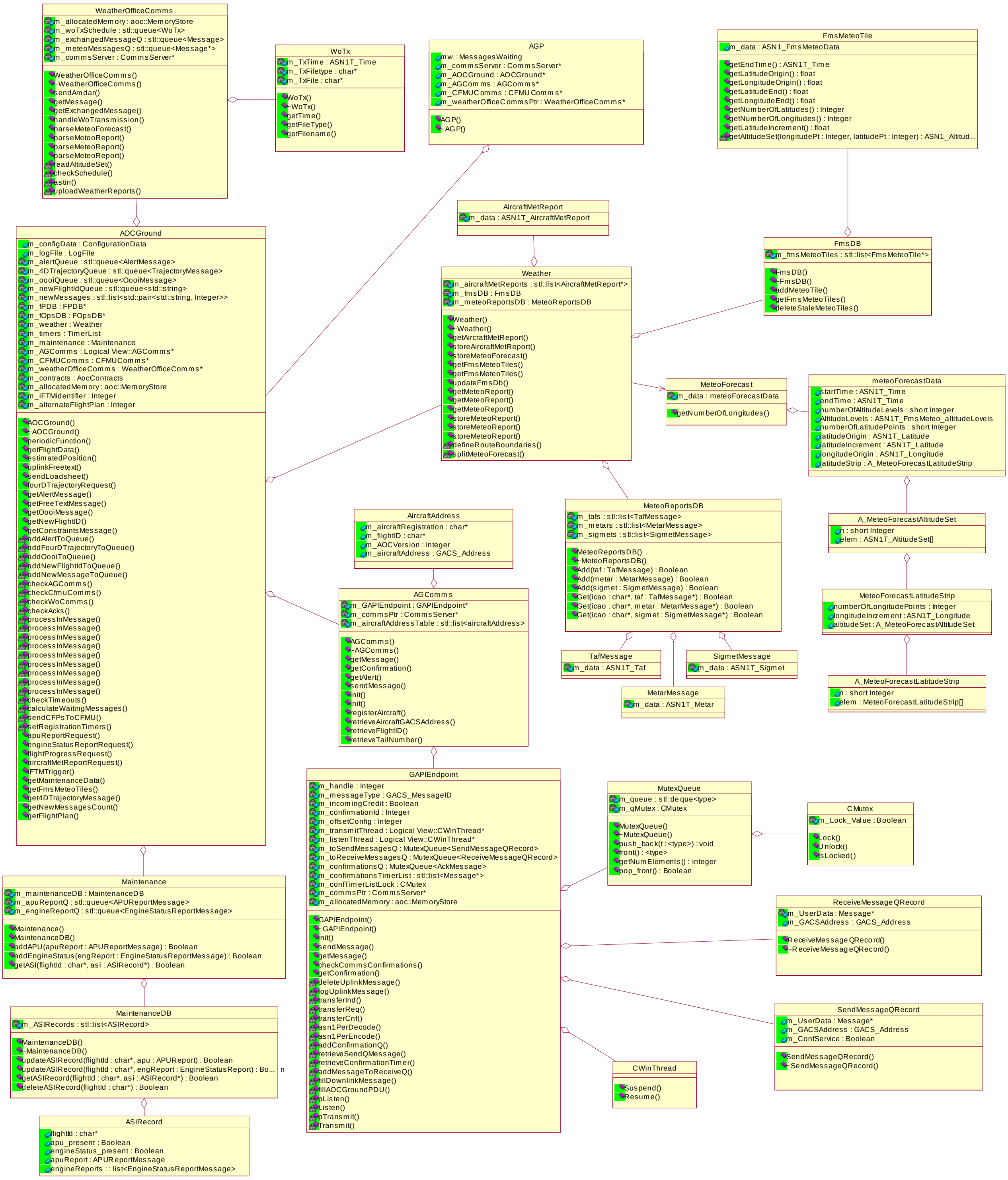
The following diagrams show the detailed design of the AGP expressed as a set of UML class diagrams. The design is split into several separate diagrams, the first being a high level overview of the system, and the subsequent diagrams breaking the design down further, providing more detail.

The author of each class – AMS Frimley or Skysoft Portugal, can be found in section B.1.

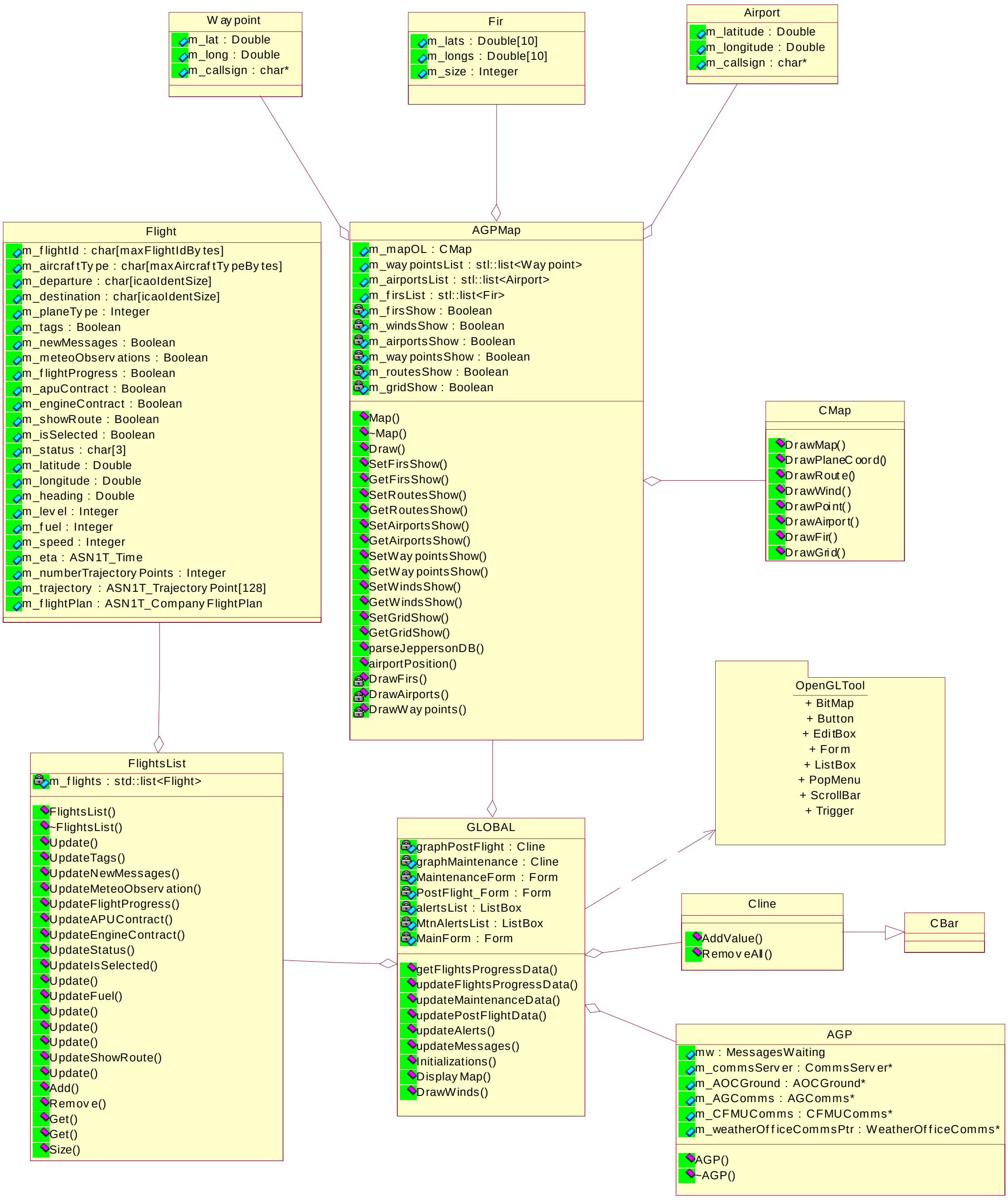

ResponseReceiveEvent
- m_implementOrCancel: enum
ReportReceiveEvent
+ m_expected: bool
- m_timeExpected: ASN1T_Time
- m_dateExpected: ASN1T_Date
AocContracts
- m_contracts: std::list<Contract>
+ request(RequestMessage *request)
+ response(ResponseReportMessage *response)
+ report(ResponseReportMessage *report)
+ numAlerts() : int
+ getAlert() : AlertMessage
+ cancel(flightId)
0..n
Contract
- m_flightId: char*
- m_aocContractType: enum
- m_demandPeriodic: ASN1T_ContractType
- m_request_reportingRate: int
- m_expectedReport: ReportReceiveEvent
- m_expectedResponse: ResponseReceiveEvent
+ Contract()
+ newCancelRequest()
+ newResponseMessage()
+ newReportMessage()
+ reportTimedOut()
+ responseTimedOut()
LogFile
+ log(string)
ConfigurationData
+ configLogFilename()
+ slotAllocationFileName()
+ flightPlanDirectory()
+ registrationTimeOut()
+ fpAckTimeOut()
+ messageTimeOut()
+ slotAckTimeOut()
+ fuelBurnRate()
+ maxWaypointTimeDifferenceMins()
+ maxWaypointDifferenceMins()
Maintenance
Weather
FopsDB
MemoryStore
- m_ptrs: std::list<void*>
+ free()
FPDB
TimerList
+ timeout()
+ timeoutBeforEobt()
+ cancel()
+ numAlerts()
+ getAlerts()
0..n
TimerListEntry
+ m_flightId: string
+ m_timeStamp: ASN1T_Time
- m_dateStamp: ASN1T_Date
- m_timerType: string
+ timeExceeded() : bool
MessagesWaiting
+ m_alertMessageCount: int
+ m_4DTrajectoryMessageCount: int
+ m_freeTextMessageCount: int
- m_oooiCount:
+ m_newFlightIdCount: int
AOCGround
- m_alertQueue: stl::queue<AlertMessage>
- m_fourDTrajectoryQueue: stl::queue<TrajectoryMessage>
- m_freeTextQueue: stl::queue<FreeTextMessage>
- m_oooiQueue: stl::queue<OooiMessage>
- m_newFlightIdQueue: stl::queue<string>
- m_fPDB: FPDB*
- m_fOpsDB: FOpsDB*
- m_logFile: LogFile*
- m_weather: Weather
- m_timerList: TimerList
- m_configData: ConfigurationData
- m_AGComms: AGComms*
- m_CFMUComms: CFMUComms*
- m_weatherOfficeComms: WeatherOfficeComms*
- m_contracts: AocContracts
- m_allocatedMemory: MemoryStore
- m_maintenance: Maintenance
+ periodicFunction() : MessagesWaiting
+ getFlightData(char* flightId) : FOpsDBData
+ uplinkFreeText(char* flightId, char* fretext, bool confirmed)
+ fourDTrajectoryRequest(char* flightId) : boolean
+ getAlertMessage() : AlertMessage
+ get4DTrajectoryMessage() : Trajectory
+ getOooiMessage() : OooiMessage
+ getNewFlightId() : string
- checkAGComms()
- checkWoComms()
- checkCfmuComms()
- checkAcks()
- processInMessage(FreeTextMessage)
- processInMessage(OooiMessage)
- processInMessage(TrajectoryMessage)
- processInMessage(InitialisingMessage)
- checkTimeouts()
- checkTrajectory()
- calculateWaitingMessages()
- sendCFPsToCFMU()
- setRegistrationTimers()
+ estimatedPosition(flightId)
+ flightProgressRequest(flightId)
+ aircraftMetReportRequest(flightId)
+ engineStatusReportRequest(flightId)
+ apuReportRequest(flightId)
+ iFTMTrigger(flightId)
+ getFMSMeteoTiles()
- processInMessage(AircraftMetReportMessage)
- processInMessage(FlightProgressReportMessage)
- processInMessage(ClearedTrajectoryMessage)
- processInMessage(ConstraintsAcceptanceMessage)
- processInMessage(LoadsheetAckMessage)
- processInMessage(MeteoReportRequestMessage)
- sendLoadsheet()
+ getFlightPlan(flightId, FPDBRecordType)
CommsServer
- m_objectLock: CMutex
- m_log: CommsLogType
- m_uniqueIdCounter: int
+ logMessage(flightId, Message)
+ getNextMessage(flightId) : Message*
+ getNextMessage(flightId, MessageType) : Message*
+ getMessage(uniqueId) : Message*
+ updateMessageWithFlightId(uniqueId, flightId)
AGP
WeatherOfficeComms
AGComms
CFMUComms
- m_commsPtr: CommsServer*
- m_logFile: LogFile*
- m_cfpQ: std::queue<CompanyFlightPlanMessage>
- m_slotsList: std::list<SlotAllocationAndTimer>
- m_allocatedMemory: MemoryStore
+ sendMessage(Message*)
+ poll(SlotAllocation) : bool
+ getMessageSent() : CompanyFlightPlanMessage
- readSlotsFromFile()


FPDB
- m_DB: stl::list<FPDBRecordType>
+ get(ASN1T_InitialisingMessage, FPDBRecordtype)
+ get(FlightId, FPDBRecordtype)
+ getSize() : int
+ getRecord(int recNum) : FPDBRecordtype
+ updateSlotAllocation(flightId, CfmuSlotAllocation)
- add(FPDBRecordtype)
- readFPsFromFile()
FlightData
+ m_companyFlightPlan: ASN1T_CompanyFlightPlan
+ m_eobt: ASN1T_Time
+ m_topOfClimbPoint: int
FPDBRecordtype
+ m_tailNumber: string
+ m_flightId: sting
- m_whichFp[2]: FlightData
- m_doesFpExist[2]: bool
ASN1T_CompanyFlightPlan
+ flightId: string
+ routeIdentifier: String
+ numberPax: int
+ flightPlan: ASN1T_FlightPlan
+ fuelUsage: ASN1T_FuelUsagePoint
ASN1T_FlightPlan
+ flightRules: Enum
+ typeOfFlight: Enum
+ typeOfAircraft: string
+ wakeTurbCat: Enum
+ equipment: Enum
+ departureAerodrome: string
+ time: Time
+ cruisingSpeed: Speed
+ level: int
+ destinationAerodrome: string
+ alternateAerodrome: string
+ secondAlternateAerodrome: string
+ totalEet: Time
+ otherInformation: string
+ route: ASN1T_FlightPlan_route
128
ASN1T_FuelUsagePoint
+ pointNumber: int
+ fuelUsed: int
ASN1T_Speed
+ speedGround: int
+ speedMach: int
ASN1T_FlightPlan_route
- elem: ASN1T_ConstraintPoint
ASN1T_ConstraintPoint
+ pointNumber: int
+ posiiton: ThreeDPosition
+ rta: Time
+ turnType: Enum
+ rnp: int
+ rtnp: int
FOpsDB
- m_DB: stl::map<FOpsDBKey,FOpsDBData>
- m_logFile: LogFile*
- m_configData: ConfigurationData
- m_allocatedMemory: MemoryStore
+ get(flightId, FOpsDBData) : boolean
+ add(FPDBRecordType)
+ updateOooi(flightId, ASN1T_OooiMessage)
+ updateTrajectory(flightId, ASN1T_Trajectory)
+ fpInDB(flightId) : bool
+ updateWithNewFlightPlan(flightId, ASN1T_CompanyFlightPlan)
- generateTrajectory(ASN1T_CompanyFlightPlan) : ASN1T_Trajectory
+ getTrajectory() : TrajectoryMessage
+ updateFuelData() : bool
+ updateBaggage() : bool
+ updatePaxCompartments() : bool
+ fuelPresent() : bool
+ paxBaggagePresent() : bool
+ getNextWaypointNumber(char *flightId, ASN1T_Time time) : int
+ getLastWaypointNumber(char *flightId, ASN1T_Time time) : int
+ updateLastKnownPositionValidity(char *flightId)
+ getWaypoint(char *flightId, int pointNumber) : ASN1T_FourDPosition
+ getExpectedFuel(char *flightId, ASN1T_Time time, ASN1T_FuelQuantity &expectedFuel)
+ getEta(char *flightId) : ASN1T_Time
+ updateWaypoint(char *flightId, int pointNumber, ASN1T_FourDPosition newPosition)
+ updateWaypointEtas(char *flightId, ASN1T_SpeedGround speed, int pointNumber)
+ updateWaypointEtas(char *flightId, float delayInHours)
+ updateCurrentPosition(char *flightId, ASN1T_ThreeDPosition newPosition, ASN1T_Time time)
+ updateEta(char* flightId, ASN1T_Time newEta)
+ updateSpeed(char *flightId, ASN1T_Speed speed)
+ getIftmConstraints(char *flightId)
+ storeIftmConstraints((char *flightId, ASN1T_ConstraintsList constraints)
+ getTopOfClimbWaypoint(char *flightId)
FOpsDBData
+ m_tailNumber: String
+ m_Trajectory: ASN1T_Trajectory
+ m_oooi: queue<ASN1T_OooiMessage>
+ m_expectedFuel: ASN1T_CompanyFlightPlan_fuelUsage
+ m_eta: ASN1T_Time
+ m_lastKnownPosition: ASN1T_FourDPosition
+ m_lastKnownPositionValid: bool
+ m_lastKnownPositionValidForSegment: int
+ m_lastKnownSpeed: ASN1T_Speed
+ m_paxTotal: ASN1T_NumberPax
+ m_baggageWeight: int
+ m_fuelWeight: ASN1T_FuelQuantity
+ m_compartments: ASN1T_CompartmentLoad
+ m_numberOfCompartments: int
+ m_fuelPresent: bool
+ m_paxBaggagePresent: bool
+ m_constraints: ASN1T_ConstraintsList
+ m_topOfClimbWaypoint: int
FOpsDBKey
+ m_flightId: string
ASN1T_OooiMessage
+ outMessage:
+ offMessage:
+ onMessage:
+ inMessage:
ASN1T_TrajectoyPoint
+ pointNumber: int
+ position: ASN1T_ThreeDPosition
+ rtaEta:
+ turntype:
+ rnp:
+ rtnp:
+ cleared: boolean
128
ASN1T_Trajectory
+ iFTMIdentifer:
+ airportDeparture: string
+ airportDestination: string
+ trajectoryStatus: Enum
+ constraintsStatus: Enum
+ runwayDeparture: RunwayTime
+ procedureDeparture: ProcedureNameTime
+ runwayArrival: RunwayTime
+ procedureApproach: ProcedureNameTime
+ procedureArrival: ProcedureNameTime
+ trajectoryPoint: TrajectoryPoint
+ iFTMInfromationAdditional:
ASN1T_ThreeDPosition
+ latitude: Latitude
+ longitude: Longitude
+ level: int


Message
- m_messageType: Enum
- m_uniqueId: int
- m_timestamp: Time
- m_datestamp: ASN1T_Date
- m_flightId: string
- m_tailNumber: char*
- m_transmissionType: string
ResponseReportMessage
+* contractType()
+* requestResult()
+ requestResultPresent()
RequestMessage
+* contractType()
+* reportingRate()
+* requestType()
+* reportingRatePresent()
+* requestTypePresent()
InitialisingMessage
+ m_data: ASN1T_InitialisingMessage
FmsMeteoMessage
+ m_data: ASN1T_FmsMeteo
SlotAllocationMessage
+ m_data: ASN1T_SlotAllocation
CompanyFlightPlanMessage
+ m_data: ASN1T_CompanyFlightPlan
TrajectoryMessage
+ m_data: ASN1T_Trajectory
ConstraintsAcceptanceMessage
+ m_data: ASN1T_ConstraintsAcceptance
ConstraintsListMessage
+ m_data: ASN1T_ConstraintsList
EngineStatusResponseMessage
+ m_data: ASN1T_EngineStatusResponse
EngineStatusReportMessage
+ m_data: ASN1T_EngineStatusReport
APUResponseMessage
+ m_data: ASN1T_APUResponse
APUReportMessage
+ m_data: ASN1T_APUReport
FlightProgressResponseMessage
+ m_data: ASN1T_FlightProgressResponse
FlightProgressReportMessage
+ m_data: ASN1T_FlightProgressReport
AircraftMetReportMessage
+ m_data: ASN1T_AircraftMetReport
AircraftMetReportsResponseMessage
+ m_data: ASN1T_AircraftMetreportsResponse
TafMessage
- m_data: ASN1T_Taf
AlertMessage
+ m_type: Enum
OooiMessage
+ m_data: ASN1T_Oooi
MeteoForecast
+ m_data: MeteoForecastData
FourDTrajectoryMessage
+ m_data: ASN1T_Trajectory
FreeTextMessage
+ m_data: ASN1T_Freetext
TrajectoryRequestMessage
SnagReportMessage
- m_data: ASN1T_SNAGReport
CfmuSlotAllocationMessage
+ m_data: CfmuSlotAllocation
MeteoReportRequestMessage
- m_data: ASN1T_MeteoRequestType
LoadsheetAckMessage
+ m_data: ASN1T_LoadsheetAcknowledgement
LoadsheetMessage
+ m_data: ASN1T_Loadsheet
MeteoReportUnavailableMessage
- m_data: ASN1T_MeteoSpecification
MetarMessage
- m_data: ASN1T_Metar
SigmetMessage
- m_data: ASN1T_Sigmet
APURequestMessage
+ m_data: ASN1T_APURequest
AircraftMetReportsRequestMessage
+ m_data: ASN1T_AircraftMetReportsRequest
EngineStatusRequestMessage
+ m_data: ASN1T_EngineStatusRequest
FlightProgressRequestMessage
+ m_data: ASN1T_FlightProgressRequest


WeatherOfficeComms
m_allocatedMemory : aoc::MemoryStore
m_woTxSchedule : stl::queue<WoTx>
m_exchangedMessageQ : stl::queue<Message>
m_meteoMessagesQ : stl::queue<Message*>
m_commsServer : CommsServer*
WeatherOfficeComms()
~WeatherOfficeComms()
sendAmdar()
getMessage()
getExchangedMessage()
handleWoTransmission()
parseMeteoForecast()
parseMeteoReport()
parseMeteoReport()
parseMeteoReport()
readAltitudeSet()
checkSchedule()
astin()
uploadWeatherReports()
WoTx
m_TxTime : ASN1T_Time
m_TxFiletype : char*
m_TxFile : char*
WoTx()
~WoTx()
getTime()
getFileType()
getFilename()
AGP
mw : MessagesWaiting
m_commsServer : CommsServer*
m_AOCGround : AOCGround*
m_AGComms : AGComms*
m_CFMUComms : CFMUComms*
m_weatherOfficeCommsPtr : WeatherOfficeComms*
AGP()
~AGP()
FmsMeteoTile
m_data : ASN1_FmsMeteoData
getEndTime() : ASN1T_Time
getLatitudeOrigin() : float
getLongitudeOrigin() : float
getLatitudeEnd() : float
getLongitudeEnd() : float
getNumberOfLatitudes() : Integer
getNumberOfLongitudes() : Integer
getLatitudeIncrement() : float
getAltitudeSet(longitudePt : Integer, latitudePt : Integer) : ASN1_Altitud...
AircraftMetReport
m_data : ASN1T_AircraftMetReport
FmsDB
m_fmsMeteoTiles : stl::list<FmsMeteoTile*>
FmsDB()
~FmsDB()
addMeteoTile()
getFmsMeteoTiles()
deleteStaleMeteoTiles()
AOCGround
m_configData : ConfigurationData
m_logFile : LogFile
m_alertQueue : stl::queue<AlertMessage>
m_4DTrajectoryQueue : stl::queue<TrajectoryMessage>
m_oooiQueue : stl::queue<OooiMessage>
m_newFlightIdQueue : stl::queue<std::string>
m_newMessages : stl::list<std::pair<std::string, Integer>>
m_fPDB : FPDB*
m_fOpsDB : FOpsDB*
m_weather : Weather
m_timers : TimerList
m_maintenance : Maintenance
m_AGComms : Logical View::AGComms*
m_CFMUComms : CFMUComms*
m_weatherOfficeComms : WeatherOfficeComms*
m_contracts : AocContracts
m_allocatedMemory : aoc::MemoryStore
m_iFTMidentifier : Integer
m_alternateFlightPlan : Integer
AOCGround()
~AOCGround()
periodicFunction()
getFlightData()
estimatedPosition()
uplinkFreetext()
sendLoadsheet()
fourDTrajectoryRequest()
getAlertMessage()
getFreeTextMessage()
getOooiMessage()
getNewFlightID()
getConstraintsMessage()
addAlertToQueue()
addFourDTrajectoryToQueue()
addOooiToQueue()
addNewFlightIdToQueue()
addNewMessageToQueue()
checkAGComms()
checkCfmuComms()
checkWoComms()
checkAcks()
processInMessage()
processInMessage()
processInMessage()
processInMessage()
processInMessage()
processInMessage()
processInMessage()
processInMessage()
processInMessage()
checkTimeouts()
calculateWaitingMessages()
sendCFPsToCFMU()
setRegistrationTimers()
apuReportRequest()
engineStatusReportRequest()
flightProgressRequest()
aircraftMetReportRequest()
IFTMTrigger()
getMaintenanceData()
getFmsMeteoTiles()
get4DTrajectoryMessage()
getNewMessagesCount()
getFlightPlan()
Weather
m_aircraftMetReports : stl::list<AircraftMetReport*>
m_fmsDB : FmsDB
m_meteoReportsDB : MeteoReportsDB
Weather()
~Weather()
getAircraftMetReport()
storeAircraftMetReport()
storeMeteoForecast()
getFmsMeteoTiles()
getFmsMeteoTiles()
updateFmsDb()
getMeteoReport()
getMeteoReport()
getMeteoReport()
storeMeteoReport()
storeMeteoReport()
storeMeteoReport()
defineRouteBoundaries()
splitMeteoForecast()
MeteoForecast
m_data : meteoForecastData
getNumberOfLongitudes()
meteoForecastData
startTime : ASN1T_Time
endTime : ASN1T_Time
numberOfAltitudeLevels : short Integer
AltitudeLevels : ASN1T_FmsMeteo_altitudeLevels
numberOfLatitudePoints : short Integer
latitudeOrigin : ASN1T_Latitude
latitudeIncrement : ASN1T_Latitude
longitudeOrigin : ASN1T_Longitude
latitudeStrip : A_MeteoForecastLatitudeStrip
AircraftAddress
m_aircraftRegistration : char*
m_flightID : char*
m_AOCVersion : Integer
m_aircraftAddress : GACS_Address
MeteoReportsDB
m_tafs : stl::list<TafMessage>
m_metars : stl::list<MetarMessage>
m_sigmets : stl::list<SigmetMessage>
MeteoReportsDB()
~MeteoReportsDB()
Add(taf : TafMessage) : Boolean
Add(metar : MetarMessage) : Boolean
Add(sigmet : SigmetMessage) : Boolean
Get(icao : char*, taf : TafMessage*) : Boolean
Get(icao : char*, metar : MetarMessage*) : Boolean
Get(icao : char*, sigmet : SigmetMessage*) : Boolean
A_MeteoForecastAltitudeSet
n : short Integer
elem : ASN1T_AltitudeSet[]
MeteoForecastLatitudeStrip
numberOfLongitudePoints : Integer
longitudeIncrement : ASN1T_Longitude
altitudeSet : A_MeteoForecastAltitudeSet
A_MeteoForecastLatitudeStrip
n : short Integer
elem : MeteoForecastLatitudeStrip[]
AGComms
m_GAPIEndpoint : GAPIEndpoint*
m_commsPtr : CommsServer*
m_aircraftAddressTable : stl::list<aircraftAddress>
AGComms()
~AGComms()
getMessage()
getConfirmation()
getAlert()
sendMessage()
init()
init()
registerAircraft()
retrieveAircraftGACSAddress()
retrieveFlightID()
retrieveTailNumber()
TafMessage
m_data : ASN1T_Taf
SigmetMessage
m_data : ASN1T_Sigmet
MetarMessage
m_data : ASN1T_Metar
GAPIEndpoint
m_handle : Integer
m_messageType : GACS_MessageID
m_incomingCredit : Boolean
m_confirmationId : Integer
m_offsetConfig : Integer
m_transmitThread : Logical View::CWinThread*
m_listenThread : Logical View::CWinThread*
m_toSendMessagesQ : MutexQueue<SendMessageQRecord>
m_toReceiveMessagesQ : MutexQueue<ReceiveMessageQRecord>
m_confirmationsQ : MutexQueue<AckMessage>
m_confirmationsTimerList : stl::list<Message*>
m_confTimerListLock : CMutex
m_commsPtr : CommsServer*
m_allocatedMemory : aoc::MemoryStore
GAPIEndpoint()
~GAPIEndpoint()
init()
sendMessage()
getMessage()
checkCommsConfirmations()
getConfirmation()
deleteUplinkMessage()
logUplinkMessage()
transferInd()
transferReq()
transferCnf()
asn1PerDecode()
asn1PerEncode()
addConfirmationQ()
retrieveSendQMessage()
retrieveConfirmationTimer()
addMessageToReceiveQ()
fillDownlinkMessage()
fillAOCGroundPDU()
pListen()
Listen()
pTransmit()
Transmit()
MutexQueue
m_queue : stl::deque<type>
m_qMutex : CMutex
MutexQueue()
~MutexQueue()
push_back(t : <type>) : void
front() : <type>
getNumElements() : integer
pop_front() : Boolean
CMutex
m_Lock_Value : Boolean
Lock()
Unlock()
isLocked()
ReceiveMessageQRecord
m_UserData : Message*
m_GACSAddress : GACS_Address
ReceiveMessageQRecord()
~ReceiveMessageQRecord()
SendMessageQRecord
m_UserData : Message*
m_GACSAddress : GACS_Address
m_ConfService : Boolean
SendMessageQRecord()
~SendMessageQRecord()
CWinThread
Suspend()
Resume()
Maintenance
m_maintenanceDB : MaintenanceDB
m_apuReportQ : stl::queue<APUReportMessage>
m_engineReportQ : stl::queue<EngineStatusReportMessage>
Maintenance()
~MaintenanceDB()
addAPU(apuReport : APUReportMessage) : Boolean
addEngineStatus(engReport : EngineStatusReportMessage) : Boolean
getASI(flightId : char*, asi : ASIRecord*) : Boolean
MaintenanceDB
m_ASIRecords : stl::list<ASIRecord>
MaintenanceDB()
~MaintenanceDB()
updateASIRecord(flightId : char*, apu : APUReport) : Boolean
updateASIRecord(flightId : char*, engReport : EngineStatusReport) : Bo...
getASIRecord(flightId : char*, asi : ASIRecord*) : Boolean
deleteASIRecord(flightId : char*) : Boolean
n
ASIRecord
flightId : char*
apu_present : Boolean
engineStatus_present : Boolean
apuReport : APUReportMessage
engineReports : : list<EngineStatusReportMessage>

(The HMI class mentioned in some AMS sequence diagrams is an abstraction of the overall HMI class structure depicted in the class diagram below)



2.2. MutexQueue Class

*2.2.1.*Class Description

<u>This class will be responsible for:</u>
N/A

<u>This class implements:</u>
A mutual exclusion access (read or write) queue.

<u>This class creates other classes:</u>
CMutex[†] *([†] Microsoft Foundation Classes class)*
deque[†] *([†] Standart Template Library class)*

2.2.2.Class Methods

Name	Inputs	Outputs	Comment
MutexQueue			*Constructor*
~MutexQueue			*Destructor*
push_back	Type t		*Adds an element to queue*
front		Type	*Retrieves a reference to the first element in queue*
pop_front		bool	*Remove first element from queue*
getNumElements		int	*Retrieves the number of elements in queue*

2.2.3.Class Attributes

deque<Type> *m_queue*
CMutex *m_qMutex*

2.3.SendMessageQRecord Class

2.3.1.Class Description

This class will be responsible for:
N/A

This class implements:
A record containing a pointer to a message, the GACS address of the message recipient and the requested GACS service.

This class instances other classes:
N/A

2.3.2.Class Methods

Name	Inputs	Outputs	Comment
SendMessageQRecord			*Constructor*
~SendMessageQRecord			*Destructor*

2.3.3.Class Attributes

*Message**	*m_UserData*
GACS_Address	*m_GACSAddress*
bool	*m_ConfService*

2.4.ReceiveMessageQRecord Class

2.4.1.Class Description

This class will be responsible for:
N/A

This class implements:
A record containing a pointer to a message and the message sender GACS address.

This class instances other classes:
N/A

2.4.2.Class Methods

Name	Inputs	Outputs	Comment
ReceiveMessageQRecord			*Constructor*
~ReceiveMessageQRecord			*Destructor*

2.4.3.Class Attributes

*Message**	*m_UserData*
GACS_Address	*m_GACSAddress*

2.5.GAPIEndpoint Class

2.5.1.Class Description

This class will be responsible for:

1. *Opening and bind a GACS endpoint.*
2. *Keeping a list of received messages.*
3. *Keeping messages waiting transmission.*
4. *Transmiting messages to an aircraft type GACS endpoint.*
5. *Listening for GACS events such as: incoming credit, arriving message and arriving confirmation.*
6. *Encoding uplink messages using ASN.1 PER.*
7. *Decoding downlink messages using ASN.1 PER.*
8. *Logging transmitted messages.*
9. *Managing confirmations.*
10. *Checking messages for which confirmation was not received.*
11. *Pausing and resuming communications between the AGP and GACS stack.*

This class implements:
A bi-directional GACS endpoint.

This class instances other classes:
MutexQueue
List[†] ([†] STL class)
CMutex[†] ([†] MFC class)
CWinThread[†] ([†] MFC class)
MemoryStore

2.5.2.Class Methods

Name	Inputs	Outputs	Comment
GAPIEndpoint	CommsServer &commsObj		*Constructor.*
~GAPIEndpoint			*Destructor.*
init	char* agpStationName, Uint8* gacsServer, int endpointMessageType	bool	*This method is responsible for opening and binding a GACS endpoint, and for creating the transmit and listen threads.*
sendMessage	const Message* agMessage, GACS_Address* GACSAddress, bool confServ	bool	*Puts a new message in send messages queue for further sending.*
getMessage		bool, Message*& returnedMessage, GACS_Address* GACSAddress	*Gets last received message from received messages queue.*
checkCommsConfirmations		Message*& notConfirmedMessage	*Searches for a timed out message, and returns it.*
getConfirmation		AckMessage&	*Retrieves last confirmation arrived.*
deleteUplinkMessage	Message* agMessage		*Deletes uplink messages after transmission.*

logUplinkMessage	Message* agMessage		*Logs a message to the Comms Server.*
transferInd			*Calls for GACS primitive G_transferInd, decodes and stores the arrived message.*
transferReq	bool	GACS_Address GACSAddress, char* encodedData, int encodedDataLenght, bool confServ	*Calls for GACS primitive G_transferReq for send a message using GACS.*
transferCnf			*Calls for GACS primitive G_transferCnf.*
asn1PerDecode	GACS_UserData* encodedMessage	bool, Message*& agMessage,	*Decodes a new received message using ASN.1 PER.*
asn1PerEncode	Message* decodedMessage	bool, char*& encodedMessage, int* len	*Encodes an uplink message using ASN.1 PER.*
retrieveSendQMessage		bool, Message*& userData GACS_Address& GACSAddress bool* confServ	*Retrieves a message from m_toSendQMessage.*
addMessageToReceiveQ	GACS_Address* GACSAddress, Message* userData	bool	*Adds a message to m_toReceivedQMessage.*
retrieveConfirmationTimer		bool, int messageIndex, Message*& confirmation	*Retrieves a confirmation timer from m_confirmationsTimerList.*
fillDownlinkMessage	ASN1T_AOCAircraftPDUs* a2gMessage	bool, Message** agMessage	*Fills all message fields based in a received downlink PDU.*
fillAOCGroundPDU	Message* agMessage	ASN1T_AOCGroundPDUs * g2aMessage	*Fills all PDU fields, using a message.*
pListen	void *param	UINT	*Returns a pointer to the function implementing the listen thread loop.*
Listen		UINT	*Listen thread loop function.*
pTransmit	void *param	UINT	*Returns a pointer to the function implementing the transmit thread loop.*
Transmit		UINT	*Transmit thread loop function.*
pause		bool	*Pauses communications between AGP and GACS.*
resume		bool	*Resumes communications between AGP and GACS.*

2.5.3.Class Attributes

int	*m_handle*
*CommsServer**	*m_commsPtr*
MutexQueue<SendMessageQRecord>	*m_toSendMessagesQ*

```
MutexQueue<ReceiveMessageQRecord>   m_toReceiveMessagesQ
list<Message*>                      m_confirmationsTimerList
CMutex                              m_confTimerListLock
bool                                m_incomingCredit
CWinThread                          m_transmitThread
CWinThread                          m_listenThread
GACS_MessageID                      m_messageType
int                                 m_confirmationId
MutexQueue<AckMessage>              m_confirmationsQ
int                                 m_offsetConfig
MemoryStore                         m_allocatedMemory
```

2.6.AircraftAddress Class

2.6.1.Class Description

This class will be responsible for:
N/A

This class implements:
A record containing flight number, tail number, and the A/C GACS address.

This class instances other classes:
N/A

2.6.2.Class Methods

Name	Inputs	Outputs	Comment
AircraftAddress	GACS_Address aircraftAddress,const char* tailNumber, const char* flightId		*Constructor*
AircraftAddress	GACS_Address aircraftAddress, char* tailNumber		*Constructor*
~AircraftAddress			*Destructor*

2.6.3.Class Attributes

*char**	*m_aircraftRegistration*
*char**	*m_flightId*
GACS_Address	*m_aircraftAddress*

2.7.AGComms Class

2.7.1.Class Description

This class will be responsible for:

1. *Generating a GAPIendpoint object.*
2. *Getting received messages from the GAPIendpoint object.*
3. *Sending to-be-transmitted messages to the GAPIendpoint object.*
4. *Generating alerts regarding the non-acknowledgment of messages.*
5. *Maintaining records correlating flightId, tail number and GACS address.*
6. *Pausing and resuming air-ground communications.*

This class implements:
An interface between the AOC ground Platform and the infrastructure supporting the communication with aircraft.

This class instances other classes:
GAPIEndpoint
AircraftAddress

2.7.2.Class Methods

Name	Inputs	Outputs	Comment
AGComms	CommsServer &commsObj		*Constructor*
~AGComms			*Destructor*
getMessage	Message*& agMessageRtn	char* messageId	*Gets a new received message (if it exists) from m_GAPIendpoint object.*
getConfirmation		AckMessage& confirmation	*Gets a confirmation from m_GAPIendpoint object.*
getAlert		AlertMessage &alert	*Returns an alert if a timeout has occurred.*
sendMessage	Message* agMessage, char* messageId		*Passes a message to m_GAPIendpoint in order to be transmitted.*
init	std::string gacsServer, std::string gacsServerPort, std::string agpGacsAddress, int messageType, int gacsConfirmationsTimeout		*Creates a GAPIendpoint object with the given parameters.*
registerAircraft	char* acRegistration	char* flightId	*Inserts flightId into the record with acRegistration as tail number .*
retrieveAircraftGACSAddress	char* flightID	GACS_Address* acAddress	*Gets GACS address based on flightId.*
retrieveFlightId	GACS_Address* acAddress	char* flightId, char* tailNumber	*Gets flightId and tail number based on a provided GACS address.*
retrieveTailNumber	char* flightId	char* tailNumber	*Gets tail number based on a provided flightId.*
PauseEndpoint		bool	*Disables air-ground communications.*

ResumeEndpoint		bool	*Enables air-ground communications.*

2.7.3.Class Attributes

*GAPIEndpoint**	*m_GAPIEndpoint*
*CommsServer**	*m_commsPtr*
list<AircraftAddress>	*m_aircraftAddressTable*

2.8.FmsConverter Class

2.8.1.Class Description

This class will be responsible for:

1. *performing all conversions between ASN.1 data types (latitudes and longitudes) and C/C++ data types.*
2. *supporting a set of operations among ASN.1 latitude and longitude variables, like comparison, sum, subtraction, multiplication*

This class implements:
A set of methods aimed at the manipulation of latitude and longitude values.

This class instances other classes:
N/A

2.8.2.Class Methods

Name	Inputs	Outputs	Comment
FmsConverter			*Constructor*
convertLatitude	float lat	ASN1T_Latitude	*Convert from float to ASN.1 Latitude.*
convertLatitude	ASN1T_Latitude lat	float	*Convert from ASN.1 Latitude to float.*
convertLongitude	float lon	ASN1T_Longitude	*Convert from float to ASN.1 Longitude.*
convertLongitude	ASN1T_Longitude lon	float	*Convert from ASN.1 Longitude to float.*
multiplyLatitude	int factor, ASN1T_Latitude lat	ASN1T_Latitude	*Multiply two ASN.1 Latitudes.*
multiplyLongitude	int factor, ASN1T_Longitude lon	ASN1T_Longitude	*Multiply two ASN.1 Longitudes.*
addLatitude	ASN1T_Latitude lat1, ASN1T_Latitude lat2	ASN1T_Latitude	*Add two ASN.1 Latitudes.*
addLongitude	ASN1T_Longitude lon1, ASN1T_Longitude lon2	ASN1T_Longitude	*Add two ASN.1 Longitudes.*
subtractLatitude	ASN1T_Latitude lat1, ASN1T_Latitude lat2	ASN1T_Latitude	*Subtract two ASN.1 Latitudes.*
subtractLongitude	ASN1T_Longitude lon1, ASN1T_Longitude lon2	ASN1T_Longitude	*Subtract two ASN.1 Longitudes.*
greaterEqual	ASN1T_Latitude lat1, ASN1T_Latitude lat2	bool	*Compare two ASN.1 Latitudes.*
greaterEqual	ASN1T_Longitude lon1, ASN1T_Longitude lon2	bool	*Compare two ASN.1 Longitudes.*
twoDintersection	float y1Min, float y1Max, float x1Min, float x1Max, float y2Min, float y2Max, float x2Min, float x2Max	bool	*Finds if two rectangles intersect each other.*

2.8.3.Class Attributes

N/A

2.9.FmsMeteoTile Class

2.9.1.Class Description

This class will be responsible for:
N/A

This class implements:
A data structure holding forecast and geographical information regarding points located inside a limited airspace region.

This class instances other classes:
ASN1_FmsMeteoData^{+}($^{+}$ Data structure generated by ASN.1 compiler)

2.9.2.Class Methods

Name	Inputs	Outputs	Comment
FmsMeteoTile			*Constructor*
getEndTime		ASN1T_Time	*Returns the end of validity of the data within the meteo tile*
getLatitudeOrigin		float	*Returns the latitude origin of the points reported within the meteo tile*
getLongitudeOrigin		float	*Returns longitude origin of the points reported within the meteo tile*
getLatitudeEnd		float	*Returns latitude end of the points reported within the meteo tile*
getLongitudeEnd		float	*Returns longitude end of the points reported within the meteo tile*
getNumberOfLatitudes		int	*Returns the number of latitudes reported within the meteo tile*
getNumberOfLongitudes		int	*Returns the number of longitudes reported within the meteo tile*
getLatitudeIncrement		float	*Returns the latitude increment between consecutive longitude values*
getAltitudeSet	int longitudePt, int latitudePt		*Returns the data related to one (latitude, longitude) point.*

2.9.3.Class Attributes

ASN1T_FmsMeteo m_data

2.10.FmsDB Class

2.10.1.Class Description

This class will be responsible for:

1. *Adding meteo tiles to the database.*
2. *Getting meteo tiles from the database.*
3. *Deleting meteo tiles from the database.*

This class implements:
A meteo tile database.

This class instances other classes:
FmsMeteoTile

2.10.2.Class Methods

Name	Inputs	Outputs	Comment
FmsDB			*Constructor*
~FmsDB			*Destructor*
deleteStaleMeteoTiles			*Finds and deletes stale meteo tiles.*
getFmsMeteoTiles	float rteLatMin, float rteLatMax, float rteLonMin, float rteLonMax	int, queue<FmsMeteoTile*>& tileArray	*Returns a list of meteo tiles intersecting the flight area of interest.*
addMeteoTile	FmsMeteoTile* meteoTile		*Adds a new meteo tile to meteo tiles list.*

2.10.3.Class Attributes

std::list<FmsMeteoTile>   m_fmsMeteoTiles*

2.11.Weather Class

2.11.1.Class Description.

This class will be responsible for:

1. *Converting meteo forecasts into fms meteo tiles.*
2. *Storing meteo tiles in database.*
3. *Retrieving fms meteo tiles from database.*
4. *Storing meteo reports (TAF, METAR, SIGMET) in database.*
5. *Retrieving meteo reports from the database.*
6. *Storing aircraft meteo reports in database.*
7. *Retrieving aircraft meteo reports from database.*

This class implements:
A module responsible for providing weather information to the AOC ground system.

This class instances other classes:
FmsDB
MeteoReportsDB

2.11.2.Class Methods

Name	Inputs	Outputs	Comment
Weather			*Constructor*
~Weather			*Destructor*
getAircraftMetReport	char* flightId	bool, AircraftMetReportMessage& report	*Gets a A/C meteo report.*
storeAircraftMetReport	AircraftMetReportMessage report		*Stores a received A/C meteo report.*
storeMeteoForecast	MeteoForecast& mtf		*Converts the meteo forecast into fms meteo tiles and stores them in the fms DB*
getFmsMeteoTiles	ASN1T_FlightPlan_route& rte	int, queue<FmsMeteoTile*>& mtfArray	*Gets the fms meteo tiles from the fms DB that have an impact on an A/C route.*
getFmsMeteoTiles	float latMin, float longMin, float latMax, float longMax	int, queue<FmsMeteoTile*>& mtfArray	*Gets the fms meteo tiles from the fms DB intersecting the rectangle defined by the input parameters.*
updateFmsDb			*Checks for stale meteo tiles and deletes them.*
getMeteoReport	ASN1T_MeteoSpecification& presentReports	bool, TafMessage* taf	*Gets a TAF.*
getMeteoReport	ASN1T_MeteoSpecification& presentReports	bool, MetarMessage* metar	*Gets a METAR.*
getMeteoReport	ASN1T_MeteoSpecification& presentReports	bool, SigmetMessage* sigmet	*Gets a SIGMET.*

storeMeteoReport	TafMessage taf		*Stores a TAF in the meteo reports DB.*
storeMeteoReport	MetarMessage metar		*Stores a METAR in the meteo reports DB.*
storeMeteoReport	SigmetMessage sigmet		*Stores a SIGMET in the meteo reports DB.*
defineRouteBoundaries	ASN1T_FlightPlan_route& rte, float& latMin, float& latMax, float& lonMin, float& lonMax		*Retrieves the minimum and maximum coordinates of the waypoints present in an A/C route.*
splitMeteoForecast	MeteoForecast& mtf, int initialLatitudePt, int finalLatitudePt, int initialLongitudePt, int finalLongitudePt	FmsMeteoTile*	*Splits the meteo forecast message and retrieves the meteo tile starting on initialLatitudePt and initialLongitudePt and ending on finalLatitudePt and finalLongitudePt.*

2.11.3.Class Attributes

FmsDB	*m_fmsDB*
list<AircraftMetReports>*	*m_aircraftMetReports*
MeteoReportsDB	*m_meteoReportsDB*

2.12.WoTx Class

2.12.1.Class Description

This class will be responsible for:
N/A

This class implements:
A record containing the time of transmission, type and concerning location of a weather office report.

This class instances other classes:
N/A

2.12.2.Class Methods

Name	Inputs	Outputs	Comment.
WoTx			*Constructor.*
~WoTx			*Destructor.*
getTime		ASN1T_Time	*Returns time of transmission.*
getFiletype		char*	*Returns type of scheduled file.*
getFilename		char*	*Returns name of scheduled file.*

2.12.3.Class Attributes

ASN1T_Time	*m_TxTime*
*char**	*m_TxFiletype*
*char**	*m_TxFilename*

2.13.WeatherOfficeComms Class

2.13.1.Class Description

This class will be responsible for:

1. *Handling the reception of meteorological data from the weather office.*
2. *Implementing a schedule simulating the transmission of meteorological reports from the weather office.*
3. *Logging with the Comms Server the received meteo reports.*
4. *Keeping a list of messages exchanged with the weather office.*

This class implements:
An Interface between the AOC ground platform and a source of meteorological data.

This class instances other classes:
WoTx
Message
MemoryStore

2.13.2.Class Methods

Name	Inputs	Outputs	Comment
WeatherOfficeComms	CommsServer* commsServer		*Constructor*
~WeatherOfficeComms			*Destructor*
sendAmdar	AmdarMessage amdar		*Sends an AMDAR message*
getMessage		bool, Message* &message char* messageId	*Returns a message transmitted by the weather office*
getExchangedMessage		bool, Message* ptr	*Retrieves a message from m_exchangedMessageQ.*
handleWoTransmission	char* woMsgFilename, char* woMsgFiletype, char* messageId	bool, Message*& message	*Handles and logs to the Comms Server a message that was transmitted by the weather office.*
parseMeteoForecast	char* woMsgFilename	int, meteoForecastData& mtf	*Parses a meteo forecast file received from the weather office and fills meteo forecast uplink message.*
readAltitudeSet	char line[MAX_LINE_LEN], int no_altitudes	ASN1T_AltitudeSet &altitudeSet	*Reads altitude sets from a UK Meteo Office formatted file.*
parseMeteoReport	char* woMsgFilename	TAFMessage& taf	*Reads a TAF message from a standard file.*
parseMeteoReport	char* woMsgFilename	METARMessage& metar	*Reads a METAR message from a standard file.*
parseMeteoReport	char* woMsgFilename	SIGMETMessage& sigmet	*Reads a SIGMET message from a standard file.*

checkSchedule		bool, char* woMsgFilename, char* woMsgFiletype	*Returns the filename and type of a scheduled file.*
astin	char* str, short len	int	*Converts a string to a signed integer.*
uploadWeatherReports			*Reads all meteo reports from a folder to m_meteoMessagesQ.*

2.13.3.Class Attributes

queue<WoTx>	*m_woTxSchedule*
queue<Message>	*m_exchangedMessageQ*
*CommsServer**	*m_commsServer*
queue<Message>*	*m_meteoMessagesQ*
MemoryStore	*m_allocatedMemory*

2.14.MeteoReportsDB Class

2.14.1.Class Description

<u>This class will be responsible for:</u>

1.Adding meteo reports to the database.
2.Getting meteo reports from the database.

<u>This class implements:</u>

A meteo reports database, which is formed by three distinct lists, one for each type of meteo report message.

<u>This class instances other classes:</u>

list (STL class)
TafMessage
MetarMessage
SigmetMessage

2.14.2.Class Methods

Name	Inputs	Outputs	Comment
MeteoReportsDB			*Constructor.*
~MeteoReportsDB			*Destructor.*
Add	TafMessage taf	bool	*Adds a TafMessage to m_tafs list.*
Add	MetarMessage metar	bool	*Adds a MetarMessage to m_metars list.*
Add	SigmetMessage sigmet	bool	*Adds a SigmetMessage to m_sigmets list.*
Get	char* icao	bool, TafMessage* taf	*Gets a TafMessage from m_tafs list.*
Get	char* icao	bool, MetarMessage* metar	*Gets a MetarMessage from m_metars list.*
Get	char* icao	bool, SigmetMessage* sigmet	*Gets a SigmetMessage from m_sigmets list.*

2.14.3.Class Attributes

list<TafMessage>	*m_tafs*
list<MetarMessage>	*m_metars*
list<SigmetMessage>	*m_sigmets*

2.15.ASIRecord Class

2.15.1.Class Description

This class will be responsible for:

1.Storing a maintenance flight record

This class implements:

A datatype able to store the latest APU report and all engine status reports received during a flight.

This class instances other classes:

APUReportMessage
EngineStatusReportMessage

2.15.2.Class Methods

N/A

2.15.3.Class Attributes

char[maxFlightIdBytes]	*m_flightId*
bool	*m_apuPresent*
APUReportMessage	*m_apuReport*
bool	*m_engineStatusPresent*
list<EngineStatusReportMessage>	*m_engineReports*

2.16.MaintenanceDB Class

2.16.1.Class Description

This class will be responsible for:

1. *Creating an ASIRecord for each registered flight.*
2. *Retrieving an ASIRecord.*
3. *Deleting an ASIRecord.*

This class implements:
A database containing ASI records..

This class instances other classes:
ASIRecord

2.16.2.Class Methods

Name	Inputs	Outputs	Comment
MaintenanceDB			*Constructor.*
~MaintenanceDB			*Destructor.*
updateASIRecord	char* flightId , APUReport apu	bool	*Searches and updates an ASI record with regard to the latest received APU report.*
updateASIRecord	char* flightId, EngineStatusReport engReport	bool	*Searches and updates an ASI record with regard to the list of received engine reports.*
getASIRecord	char* flightId	bool, ASIRecord asi	*Retrieves the ASI record corresponding to flightId.*
deleteASIRecord	char* flightId	bool,	*Deletes an ASI record.*

2.16.3.Class Attributes

List<ASIRecord>   m_ASIRecords

2.17.Maintenance Class

2.17.1.Class Description

This class will be responsible for:

1.Adding APU and Engine Status reports to the maintenance database.
2.Retrieving an ASI Record from the maintenance database.

This class implements:
An interface between the AGP and the maintenance database.

This class instances other classes:
MaintenanceDB

2.17.2.Class Methods

Name	Inputs	Outputs	Comment
Maintenance			*Constructor.*
~Maintenance			*Destructor.*
addApuReport	APUReportMessage apuReport	bool	*Adds an APU report to the maintenance database.*
addEngineStatusReport	EngineStatus ReportMessage engReport	bool	*Adds an engine status report to the maintenance database.*
getASI	char* flightId	ASIRecord* asi	*Retrieves an ASI record from the maintenance database.*

2.17.3.Class Attributes

MaintenanceDB *m_maintenanceDB*

2.18.AGPMap Class

2.18.1.Class Description

This class will be responsible for:

1.Drawing the AGP map.
2.Setting “show” flags.
3.Retrieving “show” flags.
4.Drawing FIRs, airports, and waypoints.
5.Retrieving airport position.

This class implements:
An interface between AGP and Skysoft OpenGL CMap classes.

This class instances other classes:
Fir
Waypoint
Airport

2.18.2.Class Methods

Name	Inputs	Outputs	Comment
AGPMap			*Constructor.*
~AGPMap			*Destructor.*
Draw			*Draws the AGP map.*
SetFirsShow	bool value		*Sets show FIRs flag.*
GetFirsShow		bool	*Retrieves show FIRs flag.*
SetRoutesShow	bool value		*Sets show routes flag.*
GetRoutesShow		bool	*Retrieves show routes flag.*
SetAirportsShow	bool value		*Sets show airports flag.*
GetAirportsShow		bool	*Retrieves show airports flag.*
SetWaypointsShow	bool value		*Sets show waypoints flag.*
GetWaypointsShow		bool	*Retrieves show waypoints flag*
SetWindsShow	bool value		*Sets show winds flag.*
GetWindsShow		bool	*Retrieves show winds flag.*
SetGridShow	bool value		*Sets show grid flag.*
GetGridShow		bool	*Retrieves show grid flag.*
parseJeppersonDB	char* filename	bool	*Reads FIRs, airports, waypoints data from parsed Jepperson database files.*
airportPosition	char* airport	bool, ASN1_ThreeDPosition &position	*Retrieves airports position based on airport ICAO code.*
DrawFirs			*Draws FIRs.*
DrawAirports			*Draws airports.*
DrawWaypoints			*Draws waypoints.*

SetFirColours	float* colours		*Sets the RBG components of the colour used to draw FIRs.*

2.18.3.Class Attributes

Cmap	*m_mapOL*
list<Waypoint>	*m_waypointsList*
list<Airport>	*m_airportsList*
list<Fir>	*m_firsList*
bool	*m_firsShow*
bool	*m_windsShow*
bool	*m_airportsShow*
bool	*m_waypointsShow*
bool	*m_routesShow*
bool	*m_gridShow*
float	*FirColours[3]*

2.19.AGP Class

2.19.1.Class Description

This class will be responsible for:

1.Storing pointers to AGP “core” classes.

This class implements:

An interface between the AGP processing core and the AGP HMI.

This class instances other classes:

MessagesWaiting

2.19.2.Class Methods

Name	Inputs	Outputs	Comment
AGP			*Constructor.*
~AGP			*Destructor.*

2.19.3.Class Attributes

MessagesWaiting	*mw*
*CommsServer**	*m_commsServer*
*AOCGround**	*m_AOCGround*
*AGComms**	*m_AGComms*
*CFMUComms**	*m_CFMUComms*
*WeatherOfficeComms**	*m_weatherOfficeCommsPtr*

2.20.Flight Class

2.20.1.Class Description

This class will be responsible for:
N/A

This class implements:
A flight as viewed by the HMI, i.e., an object containing all the flight related information the HMI needs to update the AGP displays.

This class instances other classes:
N/A

2.20.2.Class Methods

Name	Inputs	Outputs	Comment
Flight			*Constructor.*
~Flight			*Destructor.*

2.20.3.Class Attributes

char	*m_flightId[maxFlightIdBytes]*
char	*m_aircraftType[maxAircraftTypeBytes]*
char	*m_departure[icaoIdentSize]*
char	*m_destination[icaoIdentSize]*
int	*m_planeType*
bool	*m_tags*
bool	*m_newMessages*
bool	*m_meteoObservations*
bool	*m_flightProgress*
bool	*m_apuContract*
bool	*m_engineContract*
bool	*m_showRoute*
bool	*m_isSelected*
char	*m_status[3]*
double	*m_latitude*
double	*m_longitude*
double	*m_heading*
int	*m_level*
int	*m_fuel*
int	*m_speed*
ASN1T_Time	*m_eta*
int	*m_numberTrajectoryPoints*
ASN1T_TrajectoryPoint	*m_trajectoryElem[128]*
ASN1T_CompanyFlightPlan	*m_flightPlan*

2.21.FlightsList Class

2.21.1.Class Description

This class will be responsible for:

1. *Updating the HMI flights database.*
2. *Inserting a new flight in the HMI flights database.*
3. *Removing a flight from the HMI flights database.*
4. *Retrieving a flight from the HMI flights database*

This class implements:
A database containing all the flights that are shown in the AGP HMI.

This class instances other classes:
Flight

2.21.2.Class Methods

Name	Inputs	Outputs	Comment
FlightsList			*Constructor.*
~FlightsList			*Destructor.*
Update	char* flightId, char* aircraftType	bool	*Updates aircraft type.*
UpdateNewMessages	char* flightId, bool newMesssages	bool	*Checks if new messages have arrived.*
UpdateMeteoObservations	char* flightId, bool meteoObservations	bool	*Updates aircraft meteo periodic contract flag.*
UpdateFlightProgress	char* flightId, bool flightProgress	bool	*Updates flight progress periodic contract flag.*
UpdateAPUContract	char* flightId, bool apu	bool	*Updates APU periodic contract flag.*
UpdateEngineContract	char* flightId, bool eng	bool	*Updates engine status report periodic contract flag.*
UpdateStatus	char* flightId, char* status	bool	*Updates flight flying status.*
UpdateIsSelected	char* flightId, bool isSelected	bool	*Updates the "is selected" flag.*
Update	char* flightId, double latitude, double longitude, double heading, int level	bool	*Updates flight position, flight heading and flight level.*
UpdateFuel	char* flightId, int fuel	bool	*Updates the value of the amout of fuel still available.*
Update	char* flightId, ASN1T_Time eta	bool	*Updates flight ETA.*
Update	char* flightId, ASN1_Trajectory trajectory	bool	*Updates trajectory.*
Update	char* flightId, ASN1_CompanyFlightPlan flightPlan	bool	*Updates flight plan.*
Update	char* flightId, int speed	bool	*Updates current speed.*

UpdateShowRoute	char* flightId, bool showRoute	bool	*Updates draw route flag.*
Add	Flight flight	bool	*Inserts a new flight in the flights list.*
Remove	char* flightId	bool	*Removes a flight from the flights list.*
Get	char* flightId, Flight& flight	bool	*Retrieves a flight from the flights list based on the flight id.*
Get	int element, Flight& flight	bool	*Retrieves a flight from the flights list based on the provided list index.*
Size		Int	*Returns the number of elements within the flights list.*

2.21.3.Class Attributes

list<Flight>   m_flights

2.22.Fir Class

2.22.1.Class Description

This class will be responsible for:
N/A

This class implements:
A datatype able to store a set of points defining a FIR.

This class instances other classes:
N/A

2.22.2.Class Methods

N/A

2.22.3.Class Attributes

double	*m_lats[]*
double	*m_longs[]*
int	*m_size*

2.23.Waypoint Class

2.23.1.Class Description

This class will be responsible for:
N/A

This class implements:
A datatype able to store a waypoint.

This class instances other classes:
N/A

2.23.2.Class Methods

N/A

2.23.3.Class Attributes

double	*m_lat*
double	*m_long*
char	*m_callSign[]*

2.24.Airport Class

2.24.1.Class Description

This class will be responsible for:
N/A

This class implements:
A datatype able to store an airport location, ICAO code and zoom level after which the airport becomes visible.

This class instances other classes:
N/A

2.24.2.Class Methods

N/A

2.24.3.Class Attributes

double	*m_lat*
double	*m_long*
char	*m_callSign[]*
int	*zoom_factor*

2.25.TAFMessage Class

2.25.1.Class Description

This class will be responsible for:

1.*Holding the data for a TAF message.*

This class implements:

A TAF message object.

This class instances other classes:

Public inheritance from the Message class.

2.25.2.Class Methods

Name	Inputs	Outputs	Comment
TAFMessage			*Constructor*

2.25.3.Class Attributes

ASN1T_Taf m_data

2.26.METARMessage Class

2.26.1.Class Description

This class will be responsible for:

1.*Holding the data for a METAR message.*

This class implements:

A METAR message object.

This class instances other classes:

Public inheritance from the Message class.

2.26.2.Class Methods

Name	Inputs	Outputs	Comment
METARMessage			*Constructor*

2.26.3.Class Attributes

ASN1T_Metar m_data

2.27.SIGMETMessage Class

2.27.1.Class Description

This class will be responsible for:

1.*Holding the data for a SIGMET message.*

This class implements:

A SIGMET message object.

This class instances other classes:

Public inheritance from the Message class.

2.27.2.Class Methods

Name	Inputs	Outputs	Comment
SIGMETMessage			*Constructor*

2.27.3.Class Attributes

ASN1T_Sigmet m_data

2.28.A_MeteoForecastAltitudeSet Class

2.28.1.Class Description

This class will be responsible for:
N/A

This class implements:
A set of ASN1T_AltitudeSet elements.

This class instances other classes:
N/A

2.28.2.Class Methods

N/A

2.28.3.Class Attributes

short int	*n*
ASN1T_AltitudeSet	*elem[1000]*

2.29.MeteoForecastData Class

2.29.1.Class Description

This class will be responsible for:
N/A

This class implements:
A datatype able to store meteo forecast data.

This class instances other classes:
N/A.

2.29.2.Class Methods

N/A

2.29.3.Class Attributes

ASN1T_Time	*startTime*
ASN1T_Time	*endTime*
short int	*numberOfAltitudeLevels*
ASN1T_FmsMeteo_altitudeLevels	*altitudeLevels*
short int	*numberOfLatitudePoints*
ASN1T_Latitude	*latitudeOrigin*
ASN1T_Latitude	*latitudeIncrement*
short int	*numberOfLongitudePoints*
A_MeteoForecastAltitudeSet	*altitudeSet*

2.30.MeteoForecast Class

2.30.1.Class Description

This class will be responsible for:
N/A

This class implements:
A meteo forecast message received from a weather office.

This class instances other classes:
Public inheritance from the Message class.

2.30.2.Class Methods

Name	Inputs	Outputs	Comment
MeteoForecast			*Constructor*
~MeteoForecast			*Destructor*
getNumberOfLongitudes		int	*Retrieves the number of longitudes in meteo forecast data.*

2.30.3.Class Attributes

meteoForecastData m_data

2.31.AckMessage Class

2.31.1.Class Description

This class will be responsible for:
N/A

This class implements:
A structure used to pass GACS acknowledgments received by the AGComms class to the AOC Ground class.

This class instances other classes:
N/A

2.31.2.Class Methods

Name	Inputs	Outputs	Comment
AckMessage			*Constructor*
AckMessage	const char* const flightId, const AckMessageType ackType		*Constructor*
~ AckMessage			*Destructor*
getAckMessageType		AckMessageType	*Returns the type of the message for which the acknowledgment was transmitted.*
getFlightId		char*	*Returns the flightId of the A/C that sent the ackowledgment.*

2.31.3.Class Attributes

AckMessageType	*m_ackMessageType*
char	*m_flightId[maxFlightIdBytes]*

2.32.AlertMessage Class

2.32.1.Class Description

This class will be responsible for:

1.Matching an alert with the text that is displayed to the user or that alert.

This class implements:
A structure holding details of a generated alert.

This class instances other classes:
public inheritance from the Message class.

2.32.2.Class Methods

Name	Inputs	Outputs	Comment
AlertMessage			*Constructor*
~AlertMessage			*Destructor*
setAlertId	AlertType		*Sets the type of alert*
getAlertId		AlertType	*Returns the type of alert*
getAlertString			*Returns the string associated with a given alert type.*

2.32.3.Class Attributes

AlertType m_alertId

2.33.MemoryStore Class

2.33.1.Class Description

<u>This class will be responsible for:</u>

1. *Allocating dynamic memory*
2. *Storing details of the memory allocated*
3. *Deallocating memory previously allocated*

<u>This class implements:</u>
A module capable of managing dynamic memory.

<u>This class instances other classes:</u>
N/A

2.33.2.Class Methods

Name	Inputs	Outputs	Comment
MemoryStore			*Constructor*
~MemoryStore			*Destructor*
free()			*Frees memory assigned to this store.*

2.33.3.Class Attributes

*std::list<void *>   m_ptrs*

2.34.AocContracts Class

2.34.1.Class Description

This class will be responsible for:

1.Managing the generation of alerts for contract based messages.

This class implements:

A module that stores details of contracts and generates any necessary alerts

This class instances other classes:

Contract
AlertMessage

2.34.2.Class Methods

Name	Inputs	Outputs	Comment
AocContracts			*Constructor*
~ AocContracts			*Destructor*
request	RequestMessage *request		*Called when a request message sent*
response	ResponseReportMessage *response		*Called when a response message received*
report	ResponseReportMessage *report		*Called when a report received*
numAlerts		int	*Generates alerts and returns the number of times to call getAlert()*
getAlert		AlertMessage	*Retrieve an alert message*
cancel	char* flightId		*Cancel all contracts for a flight*

2.34.3.Class Attributes

ContractListType m_contracts
std::queue<AlertMessage> m_alertQ

2.35.Contract Class

2.35.1.Class Description

This class will be responsible for:

1.Holding details of an individual contract

This class implements:

An object capable of holding all details of a contract.

This class instances other classes:

ReportReceiveEvent
ResponseReceiveEvent

2.35.2.Class Methods

Name	Inputs	Outputs	Comment
Contract			*Constructor*
~Contract			*Destructor*
responseTimedOut		AlertMessage &alert, bool &deleteMe	*Call to check if a response not received in time, Alert generated. This object should be deleted if flag is true.*
reportTimedOut		AlertMessage &alert, bool &deleteMe	*Call to check if a report not received in time, Alert generated. This object should be deleted if flag is true.*
flightId		char*	*FlightId of flight this contract applies to.*
contractType		AocContractType	*Contract Type : apu, engine, flight progress / air met rep.*
demandOrPeriodic		ASN1T_ContractType	*Returns demand or periodic*
newCancelRequest			*Called when a request to cancel has been sent*
newResponseMessage	ASN1T_ContractRequestResult result	bool &alertGenerated, AlertMessage &alert	*Called when a response has been sent. Can generate an alert.*
newReportMessage		bool	*Called when a report is received*
startPeriodicReports			*Starts the timing of periodic reports.*

2.35.3.Class Attributes

char	*m_flightId[maxFlightIdBytes]*
AocContractType	*m_aocContractType*
ASN1T_ContractType	*m_demandPeriodic*
int	*m_request_reportingRate*
ReportReceiveEvent	*m_expectedReport*
ResponseReceiveEvent	*m_expectedResponse*

2.36.ResponseReceiveEvent Class

2.36.1.Class Description

This class will be responsible for:

1.Recording the time that a response is expected

This class implements:

A module that stores time and date of a response receive event

This class instances other classes:

Public Inheritance from Message.

2.36.2.Class Methods

Name	Inputs	Outputs	Comment

2.36.3.Class Attributes

ImplementOrCancel m_implementOrCancel

2.37.ReportReceiveEvent Class

2.37.1.Class Description

This class will be responsible for:

1.Recording the time that a report is expected

This class implements:

A module that stores time and date of a report receive event

This class instances other classes:

N/A

2.37.2.Class Methods

Name	Inputs	Outputs	Comment

2.37.3.Class Attributes

bool	*m_expected*
ASN1T_Time	*m_timeExpected*
ASN1T_Date	*m_dateExpected*

2.38.FlightData Class

2.38.1.Class Description

This class will be responsible for:

1.Holding a flight plan, an EOBT and a top of climb point.

This class implements:
A module to store above data.

This class instances other classes:
N/A

2.38.2.Class Methods

Name	Inputs	Outputs	Comment

2.38.3.Class Attributes

ASN1T_CompanyFlightPlan	*m_companyFlightPlan*
ASN1T_Time	*m_eobt*
int	*m_topOfClimbPoint*

2.39.LoadsheetMessage Class

2.39.1.Class Description

This class will be responsible for:

1.Holding the data for a loadsheet message.

This class implements:

A loadsheet message object.

This class instances other classes:

Public Inheritance from Message.

2.39.2.Class Methods

N/A

2.39.3.Class Attributes

ASN1T_Loadsheet m_data

2.40.LoadsheetAckMessage Class

2.40.1.Class Description

This class will be responsible for:

1.Holding the data for a LoadsheetAck message.

This class implements:

A LoadsheetAck message object.

This class instances other classes:

Public Inheritance from Message.

2.40.2.Class Methods

N/A

2.40.3.Class Attributes

ASN1T_LoadsheetAck m_data

2.41.FlightProgressRequestMessage Class

2.41.1.Class Description

This class will be responsible for:

1.Holding the data for a Flight Progress Request message.

This class implements:
A Flight Progress Request Message object.

This class instances other classes:
Public Inheritance from RequestMessage.

2.41.2.Class Methods

Name	Inputs	Outputs	Comment
FlightProgressRequestMessage			*Constructor*
contractType		ASN1T_ContractType	*Returns the contract type of the object.*
reportingRate		ASN1T_ReportingRate	*Returns the reporting rate of the object.*
requestType		ASN1T_RequestType	*Returns the request type of the object.*
reportingRatePresent		bool	*If reporting rate is specified.*
requestTypePresent		bool	*If request type is specified.*

2.41.3.Class Attributes

ASN1T_FlightProgressRequest m_data

2.42.EngineStatusRequestMessage Class

2.42.1.Class Description

This class will be responsible for:

1. *Holding the data for an Engine Status Request message.*

This class implements:

An Engine Status Request message object.

This class instances other classes:

Public Inheritance from RequestMessage.

2.42.2.Class Methods

Name	Inputs	Outputs	Comment
EngineStatusRequestMessage			*Constructor*
contractType		ASN1T_ContractType	*Returns the contract type of the object.*
reportingRate		ASN1T_ReportingRate	*Returns the reporting rate of the object.*
requestType		ASN1T_RequestType	*Returns the request type of the object.*
reportingRatePresent		bool	*If reporting rate is specified.*
requestTypePresent		bool	*If request type is specified.*

2.42.3.Class Attributes

ASN1T_EngineStatusRequestMessage m_data

2.43.AircraftMetReportsRequestMessage Class

2.43.1.Class Description

This class will be responsible for:

1.*Holding the data for an Aircraft Met Reports Request message.*

This class implements:

An Aircraft Met Reports Request message object.

This class instances other classes:

Public Inheritance from RequestMessage.

2.43.2.Class Methods

Name	Inputs	Outputs	Comment
AircraftMetReportsRequestMessage			*Constructor*
contractType		ASN1T_ContractType	*Returns the contract type of the object.*
reportingRate		ASN1T_ReportingRate	*Returns the reporting rate of the object.*
requestType		ASN1T_RequestType	*Returns the request type of the object.*
reportingRatePresent		bool	*If reporting rate is specified.*
requestTypePresent		bool	*If request type is specified.*

2.43.3.Class Attributes

ASN1T_AircraftMetReportsRequest m_data

2.44.APURequestMessage Class

2.44.1.Class Description

This class will be responsible for:

1.Holding the data for an APU Request message.

This class implements:

An APU Request message object.

This class instances other classes:

Public Inheritance from RequestMessage.

2.44.2.Class Methods

Name	Inputs	Outputs	Comment
APURequestMessage			*Constructor.*
contractType		ASN1T_ContractType	*Returns the contract type of the object.*
reportingRate		ASN1T_ReportingRate	*Returns the reporting rate of the object.*
requestType		ASN1T_RequestType	*Returns the request type of the object.*
reportingRatePresent		bool	*If reporting rate is specified.*
requestTypePresent		bool	*If request type is specified.*

2.44.3.Class Attributes

ASN1T_APURequest m_data

2.45.RequestMessage Class

2.45.1.Class Description

This class will be responsible for:

1.Providing an abstract class with pure virtual functions for request messages.

This class implements:

An abstract class which all contract request messages are derived from.

This class instances other classes:

Public Inheritance from Message.

2.45.2.Class Methods

Name	Inputs	Outputs	Comment
virtual contractType		ASN1T_ContractType	*Returns the contract type of the derived object.*
virtual reportingRate		ASN1T_ReportingRate	*Returns the reporting rate of the derived object.*
virtual requestType		ASN1T_RequestType	*Returns the request type of the derived object.*
virtual reportingRatePresent		bool	*If reporting rate is specified.*
virtual requestTypePresent		bool	*If request type is specified.*

2.45.3.Class Attributes

N/A

2.46.ResponseReportMessage Class

2.46.1.Class Description

This class will be responsible for:

1.*Providing an abstract class with pure virtual functions for response and report messages.*

This class implements:
An abstract class which all contract response and report messages are derived from.

This class instances other classes:
Public Inheritance from Message.

2.46.2.Class Methods

Name	Inputs	Outputs	Comment
virtual contractType		ASN1T_ContractType	*Returns the contract type of the derived object*
virtual requestResult		ASN1T_ContractRequestResult	*Returns the result of the request*
virtual requestResultPresent		bool	*Optional in a report but always in a response*

2.46.3.Class Attributes

N/A

2.47.AircraftMetReportsResponseMessage Class

2.47.1.Class Description

This class will be responsible for:

1.Holding the data for an Aircraft Met Reports Response message.

This class implements:

An Aircraft Met Reports Response message object.

This class instances other classes:

Public Inheritance from ReponseReportMessage.

2.47.2.Class Methods

Name	Inputs	Outputs	Comment
AircraftMetReportsResponseMessage			*Constructor*
AircraftMetReportsResponseMessage	char* flightId, ASN1T_ContractType demandPeriodic, ASN1T_ContractRequestResult result		*Constructor*
contractType		ASN1T_ContractType	*Returns the contract type of the message*
requestResult		ASN1T_ContractRequestResult	*Returns the result of the request*
requestResultPresent		bool	*Optional in a report but always in a response*

2.47.3.Class Attributes

ASN1T_AircraftMetReportsResponse m_data

2.48.AircraftMetReportMessage Class

2.48.1.Class Description

This class will be responsible for:

1.Holding the data for an Aircraft Met Report message.

This class implements:

An Aircraft Met Report message object.

This class instances other classes:

Public Inheritance from ResponseReportMessage.

2.48.2.Class Methods

Name	Inputs	Outputs	Comment
AircraftMetReportMessage			*Constructor*
contractType		ASN1T_ContractType	*Returns the contract type of the message*
requestResult		ASN1T_ContractRequestResult	*returns the result of the request*
requestResultPresent		bool	*Optional in a report but always in a response*

2.48.3.Class Attributes

ASN1T_AircraftMetReport   m_data

2.49.FlightProgressReportMessage Class

2.49.1.Class Description

This class will be responsible for:

1.Holding the data for a Flight Progress Report message.

This class implements:

A Flight Progress Report message object.

This class instances other classes:

Public Inheritance from ResponseReport Message.

2.49.2.Class Methods

Name	Inputs	Outputs	Comment
FlightProgressReport Message			*Constructor*
contractType		ASN1T_ContractType	*Returns the contract type of the message*
requestResult		ASN1T_ContractRequestResult	*Returns the result of the request*
requestResultPresent		bool	*Optional in a report but always in a response*

2.49.3.Class Attributes

ASN1T_FlightProgressReport   m_data

2.50.FlightProgressResponseMessage Class

2.50.1.Class Description

This class will be responsible for:

1.Holding the data for a Flight Progress Response message.

This class implements:
A Flight Progress Response message object.

This class instances other classes:
Public Inheritance from Message.

2.50.2.Class Methods

Name	Inputs	Outputs	Comment
FlightProgressResponseMessage			*Constructor*
FlightProgressResponseMessage	char* flightId, ASN1T_ContractType demandPeriodic, ASN1T_ContractRequestResult result		*Constructor*
contractType		ASN1T_ContractType	*Returns the contract type of the message*
requestResult		ASN1T_ContractRequestResult	*Returns the result of the request*
requestResultPresent		bool	*Optional in a report but always in a response*

2.50.3.Class Attributes

ASN1T_FlightProgressResponse m_data

2.51.APUReportMessage Class

2.51.1.Class Description

This class will be responsible for:

1.Holding the data for an APU Report message.

This class implements:

An APU Report message object.

This class instances other classes:

Public Inheritance from Message.

2.51.2.Class Methods

Name	Inputs	Outputs	Comment
APUReportMessage			*Constructor*
contractType		ASN1T_ContractType	*Returns the contract type of the message*
requestResult		ASN1T_ContractRequestResult	*returns the result of the request*
requestResultPresent		bool	*Optional in a report but always in a response*

2.51.3.Class Attributes

ASN1T_APUReportMessage m_data

2.52.APUResponseMessage Class

2.52.1.Class Description

This class will be responsible for:

1. *Holding the data for an APU Response message.*

This class implements:
An APU Response message object.

This class instances other classes:
Public Inheritance from ResponseReportMessage.

2.52.2.Class Methods

Name	Inputs	Outputs	Comment
APUResponseMessage			*Constructor*
APUResponseMessage	char* flightId, ASN1T_ContractType demandPeriodic, ASN1T_ContractRequestResult result		*Constructor*
contractType		ASN1T_ContractType	*Returns the contract type of the message*
requestResult		ASN1T_ContractRequestResult	*Returns the result of the request*
requestResultPresent		bool	*Optional in a report but always in a response*

2.52.3.Class Attributes

ASN1T_APUResponse m_data

2.53.EngineStatusReportMessage Class

2.53.1.Class Description

This class will be responsible for:

1.*Holding the data for an Engine Status Report message.*

This class implements:

An Engine Status Report message object.

This class instances other classes:

Public Inheritance from ResponseReportMessage.

2.53.2.Class Methods

Name	Inputs	Outputs	Comment
EngineStatusReportMessage			*Constructor*
contractType		ASN1T_ContractType	*Returns the contract type of the message*
requestResult		ASN1T_ContractRequestResult	*Returns the result of the request*
requestResultPresent		bool	*Optional in a report but always in a response*

2.53.3.Class Attributes

ASN1T_EngineStatusReport m_data

2.54.EngineStatusResponseMessage Class

2.54.1.Class Description

This class will be responsible for:

1.Holding the data for an Engine Status Response message.

This class implements:

An Engine Status Response message object.

This class instances other classes:

Public Inheritance from ResponseReportMessage.

2.54.2.Class Methods

Name	Inputs	Outputs	Comment
EngineStatusResponseMessage			*Constructor*
EngineStatusResponseMessage	char* flightId, ASN1T_ContractType demandPeriodic, ASN1T_ContractRequestResult result		*Constructor*
contractType		ASN1T_ContractType	*Returns the contract type of the message*
requestResult		ASN1T_ContractRequestResult	*Returns the result of the request*

2.54.3.Class Attributes

ASN1T_EngineStatusResponse m_data

2.55.ConstraintsListMessage Class

2.55.1.Class Description

This class will be responsible for:

1.Holding the data for a Constraints List message.

This class implements:

A Constraints List message object.

This class instances other classes:

Public Inheritance from Message.

2.55.2.Class Methods

Name	Inputs	Outputs	Comment
ConstraintsListMessage			*Constructor*

2.55.3.Class Attributes

ASN1T_ConstraintsList m_data

2.56.ConstraintsAcceptanceMessage Class

2.56.1.Class Description

This class will be responsible for:

1.Holding the data for a Constraints Acceptance message.

This class implements:

A Constraints Acceptance message object.

This class instances other classes:

Public Inheritance from Message.

2.56.2.Class Methods

Name	Inputs	Outputs	Comment
ConstraintsAcceptanceMessage			*Constructor*

2.56.3.Class Attributes

ASN1T_ConstraintsAcceptance   m_data

2.57.AOCGround Class

2.57.1.Class Description

This class will be responsible for:

1. *Providing an interface to the databases for the HMI*
2. *Enabling the HMI to send messages to an aircraft*
3. *Processing messages received from an aircraft*
4. *Generating alert messages*
5. *Processing messages received from the weather office*
6. *Processing messages received from the CFMU*
7. *Sending messages to the CFMU*
8. *Managing any timed events*

This class implements:
The main functionality of the AOC.

This class instances other classes:
FPDB
FopsDB
LogFile
Weather
TimerList
ConfigurationData
AGComms
CFMUComms
WeatherOfficeComms
MemoryStore
CFMUSlotAllocationMessage
CompanyFlightPlanMessage
ConfigurationData
FmsMeteoMessage
FreeTextMessage
MessagesWaiting
OooiMessage
SlotAllocationMessage
TimerList
TrajectoryMessage
TrajectoryRequestMessage
Maintenance
AocContracts

2.57.2.Class Methods

Name	Inputs	Outputs	Comment
AOCGround			*Constructor*
~AOCGround			*Destructor*
periodicFunction		MessagesWaiting	
uplinkFreetext	char *flightId, char *freetext, bool confirmed	bool	*Send freetext to AGComms Interface*
fourDTrajectoryRequest	char *flightId	bool	*Send a 4d trajectory request to the AGComms Interface.*

getFlightData	char *flightId	FOpsDBData &fOpsRecord)	*Returns a record from the FopsDB.*
getAlertMessage	AlertMessage &alertMessage	bool	*Pass an alert from AOCGround to the HMI.*
get4DtrajectoryMessage	TrajectoryMessage &TrajectoryMessage	bool	*Pass a trajectory from AOCGround to the HMI.*
getOooiMessage	OooiMessage &oooiMessage	bool	*Pass an OOOI message from AOCGround to the HMI.*
getNewFlightId	char* flightId	bool	*Notifies HMI of a newly registered A/C.*
addAlertToQueue	AlertMessage& alertMessage	bool	*Store an alert to be passed to the HMI.*
addFourDTrajectoryToQueue	TrajectoryMessage & fourDTrajectoryMessage	bool	*Store a trajectory to be passed to the HMI.*
addFreeTextToQueue	FreeTextMessage &freeTextMessage	bool	*Store a freetext message to be passed to the HMI.*
addOooiToQueue	OooiMessage& oooiMessage	bool	*Store an OOOI message to be passed to the HMI.*
addNewFlightIdToQueue	std::string flightId	bool	*Store the flightId of a newly registered A/C, to pass to the HMI.*
checkAGComms		int	*check for new downlink messages.*
checkWoComms			*Check for messages from the Weather Office*
checkCfmuComms			*Check for messages from the CFMU*
checkAcks			*Check AGComms for details of GACS level ACK messages.*
processInMessage	FreeTextMessage &freeTextMessage		*Performs processing required on receipt of a free text message.*
processInMessage	OooiMessage &oooiMessage		*Performs processing required on receipt of an OOOI message.*
processInMessage	TrajectoryMessage &trajectoryMessage		*Performs processing required on receipt of a trajectory message.*
processInMessage	InitialisingMessage &initMessage		*Performs processing required on receipt of an initalising message.*
checkTimeouts			*Check timerList object for timeouts.*
checkTrajectory	TrajectoryMessage &trajectory	bool	*Checks validity of received trajectory.*
calculateWaitingMessages	MessagesWaiting& messagesWaiting		*Generate structure notifying HMI of number of messages to retrieve from the queues.*
sendCFPsToCFMU		bool	*Files flightplans with the CFMU*
setRegistrationTimers			*Set a timeout for registration of an A/C.*

estimatedPosition	char *flightId	ASN1T_ThreeDPosition &estimatedPosition, double &heading	*returns estimated position and heading of an aircraft*
flightProgressRequest	char *flightId, ASN1T_FlightProgressRequest request	bool	*send a flight progress request message.*
aircraftMetReportRequest	char *flightId, ASN1T_AircraftMetReportsRequest request	bool	*send an aircraft met report request message.*
engineStatusReportRequest	char *flightId, ASN1T_EngineStatusRequest request	bool	*send an engine status report request message.*
apuReportRequest	char *flightId, ASN1T_APURequest request	bool	*send an apu report request message.*
iFTMTrigger	char *flightId	bool	*request In-Flight Traffic Management be performed*
getFMSMeteoTiles	double latMin, double longMin, double latMax, double longMax,	queue<FmsMeteoTile*>& mtfArray	*for access to the weather object*
processInMessage	AircraftMetReport Message &aircraftMetReport Message		*Performs processing required on receipt of an initalising message.*
processInMessage	FlightProgressReportMessage &report		*Performs processing required on receipt of an aircraft met report message.*
processInMessage	ClearedTrajectory Message &clearedTrajectory		*Performs processing required on receipt of a cleared trajectory message.*
processInMessage	ConstraintsAcceptanceMessage &constraintsAcceptance		*Performs processing required on receipt of a constraints acceptance message.*
processInMessage	LoadsheetAckMessage &loadsheetAck		*Performs processing required on receipt of a loadsheet ack message.*
processInMessage	MeteoReportRequestMessage &meteoReportRequest		*Performs processing required on receipt of a meteo report request message.*
sendLoadsheet	const char flightId, ASN1T_FuelQuantity fuelQuantity, int totalBaggageWeight, char compartmentNames[totalCompartments][maxCompartmentNameSize], int compartmentLoads		*Sends a Loadsheet.*

2.57.3.Class Attributes

AlertMessageQueue	*m_alertQueue*
TrajectoryMessageQueue	*m_4DtrajectoryQueue*
FreeTextMessageQueue	*m_freeTextQueue*
OooiMessageQueue	*m_oooiQueue*

NewFlightIdQueue	*m_newFlightIdQueue*
*FPDB**	*m_fPDB*
*FopsDB**	*m_fOpsDB*
*LogFile**	*m_logFile*
Weather	*m_weather*
*TimerList**	*m_timers*
ConfigurationData	*m_configData*
AGComms const*	*m_AGComms*
CFMUComms const*	*m_CFMUComms*
WeatherOfficeComms const*	*m_weatherOfficeComms*
aoc::MemoryStore	*m_allocatedMemory*

2.58.CFMUComms Class

2.58.1.Class Description

This class will be responsible for:

1. *Receiving filed flight plans.*
2. *Simulating transmission of slot allocation messages according to a schedule.*
3. *Reading details of the slot allocations from file.*

This class implements:
An interface that simulates transmissions to and from the CFMU.

This class instances other classes:
CommsServer
LogFile

2.58.2.Class Methods

Name	Inputs	Outputs	Comment
CFMUComms	CommsServer *commsObj, LogFile *aocGroundLogObj		*Constructor*
~CFMUComms			*Destructor*
sendMessage	CompanyFlightPlan Message &message, char *messageId	bool	*Send a flight plan to the CFMU*
poll	CfmuSlotAllocationM essage &message	bool	*Check to see if there is a slot allocation from the CFMU.*
getMessageSent	CompanyFlightPlan Message &message	bool	*Used by HMI to recall messages sent to the CFMU.*
readSlotsFromFile			*Read details of slot messages and send times from file.*

2.58.3.Class Attributes

CommsServer const*	*m_commsPtr*
*LogFile**	*m_logFile*
std::queue<CompanyFlightPlanMessage>	*m_cfpQ*
std::list<SlotAllocationAndTimer>	*m_slotsList*
aoc::MemoryStore	*m_allocatedMemory*

2.59.CFMUSlotAllocationMessage Class

2.59.1.Class Description

This class will be responsible for:
N/A

This class implements:
The structure used to hold details of a slot allocation from the CFMU.

This class instances other classes:
public inheritance from Message

2.59.2.Class Methods

Name	**Inputs**	**Outputs**	**Comment**
CfmuSlotAllocationMessage			*Constructor*

2.59.3.Class Attributes

CfmuSlotAllocation m_data

2.60.CommsServer Class

2.60.1.Class Description

This class will be responsible for:

1. *Storing all messages transmitted to or received from external entities.*
2. *Enabling the HMI to retrieve messages given the flightId.*
3. *Enabling the HMI to retrieve messages given the flightId and message type.*
4. *Enabling the HMI to retrieve messages given a unique identifier.*

This class implements:

A database storing all messages transmitted and received by the AGP.

This class instances other classes:

2.60.2.Class Methods

Name	Inputs	Outputs	Comment
CommsServer			*Constructor*
~CommsServer			*Destructor*
logMessage	const TrajectoryMessage &message		*Log a trajectory message*
logMessage	const SlotAllocationMessage &message		*Log a slot allocation message*
logMessage	const InitialisingMessage &message		*Log an initialising message*
logMessage	const OooiMessage &message		*Log an OOOI message*
logMessage	const TrajectoryRequestMessage &message		*Log a trajectory message*
logMessage	const CompanyFlightPlanMessage &message		*Log a company flight plan message*
logMessage	const FreeTextMessage &message)		*Log a freetext message*
logMessage	const FmsMeteoMessage &message)		*Log an FMS Meteo message*
logMessage	const MeteoForecast &message		*Log a Meteo Forecast message*
logMessage	const CfmuSlotAllocationMessage &message		*Log a CFMU slot allocation message*
updateRecordWithFlightId	const ASN1T_InitialisingMessage initMessage, const char* const flightId		*Add the flight Id to an loadsheet message that didn't contain the flight Id.*
getNextMessage	const char* const flightId	Message* &returnValue, bool	*Called until returns false. Returns messages stored for a given flight.*
getNextMessage	const char* const flightId, const MessageType type	Message* &returnValue, bool	*Called until returns false. Returns messages of a given type stored for a given flight.*

getMessage	const int uniqueId	Message* &returnValue	*Returns a message stored given its unique Id.*
getTotalMessages		Int	*Returns total message stored*
getNumMessagesForLastFlightSearchedOn		Int	*Called only after getNextMessage has returned false.* *Returns number of messages stored for a given flight.*
getNumMessagesForLastTypeSearchedOn		Int	*Called only after getNextMessage has returned false.* *Returns number of a given type of message stored for a given flight.*

2.60.3.Class Attributes

CommsLogType	*m_log*
int	*m_uniqueIdCounter*
CommsLogType::iterator	*m_lastMessageGot*
char	*m_lastFlightGot[maxFlightIdBytes]*
int	*m_messageCount*
bool	*m_firstCall_getNextMessageFlightId*
CommsLogType::iterator	*m_lastMessageGot_Type*
MessageType	*m_lastTypeGot*
int	*m_messageCount_Type*
bool	*m_firstCall_getNextMessageType*

2.61.CompanyFlightPlanMessage Class

2.61.1.Class Description

This class will be responsible for:
N/A

This class implements:
The structure used to hold details of a company flight plan uplink message.

This class instances other classes:
Public inheritance from Message.

2.61.2.Class Methods

Name	Inputs	Outputs	Comment
CompanyFlightPlanMessage			*Constructor*
CompanyFlightPlanMessage	const ASN1T_Time timestamp, const char* flightId		*Constructor*

2.61.3.Class Attributes

ASN1T_CompanyFlightPlan   m_data

2.62.ConfigurationData Class

2.62.1.Class Description

This class will be responsible for:

1.Loading constants from file.
2.Storing constants and enable them to be retrieved.

This class implements:
An object that loads stores program constants from file.

This class instances other classes:
N/A

2.62.2.Class Methods

Name	Inputs	Outputs	Comment
ConfigurationData	std::string configFilename = defaultConfigFilename		*Constructor*
~ConfigurationData			*Destructor*
configLogFilename		std::string	*Returns the log filename..*
slotAllocationFileName		std::string	*Returns the slot allocation filename.*
flightPlanDirectory		std::string	*Returns the flight plan directory.*
registrationTimeOut		int	*Returns the registration time before EOBT.*
fpAckTimeOut		int	*Returns the flight plan ack timeout before EOBT.*
messageTimeOut		int	*Returns timeout waiting for a downlink message.*
slotAckTimeOut		int	*Returns slot allocation ack timeout before EOBT.*
fuelBurnRate		float	*Returns the estimated fuel burn rate read from file.*
maxWaypointTimeDifferenceMins		int	*Returns the max difference in rta for a waypoint in a downlinked trajectory over the one stored in the AOC.*
maxWaypointDifferenceMins		int	*Returns the max difference in position for a waypoint in a downlinked trajectory over the one stored in the AOC.*
loadDefaults	std::string configFilename		*Loads the constant from file.*
readString		std::string	*Read a string form file*
readInt	std::string toConvert	int	*Convert a string to an int.*
readByteValue	std::string toConvert	int	*Convert a string to a byte*
readBool	std::string toConvert	bool	*Convert a string to a bool*
readDouble	std::string toConvert	double	*Convert a string to a double*
removeLeadingAndTrailingSpaces	std::string inString	std::string	*Remove leading and trailing spaces from a string.*

makeUpper	std::string inputString	std::string	*make a sting upper case.*

2.62.3.Class Attributes

std::string	*m_logFilename*
std::string	*m_slotFilename*
std::string	*m_fpDirectory*
int	*m_RegistrationTimeOut*
int	*m_FPAckTimeOut*
int	*m_messageTimeOut*
int	*m_SlotAckTimeOut*
int	*m_maxWaypointTimeDifferenceMins*
int	*m_maxWaypointDifferenceMins*
float	*m_fuelBurnRate*
std::ifstream	*m_fileHandle*

2.63.FmsMeteo Class

2.63.1.Class Description

This class will be responsible for:
N/A

This class implements:
The structure used to hold details of uplinked FMS Meteo.

This class instances other classes:
Public Inheritance from Message.

2.63.2.Class Methods

Name	Inputs	Outputs	Comment
FmsMeteoMessage			*Constructor*

2.63.3.Class Attributes

ASN1T_FmsMeteo m_data

2.64.FopsDB Class

2.64.1.Class Description

<u>This class will be responsible for:</u>

1. *Storing details of the current trajectory of a flight.*
2. *Storing OOOI information.*
3. *Generating an initial trajectory from the flight plan.*

<u>This class implements:</u>
The flight operations database.

<u>This class instances other classes:</u>
MemoryStore
LogFile
ConfigurationData
FOpsDBKey
FOpsDBData

2.64.2.Class Methods

Name	Inputs	Outputs	Comment
FopsDB	LogFile& logFile		*Constructor*
~FopsDB			*Destructor*
get	const char *flightId	FOpsDBData &fopsRecord	*Get flight data for a flight.*
add	const FPDBRecordType fpdbRecord		*Add a record from the flight plan stored in the Flight Planning database.*
updateOooi	OooiMessage &oooi	bool	*Store an OOOI message.*
updateTrajectory	const char *flightId, ASN1T_Trajectory &trajectory	bool	*Store a new trajectory.*
fpInDB	const char *flightId	bool	*Are a flight details in the DB?*
updateWithNewFlightPlan	const char *flightId, ASN1T_CompanyFlightPlan &cfp	bool	*Modify data due to new flight plan.*
generateTrajectory	const ASN1T_CompanyFlightPlan &cfp	ASN1T_Trajectory	*Create initial trajectory from the flight plan.*
updateFuelData	const char* const flightId, const ASN1T_FuelQuantity &fuelQuantity	bool	*Store the fuel loaded onto the aircraft*
updateBaggage	const char* const flightId, const int baggageWeight	bool	*Store the baggage loaded onto the aircraft*
updatePaxCompartments	const char* const flightId, char compartmentNames[totalCompartments][maxCompartmentNameSize], const int* compartmentLoads	bool	*Store the passengers loaded onto the aircraft*
fuelPresent	const char *flightId	bool	*Has fuel been loaded?*
paxBaggagePresent	const char *flightId	bool	*Has baggage been loaded?*

getNextWaypointNumber	char *flightId, ASN1T_Time time	int	*point number of the next waypoint the aircraft is heading for*
getLastWaypointNumber	char *flightId, ASN1T_Time time	int	*last waypoint passed.*
updateLastKnownPositionValidity	char *flightId		*mark last known position invalid if in a different segment.*
getWaypoint	char *flightId, int pointNumber	ASN1T_FourDPosition	*get a wapoint's four d cooords*
getExpectedFuel	char *flightId, ASN1T_Time time,	ASN1T_FuelQuantity &expectedFuel, bool	*the expected fuel at a given time.*
getEta	char *flightId	ASN1T_Time	*returns the eta for a flight*
updateWaypoint	char *flightId, int pointNumber, ASN1T_FourDPosition newPosition		*store a new 4d position for a waypoint.*
updateWaypointEtas	char *flightId, ASN1T_SpeedGround speed, int pointNumber		*update the etas based on the current speed. Only those past pointnumber are updated.*
updateWaypointEtas	char *flightId, float delayInHours		*update all etas by a given delay factor.*
updateCurrentPosition	char *flightId, ASN1T_ThreeDPosition newPosition, ASN1T_Time time		*store a new current position for an aircraft.*
updateEta	char* flightId, ASN1T_Time newEta		*update the eta for a flight.*
updateSpeed	char *flightId, ASN1T_Speed speed		*update the speed for a flight.*
getIftmConstraints	char *flightId	ASN1T_ConstraintsList	*return stored constraints sent up to aircraft.*
storeIftmConstraints	char *flightId, ASN1T_ConstraintsList constraints		*store constraints sent to an aircraft.*
getTopOfClimbWaypoint	char *flightId	int	*the waypoint at which the aircraft has reached top of climb.*

2.64.3.Class Attributes

std::map<FOpsDBKey,FOpsDBData>	*m_DB*
aoc::MemoryStore	*m_allocatedMemory*
aoc::MemoryStore	*m_initialTrajectoryMemory*
*LogFile**	*m_logFile*
ConfigurationData	*m_configData*

2.65.FOpsDBData Class

2.65.1.Class Description

This class will be responsible for:
N/A

This class implements:
A record in the Flight Operations Database.

This class instances other classes:
N/A

2.65.2.Class Methods

Name	Inputs	Outputs	Comment
FOpsDBData			*Constructor*

2.65.3.Class Attributes

char	*m_tailNumber[maxTailIdBytes]*
ASN1T_Trajectory	*m_trajectory*
std::queue<ASN1T_OooiMessage>	*m_oooi*
ASN1T_CompanyFlightPlan_fuelUsage	*m_expectedFuel*
ASN1T_Time	*m_eta*
ASN1T_FourDPosition	*m_lastKnownPosition*
bool	*m_lastKnownPositionValid*
int	*m_lastKnownPositionValidForSegment*
ASN1T_Speed	*m_lastKnownSpeed*
ASN1T_NumberPax	*m_paxTotal*
int	*m_baggageWeight*
ASN1T_FuelQuantity	*m_fuelWeight*
ASN1T_CompartmentLoad	*m_compartments[maxCompartments]*
int	*m_numberOfCompartments*
bool	*m_fuelPresent*
bool	*m_paxBaggagePresent*
ASN1T_ConstraintsList	*m_constraints*
int	*m_topOfClimbWaypoint*

2.66.FOpsDBKey Class

2.66.1.Class Description

This class will be responsible for:
N/A

This class implements:
A record's key in the Flight Operations Database.

This class instances other classes:
N/A

2.66.2.Class Methods

Name	Inputs	Outputs	Comment
FopsDBkey			*Constructor*

2.66.3.Class Attributes

char m_flightId[maxFlightIdBytes]

2.67.FPDB Class

2.67.1.Class Description

This class will be responsible for:

1.Loading flight plans from file
2.Storing flight plans
3.Enabling a flight plan to be retrieved given a flight Id and tail number.

This class implements:
The Flight Planning Database.

This class instances other classes:
MemoryStore
LogFile
FPDBRecordType

2.67.2.Class Methods

Name	Inputs	Outputs	Comment
FPDB	LogFile &logFile		*Constructor*
~FPDB			*Destructor*
get	ASN1T_InitialisingMessage initMessage	bool, FPDBRecordType &fpdbRecord	*returns a loadsheet given an loadsheet message.*
get	const char* const flightId	bool, FPDBRecordType &F	*get a record given the flight Id.*
getSize		int	*get number of flight plans stored.*
getRecord	const int reqRecord	bool, FPDBRecordType &fpdbRecord	*get a record given the position in the database.*
updateSlotAllocation	const char* const flightId, const CfmuSlotAllocation &sa	bool	*update a flight plan due to a new slot allocation.*
get	const char* const tailNumber, const char* const flightId	bool, FPDBRecordType &FP	*get a record given the flight id and tail number.*
getTN	const char* const tailNumber	int	*number of flight plans for a given tail number*
searchResult	int I	FPDBRecordType	*get one of the flight plans identified by getTN.*
add	const FPDBRecordType FP		*add a flight plan to the database.*
readFPsFromFile			*read flight plans from file.*

2.67.3.Class Attributes

std::list<FPDBRecordType>	m_DB
std::list<FPDBRecordType>::iterator	m_ptrToFirstSearchResult
LogFile*	m_logFile
aoc::MemoryStore	m_allocatedMemory

2.68.FPDBMemBlock Class

2.68.1.Class Description

This class will be responsible for:
N/A

This class implements:
A structure dynamically allocated to store flight plan information

This class instances other classes:
N/A

2.68.2.Class Methods

N/A

2.68.3.Class Attributes

char	*m_flightId[maxFlightIdBytes]*
char	*m_routeIdentifier[maxRouteIdentifierBytes]*
char	*m_typeOfAircraft[maxAircraftTypeBytes]*
char	*m_departureAerodrome[icaoIdentSize]*
char	*m_destinationAerodrome[icaoIdentSize]*
char	*m_alternateAerodrome[icaoIdentSize]*
char	*m_secondAlternateAerodrome[icaoIdentSize]*
char	*m_otherInformation[maxFreeTextBytes]*

2.69.FPDBRecordType Class

2.69.1.Class Description

This class will be responsible for:
N/A

This class implements:
A record in the flight planning database.

This class instances other classes:
N/A

2.69.2.Class Methods

Name	Inputs	Outputs	Comment
FPDBRecordType			*Constructor*
operator <	FPDBRecordType b	bool	*Comparision needed for stl::list::sort function*

2.69.3.Class Attributes

char	*m_tailNumber[maxTailIdBytes]*
char	*m_flightId[maxFlightIdBytes]*
FlightData	*m_whichFp[maxExtraFlightPlans + 1]*
bool	*m_doesFpExist[maxExtraFlightPlans + 1]*

2.70.FreeTextMessage Class

2.70.1.Class Description

This class will be responsible for:
N/A

This class implements:
A free text message.

This class instances other classes:
Public Inheritance from Message.

2.70.2.Class Methods

Name	Inputs	Outputs	Comment
FreeTextMessage			*Constructor*
setConfirmed	bool confirmed		*Whether a GACS ACK is expected or not.*

2.70.3.Class Attributes

ASN1T_Freetext	*m_data*
bool	*m_confirmed*

2.71.CFMUSlotAllocation Class

2.71.1.Class Description

This class will be responsible for:
N/A

This class implements:
A slot allocation message from the CFMU.

This class instances other classes:
Public Inheritance from Message.

2.71.2.Class Methods

Name	Inputs	Outputs	Comment
CfmuSlotAllocationMessage			*Constructor*

2.71.3.Class Attributes

ASN1T_Time	*m_eobt*
ASN1T_Time	*m_ctot*
char	*m_flightId[maxFlightIdBytes]*
*char**	*m_restriction*

2.72.InitialisingMessage Class

2.72.1.Class Description

This class will be responsible for:
N/A

This class implements:
An initialising message received from the aircraft.

This class instances other classes:
Public Inheritance from Message.

2.72.2.Class Methods

Name	Inputs	Outputs	Comment
InitialisingMessage			*Constructor*

2.72.3.Class Attributes

ASN1T_InitialisingMessage m_data

2.73.Logfile Class

2.73.1.Class Description

This class will be responsible for:

1.Logging warnings to file
2.Logging errors to file
3.Logging information to file

This class implements:
The logging to file of error and warning and information messages.

This class instances other classes:
ConfigurationData

2.73.2.Class Methods

Name	Inputs	Outputs	Comment
LogFile			*Constructor*
LogFile	std::string logFileName, LogFileMode mode		*Constructor*
~LogFile			*Destructor*
openLogFileWithPrefs	std::string logFileName, LogFileMode mode		*Open the logfile for writing.*
addHeader			*Add a header to the file.*
empty			*Clear the log.*
log	LogType logKind, std::string callingFunction, std::string callingClass, std::string otherInformation, std::string processName = "(MAAFAS AGP)"		*log an error or warning cllling function, calling class and process.*
log	std::string logInformation		*Log an information message*
getDateAndTimeString		std::string	*Return time and date as a string*
getNewLine		std::string	*New line character*

2.73.3.Class Attributes

std::ofstream	*m_fileHandle*
std::string	*m_logFilename*
ConfigurationData	*m_configData*

2.74.Message Class

2.74.1.Class Description

This class will be responsible for:

1.Storing all common message parameters

This class implements:

The base class for all messages

This class instances other classes:

N/A

2.74.2.Class Methods

Name	Inputs	Outputs	Comment
Message			*Constructor*
~Message			*Destructor*
messageType		MessageType	*Returns the type of message*
getFlightId		char*	*Returns flightId messagfe is from/to*
getTimestamp		ASN1T_Time	*Returns time of send/receive*
getDatestamp		ASN1T_Date	*Returns date of send/receive*
getUniqueId		unsigned int	*Returns unique Id*
getTxRx		TransmissionType	*Returns whether this message was sent or received*
getTailNumber		char*	*Returns tail number*
setFlightId	const char* const flightId		*Sets flight id*
setTimestamp	const ASN1T_Time timestamp		*Sets send/receive time*
setDatestamp	const ASN1T_Date datestamp		*Sets send/receive time*
setUniqueId	const unsigned int value		*Sets the unique id*
setTxRx	TransmissionType txRx		*Sets whether send or receive*
setTailNumber	char* tn		*Sets the tail number*
setMessageType	MessageType		*Sets the type of message*

2.74.3.Class Attributes

unsigned int	*m_uniqueId*
ASN1T_Time	*m_timestamp*
ASN1T_Date	*m_datestamp*
MessageType	*m_messageType*
char	*m_flightId[maxFlightIdBytes]*
TransmissionType	*m_transmissionType*
char	*m_tailNumber[maxTailIdBytes]*

2.75.MessagesWaiting Class

2.75.1.Class Description

This class will be responsible for:
N/A

This class implements:
The structure passed from AOCGround to the HMI detailing new messages received and stored on queues.

This class instances other classes:
N/A

2.75.2.Class Methods

Name	Inputs	Outputs	Comment
MessagesWaiting			*Constructor*
MessagesWaiting	const int alertCount, const int fourDTrajCount, const int freeTextCount, const int oooiCount, const int flightIdCount,		
~MessagesWaiting			*Destructor*
getAlertMessageCount		int	*Returns number of alert messages waiting on the queue.*
get4DtrajectoryMessageCount		int	*Returns number of trajectory messages waiting on the queue.*
getFreeTextMessageCount		int	*Returns number of free text messages waiting on the queue.*
getOooiMessageCount		int	*Returns number of OOOI messages waiting on the queue.*
getNewFlightIdCount		int	*Returns number of new flight ids waiting on the queue.*

2.75.3.Class Attributes

int	*m_alertMessageCount*
int	*m_4DtrajectoryMessageCount*
int	*m_freeTextMessageCount*
int	*m_oooiMessageCount*
int	*m_newFlightIdCount*

2.76.OooiMessage Class

2.76.1.Class Description

This class will be responsible for:
N/A

This class implements:
An OOOI message from a flight.

This class instances other classes:
Public Inheritance from Message.

2.76.2.Class Methods

Name	Inputs	Outputs	Comment
OooiMessage			*Constructor*

2.76.3.Class Attributes

ASN1T_OooiMessage m_data

2.77.SlotAllocationMessage Class

2.77.1.Class Description

This class will be responsible for:
N/A

This class implements:
A slot allocation message uplinked to the aircraft.

This class instances other classes:
Public Inheritance from Message.

2.77.2.Class Methods

Name	Inputs	Outputs	Comment
SlotAllocationMessage			*Constructor*
SlotAllocationMessage	const ASN1T_Time timestamp, const char* const flightId		*Constructor*
SlotAllocationMessage			*Destructor*
initialise			*Set default values*

2.77.3.Class Attributes

ASN1T_SlotAllocation m_data

2.78.TimerList Class

2.78.1.Class Description

This class will be responsible for:

1. *Managing a number of timers*
2. *Generating an appropriate alert if a timer times out*
3. *Setting timers for a number of seconds in the future*
4. *Setting a timer for a time before an EOBT time*

This class implements:
Timing of various events within the AOC.

This class instances other classes:
LogFile
AlertMessage
TimerListEntry

2.78.2.Class Methods

Name	Inputs	Outputs	Comment
TimerList	LogFile &logFile		*Constructor*
TimerList			*Destructor*
timeout	char *flightId, int seconds, TimerType type		*Sets a timeout a number of seconds in the future.*
timeoutBeforeEobt	char *flightId, int minutes, ASN1T_Time eobt, TimerType type		*Sets a timeout a number of minutes before an EOBT time.*
cancel	char* flightId, TimerType type		*Cancel a timeout.*
numAlerts		int	*Generate any alerts arising from timeouts and return number.*
getAlert		bool, AlertMessage	*Get alerts. Call until returns false.*
sortList			*Sort internal list*

2.78.3.Class Attributes

std::list<TimerListEntry>	*m_list*
std::queue<AlertMessage>	*m_alertQ*
*LogFile**	*m_logFile*

2.79.TimerListEntry Class

2.79.1.Class Description

This class will be responsible for:

1. *Storing details of a single timeout*
2. *Calculating when the timeout has expired*
3. *Returning a string naming the timer*

This class implements:

A timeout event.

This class instances other classes:

N/A

2.79.2.Class Methods

Name	Inputs	Outputs	Comment
TimerListEntry			*Constructor*
~TimerListEntry			*Destructor*
getString	TimerType type	std::string	*Return string name of timer*
getFlightId		char*	*Return the flight Id the timer was set for.*
timeExceeded		bool	*Has the timer expired?*

2.79.3.Class Attributes

char	*m_flightId[maxFlightIdBytes]*
ASN1T_Time	*m_timestamp*
ASN1T_Date	*m_datestamp*
TimerType	*m_timerType*

2.80.TrajectoryMessage Class

2.80.1.Class Description

This class will be responsible for:
N/A

This class implements:
A four D trajectory message.

This class instances other classes:
Public Inheritance from Message.

2.80.2.Class Methods

Name	Inputs	Outputs	Comment
TrajectoryMessage			*Constructor*
~TrajectoryMessage	const ASN1T_Time timestamp, const char* flightId		*Constructor*
~TrajectoryMessage			*Destructor*

2.80.3.Class Attributes

ASN1T_Trajectory   m_data

2.81.TrajectoryRequestMessage Class

2.81.1.Class Description

This class will be responsible for:
N/A

This class implements:
A trajectory request message.

This class instances other classes:
Public Inheritance from Message.

2.81.2.Class Methods

Name	Inputs	Outputs	Comment
TrajectoryRequestMessage			*Constructor*
~TrajectoryRequestMessage			*Destructor*

2.81.3.Class Attributes

N/A

3.AGP CAPABILITIES

The current section provides a detailed description of the capabilities that shall be included in the AGP.

*3.1.*Air-Ground Communications: Reception

3.1.1.Overview

This functionality handles incoming GACS events, which amongst others includes the reception of a datalink message. This function is supported by a GAPIEndpoint object (the object attributes refer to a GACS endpoint) and performed on a dedicated thread called listen. The GACS endpoint is checked every cycle calling the GAPI primitive G_listen. According the event returned by G_listen, the incoming data is handled:

-If the returned event is a transfer indication (G_TRANSFER_IND), the arriving message is decoded and stored in the received messages queue.

-If the returned event is a credit indication (G_CREDIT_IND), the m_incomingCredit attribute is set to true.

-If the returned event is a confirmation (G_TRANFER_CNF), the message related to the arriving confirmation is removed from the confirmation timers list. Depending on the type of the original message the confirmation is stored on the confirmations queue.

3.1.2.Starting Point

At the beginning of the application, when the GAPIendpoint::init() method is called and the listen thread is initialised.

3.1.3.Ending Point

At the end of the application, when the GAPIendpoint object is destructed and the listen thread is destroyed.

3.1.4.Measurement Result

The measurement result depends on the event returned by the G_listen function.

-If the event was a transfer indication (G_TRANSFER_IND) and the decoding was successful, the received messages queue holds the new message.

-If the event was a credit indication (G_CREDIT_IND), the m_incomingCredit attribute is set to true.

-If the event was a confirmation (G_TRANFER_CNF), the number of elements of the confirmation timers list is reduced by 1. If the confirmation refers to the uplink of a slot allocation or to the uplink of flight plan, the confirmation queue holds a new element.

3.1.5.Outstanding Issues

The sequence diagram depicted below (Figure 3) shows the execution of one listen thread cycle (GAPIEndpoint::Listen()).

The measurement results are verified if the execution of the listen thread is not interrupted.

3.1.6.Sequence Diagram

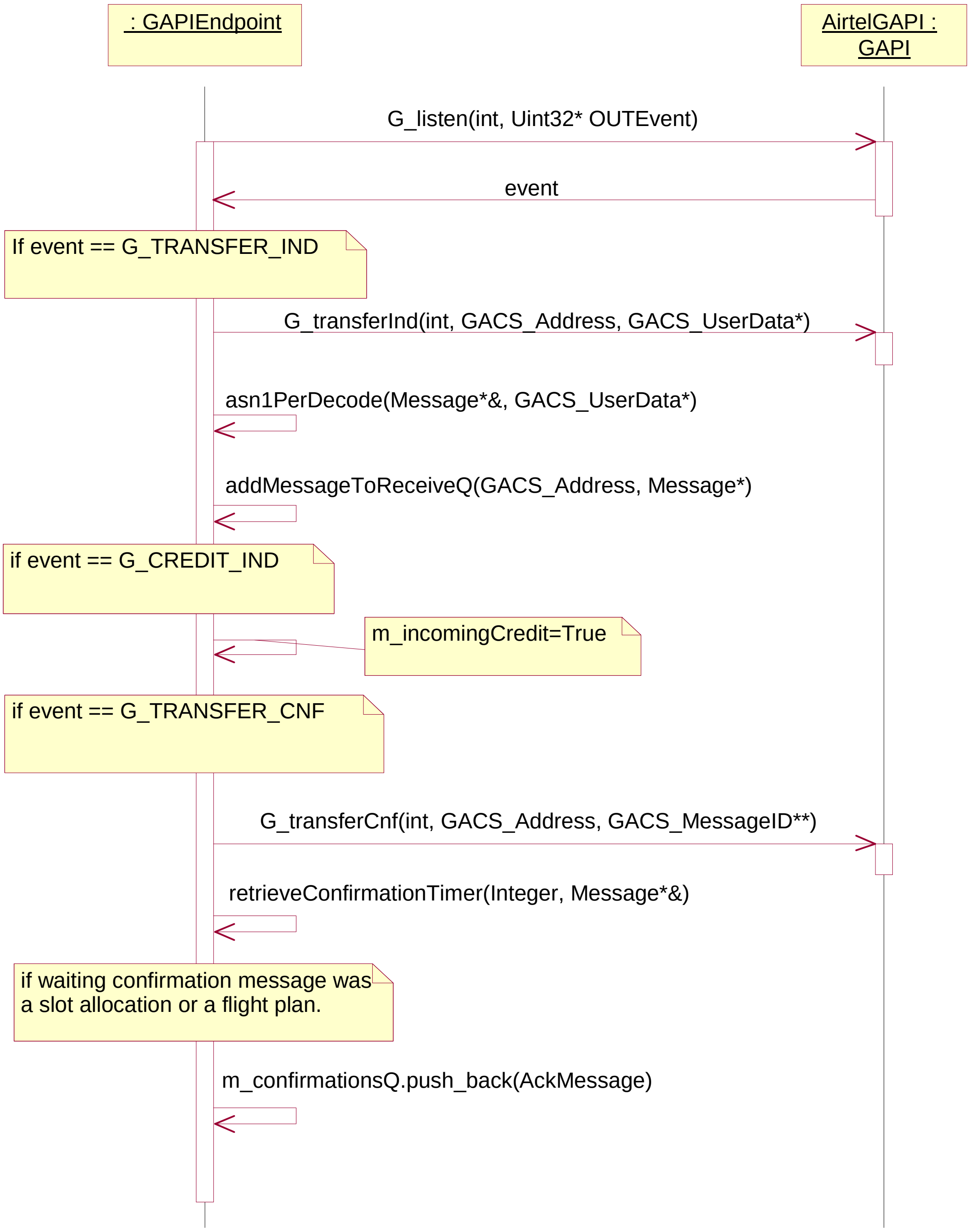

: GAPIEndpoint
AirtelGAPI : GAPI
G_listen(int, Uint32* OUTEvent)
event
If event == G_TRANSFER_IND
G_transferInd(int, GACS_Address, GACS_UserData*)
asn1PerDecode(Message*&, GACS_UserData*)
addMessageToReceiveQ(GACS_Address, Message*)
if event == G_CREDIT_IND
m_incomingCredit=True
if event == G_TRANSFER_CNF
G_transferCnf(int, GACS_Address, GACS_MessageID**)
retrieveConfirmationTimer(Integer, Message*&)
if waiting confirmation message was a slot allocation or a flight plan.
m_confirmationsQ.push_back(AckMessage)

Figure 3 – Listen thread sequence diagram

3.2.Air-Ground Communications: Transmission

3.2.1.Overview

This functionality is responsible for transmitting messages using GACS.This function is supported by a GAPIEndpoint object (object attributes refer to a GACS endpoint) and performed on a dedicated thread named transmit. The queue storing the messages that are to be transmitted is checked on every cycle. If the queue is not empty and there is a transmission credit, the head of the queue is removed and the corresponding message is transmitted. The attributes kept in the queue are the recipient of the message, the service to be used when transmitting the message, i.e. confirmed or not confirmed, and the actual message to be transmitted.
Before the transmission, the message is encoded using ASN.1 PER. If the transmission is successful, the communication server logs the message.

3.2.2.Starting Point

At the beginning of application, when the GAPIendpoint::init() method is called and the transmit thread is initialised.

3.2.3.Ending Point

At the end of the application, when the GAPIendpoint object is destructed and the transmit thread is destroyed.

3.2.4.Measurement Result

The number of elements in the queue holding the messages to transmit is reduced by 1.
If the message was to be transmitted using the GACS confirmed service, the list holding the timers regarding the receipt of confirmations holds a new element.

3.2.5.Outstanding Issues

The sequence diagram depicted below (Figure 4) shows the execution of one transmit thread cycle (GAPIEndpoint::Transmit()).
This diagram was depicted assuming that there is a transmission credit and that the queue holding the messages to transmit is not empty.

3.2.6.Sequence Diagram

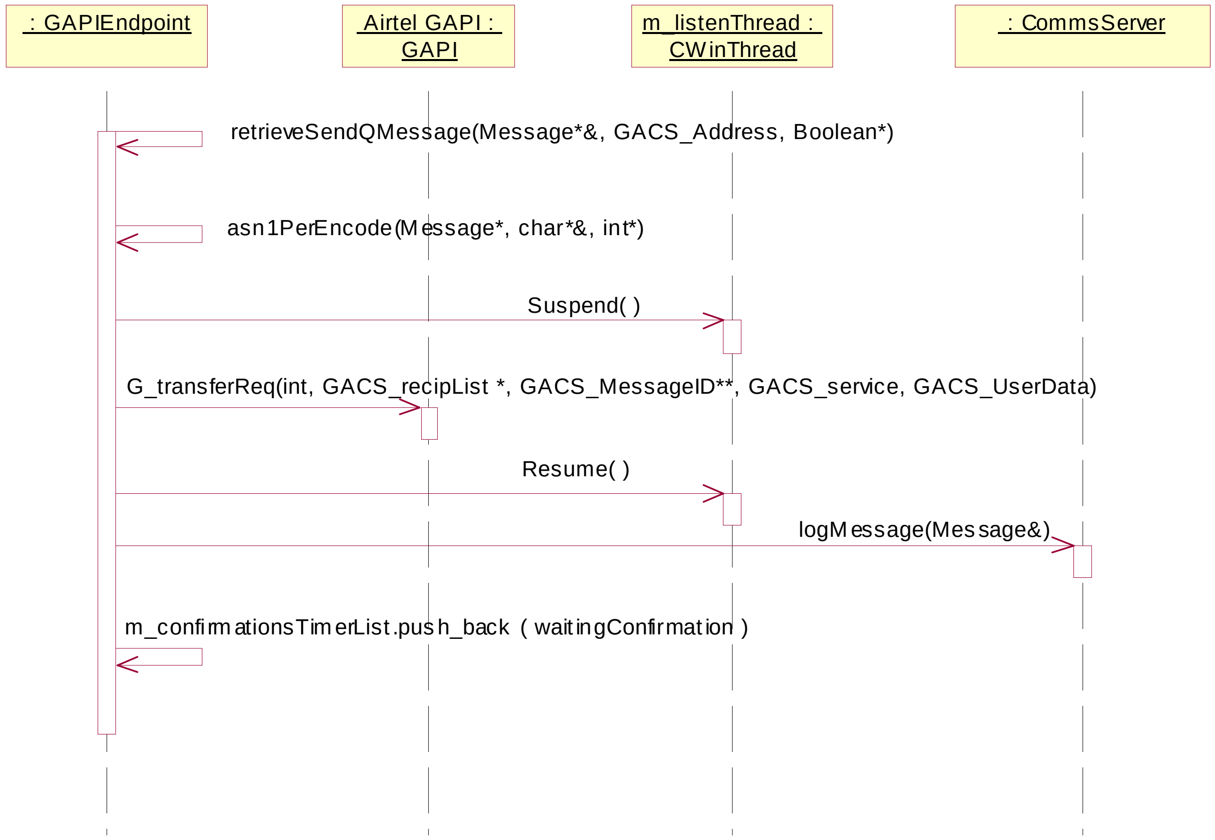

: GAPIEndpoint
Airtel GAPI : GAPI
m_listenThread : CWinThread
: CommsServer
retrieveSendQMessage(Message*&, GACS_Address, Boolean*)
asn1PerEncode(Message*, char*&, int*)
Suspend( )
G_transferReq(int, GACS_recipList *, GACS_MessageID**, GACS_service, GACS_UserData)
Resume( )
logMessage(Message&)
m_confirmationsTimerList.push_back (waitingConfirmation)

Figure 4- Transmission thread sequence diagram

3.3.Air-Ground Communications: Alerts

3.3.1.Overview

This capability checks the non-confirmation of messages transmitted using the GACS confirmed service and issues the corresponding alerts. The messages kept in the confirmation timers list of the GAPIEndpoint object are verified against a timeout value. The first message found to which the timer has expired, is returned to the AGComms instance where a corresponding alert is generated.

3.3.2.Starting Point

The AOCGround object calls the AGComms::getAlert method.

3.3.3.Ending Point

The AGComms::getAlert method returns.

3.3.4.Measurement Result

The number of elements in the waiting confirmations queue is reduced by 1.
The AGComms::getAlert method returns true and the alert message details are retrieved in the method parameter.

3.3.5.Outstanding Issues

The sequence diagram shown below (Figure 7) was depicted assuming that there was one element within the confirmation timers list to which the confirmation timer has expired.

3.3.6.Sequence Diagram

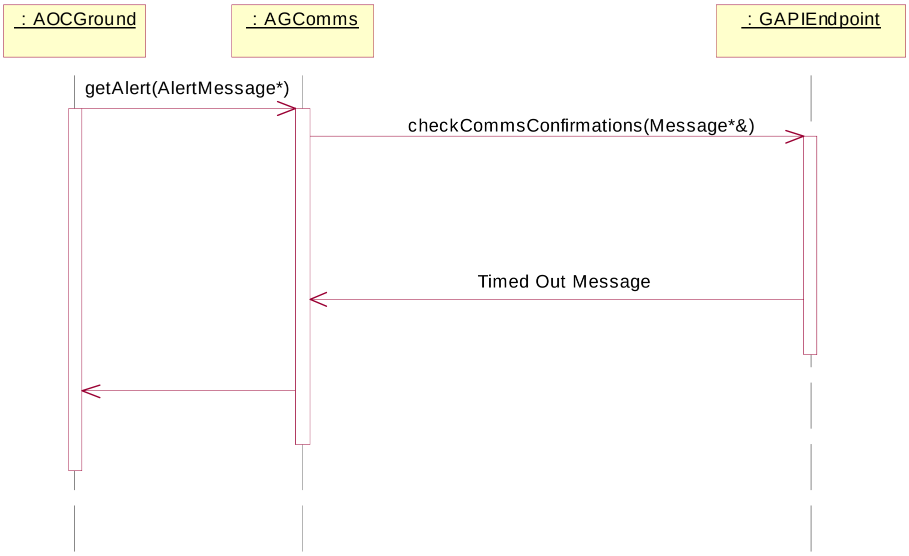


Figure 5 – Alerts sequence diagram

3.4.Communications To Meteo Office: Meteo Forecast and Meteo Reports

3.4.1.Overview

This capability handles the reception of Meteo Forecast messages and Meteo Report messages (TAFs, METARs, and SIGMETs) from an AGP external entity.
The transmission of a Meteo Forecast Message or a Meteo Report Message by an external AGP entity like a Weather Office is simulated.
Both meteo forecast and meteo report messages are read by WeatherOfficeComms object from files at system start-up, stored in the WeatherOfficeComms::m_exchangedMessageQ and logged into the AGP communications server. Later on, when the checkWoComms method is called, messages in the m_exchangedMessageQ are stored in the AGP meteo database.

3.4.2.Starting Point

First call to WeatherOfficeComms() method.

3.4.3.Ending Point

End of AOCGround::checkWoComms call.

3.4.4.Measurement Result

Meteo Forecast data is stored in the AGP meteo database as meteo tiles.
All TAFs, METARs and SIGMETs files are stored in the AGP meteo database.

3.4.5.Outstanding Issues

None

3.4.6.Sequence Diagram

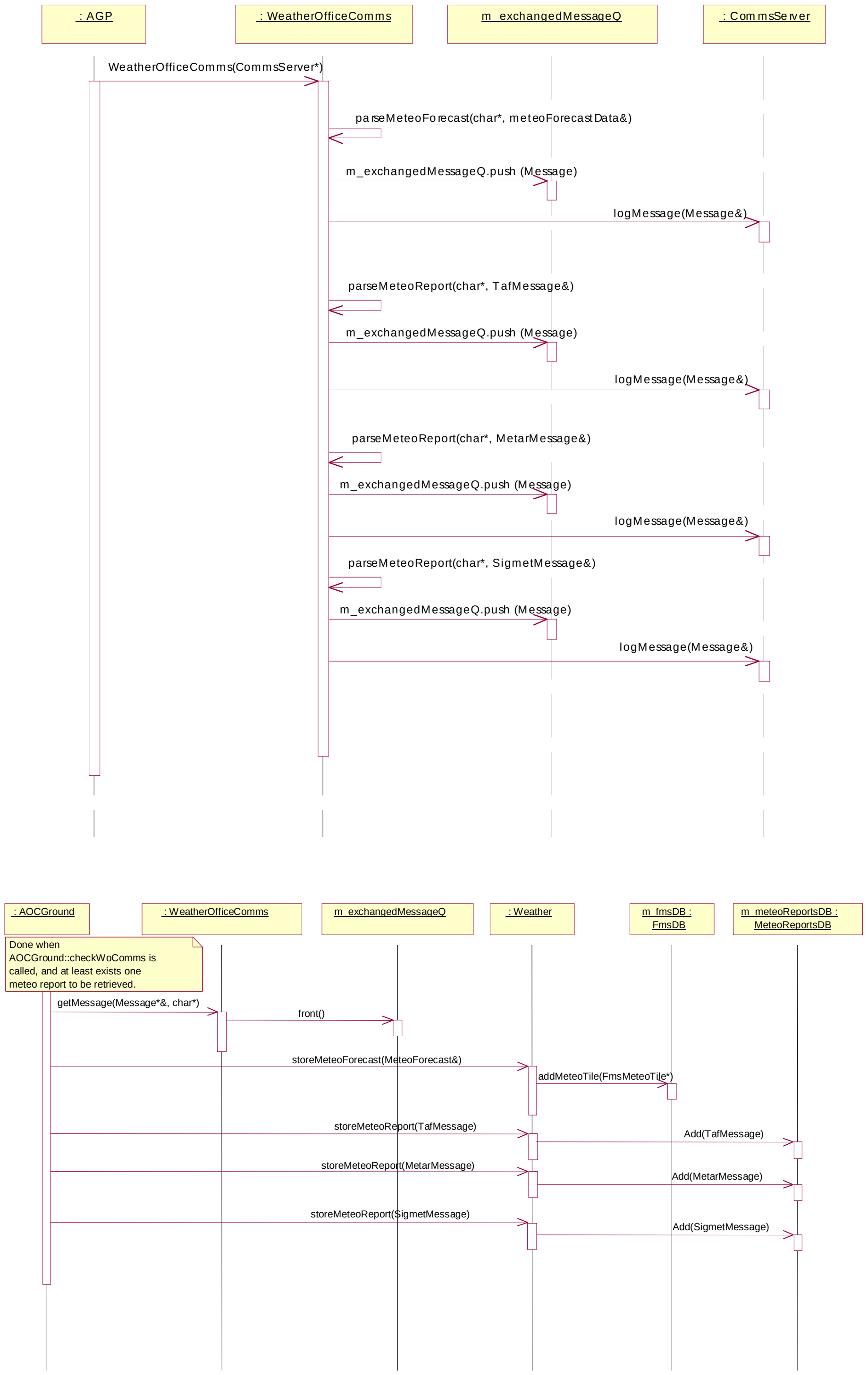

: AGP
: WeatherOfficeComms
m_exchangedMessageQ
: CommsServer
WeatherOfficeComms(CommsServer*)
parseMeteoForecast(char*, meteoForecastData&)
m_exchangedMessageQ.push (Message)
logMessage(Message&)
parseMeteoReport(char*, TafMessage&)
m_exchangedMessageQ.push (Message)
logMessage(Message&)
parseMeteoReport(char*, MetarMessage&)
m_exchangedMessageQ.push (Message)
logMessage(Message&)
parseMeteoReport(char*, SigmetMessage&)
m_exchangedMessageQ.push (Message)
logMessage(Message&)
: AOCGround
: WeatherOfficeComms
m_exchangedMessageQ
: Weather
m_fmsDB : FmsDB
m_meteoReportsDB : MeteoReportsDB
Done when AOCGround::checkWoComms is called, and at least exists one meteo report to be retrieved.
getMessage(Message*&, char*)
front()
storeMeteoForecast(MeteoForecast&)
addMeteoTile(FmsMeteoTile*)
storeMeteoReport(TafMessage)
Add(TafMessage)
storeMeteoReport(MetarMessage)
Add(MetarMessage)
storeMeteoReport(SigmetMessage)
Add(SigmetMessage)

Figure 6 – Meteo Reports and Meteo Forecast sequence diagrams

3.5.Flight Plan: FMS Meteo

3.5.1.Overview

The FMS Meteo functionality manages the uplink of FMS meteo messages to a registered aircraft. This capability scope includes matching the aircraft route with stored meteo tiles coordinates in order to find and transmit the most useful meteo data to the aircraft. After receiving the registration message from a new aircraft, the AGP uplinks flight support data to the aircraft. Flight support data includes FMS meteo data. The FMS meteo data is requested to the Weather object which holds the meteo tiles database (m_fmsDB). Within the weather object the route is matched against the meteo tiles coordinates and a meteo tile queue is filled with the meteo tiles that are found useful. After the retrieval of the FMS meteo tile queue, each meteo tile is transmitted to the aircraft.

3.5.2.Starting Point

The AOCGround object is processing an aircraft registration message.

3.5.3.Ending Point

All messages containing the retrieved meteo tiles were transmitted to the aircraft.

3.5.4.Measurement Result

The queue containing the messages to be transmitted to the aircraft holds the messages corresponding to the retrieved meteo tiles.

3.5.5.Outstanding Issues

None

3.5.6.Sequence Diagram

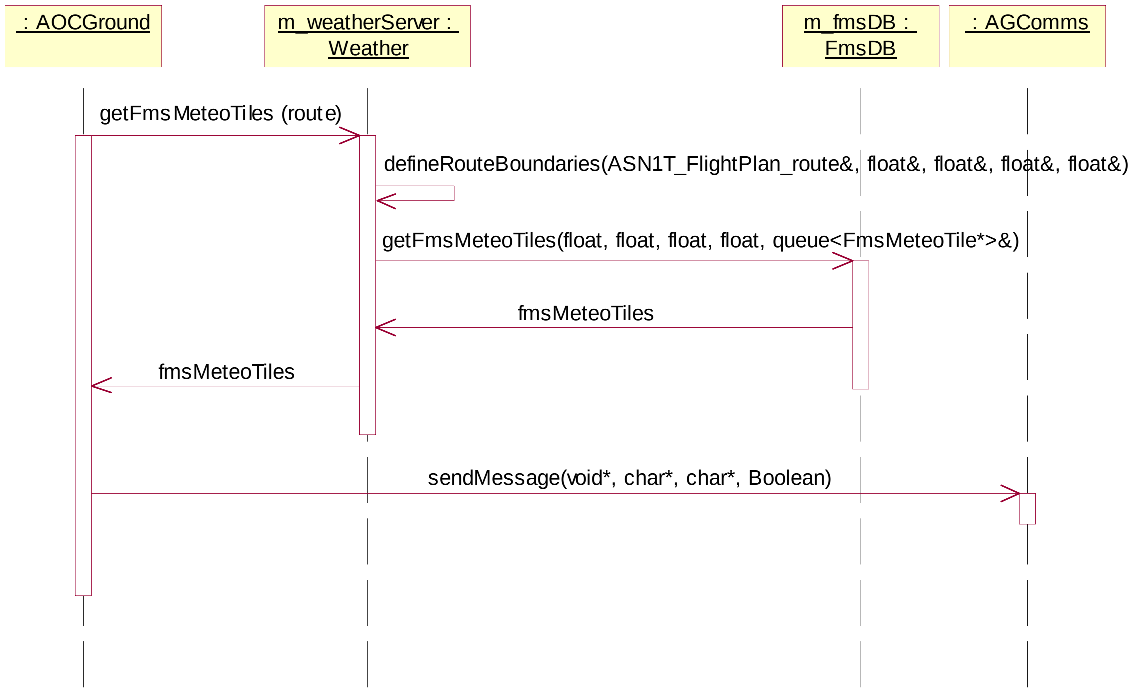

: AOCGround
m_weatherServer : Weather
m_fmsDB : FmsDB
: AGComms
getFmsMeteoTiles (route)
defineRouteBoundaries(ASN1T_FlightPlan_route&, float&, float&, float&, float&)
getFmsMeteoTiles(float, float, float, float, queue<FmsMeteoTile*>&)
fmsMeteoTiles
fmsMeteoTiles
sendMessage(void*, char*, char*, Boolean)

Figure 7 – FMS Meteo sequence diagram

3.6.Flight Plan: Meteo Reports

3.6.1.Overview

This capability handles the reception of meteo report requests from a registered aircraft, and the uplink of the requested meteo reports or unavailable messages if the requested meteo reports are not found. The AGP periodically checks if exists new messages from Air-Ground communications every cycle. Occasionally, an aircraft requests meteorological information concerning a specified location sending a meteo report request message. After receiving a meteo report request, the AGP searches the meteo reports that were requested (TAFs, METARs or SIGMETs) in the MeteoReportsDB object. If it finds at least one of the referred meteo reports, the AGP transmits the available meteo reports to the aircraft. If there is not any of the requested meteo reports in the MeteoReportsDB object, the AGP sends an unavailable message.

3.6.2.Starting Point

The AOC Ground object calls the AGComms::getMessage within the checkAGComms.

3.6.3.Ending Point

End of AOCGround::checkAGComms.

3.6.4.Measurement Result

The queue containing the messages to be transmitted to the aircraft holds an unavailable message or the requested meteo report messages.

3.6.5.Outstanding Issues

The sequence diagram shown below (Figure 8) was depicted assuming that the message that is returned by AGComms::getMessage is a meteo report request message.

3.6.6.Sequence Diagram

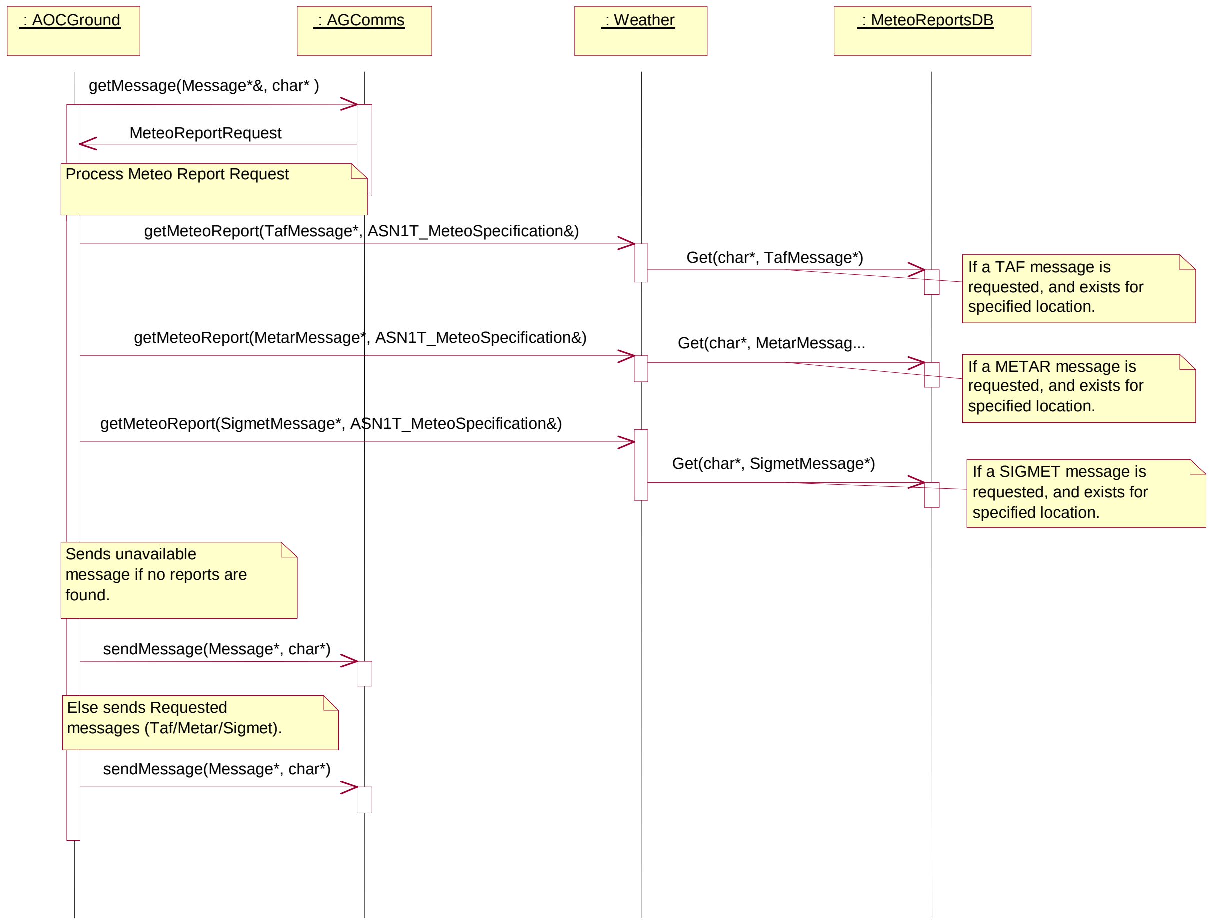

: AOCGround
: AGComms
: Weather
: MeteoReportsDB
getMessage(Message*&, char*)
MeteoReportRequest
Process Meteo Report Request
getMeteoReport(TafMessage*, ASN1T_MeteoSpecification&)
Get(char*, TafMessage*)
If a TAF message is requested, and exists for specified location.
getMeteoReport(MetarMessage*, ASN1T_MeteoSpecification&)
Get(char*, MetarMessag...
If a METAR message is requested, and exists for specified location.
getMeteoReport(SigmetMessage*, ASN1T_MeteoSpecification&)
Get(char*, SigmetMessage*)
If a SIGMET message is requested, and exists for specified location.
Sends unavailable message if no reports are found.
sendMessage(Message*, char*)
Else sends Requested messages (Taf/Metar/Sigmet).
sendMessage(Message*, char*)

Figure 8 – Meteo reports sequence diagram

3.7.Maintenance: SNAG Reports

3.7.1.Overview

This functionality handles the retrieval of reports containing description about aircraft failures or malfunctions (SNAG reports) from the AGP Communications Server. When the AGP operator selects a SNAG report from the Maintenance Display messages list, the AGP retrieves the selected SNAG message from the CommsServer object, filling the message content listbox with the SNAG message details.

3.7.2.Starting Point

Call to updateMessages function.

3.7.3.Ending Point

End of updateMessages function.

3.7.4.Measurement Result

SNAG Messages details are displayed.

3.7.5.Outstanding Issues

None

3.7.6.Sequence Diagram

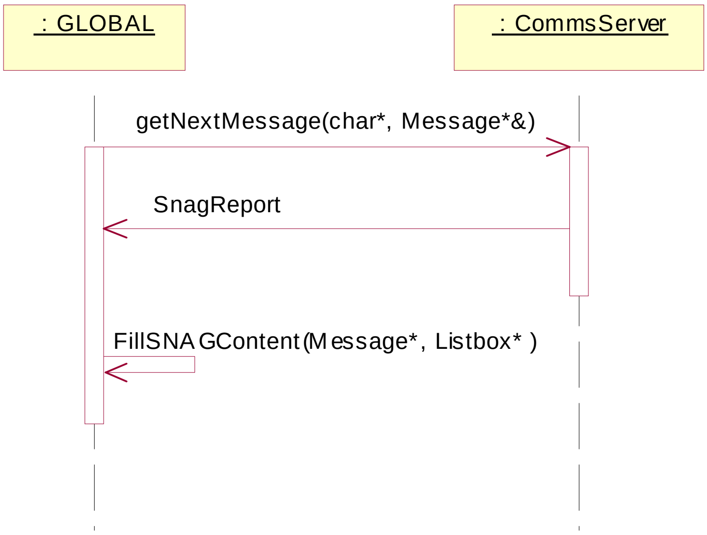

: GLOBAL
: CommsServer
getNextMessage(char*, Message*&)
SnagReport
FillSNAGContent(Message*, Listbox*)

Figure 9 – SNAG reports sequence diagram

3.8.Maintenance: APU Reports

3.8.1.Overview

This functionality enables the AGP operator to get the current data regarding the APU of a specified flight. The AGP supports the request of APU data on a demand or periodic basis. A demand request is satisfied as soon as the next APU report is received. A periodic request implies setting up a contract and the reception of reports at a specified time interval. If the aircraft cannot satisfy a request then a response is downlinked rejecting the request. When an APU report is received, the aircraft systems information (ASI) record concerning the specified flight is updated in the maintenance database, if this record does not exist a new ASI record is then created. If the anticipated message is not received then a suitable alert is generated.

3.8.2.Starting Point

AOCGround::apuReportRequest() function is called.

3.8.3.Ending Point

End of AOCGround::checkAGComms.

3.8.4.Measurement Result

Transmission and receipt of messages.
Maintenance database is updated either by inserting a new ASI record or by updating the data of an existing ASI record.

3.8.5.Outstanding Issues

None.

3.8.6.Sequence Diagram

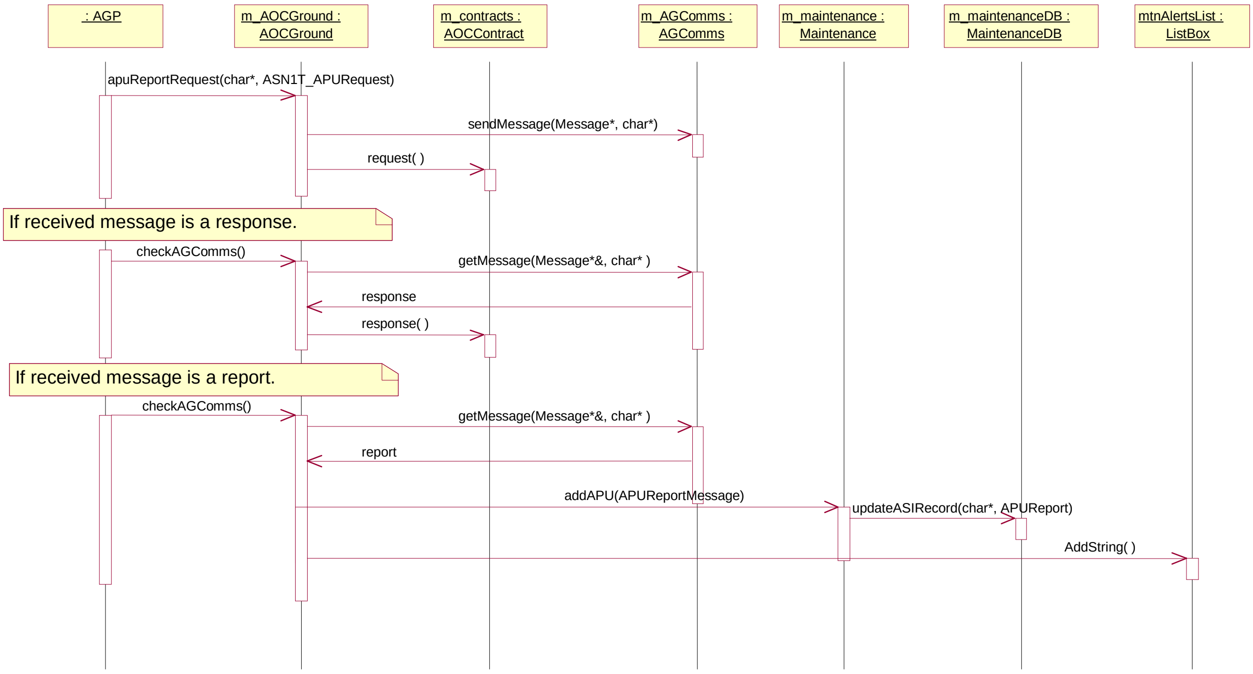

: AGP
m_AOCGround : AOCGround
m_contracts : AOCContract
m_AGComms : AGComms
m_maintenance : Maintenance
m_maintenanceDB : MaintenanceDB
mtnAlertsList : ListBox
apuReportRequest(char*, ASN1T_APURequest)
sendMessage(Message*, char*)
request( )
If received message is a response.
checkAGComms()
getMessage(Message*&, char*)
response
response( )
If received message is a report.
checkAGComms()
getMessage(Message*&, char*)
report
addAPU(APUReportMessage)
updateASIRecord(char*, APUReport)
AddString( )

Figure 10 – APU reports sequence diagram

3.9.Maintenance: Engine Status Reports

3.9.1.Overview

This functionality enables the AGP to monitor the engine status of currently flying aircrafts. The AGP supports the request of engine data on a demand or periodic basis. A demand request is satisfied as soon as the next engin report is received. A periodic request implies setting up a contract and the receipt of reports satisfies at a specified time interval. If the aircraft cannot satisfy a request then a response is downlinked rejecting the request.

When a engine status report is received, the aircraft systems information (ASI) record concerning the specified flight is updated in the maintenance database, if this record does not exist a new ASI record is then created. If the anticipated message is not received then a suitable alert is generated.

3.9.2.Starting Point

AOCGround::engineStatusReportRequest() function is called.

3.9.3.Ending Point

End of AOCGround::checkAGComms.

3.9.4.Measurement Result

Transmission and receipt of messages.

Maintenance database is updated either by inserting a new ASI record or by updating the data of an existing ASI record.

3.9.5.Outstanding Issues

None.

3.9.6.Sequence Diagram

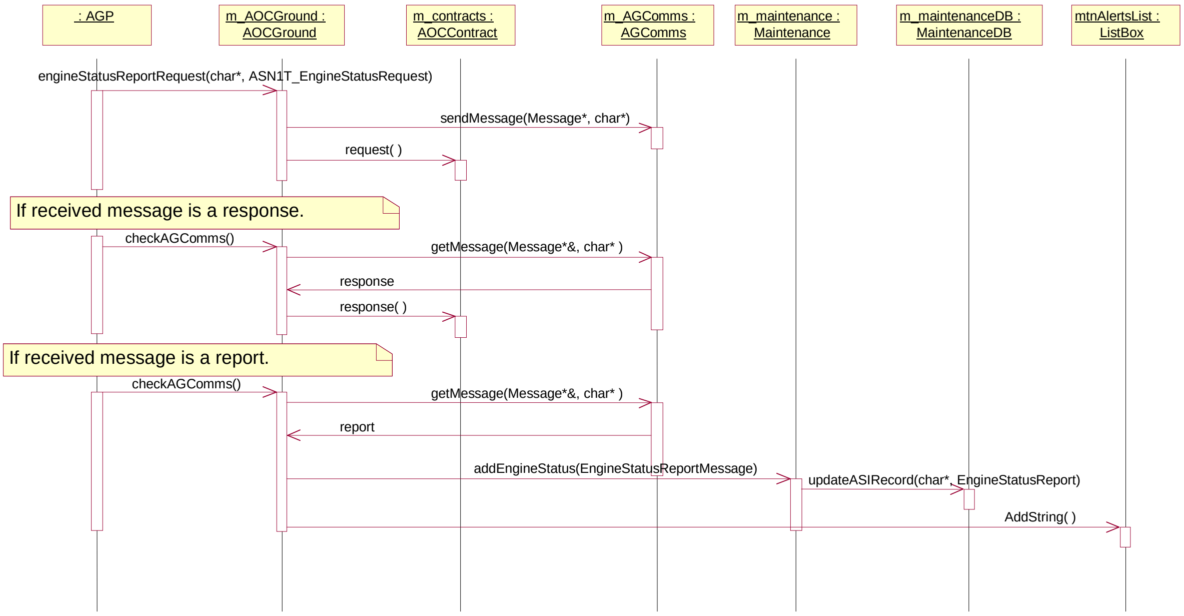

: AGP
m_AOCGround : AOCGround
m_contracts : AOCContract
m_AGComms : AGComms
m_maintenance : Maintenance
m_maintenanceDB : MaintenanceDB
mtnAlertsList : ListBox
engineStatusReportRequest(char*, ASN1T_EngineStatusRequest)
sendMessage(Message*, char*)
request( )
If received message is a response.
checkAGComms()
getMessage(Message*&, char*)
response
response( )
If received message is a report.
checkAGComms()
getMessage(Message*&, char*)
report
addEngineStatus(EngineStatusReportMessage)
updateASIRecord(char*, EngineStatusReport)
AddString( )

Figure 11 – Engine Status reports sequence diagram

3.10.Asset Management: Free Text Uplink

3.10.1.Overview

This capability handles the uplink of a free text message from the AGP to a specified A/C. When the AGP operator requests, using the AGP HMI, the transmission of a free text message, the AOCGround::uplinkFreetext method is called, a new free text message object is generated and the AGComms::sendMessage is called. The AGComms::sendMessage method finds the aircraft GACS address based on the Flight ID provided by the operator and sets the GACS service that was selected by the operator. Finally, the GAPIEndpoint::sendMessage is called, causing the insertion of the free text message on the queue holding the messages to be transmitted.

3.10.2.Starting Point

When the AGP operator requests the transmission of a free text message.

3.10.3.Ending Point

At the end of AOCGround::uplinkFreetext method.

3.10.4.Measurement Result

The queue holding the messages that are to be transmitted holds a new element.

3.10.5.Outstanding Issues

None

3.10.6.Sequence Diagram

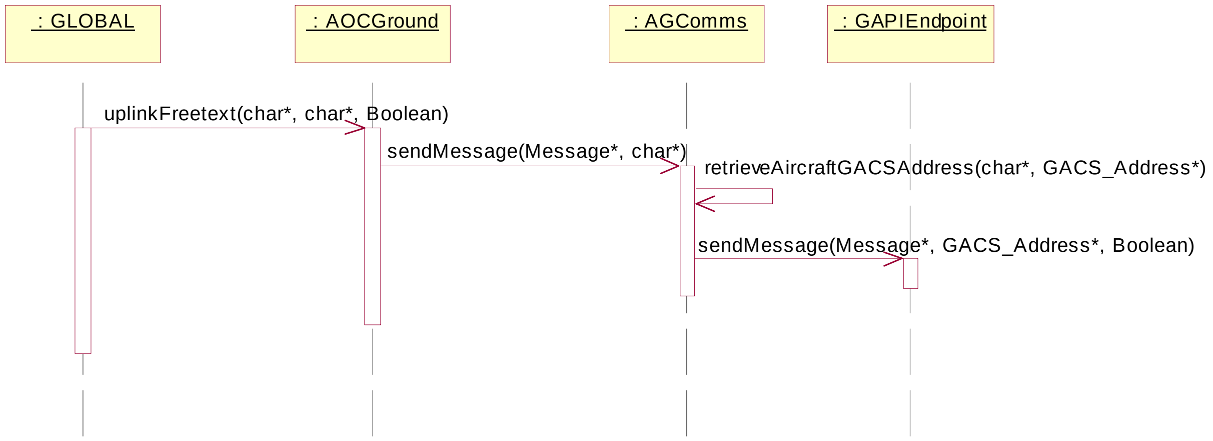

: GLOBAL
: AOCGround
: AGComms
: GAPIEndpoint
uplinkFreetext(char*, char*, Boolean)
sendMessage(Message*, char*)
retrieveAircraftGACSAddress(char*, GACS_Address*)
sendMessage(Message*, GACS_Address*, Boolean)

Figure 12 – Free Text uplink sequence diagram

3.11.Asset Management: Free Text Downlink

3.11.1.Overview

This capability handles the downlink of free text messages from currently registered A/Cs. This functionality is triggered by the return of a free text message from the GAPIEndpoint::getMessage method. After the return of a free text message, the flightId of the A/C that transmitted the message is retrieved using the message senders GACS address (GACS_Address), and the message is logged on the communications server.

3.11.2.Starting Point

The AOCGround object calls the AGComms::getMessage method within the checkAGComms.

3.11.3.Ending Point

End of AOCGround::checkAGComms.

3.11.4.Measurement Result

A received free text messages is logged into CommsServer object.

3.11.5.Outstanding Issues

The sequence diagram shown below (Figure 13) was depicted assuming that the message that is returned by GAPIEndpoint::getMessage is a free text message.

3.11.6.Sequence Diagram

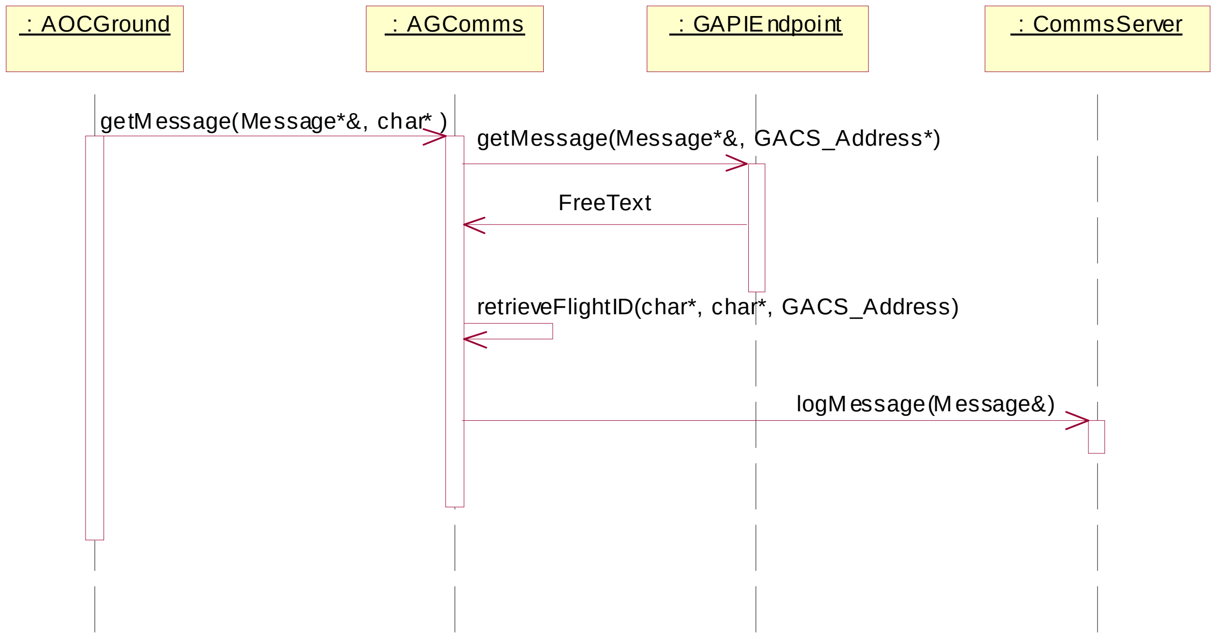

: AOCGround
: AGComms
: GAPIEndpoint
: CommsServer
getMessage(Message*&, char*)
getMessage(Message*&, GACS_Address*)
FreeText
retrieveFlightID(char*, char*, GACS_Address)
logMessage(Message&)

Figure 13- Free Text downlink sequence diagram

3.12.Asset Management: OOOI Reports

3.12.1.Overview

This capability handles the downlink of OOOI reports. This functionality is triggered by the return of a OOOI message from the GAPIEndpoint::getMessage method. After the return of a OOOI message, the flightId of the A/C that transmitted the message is retrieved using the message senders GACS address (GACS_Address), and the message is logged on the communications server. Finally, the OOOI message is added to the OOOI queue and the Flight Operations database is updated with new OOOI data.

3.12.2.Starting Point

The AOCGround object calls the AGComms::getMessage method within the checkAGComms.

3.12.3.Ending Point

End of AOCGround::checkAGComms.

3.12.4.Measurement Result

The queue holding received OOOI messages holds a new element.
Flight Operations database is updated.

3.12.5.Outstanding Issues

The sequence diagram shown below (Figure 14*) was depicted assuming that the message that is returned by GAPIEndpoint::getMessage is a OOOI message.*

3.12.6.Sequence Diagram

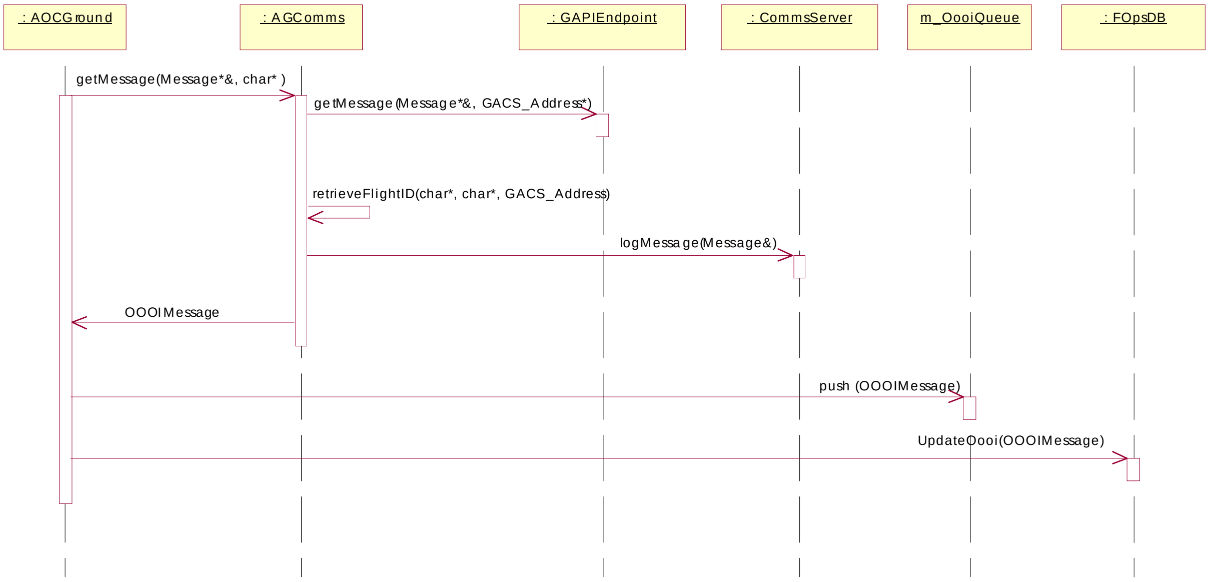


Figure 14 – OOOI reports sequence diagram

3.13.AOC Operator HMI: AGP Main Loop

3.13.1.Overview

This capability is responsible for updating data in the main AGP displays : Flight Progress Display, Maintenance Display and Post-Flight Display. A timer triggers this functionality automatically. The update is done in six major steps. The first step corresponds to the AOCGround::periodicFunction call, which causes the polling of the messages that have arrived, thus updating the AOCGround message queues. The remaining steps correspond to the procedures related with the update of the AGP Displays (getFlightsProgressData call, updateMessages call, updateFlightsProgressData call, updateAlerts call and updateMaintenanceData call). At the end of this sequence, the AGP displays should be refreshed.

3.13.2.Starting Point

Call to theTimerFunction function.

3.13.3.Ending Point

End of theTimerFunction function.

3.13.4.Measurement Result

All AGP displays are updated.

3.13.5.Outstanding Issues

For more details on each AGP main loop step, refer to the following sections:

- *getFlightsProgressData - section 3.14*
- *updateMessages - section 3.18*
- *updateFlightsProgressData – section 3.15*
- *updateAlerts – 3.20*
- *updateMaintenanceData – 3.16*

3.13.6.Sequence Diagram

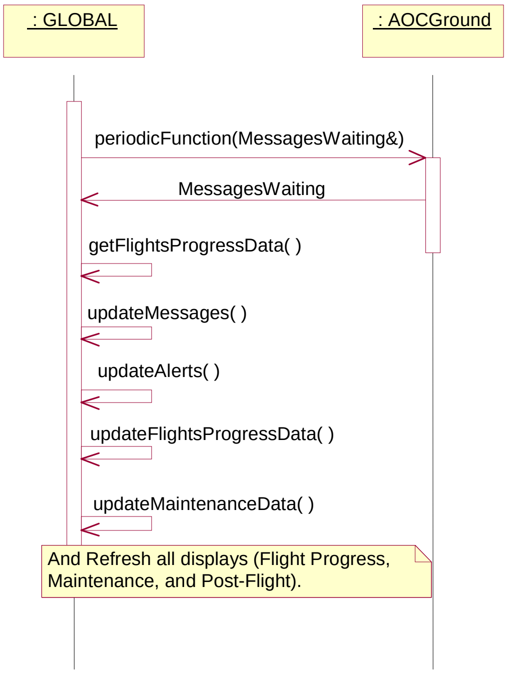

: GLOBAL
: AOCGround
periodicFunction(MessagesWaiting&)
MessagesWaiting
getFlightsProgressData( )
updateMessages( )
updateAlerts( )
updateFlightsProgressData( )
updateMaintenanceData( )
And Refresh all displays (Flight Progress, Maintenance, and Post-Flight).

Figure 15 – AGP main loop sequence diagram

3.14.AOC Operator HMI: Flight Progress Display Data Retrieval

3.14.1.Overview

This capability is responsible for getting updated flight progress data from AOCGround in order to enable the AGP displays update. Flight progress data is considered to be OOOI data, trajectory data, and information about new registered flights and new received messages from registered flights. This functionality is performed on every AGP main loop cycle and consists in the processing and retrieval of the AOCGround queues data. This process is done in four steps as explained below. The first step consists in processing the data retrieved from the queue storing information about new registered flights in AGP. When a new flight Id is retrieved from this queue, the AGP gets all the available operational data (trajectory, passengers, fuel, ETA, etc…) and compiles it into a flight record that is inserted into the list of registered flights. The second step consists in processing the data retrieved from the queue storing new OOOI data. When new OOOI data is retrieved from this queue, the current OOOI status of the respective flight updated with the new OOOI data. If the new flight OOOI status data is "IN", this flight is deleted from the Flight Progress Display and shown instead in the Post-Flight Display. The third step consists in processing the data retrieved from the queue storing new trajectory data. When new trajectory data is retrieved from this queue, the current trajectory of the respective flight is replaced by the new received trajectory. Afterwards, ETA and fuel weight are also updated according the new trajectory. The latest step consists in processing the data retrieved from the queue storing new downlink messages. When this data is retrieved the boolean new messages attribute of the respective flight is updated accordingly.

3.14.2.Starting Point

Call to getFlightsProgressData global function.

3.14.3.Ending Point

End of getFlightsProgressData global function

3.14.4.Measurement Result

FlightsList object is updated with the new data.

3.14.5.Outstanding Issues

Retrieving AOCGround queue messages is done until these queues are empty.

3.14.6.Sequence Diagram

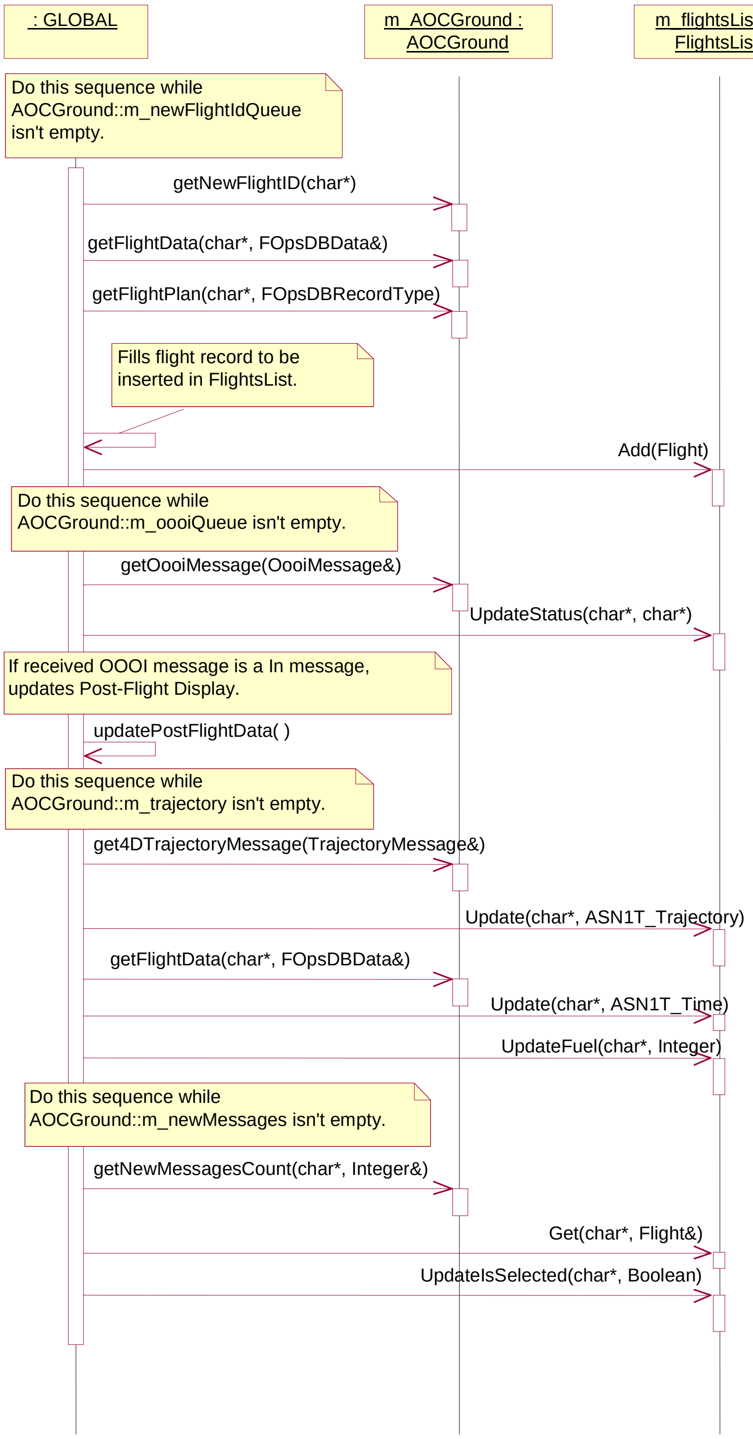

: GLOBAL
m_AOCGround : AOCGround
m_flightsList : FlightsList
Do this sequence while AOCGround::m_newFlightIdQueue isn't empty.
getNewFlightID(char*)
getFlightData(char*, FOpsDBData&)
getFlightPlan(char*, FOpsDBRecordType)
Fills flight record to be inserted in FlightsList.
Add(Flight)
Do this sequence while AOCGround::m_oooiQueue isn't empty.
getOooiMessage(OooiMessage&)
UpdateStatus(char*, char*)
If received OOOI message is a In message, updates Post-Flight Display.
updatePostFlightData( )
Do this sequence while AOCGround::m_trajectory isn't empty.
get4DTrajectoryMessage(TrajectoryMessage&)
Update(char*, ASN1T_Trajectory)
getFlightData(char*, FOpsDBData&)
Update(char*, ASN1T_Time)
UpdateFuel(char*, Integer)
Do this sequence while AOCGround::m_newMessages isn't empty.
getNewMessagesCount(char*, Integer&)
Get(char*, Flight&)
UpdateIsSelected(char*, Boolean)

Figure 16 – Flight Progress Display Data Retrieval sequence diagram

3.15.AOC Operator HMI: Flight Progress Display Data Updating

3.15.1.Overview

This functionality updates the Flight Progress Display (FPD) console. First, the FPD flightsList list box is cleared. Then, for each flight in m_flightsList, flight data is retrieved from m_flightsList and inserted in the FPD flightsList listbox. Within this capability scope, flight data retrieved from m_ flightsList includes: flight id, aircraft type, ETA, departure, destination, OOOI status, flags providing the current status of message contracts and a flag indicating if there are unread messages concerning the respective flight. Finnaly, the current estimated position and heading are retrieved from the AOCGround object in order to update the corresponding fields in the m_flightsList.

3.15.2.Starting Point

Call to updateFlightsProgressData global function.

3.15.3.Ending Point

End of updateFlightsProgressData global function.

3.15.4.Measurement Result

FPD flightsList listbox contents are updated.

3.15.5.Outstanding Issues

None.

3.15.6.Sequence Diagram

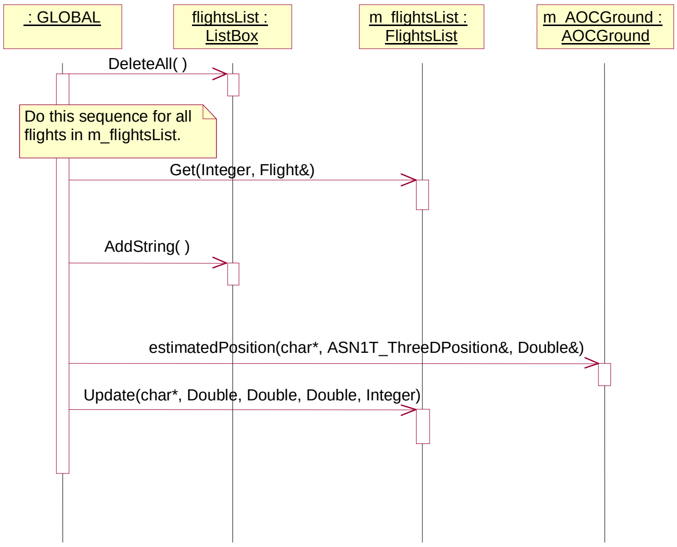

: GLOBAL
flightsList : ListBox
m_flightsList : FlightsList
m_AOCGround : AOCGround
DeleteAll( )
Do this sequence for all flights in m_flightsList.
Get(Integer, Flight&)
AddString( )
estimatedPosition(char*, ASN1T_ThreeDPosition&, Double&)
Update(char*, Double, Double, Double, Integer)

Figure 17 – Flight Progress Display Data Updating sequence diagram

3.16.AOC Operator HMI: Maintenance Display Data Updating

3.16.1.Overview

This functionality manages the update of the Maintenance Display (MD) and all data associated including APU data and Engine Status data. First, MD mtnFlightsList list box is cleared. Then, for each flight in m_flightsList, flight data is retrieved from m_flightsList and inserted in the MD mtnFlightsList listbox. Within this capability scope, flight data includes: flight id, aircraft type, ETA, departure, destination and flags providing the current status of message contracts. If the flight is selected and there is ASI data available, the corresponding ASI record is retrieved the APU text boxes are updated and, if one of the Maintenance Display buttons (EGT, N1, Oil Temperature or Fuel Flow), is pressed the the maintenance graph values are also updated.

3.16.2.Starting Point

Call to updateMaintenanceData global function.

3.16.3.Ending Point

End of updateMaintenanceData global function.

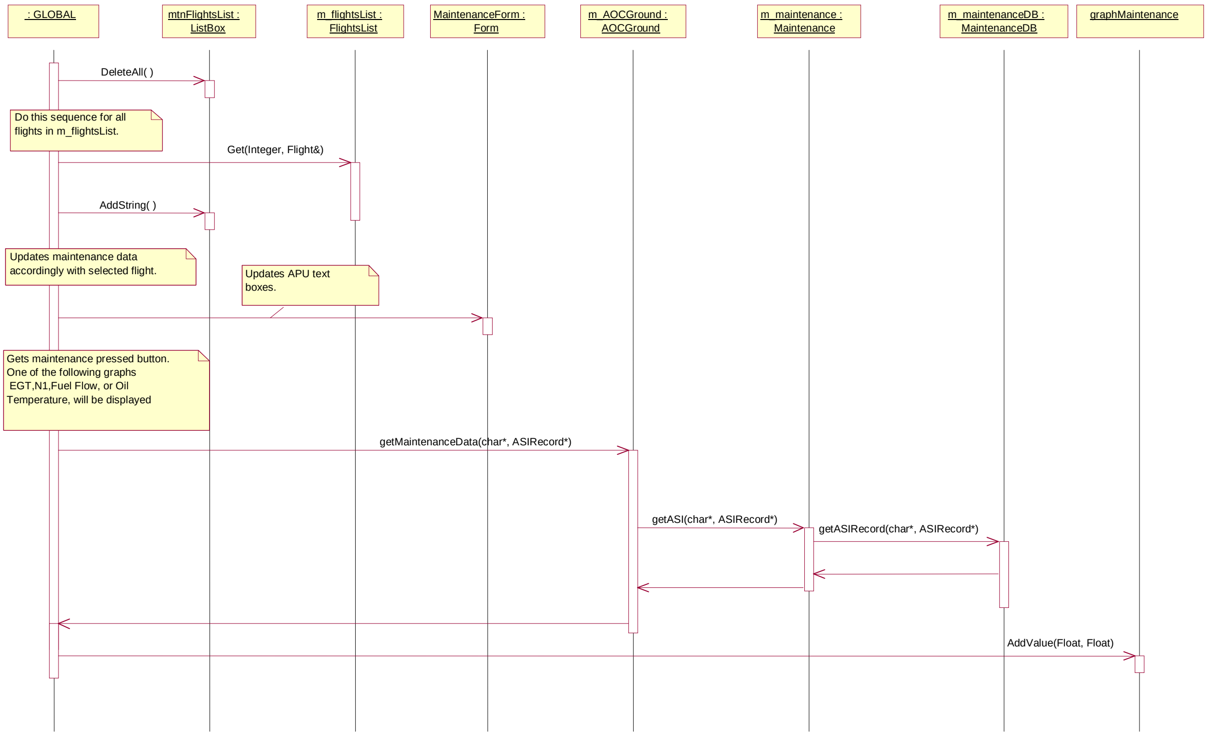
3.16.4.Measurement Result

The Maintenance APU text fields are updated.
The Maintenance data graph shows a modified graph line.
Fields in MD flightsList listbox are updated.

3.16.5.Outstanding Issues

None.

3.16.6.Sequence Diagram



: GLOBAL
mtnFlightsList : ListBox
m_flightsList : FlightsList
MaintenanceForm : Form
m_AOCGround : AOCGround
m_maintenance : Maintenance
m_maintenanceDB : MaintenanceDB
graphMaintenance
DeleteAll( )
Do this sequence for all flights in m_flightsList.
Get(Integer, Flight&)
AddString( )
Updates maintenance data accordingly with selected flight.
Updates APU text boxes.
Gets maintenance pressed button. One of the following graphs EGT,N1,Fuel Flow, or Oil Temperature, will be displayed
getMaintenanceData(char*, ASIRecord*)
getASI(char*, ASIRecord*)
getASIRecord(char*, ASIRecord*)
AddValue(Float, Float)

Figure 18 – Maintenance Display Data Updating sequence diagram

3.17.AOC Operator HMI: Post-Flight Display Data Updating

3.17.1.Overview

This functionality manages the update of the Post-Flight Display (PFD), a facility to aid the analysis of finished flights. First, PFD pfFlightsList listbox is cleared. Then, for each flight in m_flightsList, flight data is retrieved from m_flightsList and inserted in the PFD pfFlightsList listbox. Within this capability scope, flight data includes: flight id, aircraft type, EOBT, departure and destination. Finally, if the flight is selected and one of the Post-Flight graph buttons (Est. or Real) is pressed the post-flight graph, a graph showing real or estimated fuel usage, is updated. If the Est. button is pressed the fuel usage data that was forecast for the flight is extracted from the corresponding flight plan and displayed. Else if the Real button is pressed, the OUT and IN message are retrieved from the CommsServer object in order to acquire the initial and final fuel values, along with all available flight progress reports reporting intermediary fuel values. These values are inserted into the Post-Flight graph.

3.17.2.Starting Point

Call to updatePostFlightData() function.

3.17.3.Ending Point

End of updatePostFlightData() function.

3.17.4.Measurement Result

The Post-Flight graph is updated.
Fields in PFD pfFlightsList listbox are updated.

3.17.5.Outstanding Issues

None.

3.17.6.Sequence Diagram

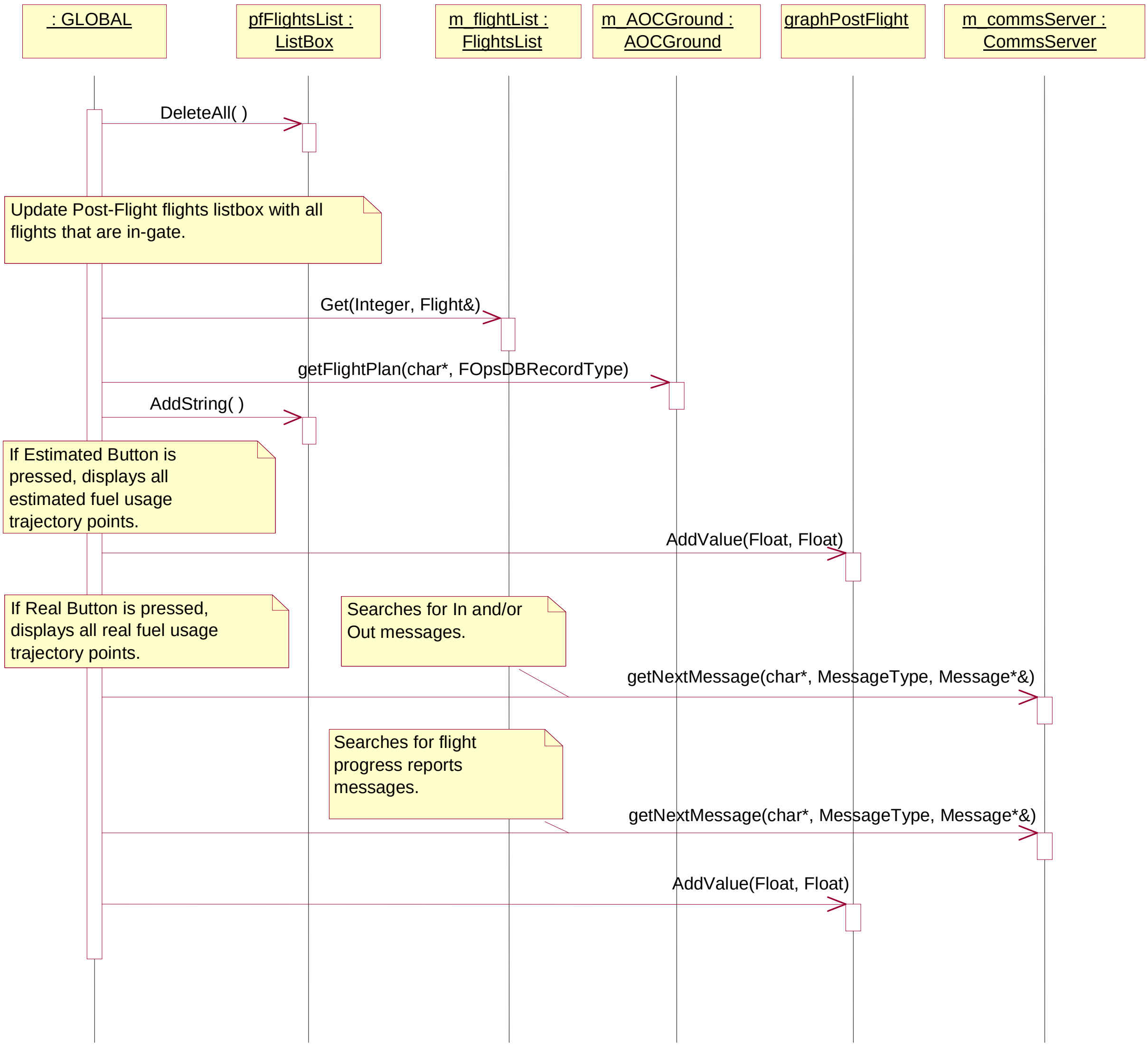

: GLOBAL
pfFlightsList : ListBox
m_flightList : FlightsList
m_AOCGround : AOCGround
graphPostFlight
m_commsServer : CommsServer
DeleteAll( )
Update Post-Flight flights listbox with all flights that are in-gate.
Get(Integer, Flight&)
getFlightPlan(char*, FOpsDBRecordType)
AddString( )
If Estimated Button is pressed, displays all estimated fuel usage trajectory points.
AddValue(Float, Float)
If Real Button is pressed, displays all real fuel usage trajectory points.
Searches for In and/or Out messages.
getNextMessage(char*, MessageType, Message*&)
Searches for flight progress reports messages.
getNextMessage(char*, MessageType, Message*&)
AddValue(Float, Float)

Figure 19 – Post-Flight Display Data Updating sequence diagram

3.18.AOC Operator HMI: Messages Updating

3.18.1.Overview

This capability is responsible for updating the messages and message details listboxes contained in the Flight Progress Display and in the Maintenance Display. When UpdateMessages is called, the selected flight id is retrieved from the FPD and the FPD Messages and message details boxes are cleared. Then, all messages concerning the selected flight Id and the FPD are retrieved from CommsServer and inserted into the FPD Messages listbox. The MD is updated in the same way. The addressed HMI components should be the ones belonging to the MD.

3.18.2.Starting Point

Call to UpdateMessages function.

3.18.3.Ending Point

End of UpdateMessages function.

3.18.4.Measurement Result

The Messages listboxes of the Flight Progress Display and Maintenance Display are updated.

3.18.5.Outstanding Issues

None.

3.18.6.Sequence Diagram

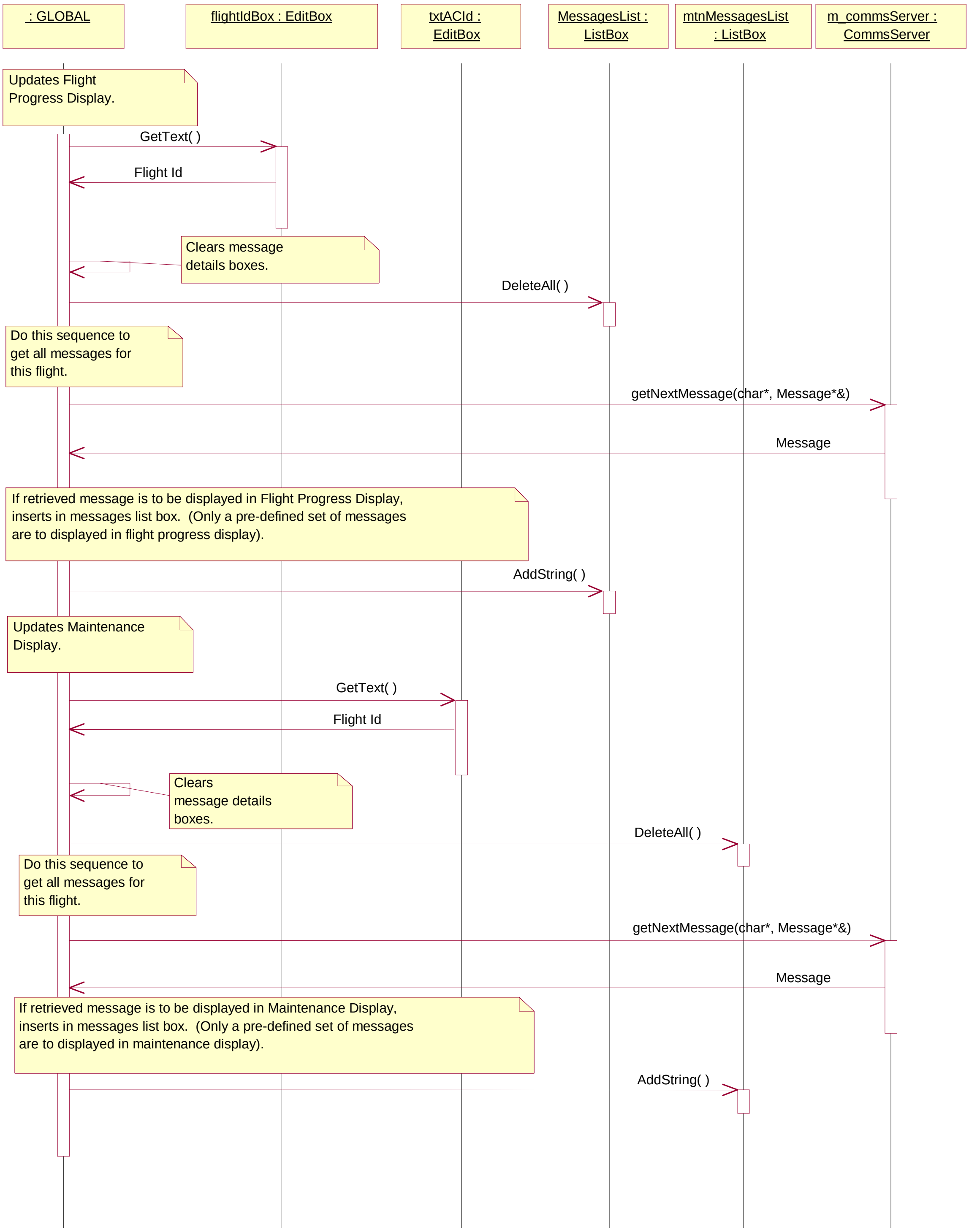

: GLOBAL
flightIdBox : EditBox
txtACId : EditBox
MessagesList : ListBox
mtnMessagesList : ListBox
m_commsServer : CommsServer
Updates Flight Progress Display.
GetText( )
Flight Id
Clears message details boxes.
DeleteAll( )
Do this sequence to get all messages for this flight.
getNextMessage(char*, Message*&)
Message
If retrieved message is to be displayed in Flight Progress Display, inserts in messages list box. (Only a pre-defined set of messages are to displayed in flight progress display).
AddString( )
Updates Maintenance Display.
GetText( )
Flight Id
Clears message details boxes.
DeleteAll( )
Do this sequence to get all messages for this flight.
getNextMessage(char*, Message*&)
Message
If retrieved message is to be displayed in Maintenance Display, inserts in messages list box. (Only a pre-defined set of messages are to displayed in maintenance display).
AddString( )

Figure 20 - Messages updating sequence diagram

3.19.AOC Operator HMI: Message Display

3.19.1.Overview

This capability enables the display of the contents of a message selected by the operator from one of the AGP Messages listbox (there is one Messages listbox for each AGP display). The selected message contains a unique ID that is used to search the communications server in order to retrieve the full contents of the message. When it is found, the contents of the selected message are displayed in the proper display.

3.19.2.Starting Point

Call to SelectMessagesList function.

3.19.3.Ending Point

Return of SelectMessagesList function.

3.19.4.Measurement Result

Message contents are displayed in the respective AGP display.

3.19.5.Outstanding Issues

The processing of selecting a message from the Maintenance Display or from the Post-Flight Display is analogous to the processing regarding the selection of a message from the Flight Progress Display which is depicted below. However the starting and ending points for Maintenance Display are the call and return of the SelectMtnMessagesList function, and the starting and ending points for Post-Flight Display are the call and return of the SelectPFMessagesList function.

3.19.6.Sequence Diagram

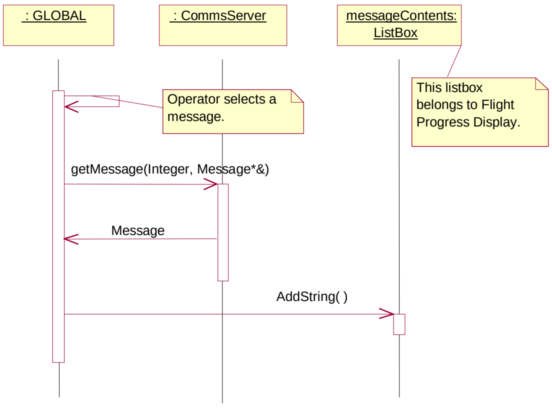

: GLOBAL
: CommsServer
messageContents: ListBox
Operator selects a message.
This listbox belongs to Flight Progress Display.
getMessage(Integer, Message*&)
Message
AddString( )

Figure 21 – Message Display sequence diagram

*3.20.*AOC Operator HMI: Alert Display

3.20.1.Overview

This capability displays all alerts generated by the AGP in the Flight Progress Display and in the Maintenance Display.
This functionality is performed whenever there are alerts in the AOCGround alert queue. Processing of each alert retrieved from the AOCGround alert queue consists basically on building the string that will and then determining in which AGP display the alert should be displayed. The final step is the insertion of the alert string in the corresponding listbox.

3.20.2.Starting Point

Call to updateAlerts function.

3.20.3.Ending Point

End of updateAlerts function.

3.20.4.Measurement Result

All alerts are displayed in the respective AGP display.

3.20.5.Outstanding Issues

None.

3.20.6.Sequence Diagram

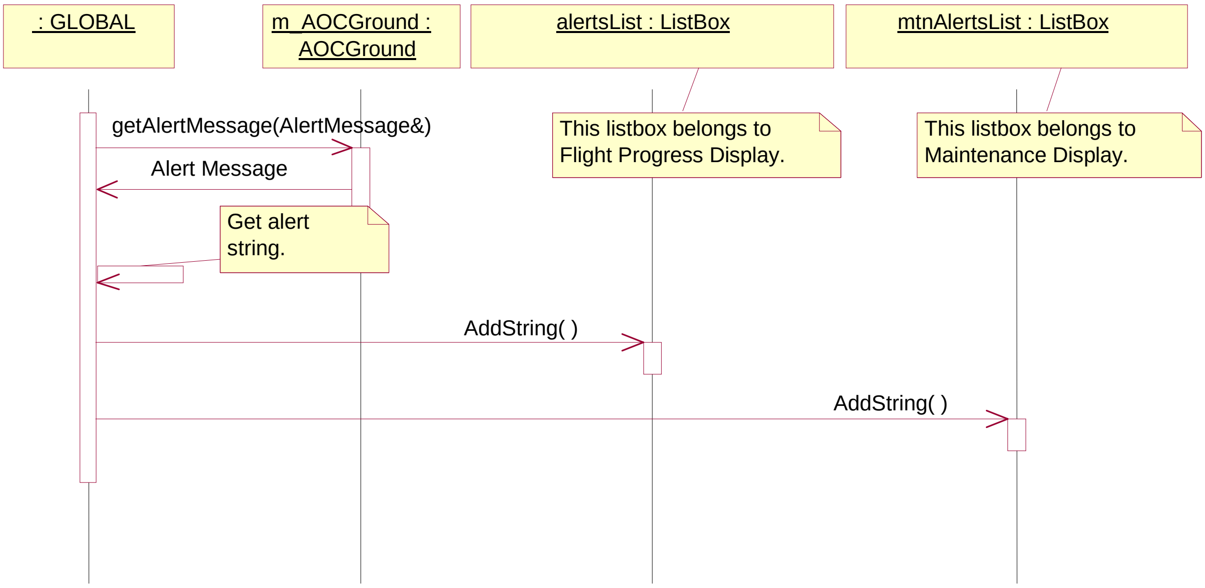


Figure 22 – Alerts Display sequence diagram

*3.21.*AOC Operator HMI: Display Map

3.21.1.Overview

This functionality is responsible to update the Flight Progress Display map.
This is performed in the five steps explained below. In the first step the "show routes" flag is checked. If this flag is set the routes of all flights registered with the AGP are drawn, if this flag is not set only the route of the selected flight is drawn. In the second step all aircraft symbols are drawn. The colour used to draw the aircraft symbol varies according the flight status. In the third step the "show winds" flag is checked. If this flag is set and if there are winds available for the displayed area, the wind symbols are drawn. In step four all non-dynamic data (FIRs, waypoints and airports) are displayed, if the respective "show flags" are set. In the last step, the "show grid" flag is checked. If this flag is set the map grid is drawn.

3.21.2.Starting Point

Call to drawObjects function.

3.21.3.Ending Point

Return of drawObjects function.

3.21.4.Measurement Result

AGP Map is updated.

3.21.5.Outstanding Issues

None

3.21.6.Sequence Diagram

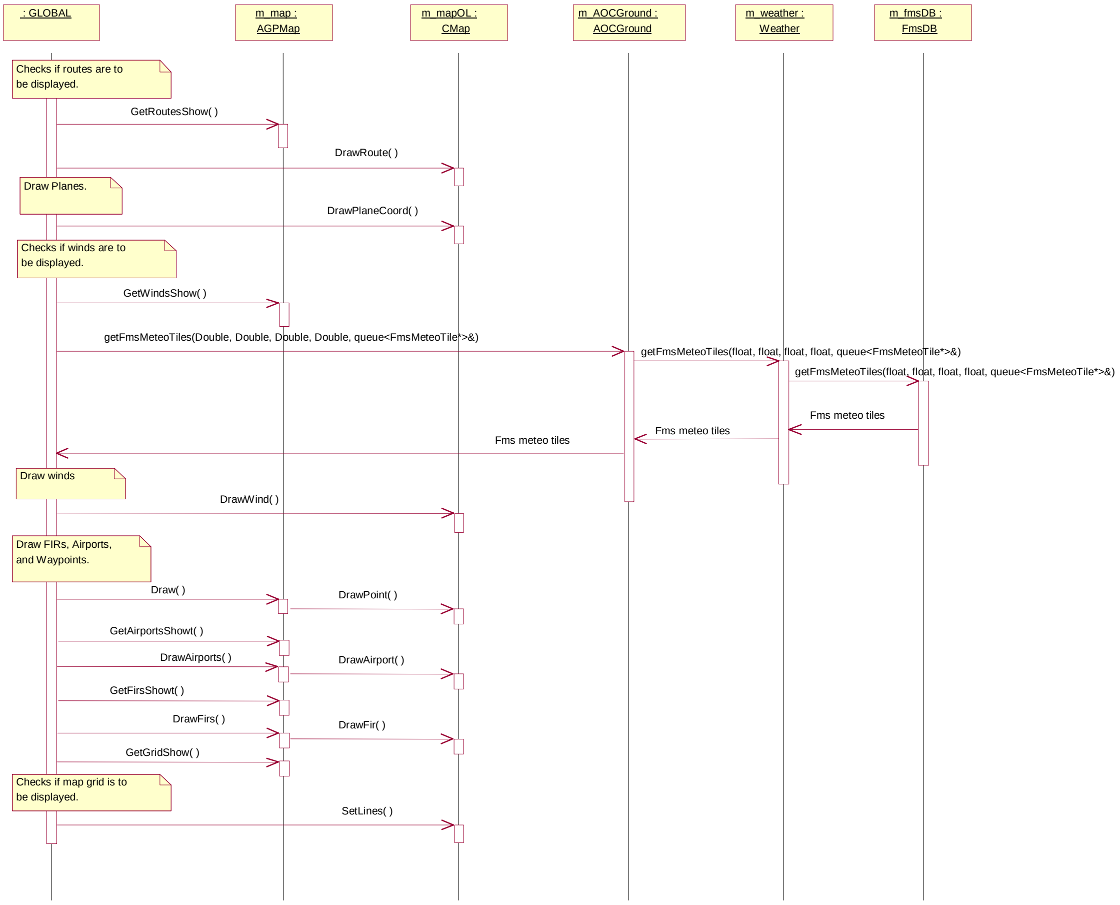


Figure 23 – Display Map sequence diagram

3.22.CFMU Communications: Slot Allocation Message

3.22.1.Overview

This capability handles the receipt of slot allocation messages for a flight from the (simulated) CFMU. Initially CFMUComms loads a schedule of slot allocation transmissions from file, simulating sends from a real CFMU. Periodically AOCGround polls the CFMUComms object to see if a slot allocation has been transmitted by the CFMU. If a slot allocation is received then it triggers a number of subsequent operations. First the flight plan stored in the Flight Planning Database (FPDB) is updated. This involves updating the departure time and the RTAs for each waypoint in the route. Secondly the Flight Operations Database (FopsDB) is checked to see if there is a record corresponding to the flight for which the slot allocation was received for. If the record is present the flight has registered and so a slot allocation is sent out to AGComms for uplink to the aircraft. A timer is created to make sure the uplink is acknowledged before EOBT. The EOBT and ETA and ETA at each waypoint are updated in the FopsDB. If the flight record wasn't found in the FopsDB then the timer for registration is modified to a pre-defined time before the new EOBT.

3.22.2.Starting Point

The function AOCGround::checkCfmuComms() is called.

3.22.3.Ending Point

End of the AOCGround::checkCfmuComms() function.

3.22.4.Measurement Result

FPDB is updated. If flight has registered, a slot allocation is up-linked to the aircraft and the FopsDB updated. If the flight hasn't registered the registration timer is modified.

3.22.5.Outstanding Issues

None

3.22.6.Sequence Diagram

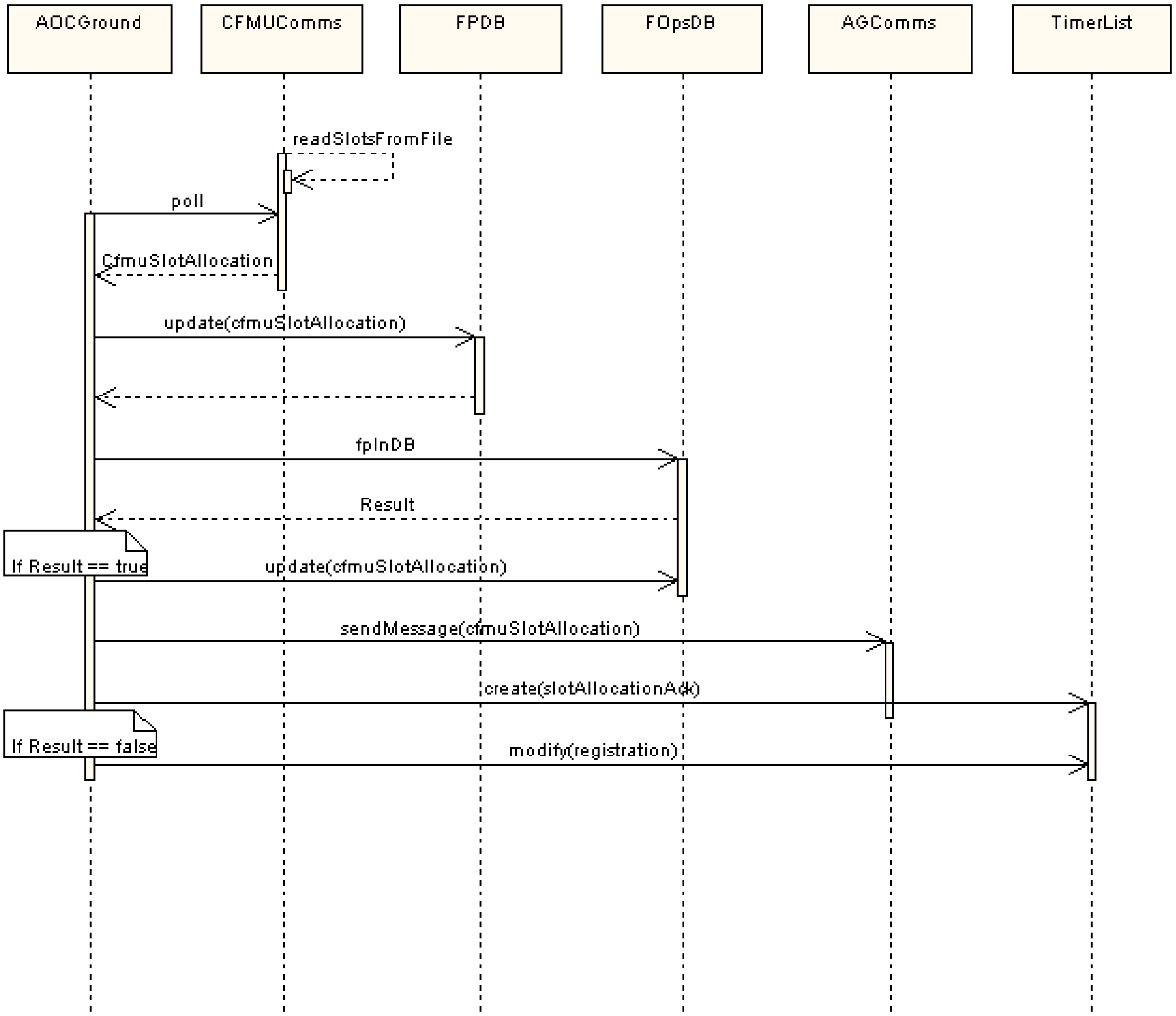


Figure 24 – Slot Allocation from CFMU sequence diagram

3.23.CFMU Communications: Filed Flight Plan

3.23.1.Overview

This capability simulates the transmission of flight plans to the CFMU. Flight plans are filed when the AOC is started. There is no reply from the CFMU in this implementation. A registration timer is set up for each flight, enabling the generation of an alert if an aircraft hasn't registered for a flight a pre-determined time before the flight's EOBT.

3.23.2.Starting Point

AOCGround is created.

3.23.3.Ending Point

End of AOCGround initialisation.

3.23.4.Measurement Result

CFMUComms has received a filed flight plan and a registration timer has been created for each flight in the FPDB.

3.23.5.Outstanding Issues

None

3.23.6.Sequence Diagram

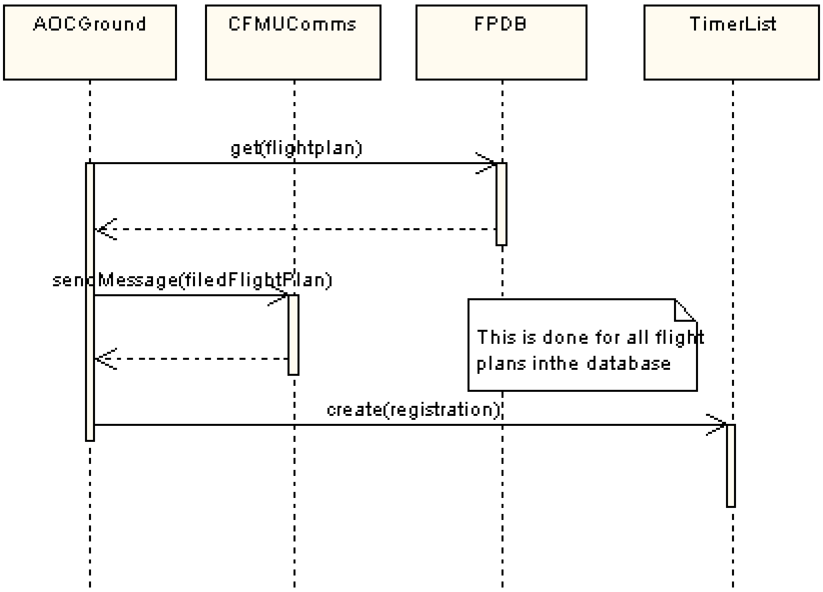

AOCGround
CFMUComms
FPDB
TimerList
get(flightplan)
sendMessage(filedFlightPlan)
This is done for all flight plans inthe database
create(registration)

Figure 25 – Filed Flight Plan sequence diagram

3.24. Flight Progress: Four-D Trajectory Request

3.24.1.Overview

This capability enables the AOC to receive a copy of the trajectory that is currently in the aircraft's FMS. The user of the AOC clicks on the HMI and this triggers the AOCGround to send a 4D trajectory request to the AGComms interface. A timer is created enabling an alert to be generated if the trajectory is not received within a pre-determined time. The down-linked trajectory is received on the AGComms interface and the trajectory timer cancelled. The trajectory is checked for validity and any significant differences from the previous trajectory held in the Flight Operations Database (FopsDB), before being stored.

3.24.2.Starting Point

The user clicks a button on the HMI to request the aircraft to downlink its trajectory.

3.24.3.Ending Point

The FopsDB is updated with the new trajectory.

3.24.4.Measurement Result

Trajectory is received and FopsDb updated.

3.24.5.Outstanding Issues

None

3.24.6.Sequence Diagram

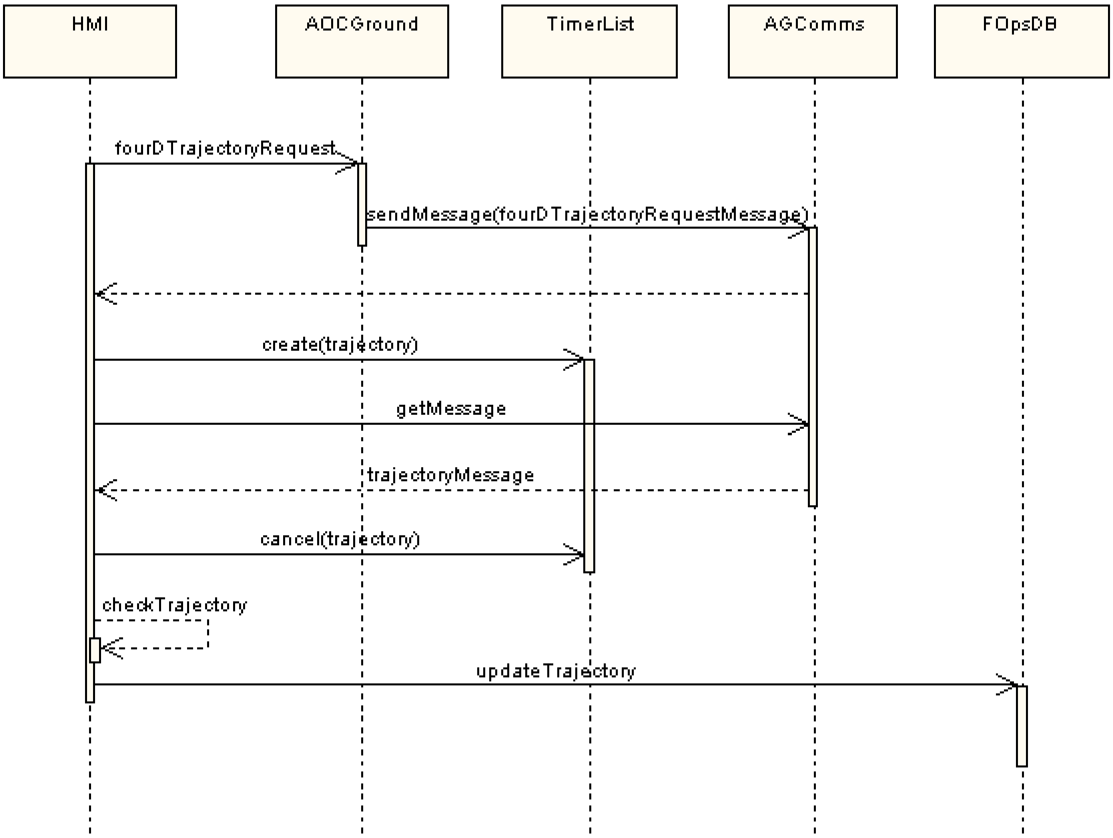


Figure 26 – Four D Trajectory Request sequence diagram

3.25.Sequencing: Periodic function

3.25.1.Overview

This functionality triggers all the processing in the AOCGround. Timeouts are checked. External interfaces checked for new messages and if received processing of messages is triggered. The function ends by notifying the caller (HMI) of the number of messages that are waiting to be removed from the internal queues in AOCGround by the HMI for further processing.

3.25.2.Starting Point

HMI calls AOCGround::periodicFunction().

3.25.3.Ending Point

AOCGround::periodicFunction() ends and the number of messages on queues is returned.

3.25.4.Measurement Result

Function returns number of messages for the HMI to extract from the queues.

3.25.5.Outstanding Issues

None

3.25.6.Sequence Diagram

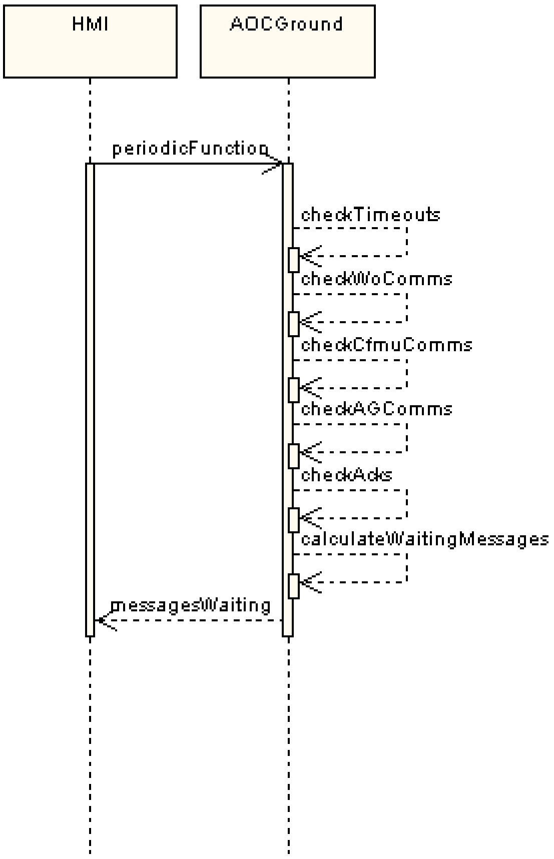


Figure 27 – Periodic Function sequence diagram

3.26.Sequencing: Check For Timeouts

3.26.1.Overview

This functionality enables alerts to be generated if a timer set for the receipt of a message has not been cancelled before the deadline has expired. AOCGound calls the numAlerts() function in TimerList which triggers TimerList to process the timers it has stored and generate alerts for those which have passed their deadline. These are alerts are retrieved by AOCGround and buffered for display by the HMI.

3.26.2.Starting Point

AOCGround::checkTimeOuts() function is called.

3.26.3.Ending Point

AOCGround::checkTimeOuts() function finishes.

3.26.4.Measurement Result

Alerts are added to the queue in AOCGround for all events which have passed their deadline.

3.26.5.Outstanding Issues

None

3.26.6.Sequence Diagram

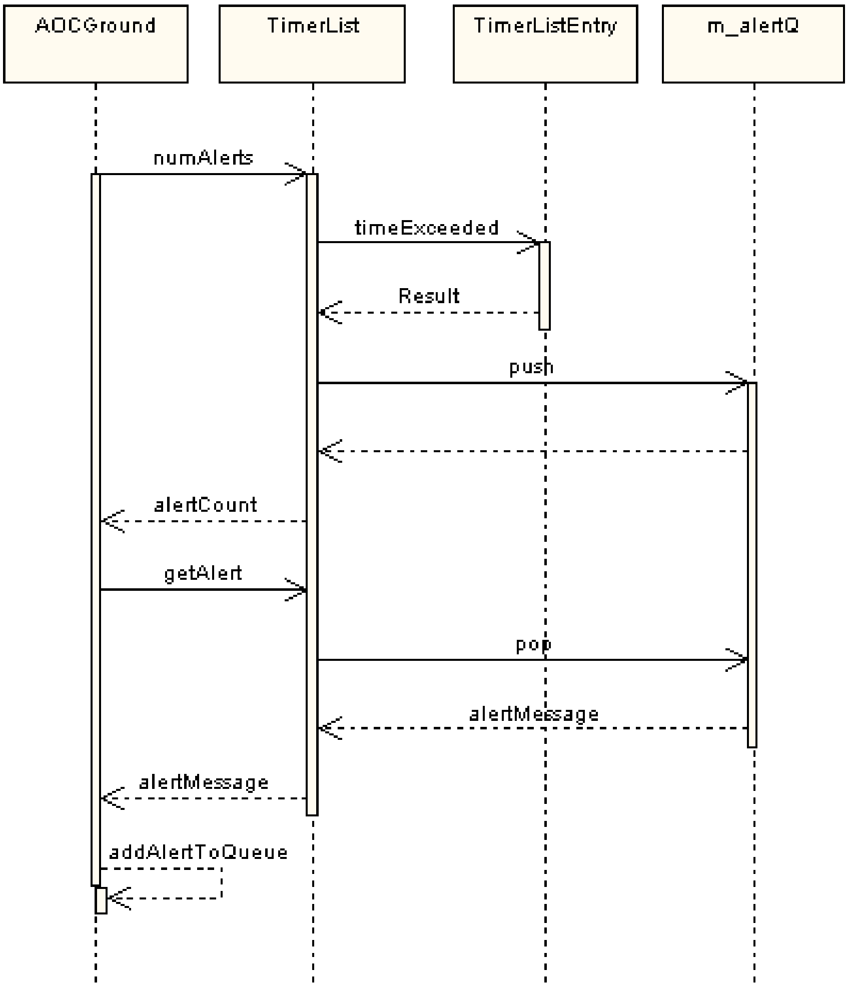


Figure 28- Checking for Timeouts sequence diagram

3.27.Asset Management: Aircraft Met Reports

3.27.1.Overview

This functionality allows the ground platform to send a request to an aircraft for a meteorological report. The request can be either demand or periodic. A demand request is satisfied when a report is received, a periodic request is satisfied by the receipt of reports at a specified time interval. If a request cannot be satisfied by the aircraft then a response is downlinked rejecting the request. If the request cannot be satisfied immediately then a response is sent accepting the request and a report is downlinked afterwards. If the anticipated message is not received then a suitable alert is generated. Han a report is received it is logged and converted to AMDAR format and sent to Weather Office Comms interface for simulated transmission to the weather office.

3.27.2.Starting Point

AOCGround::airMetReportRequest() function is called.

3.27.3.Ending Point

AMDAR sent to Weather Office Comms Interface.

3.27.4.Measurement Result

Transmission and receipt of relevant messages.
Successful AMDAR conversion.
Generation of suitable alerts.

3.27.5.Outstanding Issues

None

3.27.6.Sequence Diagram

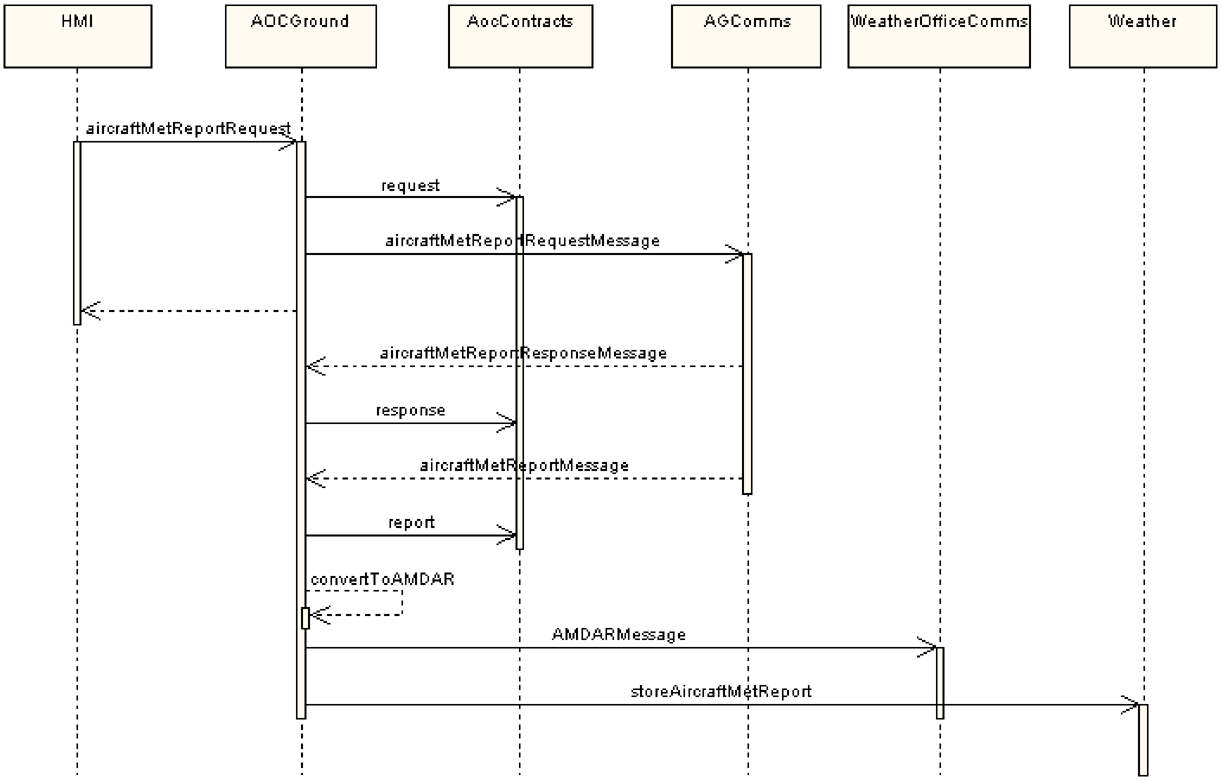


Figure 29- Aircraft Met Reports sequence diagram

3.28.Asset Management: Flight Progress Reports

3.28.1.Overview

This functionality allows the ground platform to monitor the progress of a flight. The ground platform can request flight progress data on a demand or periodic basis. A demand request is satisfied when a report is received, a periodic request is satisfied by the receipt of reports at a specified time interval. If a request cannot be satisfied by the aircraft then a response is downlinked rejecting the request. If the request cannot be satisfied immediately then a response is sent accepting the request and a report is downlinked afterwards. If the anticipated message is not received then a suitable alert is generated.
The request includes the request of the optional data items: eta, next reporting point (NRP) and speed.
When a report is received the ETAs for the next reporting point and subsequent points are updated. This is done by, in order of preference, aircraft provided data, (reported position / reported speed) derived ETA, or (reported position / derived speed) derived ETA.
If the NRP is present in the report then it is compared with the previous known NRP and if significantly different on position or time an alert is generated. If the reported position, fuel or ETA is significantly different to the estimated values then alerts are generated.

3.28.2.Starting Point

AOCGround::flightProgressRequest() function is called.

3.28.3.Ending Point

Exit of processInMessage() function

3.28.4.Measurement Result

Transmission and receipt of messages.
Generation of alerts.
Updating of stored trajectory, ETAs, position, fuel and speed.

3.28.5.Outstanding Issues

None

3.28.6.Sequence Diagram

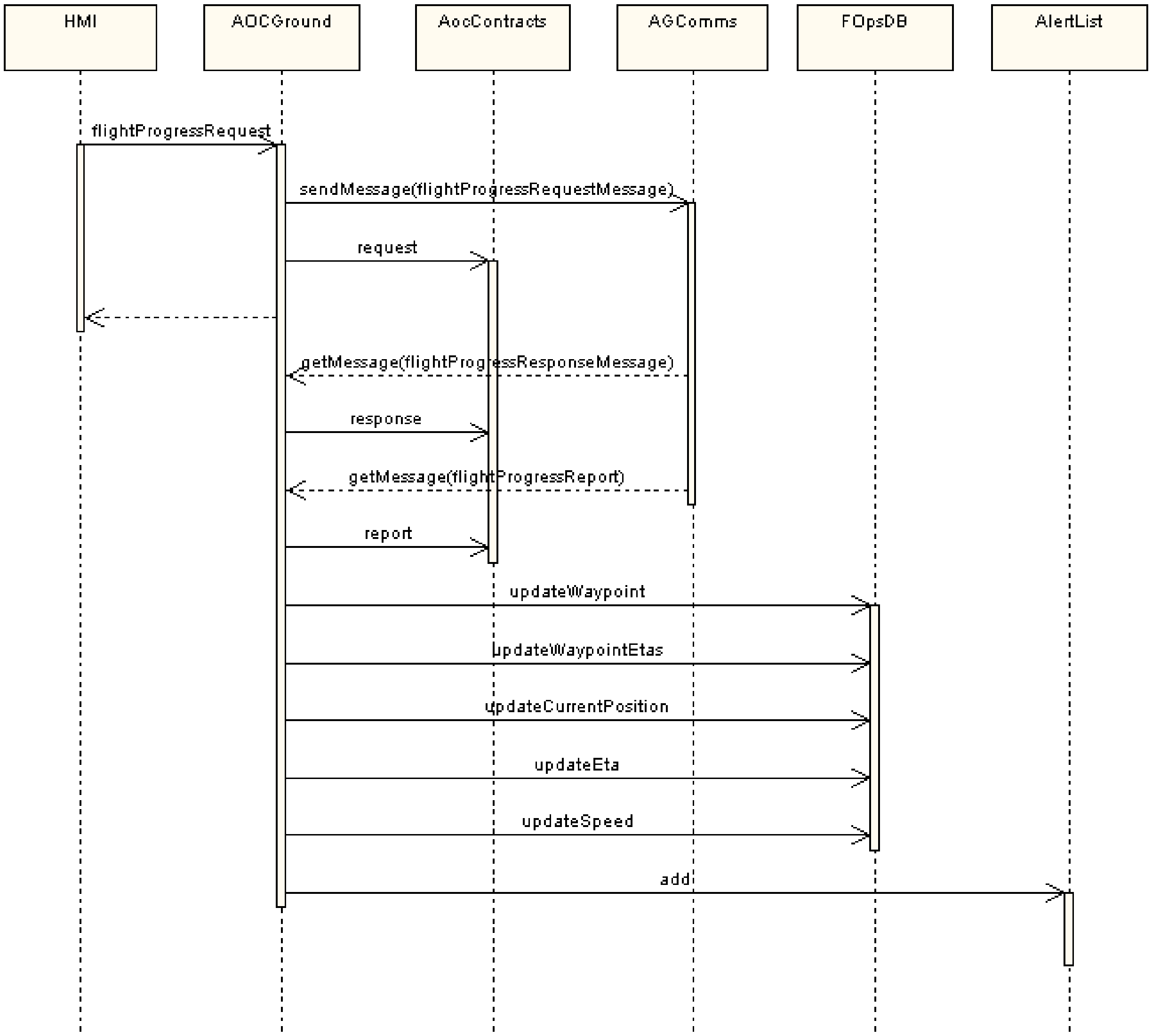


Figure 30- Flight Progress Reports sequence diagram

3.29.Collaborative Decision Making: In-Flight Traffic Management

3.29.1.Overview

This functionality allows the ground platform to uplink a set of constraint points to an aircraft to replace those uplinked as part of the company flight plan. The constraints can be uplinked before the aircraft has reached its top of climb. Following the uplink of a constraints list message the ground system will expect to receive a Constraints Acceptance message, either accepting or rejecting the request. Finally a cleared trajectory message will be downlinked confirming the new points. If the Cleared Trajectory is significantly different from the uplinked constraints then the user is notified by the generation of an alert. If either of these messages are not received within a specified timeframe then a suitable alert is generated.

3.29.2.Starting Point

AOCGround::iFTMTrigger() function is called.

3.29.3.Ending Point

1. *Constraints acceptance message received and response is rejected. Alert generated.*
2. *Cleared Trajectory received and trajectory in FopsDB is updated.*

3.29.4.Measurement Result

Successful transmission of messages.
Updating of database with new points.
Correct generation of alerts.

3.29.5.Outstanding Issues

None

3.29.6.Sequence Diagram

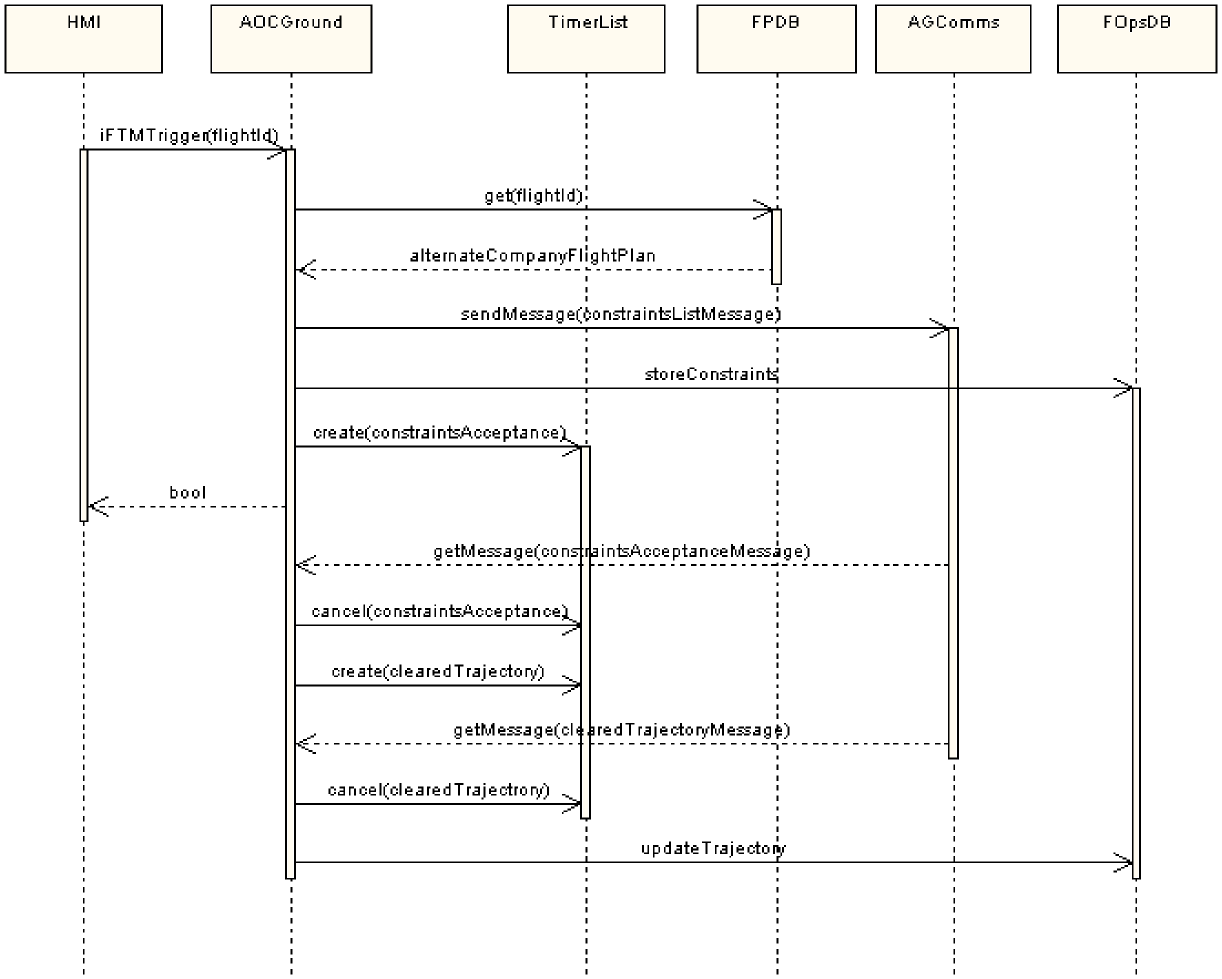

HMI
AOCGround
TimerList
FPDB
AGComms
FOpsDB
iFTMTrigger(flightId)
get(flightId)
alternateCompanyFlightPlan
sendMessage(constraintsListMessage)
storeConstraints
create(constraintsAcceptance)
bool
getMessage(constraintsAcceptanceMessage)
cancel(constraintsAcceptance)
create(clearedTrajectory)
getMessage(clearedTrajectoryMessage)
cancel(clearedTrajectrory)
updateTrajectory

Figure 31- In-Flight Traffic Management sequence diagram

3.30.Flight Plan: Loadsheet

3.30.1.Overview

This functionality enables the ground platform to uplink to the aircraft a loadsheet containing details such as fuel quantity, number of passengers, weight of baggage and balance figures. A timer is set up to notify the user if the loadsheet has not been sent to an aircraft a pre-determined time before EOBT. On receipt of a loadsheet the aircraft responds with a □oadsheet Ack message. A similar timer is set up for the loadsheet acknowledgement to ensure it is received a pre-determined time before EOBT.

3.30.2.Starting Point

AOCGround::sendLoadsheet() is called.

3.30.3.Ending Point

Loadsheet Ack message is received.

3.30.4.Measurement Result

Loadsheet can be uplinked.
Loadsheet Ack can be received.
Timers are created and alerts generated.

3.30.5.Outstanding Issues

Balance data is fixed. (Taken from an Easy-Jet Flight Plan).

3.30.6.Sequence Diagram

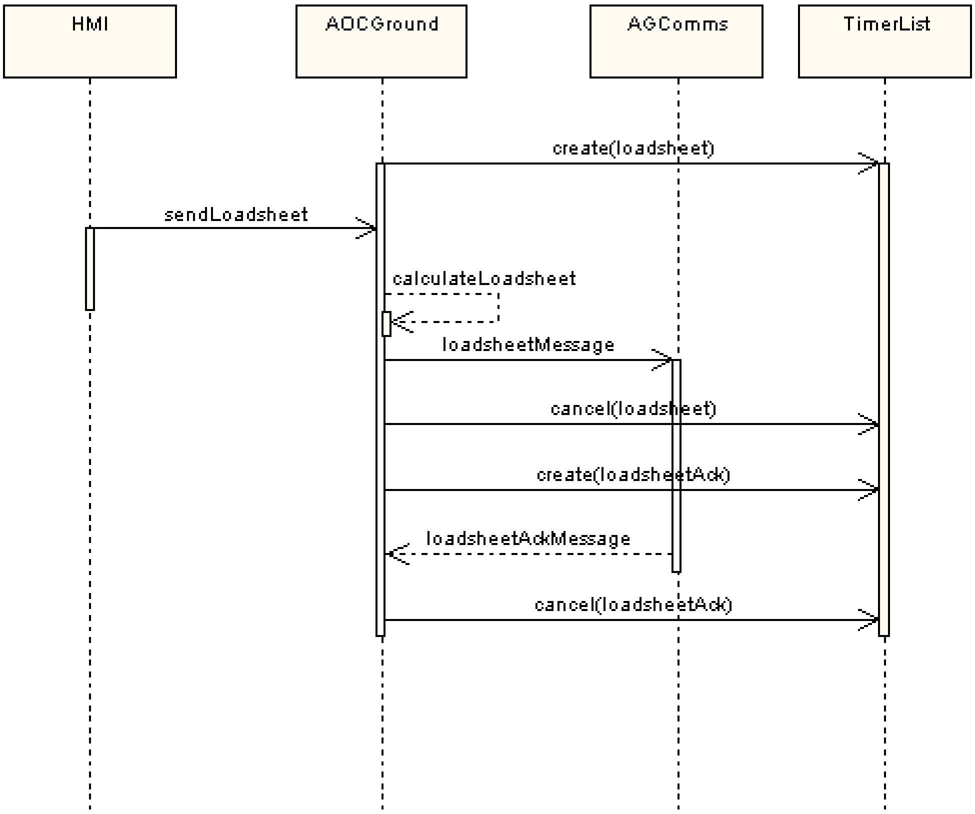


Figure 32- Loadsheet sequence diagram

4.REQUIREMENTS TRACEABILITY

AGP System Requirements are defined in MA-AFAS Ground System Requirements document [MA-AFAS D38]. The current section provides traceability between these system requirements and the AGP functionalities/capabilities described in Section 3. It should be noted that the identification number associated to each requirement is as defined in [MA-AFAS D38].

4.1.Air-Ground Communications: Reception

Requirement ID	Description:
AGP_COM_001	The AOC Ground Platform shall use the Generic ATN Communication Service (GACS) application defined in ref. (TBC) to support end-to-end air-ground communications.
AGP_COM_002	The AOC Ground Platform shall be capable of sending/receiving ATN messages through a VDLM4 air/ground data-link.
AGP_COM_003	The AOC Ground Platform shall provide a communication service to support the registration of a flight for datalink.
AGP_COM_004	The AOC Ground Platform shall provide a communication service to handle the non-receipt of confirmation for uplink messages sent using the GACS confirmed service.
AGP_COM_011	The AOC Ground Platform shall provide a communication service to support the exchange of Meteorological data with a given aircraft.
AGP_COM_013	The AOC Ground Platform shall provide a communication service to support the exchange of flight support data with a given aircraft.
AGP_COM_015	The AOC Ground Platform shall provide a suitable warning to the operator if confirmation of the Slot Allocation has not been received by a pre-determined time before the Estimated Off Block Time.
AGP_COM_021	The AOC Ground Platform shall provide a communication service to support the exchange of Aircraft Systems Information.
AGP_COM_031	The AOC Ground Platform shall provide a communication service to support the exchange of flight progress information with a given aircraft.
AGP_COM_042	The AOC Ground Platform shall provide a communication service supporting the exchange of 4D trajectory data with a given aircraft.
AGP_COM_043	The AOC Ground Platform shall provide a communication service supporting the exchange of free text messages with a given aircraft.
AGP_COM_044	The AOC Ground Platform shall provide a communication service supporting the downlink of OOOI messages from a given aircraft.
AGP_COM_045	The AOC Ground Platform shall provide a communication service supporting the exchange of 4D trajectory data with a given aircraft.
AGP_MTC_017	The AOC Ground Platform shall be able to receive APU Reports from an aircraft.
AGP_MTC_030	The AOC Ground Platform shall be able to receive Engine Reports from an aircraft.
AGP_CDM_017	The AOC Ground Platform shall be able to receive 4D trajectory data from an aircraft.
AGP_AMG_021	The AOC Ground Platform shall be able to receive Flight Progress Reports from an aircraft.
AGP_AMG_210	The AOC Ground Platform shall be able to receive Aircraft Met Reports from the aircraft.

4.2.Air-Ground Communications: Transmission

Requirement ID	Description:
AGP_COM_001	The AOC Ground Platform shall use the Generic ATN Communication Service (GACS) application defined in ref. (TBC) to support end-to-end air-ground communications.
AGP_COM_002	The AOC Ground Platform shall be capable of sending/receiving ATN messages through a VDLM4 air/ground data-link.
AGP_COM_004	The AOC Ground Platform shall provide a communication service to handle the non-receipt of confirmation for uplink messages sent using the GACS confirmed service.
AGP_COM_011	The AOC Ground Platform shall provide a communication service to support the exchange of Meteorological data with a given aircraft.
AGP_COM_013	The AOC Ground Platform shall provide a communication service to support the exchange of flight support data with a given aircraft.
AGP_COM_021	The AOC Ground Platform shall provide a communication service to support the exchange of Aircraft Systems Information.
AGP_COM_031	The AOC Ground Platform shall provide a communication service to support the exchange IFTM data with a given aircraft.
AGP_COM_042	The AOC Ground Platform shall provide a communication service supporting the exchange of 4D trajectory data with a given aircraft.
AGP_COM_043	The AOC Ground Platform shall provide a communication service supporting the exchange of free text messages with a given aircraft.
AGP_COM_045	The AOC Ground Platform shall provide a communication service supporting the exchange of 4D trajectory data with a given aircraft.
AGP_FPL_190	The AOC Ground Platform shall be able to generate and uplink a Loadsheet message to a specified aircraft.
AGP_ FPL _280	The AOC Ground Platform shall encode requested meteo reports on a format suitable for transmission through the datalink.

4.3.Air-Ground Communications: Alerts

Requirement ID	Description:
AGP_COM_004	The AOC Ground Platform shall provide a communication service to handle the non-receipt of confirmation for uplink messages sent using the GACS confirmed service.
AGP_COM_006	The AOC Ground Platform shall provide a suitable warning to the operator if confirmation of transmission of a GACS confirmed message has not been received.
AGP_COM_015	The AOC Ground Platform shall provide a suitable warning to the operator if confirmation of the Slot Allocation has not been received by a pre-determined time before the Estimated Off Block Time.
AGP_AMG_300	The AOC Ground Platform shall provide a suitable warning to the operator if confirmation of transmission of a GACS confirmed free text message has not been received.
AGP_MTC_031	The AOC Ground Platform shall provide a suitable warning to the operator if an Engine Report is not received within a pre-determined time after the expected time. The expected time will either be defined by the reporting interval of periodic reports or the time a demand report was requested, as appropriate.

4.4.Communications To Meteo Office: Meteo Forecast and Meteo Reports

Requirement ID	Description:

AGP_COM_080	The AOC Ground Platform shall be able to receive TAFs, METARs, and SIGMETs from an external provider. Note: For MA-AFAS this communication will be simulated.
AGP_COM_090	The AOC Ground Platform shall be able to receive meteo forecast reports from an external provider. Note: For MA-AFAS this communication will be simulated.
AGP_FPL_240	The AOC Ground Platform shall be able to store received meteo forecast reports enabling the AOC Ground Platform to provide them, at a later time, to aircraft.
AGP_FPL_260	The AOC Ground Platform shall be able to store meteo reports.

4.5.Flight Plan: FMS Meteo

Requirement ID	Description:
AGP_FPL_250	The AOC Ground Platform shall, when a flight registers for datalink communication, automatically associate meteo forecasts reports with that flight, and transmit them to the flight as FMS Meteo messages.

4.6.Flight Plan: Meteo Reports

Requirement ID	Description:
AGP_FPL_270	The AOC Ground Platform shall be able to retrieve a meteo report requested by a given aircraft.
AGP_FPL_290	On receipt of a Meteo report request from an aircraft which has registered with the AOC Ground Platform for datalink, the AOC Ground Platform shall transmit the requested meteo report to the aircraft.
AGP_FPL_300	If the AOC Ground Platform is unable to send a Meteo report requested by an aircraft which has registered with the AOC Ground Platform for datalink, the AOC Ground Platform shall transmit a NOT AVAILABLE message to the aircraft.

4.7.Maintenance: SNAG Reports

Requirement ID	Description:
AGP_MTC_001	The AOC Ground Platform shall be able to store all SNAG reports received from a given aircraft.
AGP_MTC_002	The AOC Ground Platform shall provide facilities to associate information about aircraft technical malfunctions based on SNAG reports with flight progress information.

4.8.Maintenance: APU Reports

Requirement ID	Description:
AGP_MTC_011	The AOC Ground Platform shall be able to store all received APU Reports from a given aircraft.
AGP_MTC_014	On Operator request, the AOC Ground Platform shall send a request for an APU report to a specified aircraft.
AGP_MTC_015	The AOC Ground Platform shall receive and process APU Report Responses from an aircraft.

4.9.Maintenance: Engine Status Reports

Requirement ID	Description:

AGP_MTC_019	The AOC Ground Platform shall provide a suitable warning to the operator if an APU report is not received within a pre-determined time after the time a demand report was request.
AGP_MTC_021	The AOC Ground Platform shall be able to store all received Engines Reports.
AGP_MTC_022	The AOC Ground Platform shall be able to provide to the user, information concerning the current status of the Aircraft Engines.
AGP_MTC_023	The AOC Ground Platform shall provide facilities to associate information about Aircraft Engines performance with flight progress information.
AGP_MTC_025	On operator request, the AOC Ground Platform shall send a request for an Engine Report to a specified aircraft.
AGP_MTC_026	The AOC Ground Platform shall allow the operator to specify what information is required from the aircraft Engine Report in the Engine Report Request.
AGP_MTC_027	The AOC Ground Platform shall support Periodic Requests and On-demand Requests for Engine reports.
AGP_MTC_028	The AOC Ground Platform shall receive and process Engine Report Responses from an aircraft.
AGP_MTC_031	The AOC Ground Platform shall provide a suitable warning to the operator if an Engine Report is not received within a pre-determined time after the expected time. The expected time will either be defined by the reporting interval of periodic reports or the time a demand report was requested, as appropriate.

4.10.Asset Management: Free Text Uplink

Requirement ID	Description:
AGP_AMG_250	The AOC Ground Platform shall allow the AOC operator to communicate with the aircraft (pilot) through text messages.
AGP_AMG_280	The AOC Ground Platform shall provide the operator facilities to select and display a free text messages.
AGP_AMG_290	The AOC Ground Platform shall enable the operator to choose between sending a free text message using the GACS confirmed service or sending it using the GACS non-confirmed service.

4.11.Asset Management: Free Text Downlink

Requirement ID	Description:
AGP_AMG_250	The AOC Ground Platform shall allow the AOC operator to communicate with the aircraft (pilot) through text messages.
AGP_AMG_260	The AOC Ground Platform shall inform the operator of the receipt of a freetext message from the aircraft.
AGP_AMG_270	The AOC Ground Platform shall be able to store all received free text messages.
AGP_AMG_280	The AOC Ground Platform shall provide the operator facilities to select and display a free text messages.

4.12.Asset Management: OOOI Reports

Requirement ID	Description:
AGP_AMG_001	The AOC Ground Platform shall be able to keep track of an aircraft flight phase using OOOI information.
AGP_AMG_003	The AOC Ground Platform shall be able to store all received OOOI reports.

4.13.AOC Operator HMI: AGP Main Loop

4.14.AOC Operator HMI: Flight Progress Data Retrieval and Flight Progress Data Updating

Requirement ID	Description:
AGP_AMG_001	The AOC Ground Platform shall be able to keep track of an aircraft flight phase using OOOI information.
AGP_AMG_250	The AOC Ground Platform shall allow the AOC operator to communicate with the aircraft (pilot) through text messages.
AGP_ AMG_260	The AOC Ground Platform shall inform the operator of the receipt of a freetext message from the aircraft.
AGP_AMG_280	The AOC Ground Platform shall provide the operator facilities to select and display a free text message.
AGP_AMG_290	The AOC Ground Platform shall enable the operator to choose between sending a free text message using the GACS confirmed service or sending it using the GACS non-confirmed service.
AGP_AMG_300	The AOC Ground Platform shall provide a suitable warning to the operator if confirmation of transmission of a GACS confirmed free text message has not been received.
AGP_HMI_004	The Ground Platform HMI shall provide functionality to enable the operator to compose, and read, text messages.
AGP_HMI_006	The AOC Ground Platform HMI shall support the configuration of the AOC Ground Platform communication interface.
AGP_HMI_007	The AOC Ground Platform HMI shall provide facilities to configure the AOC Ground Platform HMI.
AGP_HMI_015	The Flight Progress Display shall provide facilities to display a list of all registered flights.
AGP_HMI_020	The Flight Progress Display shall provide facilities to display a scrollable list of all datalink messages exchanged between the AGP and a registered aircraft.
AGP_HMI_025	The Flight Progress Display shall provide a visual alert for messages which cannot be delivered.
AGP_HMI _055	The Flight Progress Display shall include facilities to display flight related data, which shall be, as a minimum: flight Id, tail number, current estimated position, flight phase and trajectory.

4.15.AOC Operator HMI: Maintenance Display Data Updating

Requirement ID	Description:
AGP_MTC_012	The AOC Ground Platform shall be able to provide to the user, information concerning the current status of the aircraft APU.
AGP_MTC_013	The AOC Ground Platform shall provide facilities to associate information about Aircraft APU performance with flight progress information.
AGP_MTC_018	The AOC Ground Platform shall be able to correlate APU data provided by a registered aircraft during a flight, to allow monitoring of aircraft APU performance.
AGP_MTC_022	The AOC Ground Platform shall be able to provide to the user, information concerning the current status of the Aircraft Engines.
AGP_MTC_023	The AOC Ground Platform shall provide facilities to associate information about Aircraft Engines performance with flight progress information.

AGP_MTC_032	The AOC Ground Platform shall be able to correlate A/C engine data provided by a registered aircraft during a flight, to allow monitoring of aircraft engines performance.
AGP_HMI_001	The Maintenance Management Console of the AOC Ground Platform shall provide facilities allowing the access to maintenance data.

4.16.AOC Operator HMI: Post-Flight Display Data Updating

Requirement ID	Description:
AGP_AMG_002	The AOC Ground Platform shall provide facilities to monitor aircraft performance based on OOOI information.
AGP_AMG_004	The AOC Ground Platform shall calculate the time spent by aircraft in the different phases of flight.
AGP_AMG_005	The AOC Ground Platform shall calculate the fuel used by aircraft in the different phases of flight.
AGP_HMI_005	The AOC Ground Platform shall include a Flight Analysis console to aid in aircraft performance analysis.

4.17.AOC Operator HMI: Messages Updating and Message Display

Requirement ID	Description:
AGP_AMG_280	The AOC Ground Platform shall provide the operator facilities to select and display a free text message.
AGP_MTC_002	The AOC Ground Platform shall be able to provide to the user, information about aircraft technical malfunctions based on SNAG report data.
AGP_MTC_003	The AOC Ground Platform shall provide facilities to associate information about aircraft technical malfunctions based on SNAG reports with flight progress information.
AGP_MTC_012	The AOC Ground Platform shall be able to provide to the user, information concerning the current status of the Aircraft APU.
AGP_MTC_022	The AOC Ground Platform shall be able to provide to the user, information concerning the current status of the Aircraft Engines.
AGP_CDM_070	The received 4D trajectory shall be suitably displayed to the operator.

4.18.AOC Operator HMI: Alert Display

4.19.AOC Operator HMI: Display Map

Requirement ID	Description:
AGP_HMI _030	The Flight Progress Display shall provide a map displaying current estimated position of registered aircraft within a rectangular Airline Area of Interest.
AGP_HMI _040	The Flight Progress Display shall display an aircraft symbol at the current estimated position of each registered aircraft.
AGP_HMI _045	The display of aircraft symbols shall be consistent with aircraft flight progress information (current position, heading, flight phase).
AGP_HMI _035	The Flight Progress Display shall provide facilities to filter (filters TBD) the aircraft to be displayed.
AGP_HMI _050	The Flight Progress Display shall enable the display of the current 4D trajectory data, waypoint data and other flight plan data of selected registered aircraft.

AGP_HMI _055	The Flight Progress Display shall include facilities to display flight related data, which shall be, as a minimum: flight Id, tail number, current estimated position, flight phase and trajectory.
AGP_HMI _003	The AOC Ground Platform HMI shall support the display of pre-defined weather information.

4.20.CFMU Communications: Slot Allocation Message

Requirement ID	Description:
AGP_ FPL _130	On receipt of a departure Slot Allocation message from CFMU for a flight which has registered with the AOC Ground Platform for datalink, the AOC Ground Platform shall transmit the Slot Allocation to the affected aircraft.
AGP_ FPL _140	On receipt of a departure Slot Allocation message from CFMU, the AOC Ground Platform shall update the company flight plan of the affected flight.

4.21.CFMU Communications: Filed flight plan

Requirement ID	Description:
AGP_ FPL _050	The AOC Ground Platform shall send a Filed Flight Plan message to CFMU at the later of the time of generation of the initial 4D trajectory and a configurable time before the estimated time of departure of the flight. Note for MA-AFAS all the pre-defined flight-plans will be filed with a simulated CFMU at system start-up.
AGP_ FPL _230	The AOC Ground Platform shall provide a suitable warning to the operator if a flight has not registered for datalink by a pre-determined time before its Estimated Off Block Time.

4.22.Flight Progress: Four-d trajectory request

Requirement ID	Description:
AGP_COM_045	The AOC Ground Platform shall provide a communication service supporting the exchange of 4D trajectory data with a given aircraft.
AGP_ CDM _040	On operator request, the AOC Ground Platform shall send a request for 4D trajectory data to a specified aircraft.
AGP_ CDM _017	The AOC Ground Platform shall be able to receive 4D trajectory data from an aircraft.
AGP_ CDM _050	On receiving 4D trajectory data from an aircraft, the AOC Ground Platform shall check the data for validity.
AGP_ CDM _060	On receiving 4D trajectory data from an aircraft, the AOC Ground Platform shall check the trajectory against the flight plan held for that aircraft.
AGP_ CDM _080	If the received 4D trajectory data has significant differences (a configurable difference in latitude and/or longitude) from the Flight Plan, a suitable alert shall be displayed to the operator.
AGP_ CDM _090	The AOC Ground Platform shall provide a warning to the operator if the 4D trajectory data is not received within a specified time after the request.

4.23.Sequencing: Check for timeouts

Requirement ID	Description:
AGP_COM_015	The AOC Ground Platform shall provide a suitable warning to the operator if confirmation of the Slot Allocation has not been received by a pre-determined time before the Estimated Off Block Time.

AGP_COM_019	The AOC Ground Platform shall provide a suitable warning to the operator if confirmation of the Company Flight Plan has not been received by a pre-determined time before the Estimated Off Block Time.
AGP_ FPL _230	The AOC Ground Platform shall provide a suitable warning to the operator if a flight has not registered for datalink by a pre-determined time before its Estimated Off Block Time.
AGP_ CDM _090	The AOC Ground Platform shall provide a warning to the operator if the 4D trajectory data is not received within a specified time after the request.

4.24.Asset Management: Aircraft Met Reports

Requirement ID	Description:
AGP_COM_070	The AOC Ground Platform shall provide a communication service to support the transmission of aircraft meteo observations in AMDAR format to an external provider. Note for MA-AFAS this communication will be simulated.
AGP_ AMG_210	The AOC Ground Platform shall be able to receive Aircraft Met reports from the aircraft.
AGP_ AMG_220	The AOC Ground Platform shall support Periodic Requests and On-demand Requests for Aircraft Met reports.
AGP_ AMG_230	The AOC Ground Platform shall convert Aircraft Met Reports into AMDAR format Observations, and send the AMDAR format Observations to the Met Office. Note for MA-AFAS this will be a simulated transmission.
AGP_AMG_240	The AOC Ground Platform shall provide a suitable warning to the operator if the Aircraft Met Report is not received within a pre-determined time after the expected time. The expected time will either be defined by the reporting interval of the periodic reports or by the time a demand report was requested, as appropriate.

4.25.Asset Management: Flight progress Reports

Requirement ID	Description:
AGP_COM_042	The AOC Ground Platform shall provide a communication service supporting the exchange of flight progress information with a given aircraft.
AGP_ FPL _090	The AOC Ground Platform shall use 4D Position data, Next Reporting Point, ETA and Fuel Information received from aircraft to track changes to the company flight plan during the flight.
AGP_AMG_010	The AOC Ground Platform shall support sending Flight Progress Requests to an aircraft on operator request.

4.26.Collaborative Decision Making: In-Flight Traffic Management

Requirement ID	Description:
AGP_COM_031	The AOC Ground Platform shall provide a communication service to support the exchange IFTM data with a given aircraft.
AGP_ CDM _100	The AOC Ground Platform shall support the generation of Delay/Divert/Re-route proposals.
AGP_ CDM _110	The AOC Ground Platform shall be able to send Delay/Divert/Re-route proposals to an aircraft.
AGP_ CDM _120	The AOC Ground Platform shall be able to receive Re-route Responses from the aircraft. The Re-route Response shall be displayed to the operator.
AGP_ CDM _130	The AOC Ground Platform shall provide a warning to the operator if the Re-route Response is not received within a specified time after the request.
AGP_ CDM _140	The AOC Ground Platform shall be able to receive Re-route Results from the aircraft. The Re-route Result shall be displayed to the operator and passed in to

	the Flight Planning process.
AGP_ CDM _150	The AOC Ground Platform shall provide a warning to the operator if the Re-route Result is not received within a specified time after the request.
AGP_ AMG_021	The AOC Ground Platform shall be able to receive Flight Progress Reports from an aircraft.
AGP_ AMG_040	The AOC Ground Platform shall allow the operator to specify what information is required from the aircraft Flight Progress Report in the Flight Progress Request.
AGP_ AMG_050	The AOC Ground Platform shall receive and process Flight Progress Responses from an aircraft. Flight Progress Responses will be either an ACK or a NACK.
REQ_OC_075	The AOC Ground Platform shall provide capability to monitor aircraft 4D flight plans by exploiting position and intent data received from the aircraft.
AGP_AMG_060	The AOC Ground Platform shall be able to calculate ETAs when a Flight Progress Report is received.
AGP_ AMG_090	When the AOC Ground Platform receives 4D Position data it shall check it for validity and compare the data with the expected 4D Position data.
AGP_AMG_095	Significant differences (a configurable difference in latitude and/or longitude) between the received 4D Position data and the expected 4D Position data shall be reported to the operator.
AGP_AMG_100	When the AOC Ground Platform receives Next Reporting Point information it shall compare it with the flight plan.
AGP_AMG_110	Significant differences (a configurable difference in latitude and/or longitude) between the Next Reporting Point information and the flight plan shall be reported to the operator.
AGP_AMG_140	On receipt of a position report, the AOC Ground Platform shall calculate the difference between the ETA at next waypoint and the expected time.
AGP_AMG_150	Significant differences (a configurable difference in time) between the ETA at next waypoint and the expected time shall be reported to the operator.
AGP_AMG_160	When the AOC Ground Platform receives Fuel Information it shall compare it with the expected fuel information.
AGP_AMG_170	Significant differences (a configurable difference in fuel quantity) between the received Fuel Information and the expected fuel information shall be reported to the operator.
AGP_AMG_180	The AOC Ground Platform shall provide a suitable warning to the operator if the Flight Progress Response is not received within a pre-determined time after the request.
AGP_AMG_190	The AOC Ground Platform shall support Periodic Requests and On-demand Requests for Flight Progress reports.
AGP_AMG_200	The AOC Ground Platform shall provide a suitable warning to the operator if a Flight Progress Report is not received within a pre-determined time after the expected time. The expected time will either be defined by the reporting interval of periodic reports or the time a demand report was requested, as appropriate.

4.27.Flight Plan: Loadsheet

Requirement ID	Description:
AGP_ FPL _170	The AOC Ground Platform shall be able to record Pax/Baggage information.
AGP_ FPL _180	The AOC Ground Platform shall be able to record Fuel information.
AGP_ FPL _190	The AOC Ground Platform shall be able to generate and uplink a Loadsheet message to a specified aircraft.
AGP_ FPL _200	The AOC Ground Platform shall provide a suitable warning to the operator if the Loadsheet has not been sent by a pre-determined time before the Estimated Off Block Time.

AGP_ FPL _210	The AOC Ground Platform shall provide a suitable warning to the operator if confirmation of the Loadsheet has not been received by a pre-determined time before the Estimated Off Block Time.

4.28. Other Requirements

This requirement is not accomplished in any other section above, because data input is already tested.

Requirement ID	Description:
AGP_HMI_002	The AOC Ground HMI shall be able to receive inputs from a keyboard and a CCD.

A.FLEET SIMULATOR

- **INTRODUCTION**

The AGP Fleet Simulator is an application that simulates a group of aircraft in communication with the AGP. The fleet simulator is capable of simulating up to 20 aircraft at a time, and each communicates with the AGP by using the Generic ATN Communications Service (GACS).

- **FLEET SIMULATOR CLASS DIAGRAM**

The following diagram describes the architecture of the fleet simulator. An Aircraft object is created for each configuration file found. Each Aircraft opens a GACS Endpoint and then starts its main loop thread which continues until the aircraft reaches its IN state. The mainLoop() checks for uplink messages on the GACS interface and checks for downlink messages on the send schedule. Any messages generated requiring display are sent to the Globals::logMessage() function, which keeps a list. The fleet simulator dialog window, created using MFC, periodically reads this list from the Globals class and displays the messages on the screen. This keeps the MFC code separate from the Fleet Simulator classes. The messages are also appended to a log file, which is created on start-up if a previous file is not found. Full message contents are logged to file, but just the first line to the screen.

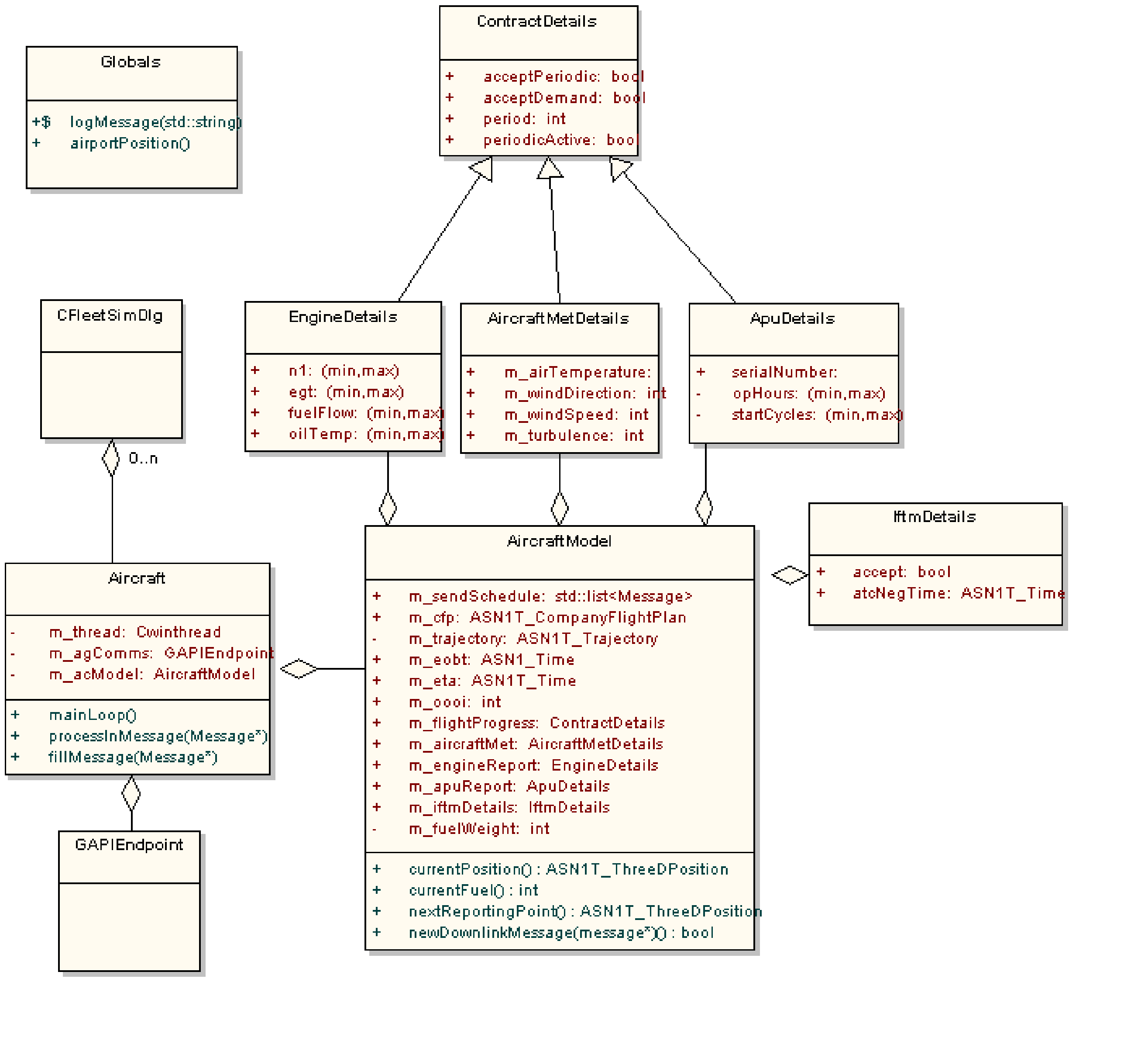


Figure A1. – Fleet Simulator Class Diagram

- **COMMUCTION WITH THE AGP**

  Each simulated aircraft within the fleet will have its own GACS endpoint. This will enable the AGP to connect simultaneously to both the avionics package and the simulated aircraft within the fleet simulator as shown in the diagram below. As far as the AGP is concerned there will be no difference between a simulated aircraft and a real aircraft using the AOC avionics package.

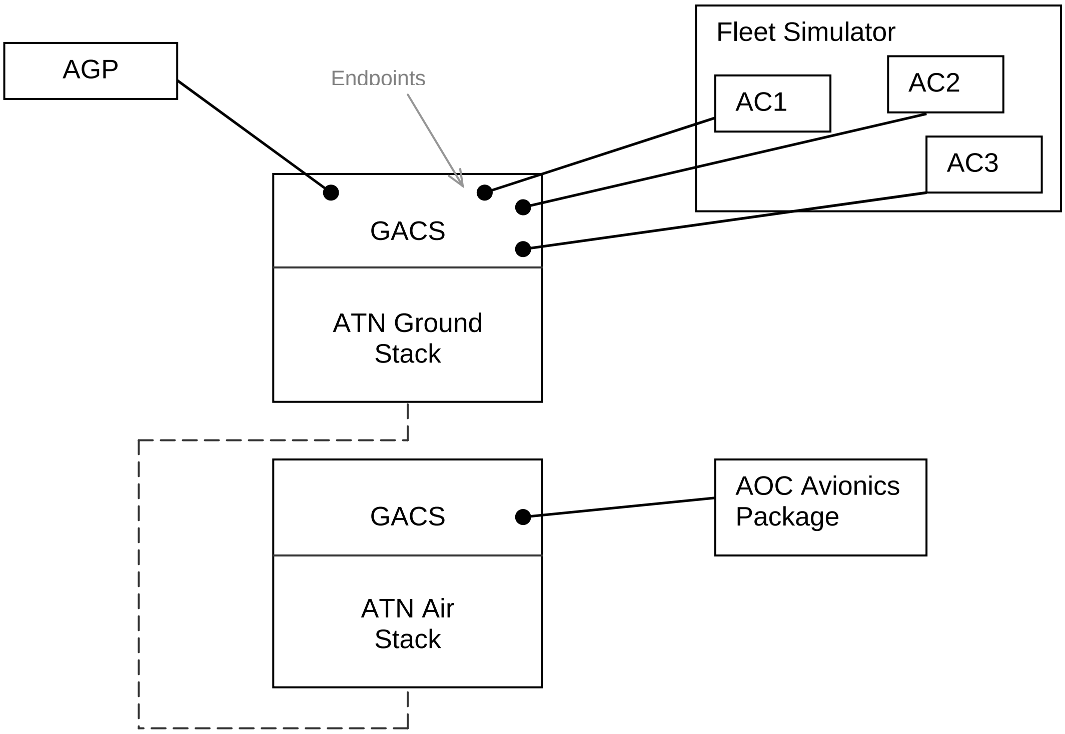


Figure A2. – Connection of the Fleet Simulator to the AGP

- **MESSAGE UPLINK**

Messages uplinked by the AGP to a simulated aircraft are received from the GAPIEndpoint class which queues messages it receives from GACS. These are then processed by the fleet simulator and any resulting downlink messages are added to the aircraft's send schedule of downlink messages. Any relevant details in the aircraft model class are updated and then the message is logged (Figure A3).

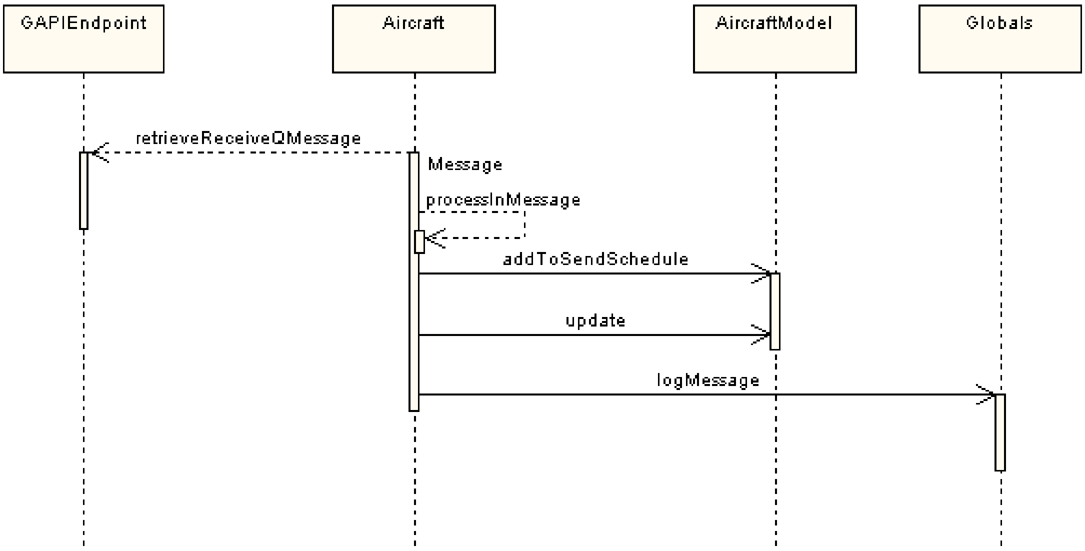


Figure A3. – Fleet Simulator – Message Uplink

- **MESSAGE DOWNLINK**

Downlink messages fall into two categories: those that occur at pre-defined times (specified in the input file), and those that occur as a response to an uplink message. The first message to be sent is an initialisation and this is relative to when the scenario is loaded, however for all subsequent messages the predefined time is relative to the flight's take-off time received as part of the company flight plan message. This has the advantage of making sure the whole system is synchronised.

When the input file is parsed a schedule of downlink messages is built up for each aircraft. The schedule is then periodically checked (several times a second) for a downlink message event. As the simulation progresses more messages will be added to the schedule in response to uplinked messages. The content of the messages is generated by the aircraft model class (Figure A4).

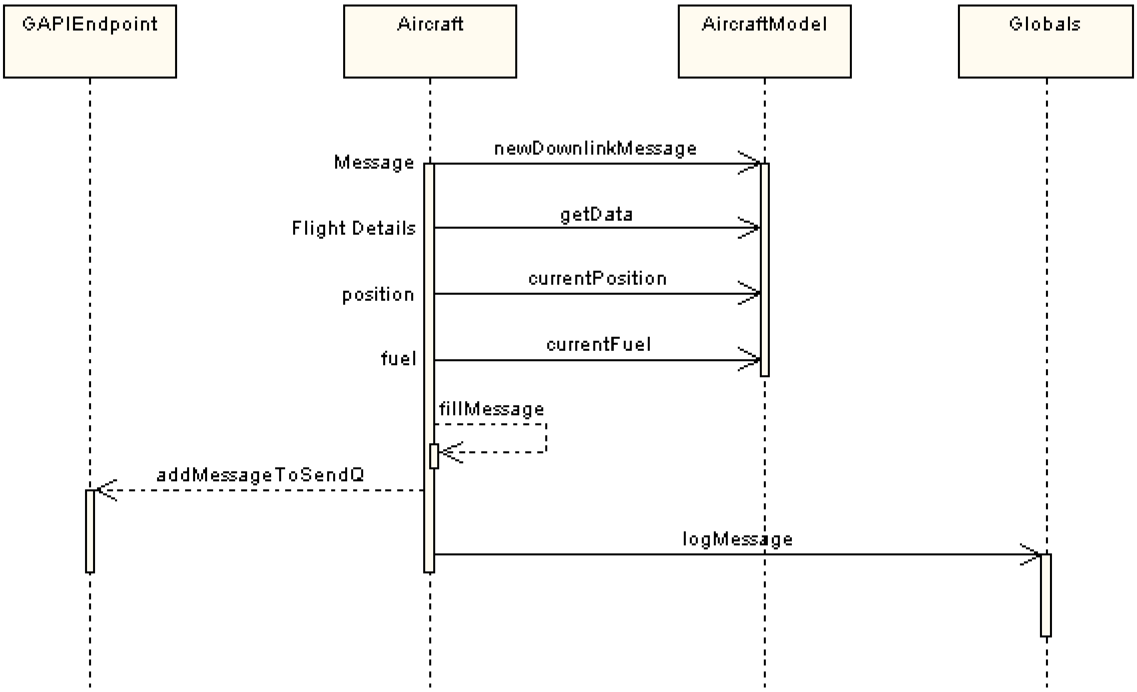


Figure A4 – Fleet Simulator – Message Downlink

- **CONFIGURATION FILES**

Aircraft Configuration Files

Each aircraft to be simulated has its own configuration file, which contains details of the flight to be simulated. The file is read in by the fleet simulator, the data in the file parsed then stored within the object in the fleet simulator representing that particular aircraft.

To read the file and parse the data, two tools are used; Flex and Bison. These tools are part of the GNU library of software. Flex reads a file containing descriptions of the tokens in the input file, and generates a tokeniser function; yylex(), which returns the next token from the input and its associated value. Bison is used to generate a parser function yyparse(), which calls the yylex() function to get tokens, and matches the tokens to semantic rules defining the input pattern.

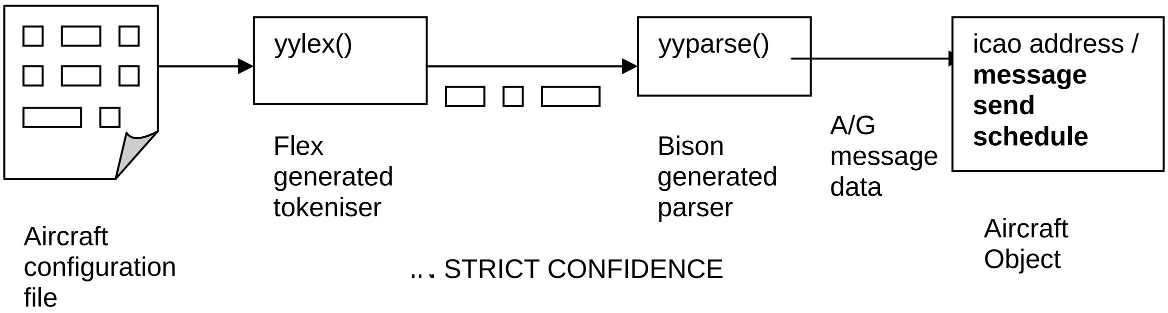


Figure A5 – Reading the configuration data from file.

The parser will only accept valid input and will provide a warning message if the input is incorrect, much in the same way a compiler does.

Configuration Data

The fleet simulator executable should be installed in the same directory as the AGP executable. The reason is that the Fleet Simulator will also read the AGP's "configurationData.dat" file located in the \data directory to read the parameters for the ground stack, to enable endpoints to be opened.

- **GUI DESIGN**

The GUI provides the capability to load different scenarios. This is achieved by each scenario having a different directory. In the directory are located the configuration files for each simulated aircraft. Having the option to load and start the scenario at the press of a button enables the user to pre-script a number of simulated flights, forming a scenario.

The GUI will also provide an indication of the running time of the simulation. This will be updated to reflect the progress of the simulated flights.

B.SCOPE OF WORK

In order to enable future support activities, the current Annex describes each partner scope of work responsibility within the AGP implementation regarding the *classes* and *AGP capabilities* described, previously, in Section 2 and Section 3, respectively.

CLASSES

Class	Partner
MutexQueue	Skysoft Portugal
SendMessageQrecord	Skysoft Portugal
ReceiveMessageQrecord	Skysoft Portugal
GAPIEndpoint	Skysoft Portugal
AircraftAddress	Skysoft Portugal
AGComms	Skysoft Portugal
FmsConverter	Skysoft Portugal
FmsMeteoTile	Skysoft Portugal
FmsDB	Skysoft Portugal
Weather	Skysoft Portugal
WoTx	Skysoft Portugal
WeatherOfficeComms	Skysoft Portugal
MeteoReportsDB	Skysoft Portugal
Maintenance	Skysoft Portugal
MaintenanceDB	Skysoft Portugal
AGPMap	Skysoft Portugal
AGP	Skysoft Portugal
Flight	Skysoft Portugal
FlightsList	Skysoft Portugal
ASIRecord	Skysoft Portugal
Fir	Skysoft Portugal
Waypoint	Skysoft Portugal
Airport	Skysoft Portugal
TAFMessage	Skysoft Portugal
METARMessage	Skysoft Portugal
SIGMETMessage	Skysoft Portugal
A_MeteoForecastAltitudeSet	Skysoft Portugal
MeteoForecastData	Skysoft Portugal
MeteoForecast	Skysoft Portugal
AckMessage	AMS Frimley
AlertMessage	AMS Frimley

Class	Partner
MemoryStore	AMS Frimley
AocGround	AMS Frimley
CFMUComms	AMS Frimley
CFMUSlotAllocationMessage	AMS Frimley
CommsServer	AMS Frimley
Companyflightplanmessage	AMS Frimley
configurationdata	AMS Frimley
FopsDB	AMS Frimley
FopsDBData	AMS Frimley
FopsDBKey	AMS Frimley
FPDB	AMS Frimley
FPDBMemBlock	AMS Frimley
FPDBRecordType	AMS Frimley
FreetextMessage	AMS Frimley
CFMUSlotAllocation	AMS Frimley
InitialisingMessage	AMS Frimley
Logfile	AMS Frimley
Message	AMS Frimley
MessagesWaiting	AMS Frimley
OooiMessage	AMS Frimley
SlotAllocationMessage	AMS Frimley
TimerList	AMS Frimley
TimerListEntry	AMS Frimley
TrajectoryMessage	AMS Frimley
TrajectoryRequestMessage	AMS Frimley
AocContracts	AMS Frimley
Contract	AMS Frimley
ResponseReceiveEvent	AMS Frimley
ReportReceiveEvent	AMS Frimley
FlightData	AMS Frimley
LoadsheetMessage	AMS Frimley
LoadsheetAckMessage	AMS Frimley
RequestMessage	AMS Frimley
FlightProgressRequestMessage	AMS Frimley
EngineStatusRequestMessage	AMS Frimley
AircraftMetReportsMessage	AMS Frimley
APURequestMessage	AMS Frimley
ResponseReportMessage	AMS Frimley

Class	**Partner**
AircraftMetReportsResponseMessage	AMS Frimley
AircraftMetReportMessage	AMS Frimley
FlightProgressReportMessage	AMS Frimley
FlightProgressResponseMessage	AMS Frimley
APUReportMessage	AMS Frimley
APUResponseMessage	AMS Frimley
EngineStatusReportMessage	AMS Frimley
EngineStatusResponseMessage	AMS Frimley
ConstraintsListMessage	AMS Frimley
ConstraintsAcceptanceMessage	AMS Frimley

AGP CAPABILITIES

Capability	Partner
Air-Ground Communications: Reception	Skysoft Portugal
Air-Ground Communications: Transmission	Skysoft Portugal
Air-Ground Communications: Alerts	Skysoft Portugal
Communications To Meteo Office: Meteo Forecast and meteo reports.	Skysoft Portugal
Flight Plan: Initialising Message	AMS Frimley
Flight Plan: FMS Meteo	Skysoft Portugal
Flight Plan: Meteo Reports	Skysoft Portugal
Maintenance: SNAG Reports	Skysoft Portugal
Maintenance: APU Reports	Skysoft Portugal
Maintenance: Engine Status Reports	Skysoft Portugal
Asset Management: Free Text Uplink	Skysoft Portugal
Asset Management: Free Text Downlink	Skysoft Portugal
Asset Management: OOOI Reports	Skysoft Portugal
AOC Operator HMI: AGP Main Loop	Skysoft Portugal
AOC Operator HMI: Flight Progress Display Data Retrieval	Skysoft Portugal
AOC Operator HMI: Flight Progress Display Data Updating	Skysoft Portugal
AOC Operator HMI: Maintenance Display Data Updating	Skysoft Portugal
AOC Operator HMI: Post-Flight Display Data Updating	Skysoft Portugal
AOC Operator HMI: Messages Updating	Skysoft Portugal
AOC Operator HMI: Message Display	Skysoft Portugal
AOC Operator HMI: Alert Display	Skysoft Portugal
AOC Operator HMI: Display Map	Skysoft Portugal
CFMU Communications: Slot Allocation Message	AMS Frimley
CFMU Communications: Filed Flight Plan	AMS Frimley
Flight Progress: Four-D Trajectory Request	AMS Frimley
Sequencing: Periodic function	AMS Frimley
Sequencing: Check For Timeouts	AMS Frimley
Asset Management: Aircraft Met Reports	AMS Frimley
Asset Management: Flight Progress Reports	AMS Frimley
Collaborative Decision Making: In-Flight Traffic Management	AMS Frimley
Flight Plan: Loadsheet	AMS Frimley

C. REFERENCE DOCUMENTS

The standards listed here will be considered when reading this document.

[RTCA 178B]	RTCA Software Considerations in Airborne Systems and Equipment Certification, December 1, 1992.
[MAAFAS D38]	MA-AFAS Ground System Requirements, Project Delivery D38, Issue 1
[MAAFAS ICD]	MA-AFAS AOC Air-Ground Interface Control Document – Issue 2
[AGP SDD]	MA-AFAS AGP Software Design Document, Version 1.1; June 16, 2002

D. ABBREVIATIONS

A/C	Aircraft
AGP	AOC Ground Platform
AOC	Airline Operational Center
ASI	Aircraft Systems Information
ASN	Abstract Syntax Notation
CFMU	Central Flow Management Unit
EGT	Exhaust Gas Temperature
ETA	Estimated Time of Arrival
FIR	Flight Information Region
FMS	Flight Management System
FOpsDB	Flight Operations Database
FPD	Flight Progress Display
FPDB	Flight Planning Database
GACS	Generic ATN Communications Services
GAPI	GACS Application Programming Interface
HMI	Human-Machine-Interface
IFTM	In Flight Traffic Management
MA-AFAS	More Autonomous Aircraft in the Future ATM System
MFC	Microsoft Foundation Classes
NRP	Next Reporting Point
PER	Packed Encoded Rules
STL	Standard Template Library