DRAFT

Connection of air traffic laboratories at NASA Ames and Langley Research Centers

Distributed Air/Ground Traffic Management

Technical Connectivity Working Document Appendices

Revision: initial Draft
Date: Created on 11/27/02

Prepared for: NASA ARC and LaRC

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, CA

Langley Research Center
Hampton, VA

TABLE OF CONTENTS

ERROR! BOOKMARK NOT DEFINED.

[1]DATA LINK ARINC 702+ DOCUMENT 3

1.1 Introduction 3
1.2 Summary of Proposed Message Set 3
1.2.1 Message identifiers: 3
1.2.2 Data fields within each IEI 5
1.2.3 Honeywell CPI data extraction 15
1.2.4 Data Extraction Logic 16
1.3 Sample CDU Message Text 17

[2]ADRS CURRENT MULTI PURPOSE INTERFACE HEADER FILE (ADRS_MPI.H) 17

[3]HLA OBJECTS, ATTRIBUTES AND INTERACTIONS IN AIRSPACE AND TRAFFIC OPERATIONS SIMULATION 39

3.1 Objects Supported 39
3.1.1 Entities for transmitting and receiving 39
3.1.2 Derived Object: Aircraft 40
3.2 Interactions in ATOS: 40
3.2.1 Data Link interactions: 40
3.2.2 Base Interaction : ModeControl 44
3.2.3 Derived Interaction: Configure 44
3.2.4 Cycle timing interactions 45

[1] DATA LINK ARINC 702+ DOCUMENT

1.1 Introduction

This document shows data link messages for the joint NASA Langley/Ames TAP CTAS/FMS Data Link experiments to be conducted in 1999. The messages were derived from the proposed CTAS/FMS Data Link scenario that was developed by Langley and Ames to test the CTAS/FMS interaction in a near-term ATC environment. This set has been reduced to the minimum necessary to support FY99 experiments. The original message set has been slightly extended to support current DAG-TM requirements. Performance preferences have been added to the ADB message set. These should be gathered from the ACTIVE Route. The CLR has been extended to include a CA element for loading of a new cruise altitude (to support altitude changes created on a CDTI for example)

1.2 Summary of Proposed Message Set

1.2.1 Message identifiers:

Below are the Imbedded Message Identifiers (IMIs, in bold type) defined for this experiment. Listed after each IMI are the associated two-letter Imbedded Element Identifiers (IEIs) that may be contained in that IMI.

ADB (ADS-B)

FN*
MS
SV*
TI
PF
TS*

AFN (ATS facility notification uplink and downlink):

FN*
LR
LC
TR
TC
SN*
TS*

CLR (clearance uplink message):

FN*
FT
DA
RM
CA
SN*
TS*

CTS (CTAS constraint uplink message):

FN*

FT
WX
SN*
TS*

FPC (Flight Plan downlink message)

FN*
RP
FL
PF
TS*

RES (Response downlink message)

FN*
AC
RJ
FT
TS*

TAC (Tactical ATC uplink)

FN*
AL
HE
VE
SN*
TS*

UPI (user preference information downlink message):

FN*
PL
PF
PT
TS*

* required IEIs.

1.2.2 Data fields within each IEI

In the tables below are listed the data fields associated with each of the IEIs. The IEI is shown in italics in the top left box. Fields that are marked with not applicable (n/a) will not be used in this experiment, but are part of the standard ACARS message.

AC (Accept)	**field length**	**Alpha / Numeric**	**comments**	**example**
message sequence number	10	AN		DFW131725A
stimulus code	3	N	n/a	

AL (altitude command)	**field length**	**Alpha / Numeric**	**comments**	**example**
altitude command action	1	A	M=maintain, D=descend, C=climb	M
altitude	4	N	feet*10	2900

CA (cruise altitude)	**field length**	**Alpha / Numeric**	**comments**	**example**
cruise altitude	4	N	Ft x 10	3300

DA (descent advisory)	**field length**	**Alpha / Numeric**	**comments**	**example**
cruise speed	4	3N,1A	kts or M*.001	790M
clearance descent speed	4	3N, 1A	kts	280K

1.2.2.1.1.1 FL (flight plan info)	**field length**	**Alpha / Numeric**	**comments**	**example**
gross takeoff weight	4	N	lbs*100	2000
performance/drag factor	4	1A, 3N	% drag factor; P/M = +/-	M005
planned cruise altitude	4	N	ft x 10	3300
planned cruise speed	4	3N, 1A	M or kts	750M
actual departure time	6	N	hhmmss	120000
route - ETA (at destination)	6	N	hhmmss	140100

FN (flight number)	**field length**	**Alpha / Numeric**	**comments**	**example**
flight number	1-10	AN		NA557
Computer ID	4	N	assigned by ATC	1111

FT (free text)	**field length**	**Alpha / Numeric**	**comments**	**example**

free text	variable	A	*each line on CDU limited to 24 characters*	**RNV18R APPR FEVER TRANS**

HE (heading command)	**field length**	**Alpha / Numeric**	**comments**	**example**
direction	1	A	L=left, R=right, F=left to, G=right to	**G**
magnetic degrees	3	N	degrees	**150**

LC (logon confirmation)	**field length**	**Alpha / Numeric**	**comments**	**example**
ATC facility name	4	A		**KDFW**
[TBD]				

LR (logon request)	**field length**	**Alpha / Numeric**	**comments**	**example**
ATC facility name	4	A		**KDFW**
[TBD]				

MS (mode status for ADS-B)	**field length**	**Alpha / Numeric**	**comments**	**example**
category	1	N	1 = light a/c 3 = med a/c 5 = heavy a/c	**3**
surveillance support code	1	A	N = normal D = default	**N**
class code	1-2	N	1 = broadcast 2 = CDTI 3 = CAS 4 = station keep 5 = free flight 6 = oceanic 7 = simul app 8 = blind taxi 9 = runway incr	**4**
[TBD]				

PF (performance preferences)	**field length**	**Alpha / Numeric**	**comments**	**example**
current weight	4	N	lbs*100	**1917**
planned final approach speed	4	3N, 1A	kts	**150K**
preferred climb speed	4	3N, 1A	M*.001 or kts	**280K**
preferred cruise speed	4	3N, 1A	M*.001 or kts	**750M**
preferred descent speed	4	3N, 1A	M*.001 or kts	**280K**
cost index	3	N		**050**

PL (performance limits)	**field length**	**Alpha / Numeric**	**comments**	**example**

time error tolerance	2	N	sec	30
climb CAS limits	6	N	kts (min/max)	240320
cruise CAS limits	6	N	kts (min/max)	280340
descent CAS limits	6	N	kts (min/max)	240320
climb Mach limits	6	N	.001*M (min/max)	500640
cruise Mach limits	6	N	.001*M (min/max)	780820
descent Mach limits	6	N	.001*M (min/max)	650820

PT (preferred trajectory)	**field length**	**Alpha / Numeric**	**comments**	**example**
1.2.2.1.1.2 Point Type	2	A	WP: waypoint HP: hold pattern PH: proc hold PT: proc turn RF: RF leg TC: TOC TD: TOD SL: start of level CA: crossover altitude TA: transition altitude	**WP**
Waypoint ID	3-5	A		**JEN**
1.2.2.1.1.3 Point Latitude and Longitude	13	1A, 5N, 1A, 6N	degrees,min*.1	**N32098W097527**
1.2.2.1.1.4 Point Turn Radius	6	1A,5N	Right or Left, feet*10	**1.2.2.2 R01500**
1.2.2.2.1.1 ETA	6	N	hhmmss	**133342**
Calibrated airspeed	4	3N, 1A	kts or .001M	**332K**
1.2.2.2.1.2 Altitude	4	N	feet*10	**1500**
Fuel remaining.	4	N	lbs*100	**0200**
OAT	3	1A, 2N	P/M degrees C	**M15**
Wind *(wind speed)* *(wind direction)*	 (3) (3)	 N N	 kts deg true N	**045090**
True airspeed	3	N	knots	**426**
True course into point.	3	N	degrees	**042**
Distance to point.	4	N	nmi*.1	**0302**
Predicted gross weight	4	N	lbs*100	**1855**

RJ (reject)	**field length**	**Alpha / Numeric**	**comments**	**example**

message sequence number	10	AN		**DFW131725A**
stimulus code	3	N	n/a	

RP (route plan) or RM (route modification)	**field length**	**Alpha / Numeric**	**comments**	**example**
:DA: departure airport (RP only)	4	AN		**KORD**
:CR: company route (RP only)	10	AN		**KORDKDFW02**
:AA: arrival airport	4	AN		**KDFW**
:R: departure runway ident	3	2N,1A		**27L**
:D: departure procedure	6	AN		**OHARE2**
:F: flight plan segment list: (published ident, optional lat/lon),or - - - - - - - - - - - (lat / lon), or - - - - - - - - - - - (place bearing/place bearing), dash or - - - - - - - - - - - (place bearing distance) dash	3-5 13 - - - - 13 - - - - 3-5 13 3 1 3-5 13 3 - - - - 3-5 13 3 1 4	AN 1A, 5N, 1A, 6N - - - - - 1A, 5N, 1A, 6N - - - - - AN 1A, 5N, 1A, 6N N AN 1A, 5N, 1A, 6N N - - - - - AN 1A, 5N, 1A, 6N N N	wpt ident degrees,min*. 1 - - - - - degrees,min*. 1 - - - - - wpt ident *or* degrees,min*. 1 degrees wpt ident *or* degrees,min*. 1 degrees - - - - - wpt ident *or* degrees,min*. 1 degrees nmi*.1	**RZC** **N32289W099518** - - - **N32289W099518** - - - **RZC** *or* **N32289W099518** **030** **-** **AUS** *or* **N32289W099518** **220** - - - **AUS** *or* **N32289W099518** **220** **-** **0250**
.. direct fix - same as :F:	(see above)	(see above)	(see above)	(see above)
:A: arrival procedure	10-12	AN		**JENX.AUS**
:AP: approach procedure	10-12	AN		**RNV18R.FEVER**
(): arrival runway ident	3	2N, 1A		**18R**
:V: waypoint speed/altitude (fix ID or lat/lon, comma (,) speed, comma (,) altitude comma (,) optional altitude)	3-5 or 13 1 3 1 6 1 6	AN *or* 1A, 5N, 1A, 6N N 2A, 4N 2A, 4N	 degrees,min kts feet*10 feet*10	**ROSEL** *or* **N32289W099518** **,** **190** **,** **AA1100** **,** **AB1300**
:H: hold at waypoint	---	---	n/a	
:WS: waypoint step climb	---	---	n/a	
:AT: along track waypoint (fix ID *dash (-)* *distance,* *comma (,)* *speed,* *comma (,)*	3-5 1 5 1 3 1	AN 1A, 4N N	wpt ident P/M, nmi*.1 kts	**ROSEL** **-** **P025** **,** **190** **,**

altitude *comma (,)* *optional altitude)*	6 1 6	2A, 4N 2A, 4N	feet*10. feet*10.	**AA1100** **,** **AB1300**
:RP: reporting points	---	---	n/a	
. transition or airway via	5	AN		**J120**
:F:. airway intrcpt (same as transition , above)	(see above)	(see above)	(see above)	(see above)
:LO: lateral offset (RM only)	3	1A, 2N	L/R, nmi	**L10**
:IC: intercept course from (published ident, PB/PB, or PBD as in :F:, followed by a comma (,) and course: *course*	3	N	degrees mag.	**050**

SN (message sequence)	**field length**	**Alpha / Numeric**	**comments**	**example**
message sequence number	10	AN		**DFW131725A**

SV (state vector for ADS-B)	**field length**	**Alpha / Numeric**	**comments**	**example**
present position latitude	7	1A, 6N	degrees,min*.01	**N322890**
present position longitude	8	1A,7N	degrees,min*.01	**W0995180**
geometric altitude	1-6	(1A),N	(M), feet	**15000** **M1000 (-1000)**
Navigation Uncertainty Category - Position (NUCp)	1	N	0 thru 9	**7**
geometric position valid	1	N	0 = invalid 1 = valid	**1**
north ground velocity	5	(1A),4N,1A	(M),kts	**19K** **M19K (-19)**
east ground velocity	5	(1A),4N,1A	(M),kts	**371K**
geometric altitude rate,	5	1A,4N	U/D, ft / min.	**U1502**
Navigation Uncertainty Category - Velocity (NUCp)	1	N	0 =	**4**
barometric pressure altitude	1-6	(1A),N	(M), feet	**15000** **M1000 (-1000)**
barometric altitude rate	5	1A,4N	U/D, ft / min.	**U1502**
airspeed,	4	3N, 1A	kts, IAS or TAS defined by /MS	**301K**
ground speed	5	4N,1A	kts	**372K**
true track angle,	3	N	degrees	**087**
turn indication	1	A	R = right L = left 0 = not turning	**R**
time of applicability	6	N	UTC	**132200**
[TBD]				

note: bold items are only ones sent by LaRC 757.

TC (terminate confirmation)	**field length**	**Alpha / Numeric**	**comments**	**example**
ATC facility name	4	A		**KDFW**
[TBD]				

TI (trajectory intent)	field length	Alpha / Numeric	comments	example
1.2.2.2.1.3 Point Type	2	A	WP: waypoint HP: hold pattern PH: proc hold PT: proc turn RF: RF leg TC: TOC TD: TOD SL: start of level CA: crossover altitude TA: transition altitude	**WP**
1.2.2.2.1.4 Waypoint ID	3-5	A		**JEN**
1.2.2.2.1.5 Point Latitude and Longitude	13	1A, 5N, 1A, 6N	degrees,min*.1	**N32098W097527**
1.2.2.2.1.6 Point Turn Radius	6	1A,5N	Right or Left, feet*10	**1.2.2.3 R01500**
1.2.2.3.1.1 ETA	6	N	hhmmss	**133342**
1.2.2.3.1.2 Calibrated airspeed	4	3N, 1A	kts or .001M	**332K**
1.2.2.3.1.3 Altitude	4	N	feet*10	**1500**
Fuel remaining.	4	N	lbs*100	**0200**
OAT	3	1A, 2N	P/M degrees C	**M15**
Wind *(wind speed)* *(wind direction)*	 (3) (3)	 N N	 kts deg true N	**045090**
True airspeed	3	N	knots	**426**
True course into point.	3	N	degrees	**042**
Distance to point.	4	N	nmi*.1	**0302**
Predicted gross weight	4	N	lbs*100	**1855**

note: bold items are only ones sent by LaRC 757.

TR (terminate request)	field length	Alpha / Numeric	comments	example
ATC facility name	4	A		**KDFW**
[TBD]				

TS (Time Stamp)	field length	Alpha / Numeric	comments	example
time of day	6	N	UTC	**132200**

date	8	N	ddmmyyyy	**21051999**

VE (speed command)	**field length**	**Alpha / Numeric**	**comments**	**example**
speed command action	1	A	M=maintain, R=reduce, I=increase	**R**
speed	4	3N, 1A	kts or M	**210K**

WX (weather matrix)	**field length**	**Alpha / Numeric**	**comments**	**example**
wx matrix time	6	N	hhmmss	**120000**
wx matrix position (lat/lon)	13	1A, 5N, 1A, 6N	degrees,min*.1	**N32000W098000**
wx matrix data list: *(altitude,* *wind direction,* *wind speed,* *outside air temperature)*	 3 3 3 3	 N N N 1A, 2N	 feet*100 degrees kts P/M deg C	**290095058M51**

1.2.3 Honeywell CPI data extraction

Data from the Honeywell CPI will be used in the UPI/PF and ADB/TI down link messages. The following tables provide a cross-reference of CPI and data link message parameters.

PF (performance preferences)	**Honeywell Parameter**	**comments**
current weight	not used	use simulation weight
planned final approach speed	FLAPS3VREF	speed range 100 to 150 knots
preferred climb speed	CLBCMDCAS	speed range 190 to 340 knots
preferred cruise speed	CRZCMDMACH	Mach range .6 to .9
preferred descent speed	DESCMDCAS	speed range 190 to 340 knots
cost index	COSTINDEX	CI range 0 to 999

TI (trajectory intent)	**Honeywell Parameter**	**comments**
1.2.3.1.1.1 Point Type	n/a	"WP" for all waypoints "TD" for top of descent
Waypoint ID	FIXIDENT	remove spaces, 5 char max, 3 char min "TOD" for inserted TD point
1.2.3.1.1.2 Point Latitude and Longitude	TOLATLON	latitude range N25 to N50, longitude range W65 to W125
1.2.3.1.1.3 Point Turn Radius	n/a	1.2.3.2 ","
1.2.3.2.1.1 ETA	PRDETATOFIX	hour range 00 to 24 minute range 00 to 59 second range 00 to 59
Calibrated airspeed	PRDAIRSPDCAS or PRDAIRSPDMACH	use depending on PRDSPDTYPE flag, Mach range .2 to .9, CAS range 100 to 400 knots
1.2.3.2.1.2 Altitude	PRDALT	altitude range 0 to 40000 feet
Fuel remaining.	n/a	1.2.3.3 ","
OAT	n/a	1.2.3.4 ","
Wind *(wind speed)* *(wind direction)*	n/a	1.2.3.5 ","
True airspeed	n/a	1.2.3.6 ","
True course into point.	n/a	1.2.3.7 ","
Distance to point.	FIXDISTODEST	subtract from a/c distance to destination to get distance to waypoint
Predicted gross weight	n/a	1.2.3.8 ","

1.2.4 Data Extraction Logic

Honeywell CPI data is extracted in complete blocks at a regular interval. The following logic and criteria should be used to determine if the extracted data is valid and if a downlink UPI/PF and ADB/TI should be sent to ATC:

1. The data set needed (referred to as trajectory record) for the /TI and /PF data link messages should be extracted from each full block of data received from the CPI. This trajectory record is referred to as the new_trajectory.
2. Waypoints after “FF18R”, “RW18R”, “FF13R”, or “RW13R” should be discarded from the new_trajectory. Waypoints associated with the missed approach do not have complete data, and cause problems in determining if the trajectory is valid or has changed.
3. Top of descent data should be inserted into the new_trajectory record using the following technique:
   - Extract the top of descent distance from the CPI data block (parameter TOD2DEST).
   - Insert TOD between waypoints based on waypoints FIXDISTODEST.
     - Compute latitude, longitude and eta at TOD by linear interpolation on distance to destination using crossing conditions at the waypoints before and after TOD.
     - Use altitude and airspeed at waypoint before TOD for crossing conditions at TOD.
4. Data is valid if all parameters required for both trajectory record fall within the data ranges specified in the above tables. If any of the required data parameters are invalid, the entire new_trajectory record should be discarded -- skip the remaining steps.
5. If data is valid, and this is the first good trajectory record, save it as the current_trajectory and down link the UPI/PF and ADB/TI messages. Skip the remaining steps.
6. If data is valid and a current_trajectory exists, compare the data to the current_trajectory.

   A trajectory has changed if any of the following are true:
   - The number of waypoints is different.
   - Climb, descent or approach speed change greater than 5 knots.
   - Cruise Mach change greater than .01.
   - At each waypoint (including TOD), any of the data parameters fall outside the following tolerances:
     - Different name (strcmp).
     - Altitude change > 50 feet.
     - Speed change greater than 5 knots.
     - Time change greater than 5 seconds.
     - Distance change greater than .1 nautical mile.
     - Latitude or longitude change greater than .002 deg.
   - If new_trajectory is different than the current_trajectory, and a provisional_trajectory does not exist, save new_trajectory as provisional_trajectory and skip remaining steps.
   - If new_trajectory is different than the current_trajectory and the same as the saved provisional_trajectory, clear the provisional_trajectory, replace the current_trajectory with the new_trajectory and data link the UPI/PF and ADB/TI to ATC. Note: it may be necessary to extract the same trajectory more than twice to ensure it is a valid change. Testing with the sample CPI files has shown 2 matching trajectories is generally sufficient.

1.3 Sample CDU Message Text

This section contains samples of how the uplinked messages should appear on the ATC uplink page on the CDU.

IMI	IEI	Message text
CLR	FT	[free text] (displayed on one or more lines on the UPLINK page)
	RM DA	note: Pilot is given LOAD> prompt at end of message. Selecting load will first autoload the route (via ACARS) as a mod to the active route (if RM data sent), then enter the *clearance descent speed* via keystroke emulation onto the DESCENT page and/or *clearance cruise speed* onto the CRZ page (if DA data sent).
RES	AC	ACCEPT (under the heading RESPONSE)
CTS	FT	[free text] (displayed on one or more lines on the UPLINK page)
	WX	note: Pilot is given LOAD> prompt at end of message. Selecting load will autoload the descent winds (via ACARS).
TAC	HE	TURN *[direction]* HEADING *[degrees]* (heading command action of F or G)
	"	TURN *[direction] [degrees] DEGREES* (heading command action of R or L)
	VE	MAINTAIN *[speed],* (speed command action of M)
	"	REDUCE SPEED TO *[speed]*, (speed command action of R)
	"	INCREASE SPEED TO *[speed]*, (speed command action of I)
	AL	MAINTAIN *[altitude]*, (altitude command action of M)
	"	DESCENT TO AND MAINTAIN *[altitude]*, (altitude command action of D)
	"	CLIMB TO AND MAINTAIN *[altitude]*, (altitude command action of C)

[2] ADRS CURRENT MULTI PURPOSE INTERFACE HEADER FILE (ADRS_MPI.H)

The following Header File is provided for Reference Only. It describes all data records and formats that are available form/to the ADRS, which extends far beyond what is needed to connect the ARC and LaRC laboratories:

```
#ifndef ADRS_MPI_H
#define ADRS_MPI_H


/*
 *==============================================================================
 * File:              adrs_mpi.h
 *
 * VERSION:  adrs_mpi.h                    12/04/2002
 *
 * Description:
 *      These functions describe the socket interface for the adrs/mpi link.
 *
 * Modified:    by:            what:
 * 3/23/2000    tprevot        original version
 *==============================================================================
```

```
 */


/*
 *-----------------------------------------------------------------------------
 * include files
 *-----------------------------------------------------------------------------
 */
#include "adrs_interface.h"
#include "adrs_macs.h"


/*
 *-----------------------------------------------------------------------------
 * constants
 *-----------------------------------------------------------------------------
 */

#define ADRS_MPI_ID_LENGTH                    12
#define ADRS_MPI_REQ_ID_LENGTH                12
#define ADRS_MPI_HOST_LENGTH                  24
#define ADRS_MPI_MAX_TYPE_CHARS                     16
#define ADRS_MPI_SECTOR_LENGTH                      8
#define ADRS_MPI_MAX_FILED_ROUTE_LENGTH  300
#define ADRS_MPI_MAX_NAME_LENGTH              20
#define ADRS_MPI_STR_SIZE_S                   80
#define ADRS_MPI_VERSION_ID_LENGTH            15
#define ADRS_MPI_PAS_COMMAND_LENGTH                 128
#define ADRS_MPI_TRAJ_TEXT_LENGTH             128
#define ADRS_MPI_PAS_COMM_SIZE                128
#define ADRS_MPI_SCENARIO_LENGTH              128
#define ADRS_MPI_MAX_AK_ROUTE_LENGTH                2048
#define ADRS_MPI_CMD_STRING_SIZE              48

#define ADRS_MPI_TRS_INITIAL_DATA_SIZE  8096

#define ADRS_MPI_CID_NOT_SET                  -1

#define ADRS_MPI_FLOAT_NOT_SET                -999999.0
#define ADRS_MPI_NOT_SET                -999999

#define ADRS_MPI_ITEM_SET (X)                 ((int) X > -999995)

#define ADRS_MPI_PAS_PLAYBACK                 99


typedef enum
{
   ADRS_MPI_INITIAL_CLIENT_MSG = 5001,
   ADRS_MPI_SERVER_RESPONSE    = 5002,
   ADRS_MPI_REQUEST_MSG        = 5101,
   ADRS_MPI_RESPONSE_MSG       = 5102,
   ADRS_MPI_CANCEL_REQUEST_MSG = 5103,
   ADRS_MPI_MACS_CONFIG_MSG    = 5151,
   ADRS_MPI_MACS_CONTROL_MSG= 5181,
   ADRS_MPI_FLIGHT_PLAN_MSG    = 5201,
   ADRS_MPI_DELETE_AC_MSG      = 5202,
   ADRS_MPI_AK_ROUTE_MSG       = 5211,
   ADRS_MPI_TRACK_MSG          = 5301,
```

```
   ADRS_MPI_STATE_MSG                = 5302,
   ADRS_MPI_RADAR_CENTER_MSG = 5303,
   ADRS_MPI_RADAR_TRACON_MSG= 5304,
   ADRS_MPI_EXTENDED_STATE_MSG         = 5305,
   ADRS_MPI_COMPLETE_STATE_MSG = 5306,
   ADRS_MPI_FREQUENCY_MSG       = 5311,
   ADRS_MPI_PILOT_CONTROL_MSG= 5321,
   ADRS_MPI_ATC_CONTROL_MSG   = 5322,
#ifndef NEW_DATALINK_FORMAT
   ADRS_MPI_DATALINK_MSG            = 5401,            /* Will be phased out */
#endif
   ADRS_MPI_DLNK_MSG                = 5406,
   ADRS_MPI_TRAJ_D_INTENT_MSG  = 5411,
   ADRS_MPI_TRAJ_D_PREF_MSG     = 5412,
   ADRS_MPI_TRAJ_D_ATC_MSG       = 5413,
   ADRS_MPI_TRAJ_INTENT_MSG      = 5421,
   ADRS_MPI_TRAJ_PREF_MSG        = 5422,
   ADRS_MPI_TRAJ_ATC_MSG         = 5423,
   ADRS_MPI_PAS_COMMAND_MSG  = 5501,
   ADRS_MPI_PAS_COMM_MSG         = 5502,
   ADRS_MPI_PILOT_INPUT    = 5510,
   ADRS_MPI_GUIDANCE_OUTPUT    = 5511,
   ADRS_MPI_INITIAL_STATE  = 5520,
   ADRS_MPI_EXT_CMD                = 5530,
   ADRS_MPI_GENERIC_CHANGE      = 5531,
   ADRS_MPI_PTA_CHANGE            = 5541,
   ADRS_MPI_SPEED_CHANGE         = 5542,
   ADRS_MPI_SPACING_CHANGE      = 5543,
#ifndef NEW_DATALINK_FORMAT
   ADRS_MPI_NR_DATALINK_MSG     = 5601,   /* non request message ->essential */
   ADRS_MPI_ATC_DATALINK_MSG   = 5602,   /* ATC COMM ROUTE UPLINK ETC. */
#endif
   ADRS_MPI_NR_DLNK_MSG= 5611,   /* non request message ->essential */
   ADRS_MPI_ATC_DLNK_MSG         = 5612,   /* ATC COMM ROUTE UPLINK ETC. */
   ADRS_MPI_PROFILE_MSG  = 5701,   /* USE ONLY FOR ADRS TO ADRS COMM */
   ADRS_MPI_METER_FIX_TIMES_CTR= 5801,
   ADRS_MPI_FAF_TIMES_CTR         = 5802,
   ADRS_MPI_THRESHOLD_TIMES_CTR= 5803,
   ADRS_MPI_METER_FIX_TIMES_TRC= 5811,
   ADRS_MPI_FAF_TIMES_TRC         = 5812,
   ADRS_MPI_THRESHOLD_TIMES_TRC= 5813,
   ADRS_MPI_ARRIVAL_TIMES         = 5821,    /* Not supported */
   ADRS_MPI_CDTI_OWNSHIP_MSG   = 5901,
   ADRS_MPI_SCENARIO_MSG       = 5911,
   ADRS_MPI_REF_POS_MSG        = 5921,
   ADRS_MPI_SYSTEM_TIME_MSG     = 5931,
   ADRS_MPI_SESSION_TIME_MSG   = 5932,
   ADRS_MPI_PROCESS_INFO_MSG   = 5941,
   ADRS_MPI_NEW_SOURCE_MSG     = 5951,
   ADRS_MPI_FORWARDING_MSG     = 5977,
   ADRS_MPI_WILD_CARD_MSG      = 5988,
   ADRS_MPI_TRANSACTION_MSG    = 5999,    /* This is to be used by ADRS only */
   ADRS_MPI_MSG_COUNT
} adrs_mpi_msg_types;

typedef enum {
    ADRS_MPI_DEPARTURE = 1,
    ADRS_MPI_ARRIVAL = 2,
    ADRS_MPI_OVERFLIGHT = 3,
```

```
    ADRS_MPI_SATELLITE_ARRIVAL = 4,
    ADRS_MPI_SATELLITE = 5,
    ADRS_MPI_DEPARTURE_ARRIVAL = 6,
    ADRS_MPI_SATELLITE_DEPARTURE = 7,
    ADRS_MPI_SATELLITE_TO_EXTERNAL = 8,
    ADRS_MPI_EXTERNAL_TO_SATELLITE = 9,
    ADRS_MPI_AMBIGUOUS_CATEGORY = 4096
} adrs_mpi_category;

typedef enum {
    ADRS_MPI_CS_NONE                = (1 << 0),
    ADRS_MPI_CS_ALT_AT      = (1 << 1),
    ADRS_MPI_CS_ALT_AA      = (1 << 2),
    ADRS_MPI_CS_ALT_AB      = (1 << 3),
    ADRS_MPI_CS_SPD                 = (1 << 4),
    ADRS_MPI_CS_FMS_ALT   = (1 << 9),
    ADRS_MPI_CS_ALT                 = (ADRS_MPI_CS_ALT_AT | \
                            ADRS_MPI_CS_ALT_AA | ADRS_MPI_CS_ALT_AB),
    ADRS_MPI_CS_TURN_AFTER          = (1 << 10),
    ADRS_MPI_CS_TURN_BEFORE         = (1 << 11),
    ADRS_MPI_CS_START_DES           = (1 << 12),
    ADRS_MPI_CS_START_CLB           = (1 << 13),
    ADRS_MPI_CS_END_CLB   = (1 << 14),
    ADRS_MPI_CS_END_DES   = (1 << 15),
    ADRS_MPI_CS_ACTIVE_LEG_ORIG = (1 << 20),
    ADRS_MPI_CS_CURRENT_WPT         = (1<<21),
    ADRS_MPI_CS_NEXT_TRAJ_PT        = (1<<22)
} adrs_mpi_constraint_types;

typedef enum {
    ADRS_MPI_TYPE_WP=0,             /* waypoint                */
    ADRS_MPI_TYPE_HP,               /* hold pattern            */
    ADRS_MPI_TYPE_PH,               /* proc hold               */
    ADRS_MPI_TYPE_PT,               /* proc turn               */
    ADRS_MPI_TYPE_RF,               /* rf leg          */
    ADRS_MPI_TYPE_TC,               /* TOC                     */
    ADRS_MPI_TYPE_TD,               /* TOD                     */
    ADRS_MPI_TYPE_SL,               /* start of level   */
    ADRS_MPI_TYPE_CA,               /* crossover altitude      */
    ADRS_MPI_TYPE_TA,               /* transition altitude  */
    ADRS_MPI_TYPE_AC,               /* Aircraft position    */
    ADRS_MPI_TYPE_CS,               /* only constraint         */
    ADRS_MPI_TYPE_RT,               /* part of current rte  */
    ADRS_MPI_TYPE_AP                /* AIRPORT DATA            */
} adrs_mpi_waypoint_types;



typedef enum
{
   ADRS_MPI_ADS_EQUIPPED       = (1 << 1),
   ADRS_MPI_ADS_AVAIL          = (1 << 2),
   ADRS_MPI_ADS_ACTIVE         = (1 << 3),
   ADRS_MPI_ADS_INTENT         = (1 << 4),
   ADRS_MPI_ADS_INTENT_EXT     = (1 << 5),
   ADRS_MPI_CPDLC_EQUIPPED     = (1 << 6),
   ADRS_MPI_CPDLC_AVAIL        = (1 << 7),
   ADRS_MPI_CPDLC_ACTIVE       = (1 << 8),
   ADRS_MPI_WTHR_SENT          = (1 << 9),
```

```
   ADRS_MPI_ROUTE_CHANGED      = (1 << 10),        /* not required for MPI */
   ADRS_MPI_ADS_PROFILE   = (1 << 11)

} adrs_mpi_dlnk_types;

typedef enum
{
   ADRS_MPI_VECTORED = 0,
   ADRS_MPI_ONROUTE = 1
} adrs_mpi_flight_status;

typedef enum
{
   ADRS_MPI_SEND_ONCE           = 1,
   ADRS_MPI_SEND_PERIODICALLY    = 2,
   ADRS_MPI_SEND_WHEN_NEW       = 3,
   ADRS_MPI_SEND_WHEN_CHANGED           = 4,
   ADRS_MPI_ACTION_COUNT
} adrs_mpi_action_types;

#if 0
typedef enum
{
   ADRS_MPI_DESCR_ALL                = 1,
   ADRS_MPI_DESCR_SECTOR             = 2,
   ADRS_MPI_DESCR_STATIC_CIRCLE = 3,
   ADRS_MPI_DESCR_AC_CIRCLE     = 4,
   ADRS_MPI_DESCR_ACID               = 5,
   ADRS_MPI_DESCR_COUNT
} adrs_mpi_descr_types;
#endif

typedef enum
{
   ADRS_MPI_DESCR_DEFAULT                          =  0,
   ADRS_MPI_DESCR_NONE                             =  (1 << 0),
   ADRS_MPI_DESCR_INCLUDE_LEAVING_AREA                     =  (1 << 1),
   ADRS_MPI_DESCR_AND_SECTOR                       =  (1 << 2),
   ADRS_MPI_DESCR_AND_STATIC_CIRCLE                =  (1 << 3),
   ADRS_MPI_DESCR_AND_AC_CIRCLE                    =  (1 << 4),
   ADRS_MPI_DESCR_AND_ACID                         =  (1 << 5),
   ADRS_MPI_DESCR_AND_DATA_SRC_HOST                        =  (1 << 6),
   ADRS_MPI_DESCR_AND_ADRS_SRC_HOST                        =  (1 << 7),
   ADRS_MPI_DESCR_AND_NOT_DATA_SRC_HOST =  (1 << 8),
   ADRS_MPI_DESCR_AND_NOT_ADRS_SRC_HOST =  (1 << 9),
   ADRS_MPI_DESCR_PROC_INCL_ACFS                   =  (1 << 10),
   ADRS_MPI_DESCR_PROC_INCL_MINIACFS                       =  (1 << 11),
   ADRS_MPI_DESCR_PROC_INCL_B757                   =  (1 << 12),
   ADRS_MPI_DESCR_PROC_INCL_PC_PLANE                       =  (1 << 13),
   ADRS_MPI_DESCR_PROC_INCL_B747                   =  (1 << 14),
   ADRS_MPI_DESCR_PROC_INCL_CDTI                   =  (1 << 15),
   ADRS_MPI_DESCR_PROC_INCL_PAS_ACTIVE                     =  (1 << 16),
   ADRS_MPI_DESCR_PROC_INCL_PAS_PLAYBACK =  (1 << 17),
   ADRS_MPI_DESCR_PROC_INCL_ACTIVE_AC                      =  (1 << 18),
   ADRS_MPI_DESCR_PROC_INCL_INACTIVE_AC            =  (1 << 19),

   ADRS_MPI_DESCR_EXCLUDE_NON_DATALINK                     = (1 << 20),

/* The following are descriptors that are aircraft independent */
```

```
   ADRS_MPI_DESCR_USE_UTC                          = (1 << 21),
   ADRS_MPI_DESCR_USE_TIME_FROM_TRIAL_START        = (1 << 22),
   ADRS_MPI_DESCR_USE_MACHINE_TIME                 = (1 << 23),

   ADRS_MPI_DESCR_PROC_INCL_ASTOR                  = (1 << 24),
   ADRS_MPI_DESCR_PROC_INCL_TMX                    = (1 << 25),
   ADRS_MPI_DESCR_PROC_INCL_DIBS                   = (1 << 26),
   ADRS_MPI_DESCR_PROC_INCL_ALL                    =  (                \
                                ADRS_MPI_DESCR_PROC_INCL_ACFS     |   \
                                ADRS_MPI_DESCR_PROC_INCL_MINIACFS  |\
                                ADRS_MPI_DESCR_PROC_INCL_B757          |       \
                                ADRS_MPI_DESCR_PROC_INCL_PC_PLANE  |        \
                                ADRS_MPI_DESCR_PROC_INCL_B747     |    \
                                ADRS_MPI_DESCR_PROC_INCL_CDTI     |    \
                                ADRS_MPI_DESCR_PROC_INCL_ASTOR     | \
                                ADRS_MPI_DESCR_PROC_INCL_TMX     |     \
                                ADRS_MPI_DESCR_PROC_INCL_DIBS     |    \
                                ADRS_MPI_DESCR_PROC_INCL_PAS_ACTIVE |       \
                                ADRS_MPI_DESCR_PROC_INCL_PAS_PLAYBACK  |\
                                ADRS_MPI_DESCR_PROC_INCL_ACTIVE_AC    |       \
                                ADRS_MPI_DESCR_PROC_INCL_INACTIVE_AC ),
   ADRS_MPI_DESCR_AND_NOT_PILOT_HOST               =  (1 << 31),

   ADRS_MPI_DESCR_COUNT
} adrs_mpi_descr_types;



typedef enum
{
   ADRS_MPI_STATUS_NEW_REQUEST                     = 0,
   ADRS_MPI_STATUS_APPROVED                        = 1,
   ADRS_MPI_STATUS_INVALID_MESSAGE_TYPE            = 2,
   ADRS_MPI_STATUS_INVALID_COMBINATION                     = 3,
   ADRS_MPI_STATUS_INVALID_SPECIFICATION           = 4,
   ADRS_MPI_STATUS_INVALID_DESCRIPTOR                      = 5,
   ADRS_MPI_STATUS_INVALID_ACTION                  = 6,
   ADRS_MPI_STATUS_LEFT_AREA                       = 10,
   ADRS_MPI_STATUS_FORWARDED                       = 20,
   ADRS_MPI_STATUS_COUNT
} adrs_mpi_status_types;

typedef enum
{
   ADRS_MPI_TI_NONE                          = 0,
   ADRS_MPI_TI_FROZEN                        = (1<<0),
   ADRS_MPI_TI_SCHEDULED           = (1<<1),
   ADRS_MPI_TI_MANUALLY_SCHEDULED            = (1<<2),
   ADRS_MPI_TI_SEQUENCE_CONSTRAINT           = (1<<3),
   ADRS_MPI_TI_POPUP                         = (1<<4),
   ADRS_MPI_TI_CENTER_AREA                   = (1<<5)
} adrs_mpi_time_info_types;

typedef enum
{
   ADRS_MPI_CON_NONE                         =0,
   ADRS_MPI_CON_INIT                         =1,
   ADRS_MPI_CON_START                        =2,
   ADRS_MPI_CON_SWITCH                       =3,
```

```
    ADRS_MPI_CON_STOP                    =4,
    ADRS_MPI_CON_INIT_SUCCESS            =5,
    ADRS_MPI_CON_INIT_ERROR              =6,
    ADRS_MPI_CON_START_SUCCESS                   =7,
    ADRS_MPI_CON_START_ERROR             =8,
    ADRS_MPI_CON_SWITCH_SUCCESS                  =9,
    ADRS_MPI_CON_SWITCH_ERROR            =10,
    ADRS_MPI_CON_STOP_SUCCESS            =11,
    ADRS_MPI_CON_STOP_ERROR              =12,
    ADRS_MPI_CON_COUNT
} adrs_mpi_control_types;

typedef enum
{
    ADRS_MPI_TT_UTC                  =1,
    ADRS_MPI_TT_MACHINE_TIME             =2,
    ADRS_MPI_TT_FROM_TRIAL_START         =3,
    ADRS_MPI_TT_GPS                  =4,
    ADRS_MPI_TT_COUNT
} adrs_mpi_time_types;

typedef enum {
     ADRS_MPI_FS_UNKNOWN = 0,            /* Probably Vectored */
     ADRS_MPI_FS_LNAV                = (1<<0),
     ADRS_MPI_FS_VNAV                = (1<<1),
     ADRS_MPI_FS_ARRIVAL    = (1<<2),
     ADRS_MPI_FS_OVERFLIGHT          = (1<<3),
     ADRS_MPI_FS_DEPARTURE           = (1<<4),
     ADRS_MPI_FS_PLAYBACK = (1<<10),
     ADRS_MPI_FS_FREE_FLIGHT         = (1<<20),
     ADRS_MPI_FS_ATC_CONTROLLED = (1<<21),
     ADRS_MPI_FS_CDTI                = (1<<30),
     ADRS_MPI_FS_COUNT
} adrs_mpi_flight_status_types;

typedef enum {
     ADRS_MPI_CTRL_UNKNOWN           = 0,
     ADRS_MPI_CTRL_REGULAR           = 1,
     ADRS_MPI_CTRL_OVERWRITE         = 2,
     ADRS_MPI_CTRL_RETURN= 3,
     ADRS_MPI_CTRL_REMOTE            = 4,
     ADRS_MPI_CTRL_LOCAL   = 5,
     ADRS_MPI_CTRL_FALLBACK          = 6,
     ADRS_MPI_CTRL_NO_FALLBACK  = 7,
     ADRS_MPI_CTRL_COUNT
} adrs_mpi_control_status_types;

#define ADRS_MPI_EXT_CMD_INFO_STRG_SIZE 24

typedef enum {
   ADRS_MPI_EC_SPEED_COMMAND,      /* speed command for spacing */
   ADRS_MPI_EC_AURAL_ALERT,             /* Aural alert */
   ADRS_MPI_EC_FMS_COMMAND,
   ADRS_MPI_EC_SPACING_ENGAGED,
   ADRS_MPI_EC_SPACING_DISENGAGED,
   ADRS_MPI_EC_PTA_ENGAGED,
   ADRS_MPI_EC_PTA_DISENGAGED,
   ADRS_MPI_EC_COUNT
} adrs_mpi_ext_cmd_types;
```

```
typedef enum
{
   ADRS_MPI_CI_NONE,
   ADRS_MPI_CI_FMS_LOAD,
   ADRS_MPI_CI_FMS_ERASE,
   ADRS_MPI_CI_FMS_EXECUTE,
   ADRS_MPI_CI_COUNT
} adrs_mpi_crew_input_types;



/*
 *-----------------------------------------------------------------------------
 * data types
 *-----------------------------------------------------------------------------
 */


typedef struct
{
    adrs_mpi_msg_types          msg_type;       /* see ADRS_MPI_..._MSG */
    int                         msg_len;
    adrs_proc_type      src_process;        /* see adrs_interface.h_... */
} adrs_mpi_msg_hdr_st;

typedef struct
{
    char                        identifier[ADRS_MPI_REQ_ID_LENGTH];
    adrs_mpi_status_types       status;
    adrs_proc_type              process;
    char                        adrs_src_host[ADRS_MPI_HOST_LENGTH];
    char                        data_src_host[ADRS_MPI_HOST_LENGTH];
} adrs_mpi_ident_st;

/* Handshaking */

typedef struct
{
   adrs_mpi_msg_hdr_st msg_hdr;
   char            host[ADRS_MPI_STR_SIZE_S];
   char            version_id[ADRS_MPI_VERSION_ID_LENGTH];
   char            site_id[ADRS_MPI_MAX_NAME_LENGTH];
   char            allign_char;
} adrs_mpi_initial_client_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st msg_hdr;
   int             version_id_ok;
   int             site_id_ok;
} adrs_mpi_initial_server_response_msg_st;

typedef struct
{
   int                 requested_msg_type;
   adrs_mpi_action_types  action;
   float               update_rate;
```

```
    adrs_mpi_descr_types            descr;
    int                 sector_id;
    float               center_latitude;
    float               center_longitude;
    float               max_dist;
    float               min_altitude;   /* delta alt if descr == AC_CIRCLE */
    float               max_altitude;
    int            max_aircraft;
    char                acid[ADRS_MPI_ID_LENGTH];
    char                adrs_src_host[ADRS_MPI_HOST_LENGTH];
    char                data_src_host[ADRS_MPI_HOST_LENGTH];
    double                      time_sent;
} adrs_mpi_request_st;

typedef struct
{
    int            data_source;         /* data_source of fp source */
    char      acid[ADRS_MPI_ID_LENGTH];
    adrs_proc_type  adrs_proc;
    int                   cid;
    char           type[ADRS_MPI_MAX_TYPE_CHARS];
    char      gate_name[ADRS_MPI_MAX_NAME_LENGTH];
    char      meter_fix_name[ADRS_MPI_MAX_NAME_LENGTH];
    char      outer_fix_name[ADRS_MPI_MAX_NAME_LENGTH];
    adrs_mpi_category category;
    char           route[ADRS_MPI_MAX_FILED_ROUTE_LENGTH];
    char           departure_fix[ADRS_MPI_ID_LENGTH];
    int            departure_time;
    int            assigned_altitude;
    float          filed_speed;         /* kts, true airspeed                  */
    int            time_enroute;
    float      approach_speed;
    float      landing_speed;
    char           coordination_frd[ADRS_MPI_MAX_NAME_LENGTH];
    char           coordination_fix[ADRS_MPI_MAX_NAME_LENGTH];
    float          coordination_x;     /* nm                                  */
    float          coordination_y;     /* nm                                  */
    int            faa_coord_time;
    int            coordination_time;
    char           destination_fix[ADRS_MPI_MAX_NAME_LENGTH];
    char           destination_name[ADRS_MPI_MAX_NAME_LENGTH];
    char           runway_name[ADRS_MPI_MAX_NAME_LENGTH];
    int            configuration;
    int            beacon;              /* transponder code                    */
    char           atc_type[ADRS_MPI_MAX_TYPE_CHARS];
    int            time_received;
                        /*
                         * 1 = proposed, 2 = estimated,
                         * 3 = departed, 4 = blocked slot
                         */
    short          status;
    char      fp_data_source;         /* R = radar, etc...                   */
    char      equipment_available;
    int                   dlnk_equipped;
}  adrs_mpi_flight_plan_st;

typedef struct
{
    char      acid[ADRS_MPI_ID_LENGTH];
```

```
    adrs_proc_type  adrs_proc;
    int                     cid;
} adrs_mpi_delete_ac_st;


typedef struct
{
    char      acid[ADRS_MPI_ID_LENGTH];
    adrs_proc_type  adrs_proc;
    int                     cid;
    int                     route_len;
    char      route[ADRS_MPI_MAX_AK_ROUTE_LENGTH];
} adrs_mpi_ak_route_st;


typedef struct
{
    char          acid[ADRS_MPI_ID_LENGTH];  /*                    */
    adrs_proc_type adrs_proc;
    int           cid;              /*                          */
    int           beacon;           /*                          */
    float     latitude;             /* degrees                  */
    float     longitude;            /* degrees                  */
    float         altitude;         /* altitude in feet         */
    float         ground_speed;     /* nm per hour              */
    float         heading;          /* degrees 0 - 360          */
    float         vertical_speed;   /* feet per second          */
    float         x;                /* x pos in facility coordinates*/
    float         y;                /* y pos in facility coordinates*/
    double        time;             /* seconds since 1970       */
    int           sector_id;        /* artcc only               */
    char          controlling_facility[4];  /* artcc only       */
    char          allign_strg[4];
    int           receiving_sector;         /* artcc only       */
    char          receiving_facility[4];    /* artcc only       */
    int           sensor_id;        /* arts only                */
    double        raw_time;         /* time in source fmt       */
    double        raw_az;           /* arts only                */
    double        raw_range;        /* arts only                */
    adrs_mpi_dlnk_types dlnk;
    char          allign_strg2[4];
} adrs_mpi_track_st;

typedef struct
{
    char          acid[ADRS_MPI_ID_LENGTH];  /*                    */
    adrs_proc_type adrs_proc;
    int           cid;              /*                          */
    int           beacon;           /*                          */
    float     latitude;             /* degrees                  */
    float     longitude;            /* degrees                  */
    float         altitude;         /* altitude in feet         */
    float         ground_speed;     /* nm per hour              */
    float         heading;          /* degrees 0 - 360          */
    float         vertical_speed;   /* feet per minute          */
    float         x;                /* x pos in facility coordinates*/
    float         y;                /* y pos in facility coordinates*/
    double        time;             /* seconds since 1970       */
    int           sector_id;        /* artcc only               */
```

```
    char          controlling_facility[4];  /* artcc only            */
    char          allign_strg1[4];
    int           receiving_sector;         /* artcc only            */
    char          receiving_facility[4];    /* artcc only            */
    int           sensor_id;          /* arts only                   */
    double        raw_time;           /* time in source fmt          */
    float         raw_az;             /* arts only                   */
    float         raw_range;          /* arts only                   */
    adrs_mpi_dlnk_types dlnk;
    float     altitude_target;
    float     heading_target;
    float     speed_target;       /* .2 < speed_target < 5. Mach else CAS*/
    float     indicated_air_speed;
    float     mach;
    float     bank_angle;
    float     pitch_angle;
    int                   flight_status;
    char      allign_strg2[4];
} adrs_mpi_state_st;

typedef struct
{
    float     yaw_angle;
    float     bank_rate;
    float     pitch_rate;
    float     yaw_rate;
    float     magnetic_heading;
    float     magnetic_variation;
    float     true_ground_track;
    float     true_air_speed;
    float     lift_coefficient;
    long      last_waypoint;
    long      equipage;
    char      sector_name[ADRS_MPI_SECTOR_LENGTH];
    int       geo_sector_id;
    int       ctas_sector_id;
    int       ac_sector_id;
} adrs_mpi_state_plus_st;

typedef struct
{
   float      wind_direction;
   float      wind_speed;
   float      external_mach_target;
   float      map_range_captain;
   float      map_range_fo;
   float      heading_bug;
} adrs_mpi_state_add_st;


typedef struct
{
    adrs_mpi_state_st         state;
    adrs_mpi_state_plus_st  state_plus;
} adrs_mpi_extended_state_st;

typedef struct
{
    adrs_mpi_state_st         state;
```

```
    adrs_mpi_state_plus_st  state_plus;
    adrs_mpi_state_add_st   state_add;
} adrs_mpi_complete_state_st;


typedef struct
{
    char acid[ADRS_MPI_ID_LENGTH];
    int      cid;
    char data[ADRS_MPI_PAS_COMM_SIZE];
} adrs_mpi_pas_comm_st;

#define MAX_ADRS_MPI_TRAJECTORY 20     /* should be enough for ctas */

/*
 * Note: The following structure first contains all waypoints
 *       that effect the horizontal path and appends all the
 *           vertical points at the end. Though this may be
 *       inconvenient for certain applications, it distinguishes
 *       between the different nature of those points.
 *           The element 0 should contain the aircraft state,
 *           when the trajectory was sent/received
 */

typedef struct
{
     /*int        misc_int_value;*/
     adrs_mpi_waypoint_types type;
     char                waypoint_id [ADRS_MPI_ID_LENGTH];
     float     latitude;
     float     longitude;
     float     turn_radius;
     /*float      misc_float_value;*/
     double     eta;
     float     calibrated_air_speed;
     float     altitude;
     float     fuel_remaining;
     float     outer_air_temperature;
     float     wind_direction;
     float     wind_speed;
     float     true_air_speed;
     float     true_course_into_point;
     float     distance_to_point;
     float     predicted_gross_weight;
     float     x;      /*Facility coord */
     float         y;
     adrs_mpi_constraint_types constraint;
} adrs_mpi_traj_point_st;

typedef struct
{
     char                acid[ADRS_MPI_ID_LENGTH];
     adrs_proc_type adrs_proc;
     int        cid;
     int        number_of_points;
     int        number_of_horizontal_points;
     float cruise_speed;              /* UPLINK only */
     float descent_speed;             /* UPLINK only */
     char text[ADRS_MPI_TRAJ_TEXT_LENGTH];
```

```
} adrs_mpi_traj_descr_st;

typedef struct
{
    char            acid[ADRS_MPI_ID_LENGTH];
    adrs_proc_type adrs_proc;
    int     cid;
    int     number_of_points;
    int     number_of_horizontal_points;
    float   climb_speed;
    float   cruise_speed;
    float   descent_speed;
    float   approach_speed;
    float   landing_speed;
    float   cruise_altitude;
    float   current_gross_weight;
    float   landing_weight;
    char text[ADRS_MPI_TRAJ_TEXT_LENGTH];
} adrs_mpi_traj_spec_st;


typedef struct
{
    adrs_mpi_traj_descr_st descr;
    adrs_mpi_traj_point_st point[MAX_ADRS_MPI_TRAJECTORY];
} adrs_mpi_traj_dyn_st;

#define ADRS_MPI_TRAJ_DYN_SIZE(X)  (sizeof(adrs_mpi_traj_descr_st) + \
                    ((X)->descr.number_of_points * sizeof (adrs_mpi_traj_point_st)))

typedef struct
{
    adrs_mpi_traj_spec_st spec;
    adrs_mpi_traj_point_st point[MAX_ADRS_MPI_TRAJECTORY];
} adrs_mpi_trajectory_st;

#define ADRS_MPI_TRAJECTORY_SIZE(X)  (sizeof(adrs_mpi_traj_spec_st) + \
                    ((X)->spec.number_of_points * sizeof (adrs_mpi_traj_point_st)))

typedef struct {
   char  acid[ADRS_MPI_ID_LENGTH];
   int             cid;
   int             frequency_1;     /* first frequency   * 1000     */
   int             frequency_1_flg;/* communication destination */
   int             frequency_2;     /* second_frequency  * 1000       */
   int             frequency_2_flg;/* communication destination */
} adrs_mpi_frequency_st;

#define ADRS_MPI_MAX_DLNK_LENGTH 8*1024         /* really big for internal use only */
#ifndef NEW_DATALINK_FORMAT

struct adrs_mpi_datalink_st
{
        char                    source_id[ADRS_MPI_ID_LENGTH];
        char            dest_id[ADRS_MPI_ID_LENGTH];
        int             type;
        int             sequence_number;
        char            message[ADRS_MPI_MAX_DLNK_LENGTH];
} ;
```

```
typedef struct adrs_mpi_datalink_st adrs_mpi_datalink_st;
#endif

struct adrs_mpi_dlnk_st
{
        char            source_id[ADRS_MPI_ID_LENGTH];
        char            dest_id[ADRS_MPI_ID_LENGTH];
        int             type;
        int             sequence_number;
        int             message_length;
        char            message[ADRS_MPI_MAX_DLNK_LENGTH];
} ;

typedef struct adrs_mpi_dlnk_st adrs_mpi_dlnk_st;


/* New buffer for Pas pilot stations */
struct adrs_mpi_pilot_input_st
{
   char  acid[ADRS_MPI_ID_LENGTH];
   int           cid;
   int           input_count;
   int           speed_mode;
   int           altitude_mode;
   int           heading_mode;
   float  target_speed;
   float  target_vertical_speed;
   float  target_altitude;
   float  target_heading;
   char  target_waypoint[ADRS_MPI_ID_LENGTH];
   float  target_spacing;
   char  lead_ac[ADRS_MPI_ID_LENGTH];
   float  climb_speed;
   float  cruise_speed;
   float  descent_speed;
   float  approach_speed;
   float  cruise_altitude;
   int           confirmed_msgs;
   int           control_status;
   int           sector_checked_in;
   char  pilot_host[ADRS_MPI_HOST_LENGTH];
} ;
typedef struct adrs_mpi_pilot_input_st adrs_mpi_pilot_input_st;

struct adrs_mpi_guidance_output_st
{
   char  acid[ADRS_MPI_ID_LENGTH];
   int           cid;
   int           pilot_input_count;
   int           speed_mode;
   int           altitude_mode;
   int           heading_mode;
   float  target_speed;
   float  target_vertical_speed;
   float  target_altitude;
   float  target_heading;
   char  target_waypoint[ADRS_MPI_ID_LENGTH];
   float  current_spacing;
```

```
};
typedef struct adrs_mpi_guidance_output_st adrs_mpi_guidance_output_st;


#define ADRS_MPI_ADJUSTED_SIZE(X) ((int)((X+7)/8)*8)

#define ADRS_MPI_DLNK_MSG_FIXED_SIZE (2*ADRS_MPI_ID_LENGTH + 3*sizeof(int))
#define ADRS_MPI_DLNK_MSG_SIZE_RAW(X) \
                (ADRS_MPI_DLNK_MSG_FIXED_SIZE + \
                        strlen(X->message)+1)
#define ADRS_MPI_DLNK_MSG_SIZE(X) ADRS_MPI_ADJUSTED_SIZE \
                        (ADRS_MPI_DLNK_MSG_SIZE_RAW(X))
#ifndef NEW_DATALINK_FORMAT

#define ADRS_MPI_DATALINK_MSG_FIXED_SIZE (2*ADRS_MPI_ID_LENGTH + 2*sizeof(int))
#define ADRS_MPI_DATALINK_MSG_SIZE_RAW(X) \
                (ADRS_MPI_DATALINK_MSG_FIXED_SIZE + \
                        strlen(X->message)+1)
#define ADRS_MPI_DATALINK_MSG_SIZE(X) ADRS_MPI_ADJUSTED_SIZE \
                        (ADRS_MPI_DATALINK_MSG_SIZE_RAW(X))
#endif
/*
 * call with pointer to struct e.g:
 *
 * adrs_mpi_datalink_st *dlnk, static_dlnk;
 *      msg_size = ADRS_MPI_DLNK_MSG_SIZE(dlnk);
 *      msg_size = ADRS_MPI_DLNK_MSG_SIZE((&static_dlnk));
 */

typedef struct
{
        char            acid[ADRS_MPI_ID_LENGTH];
        char            command[ADRS_MPI_PAS_COMMAND_LENGTH];
} adrs_mpi_pas_command_st;

typedef struct
{
        char            acid[ADRS_MPI_ID_LENGTH];
        int                     cid;
        float           tod_along_track_distance;
        float                   cruise_speed;
        float                   descent_speed;
} adrs_mpi_profile_st;

typedef struct
{
        char            acid[ADRS_MPI_ID_LENGTH];
        int                     cid;
    char                        waypoint_id [ADRS_MPI_ID_LENGTH];
        long                    slow;
        long                    nominal;
        long                    fast;
        long                    solution;
        long                    scheduled;
        adrs_mpi_time_info_types info;
} adrs_mpi_arrival_st;

typedef struct
{
```

```
        char                acid[ADRS_MPI_ID_LENGTH];
        char                        host[ADRS_MPI_HOST_LENGTH];
        int                         cid;
        adrs_mpi_control_types control;
} adrs_mpi_cdti_ownship_st;

typedef struct
{
        char                        name[ADRS_MPI_SCENARIO_LENGTH];
        char                        host[ADRS_MPI_HOST_LENGTH];
        adrs_mpi_control_types control;
} adrs_mpi_scenario_st;

typedef struct
{
        char                        facility[ADRS_MPI_ID_LENGTH];
        float                       latitude;
        float                       longitude;
        float                       altitude;
} adrs_mpi_ref_pos_st;

typedef struct
{
        adrs_mpi_time_types     type;
    long                        year;
        long                        month;
    long                        day;
    double                      time;
} adrs_mpi_date_time_st;

#define ADRS_MPI_WILD_CARD_STRING_LENGTH    80
#define ADRS_MPI_WILD_CARD_BUFFER_SIZE      3000


typedef struct {
        adrs_dest_types dest_type;
        int                 dest_int;
        adrs_proc_type  dest_adrs_proc;
        char                dest_string[ADRS_MPI_WILD_CARD_STRING_LENGTH];
    int         data_size;
        int                 dst_msg_type;
        char                data[ADRS_MPI_WILD_CARD_BUFFER_SIZE];
} adrs_mpi_wild_card_st;

#define ADRS_MPI_MAX_IO_DATA_POINTS             3

typedef struct
{
  char host[ADRS_MPI_STR_SIZE_S];
  adrs_proc_type adrs_proc;
  int comm_status;
  char service[32];
  int version;
  int active;
  double initial_connect_delay;
  double retry_delay_step;
  int is_adrs;
  /* The following are set by process control */
  int uid;
```

```
  int is_remote_server;
  /*int more;*/
  double time_online;
  char name[ADRS_MPI_STR_SIZE_S];
}adrs_mpi_connect_info_st;

typedef struct
{
   int bytes_processed;
   int bytes_to_go;
   int bytes_in_error;
   int messages_succeeded;
   int messages_failed;
   float delay;
   int errors;
   int more;
} adrs_mpi_process_io_element_st;

typedef struct
{
   int more;
   float time_interval;
   double time_last_updated;
   adrs_mpi_process_io_element_st read;
   adrs_mpi_process_io_element_st write;
} adrs_mpi_process_io_st;

typedef struct
{
  adrs_mpi_connect_info_st connection;
  adrs_mpi_process_io_st total_io;
  adrs_mpi_process_io_st io[ADRS_MPI_MAX_IO_DATA_POINTS];
  char adrs_src_name[ADRS_MPI_STR_SIZE_S];
  int  adrs_uid;
  int ident;
  char adrs_src_host[ADRS_MPI_STR_SIZE_S];
  int adrs_src_proc;
  int spare_int;
  double adrs_src_start_time;
  double time_online;
  int is_local_connection;
  char site[ADRS_MPI_STR_SIZE_S];
   int to_be_removed;
} adrs_mpi_process_info_st;


typedef struct
{
            char acid[ADRS_MPI_ID_LENGTH];
            int  cid;
            int use_original_source;
            adrs_proc_type adrs_proc;
            adrs_proc_type src_adrs_proc;
        char sid_name[ADRS_MPI_STR_SIZE_S]; /*unique id: e.g AdrsAcceptIsmC_1 */
            char adrs_src_host[ADRS_MPI_STR_SIZE_S];
            char data_src_host[ADRS_MPI_STR_SIZE_S];
} adrs_mpi_src_st;
```

```
typedef struct
{
    char acid[ADRS_MPI_ID_LENGTH];
    int  cid;
    char station_name[ADRS_MPI_STR_SIZE_S];
    char adrs_src_name[ADRS_MPI_STR_SIZE_S];
    char src_name[ADRS_MPI_STR_SIZE_S];
    int control_status;
    int misc;
    int handoff_status;
    int handoff_sector;
} adrs_mpi_ac_control_st;

typedef struct
{
   char acid[ADRS_MPI_ID_LENGTH];
   adrs_mpi_ext_cmd_types ext_command;        /* frome CiExtCmdType */
   int                          int_value;
   double                       double_value;
   char                     string_value[ADRS_MPI_EXT_CMD_INFO_STRG_SIZE];
   double                       time;
} adrs_mpi_ext_cmd_st;


/*#define DST_MSG_HDR_IMBEDDED            -33333*/

/*
 * call with pointer to struct e.g:
 *
 * adrs_wild_card_st *wild_card,static_wild_card;
 *         msg_len = WILD_CARD_SIZE((wild_card));
 *         or
 *         msg_len = WILD_CARD_SIZE((&wild_card));
 */
typedef struct
{
    adrs_mpi_msg_hdr_st              msg_hdr;
    adrs_mpi_ident_st      ident;
} adrs_mpi_ident_msg_st;

typedef struct
{
    adrs_mpi_msg_hdr_st              msg_hdr;
    adrs_mpi_ident_st      ident;
    adrs_mpi_request_st     request;
} adrs_mpi_request_msg_st;

typedef struct
{
    adrs_mpi_msg_hdr_st              msg_hdr;
    adrs_mpi_ident_st      ident;
    adrs_mpi_flight_plan_st  flight_plan;
} adrs_mpi_flight_plan_msg_st;

typedef struct
{
    adrs_mpi_msg_hdr_st              msg_hdr;
    adrs_mpi_ident_st      ident;
    adrs_mpi_delete_ac_st    delete_ac;
```

```
} adrs_mpi_delete_ac_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_ak_route_st    ak;
} adrs_mpi_ak_route_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_track_st         track;
} adrs_mpi_track_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_state_st         state;
} adrs_mpi_state_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st                     msg_hdr;
   adrs_mpi_ident_st                ident;
   adrs_mpi_extended_state_st  extended_state;
} adrs_mpi_extended_state_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st                     msg_hdr;
   adrs_mpi_ident_st                ident;
   adrs_mpi_complete_state_st  complete_state;
} adrs_mpi_complete_state_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_frequency_st   frequency;
} adrs_mpi_frequency_msg_st;


typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_traj_dyn_st    traj_dyn;
} adrs_mpi_traj_dyn_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_trajectory_st    trajectory;
} adrs_mpi_trajectory_msg_st;
```

```
#ifndef NEW_DATALINK_FORMAT
typedef struct
{
   adrs_mpi_msg_hdr_st          msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_datalink_st    datalink;
} adrs_mpi_datalink_msg_st;
#endif

typedef struct
{
   adrs_mpi_msg_hdr_st          msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_dlnk_st    dlnk;
} adrs_mpi_dlnk_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st          msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_pas_command_st pas_command;
} adrs_mpi_pas_command_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st          msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_pas_comm_st    pas_comm;
} adrs_mpi_pas_comm_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st          msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_profile_st     profile;
} adrs_mpi_profile_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st          msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_arrival_st     arrival;
} adrs_mpi_arrival_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st          msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_cdti_ownship_st cdti_ownship;
} adrs_mpi_cdti_ownship_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st          msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_scenario_st    scenario;
} adrs_mpi_scenario_msg_st;

typedef struct
```

```
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_ref_pos_st     ref_pos;
} adrs_mpi_ref_pos_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_date_time_st   date_time;
} adrs_mpi_date_time_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_process_info_st      process;
} adrs_mpi_process_info_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_pilot_input_st  pilot_input;
} adrs_mpi_pilot_input_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_guidance_output_st  guidance_output;
} adrs_mpi_guidance_output_msg_st;


typedef struct
{
   adrs_mpi_pilot_input_st  pilot_input;
   adrs_mpi_guidance_output_st  guidance_output;
}  adrs_mpi_initial_state_st;

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_initial_state_st initial;
} adrs_mpi_initial_state_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_wild_card_st      wild_card;
} adrs_mpi_wild_card_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st      ident;
```

```
   adrs_macs_config_st     macs_config;
} adrs_mpi_macs_config_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st       ident;
   adrs_macs_message_st    macs_message;
} adrs_mpi_macs_control_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st       ident;
   char                data[ADRS_MPI_WILD_CARD_BUFFER_SIZE];
} adrs_mpi_generic_data_msg_st;


typedef struct
{
   int                             adrs_msg_type;
   int                             specific_msg_type;
   adrs_dest_types                 dest_type;
   int                             dest_int;
   char                dest_string[ADRS_MPI_HOST_LENGTH];
   int                             data_size;
   adrs_proc_type                  dest_adrs_proc;
   int                             dst_msg_type;
   double                          time_to_send;
   long                transaction_number;
   char                data[ADRS_MPI_TRS_INITIAL_DATA_SIZE];
} adrs_mpi_transaction_st;

#define ADRS_MPI_TRANSACTION_FIXED_SIZE (sizeof(adrs_mpi_transaction_st) - \
          ADRS_MPI_TRS_INITIAL_DATA_SIZE)
#define ADRS_MPI_TRANSACTION_SIZE(X) (ADRS_MPI_TRANSACTION_FIXED_SIZE + \
          X->data_size)

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st       ident;
   adrs_mpi_transaction_st descr;
} adrs_mpi_transaction_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st       ident;
   adrs_mpi_src_st                 src;
} adrs_mpi_src_msg_st;

typedef struct
{
   adrs_mpi_msg_hdr_st              msg_hdr;
   adrs_mpi_ident_st       ident;
   adrs_mpi_ac_control_st  ac_control;
} adrs_mpi_ac_control_msg_st;
```

```
typedef struct
{
   adrs_mpi_msg_hdr_st          msg_hdr;
   adrs_mpi_ident_st      ident;
   adrs_mpi_ext_cmd_st info;
} adrs_mpi_ext_cmd_msg_st;

typedef struct
{
   adrs_mpi_ident_st      ident;
   char                   data[2*ADRS_MPI_MAX_DLNK_LENGTH];
} adrs_mpi_forward_trs_st;

/*
 *-----------------------------------------------------------------------------
 * global variables
 *-----------------------------------------------------------------------------
 */


/*
 *-----------------------------------------------------------------------------
 * function prototypes
 *-----------------------------------------------------------------------------
 */

#endif
```

[3] HLA OBJECTS, ATTRIBUTES AND INTERACTIONS IN AIRSPACE AND TRAFFIC OPERATIONS SIMULATION

This document describes the objects and interactions that are currently present in our system. These are likely to change in the future with changes in our HLA environment or re-architecture that may take place.

3.1 Objects Supported

3.1.1 Entities for transmitting and receiving

Base Object: TranceivingEntity.
This is a base class for physical entities that possess transmitting and receiving properties.

Attributes:

Name	Data Type	Description
Entity_Id	String	This is the identification of an instance of this class.
Entity_Type	Enumeration	This indicates the nature of the entity. Ex. Airborne or ground.
ModeS_Trans_Type	Enumeration	This indicates the nature of the transmitter associated with this entity. None, low power, high power
ModeS_Recvr_Type	Enumeration	This indicates the nature of the receiver associated with this entity. None, low power, high power
Uat_Recvr_Type	Enumeration	This indicates the nature of the UAT receiver associated with

		this entity. None, low power, high power
Uat_Trans_Type	Enumeration	This indicates the nature of the UAT transmitter associated with this entity. None, low power, high power
Latitude	double	Latitude of the location of the entity in degrees.
Longitude	double	Longitude of the location of the entity in degrees
Altitude	float	This is the altitude of the entity in feet.

3.1.2 Derived Object: Aircraft

This is an object derived from TranceivingEntity and represents an aircraft in the simulation. It inherits all the attributes of the base class.

Attributes:

Name	Data Type	Description
Airspeed	float	Airspeed in knots.
TrueHeading	float	True heading in degrees from north.
MagVar	float	Magnetic variation. angle difference (in deg) between true and magnetic north, at the current position. If magnetic N-pole is West of the true N-pole, the variation is West, converse for East. Positive MagVar = West. Hence MagHdg = TrueHdg + MagVar
AcType	Enumeration	This indicates the type of the aircraft Ex. B777, Columbia, C-172… etc.

3.2 Interactions in ATOS:

3.2.1 Data Link interactions:

3.2.1.1 Base Interaction : DataLink

This is the base interactions class for Data Link protocol messages.

Parameters:

Name	Data Type	Description
EntityId	string	Unique Id of the entity of the message.
DataLinkType	enumeration	Type of the data link message.

3.2.1.2 Intermediate Derived Interaction :ADSB

This represents the ADSB message base class.

Parameters:

Name	Data type	3.2.1.3 Description
ParticipantAddress	Participant address field	This is the address of the participant.
ParticipantAddressQualifier	char	
TimeOfApplicability	float	The time at which the reported values were valid.

3.2.1.4 Derived Interaction: StateVector

This is the generic ADSB state vector message

Parameters:

Name	Data type	Description
BarometricAltitude	float	The aircraft barometric pressure altitude.
NIC	char	The Navigation Integrity Categories.
ReportMode	char	Mode of the aircraft that sent the ADSB message
EastVelocity	float	East velocity of the aircraft
GeometricAltitude	float	The shortest distance from the current aircraft position to the surface of the WGS-84 ellipsoid.
Latitude	float	The latitude of the current aircraft position.
Longitude	float	The longitude of the current aircraft position.
NorthVelocity	float	North velocity of the aircraft
SurfaceGroundSpeed	float	The aircraft speed on the ground surface
SurfaceHeading	float	The direction in which the nose of the aircraft is pointing on the surface
VerticalRate	float	Either geometric or barometric - Researcher configurable. A positive value indicates that the aircraft is climbing in altitude.
DiscreteData	char	Eight Data bits: Bit# Function Encoding 1 Horizontal Position Validity Valid Invalid 2 Geometric Altitude Validity Valid Invalid 3 Airborne Velocity Validity Valid Invalid 4 Surface Ground Speed Validity Valid Invalid 5 Surface Heading Validity Valid Invalid 6 Barometric Altitude Validity Valid Invalid 7 Vertical Rate Validity Valid Invalid 8 Spare

3.2.1.5 Derived Interaction:ModeStatus

This is the generic ADSB mode status message.

Patrameters:

Name	3.2.1.6 Data type	Description
CallSign	string	The aircraft call sign.
DestinationAirport	string	Current destination airport of the aircraft.
ParticipantCategory	char	The category of the current aircraft.
AircraftSizeCode	char	The size code of the current aircraft.
EmergencyPriorityStatus	char	The current message emergence/priority status.
ClassCode	char	The current aircraft class code.
OperationalModeData	char	The current aircraft operation mode data.
NACPosition	char	The current aircraft Navigation Accuracy Category for Position
NACVelocity	char	The current aircraft Navigation Accuracy Category for Velocity
SurveillanceIntegrityLevel	char	The current aircraft Surveilliance Integrity Level
NACBarometric	char	The current aircraft Navigation Accuracy Category for barometric altitude.
IasTasMachAirspeed	char	LAS/TAS Mach Airspeed
FlightModeData	char	The flight mode data
DiscreteData	Char	Eight data bits: Bit# Function Encoding 1 Surveillance Support Code Normal Default 2 ACAS/TCAS Advisory Active Active Inactive 3 NIC Barometric Cross-checked Not cross-chkd 4 True/Magnetic Heading TRUE Magnetic 5 Baro/Geo Vertical Rate Barometric Geometric 6 Spare 7 Spare 8 Spare

Derived Interaction : AirReferencedVelocity

The generic ADS-B air referenced velocity message.

Parameters:

Name	Data type	Description
AirborneHeading	float	The direction in which the nose of the aircraft is pointing.
AirSpeed	float	The aircraft speed in the air.
DiscreteData	char	Eight data bits: Bit# Function Encoding 1 Airspeed Valid Valid Invalid 2 Airborne Heading Valid Valid Invalid 3 Spare 4 Spare

		5 Spare 6 Spare 7 Spare 8 Spare

3.2.1.7 Intermediate Derived Interaction: AirportArrivalSequence

This interaction is used to provide airport arrival sequencing advisories to aircraft from the ground.

Parameters:

Name	Data type	Description
NumberOfSequencedAircraft	short	The number of aircraft currently assumed to be planning to use the runway.
Sequence	AircraftSequence	Seqeunces of the arriving aircraft. Repeat for NumberOfSeqeuncedAircraft

3.2.1.8 Intermediate Derived Interaction: FISB

This is the Flight Information Service - Broadcast message.

Parameters:
This intermediate derived interaction does not have any parameters.

3.2.1.9 Derived Interaction: AreaHazard

This interaction provides area hazard information to ASTOR. Area hazards include Special Use Airspace (SUA) and SIGMET convective weather regions. 3-D area hazards are described as 2D polygon regions with a height.

Parameters:

Name	Data type	Description
UniqueHazardID	string	Area hazard identification
StartTime	UtcTime	UTC Time when the area hazard (typically SUA) becomes active, hours, minutes, seconds.
FinishTime	UtcTime	UTC Time when the area hazard (typically SUA) becomes inactive, hours, minutes, seconds.
Severity	AreaHazardSeverity	Specifies the severity of the Area Hazard
TopAltitude	short	Top of the area hazard in hundreds of feet
BottomAltitude	short	Bottom of area hazard, hundreds of feet
MovementSpeed	short	Speed in knots.
MovementDirection	short	The direction of area hazard movement, 10s of degrees
Region	Polygon	The polygon describing the region.

3.2.1.10 Derived Interaction: WindTempAloft

This interaction provides non-gridded wind and temperatures aloft information to ASTOR. The data is organized in table format with airport locations being one index and altitude the other.

Parameters:

Name	Data type	Description
NumberOfAirports	short	Number of airports this information is being reported for.
AirportWinds	AirportWind	Airport Wind entries

2.2 Mode Control Interactions:

3.2.2 Base Interaction : ModeControl

This interaction is for controlling the simulation mode.

Parameters:
This base interaction does not have any parameters.

3.2.3 Derived Interaction: Configure

This interaction instructs the federate to perform any non-scenario initialization.

Parameters:

Name	Data type	Description
SimulationID	string	Unique id for the simulation.
OperationalMode	OperationMode	Specifies which mode of operation the simulation will run in.
PilotMode	PilotMode	Specifies how ASTOR will be "flown"

3.2.3.1 Derived Interaction: Reset

This interaction instructs the federate to initialize (first-time) or re-initialize to the initial conditions for the scenario.

Parameters:

Name	Data type	Description
SimulationID	string	Unique identifier for the aircraft ???
ScenarioName	string	Identifies the scenario
ExperimentName	string	Identifies the experiment; used for data collection, e.g., AUTRII
ConfigFileName	string	Specifies which configure file will be used for configuration
DataFilePrefix	string	Identifies the prefix to use for data collection files
DataCollectionEnabled	boolean	Identifies whether local data collection should be performed

3.2.3.2 Derived Interaction: Trim

This interaction instructs an aircraft federate to call a trimming function to put the aircraft in a steady state.

Parameters:
This interaction does not have any parameters.

3.2.3.3 Derived Interaction: Operate

This interaction instructs the federate that the simulation clock is starting.

Parameters:

Name	Data type	Description
CurrentTime	UtcTime	The current simulation time (UTC), hour minutes, seconds
SpeedFactor	short	The speed multiplier for fast time ('speed times N'). Default is 1 (normal speed)

3.2.3.4 Derived Interaction: Freeze

This interaction instructs the federate that the simulation time is frozen and the federate should go into the Hold mode.

Parameters:

Name	3.2.3.5 Data type	3.2.3.6 Description
HidePilotDisplays	boolean	Hide the pilot displays if true.
HoldSimulationTime	UtcTime	The simulation time at which to go to hold.
HoldScenarioTime	UtcTime	?????

3.2.3.7 Derived Interaction: Shutdown

This interaction instructs the federate to terminate itself and any sub-components it is responsible for.

Parameters:
This interaction does not have any parameters.

3.2.4 Cycle timing interactions

3.2.4.1 Base: Cycle timing

This interaction provides the synchronized cycle timing message with current simulation time in UTC and time into the scenario.

Parameters:

Name	Data type	Description
CurrentTime	UtcTime	This is the current time
ScenarioTime	long	This is the scenario time.