National Aeronautics and
Space Administration
Armstrong Flight Research Center
Edwards, CA 93523-0273

Live Virtual Constructive (LVC)

Interface Control Document (ICD) for the LVC Gateway

Flight Test 3

February 26, 2015

LVC ICD-03

Release: **Rev B**

UAS-NAS
Live Virtual Constructive – Distributed Environment (LVC)
Message Interface Control Document
For the LVC Gateway

Prepared by

Srba Jovic

Science Applications International Corporation (SAIC)
NASA Ames Research Center
Moffett Field, CA 94035-0081

February 26, 2015

Prepared By:

SJovic 03/20/15
Srba Jovic – IT&E SW & Integration, SAIC/NASA ARC

Concur:

Jamie Willhite 3-6-15
Jamie Willhite – IT&E - LVC Integration & Test, NASA AFRC

Jack Connolly 3/24/15
Jack Connolly – IT&E Systems Engineer, NASA ARC

Michael A Scardello 3/6/15
Mike Scardello – IT&E Systems Engineer, NASA AFRC

3/20/15
Neil Otto – Test Operations Lead, SAIC/NASA ARC

3/10/2015
Jim Murphy – Project Engineer, NASA ARC

Sam Kim 03/06/2015
Sam Kim – Project Engineer, NASA AFRC

Approve:

MATT 4/6/15
Matt Knudson – DPMf, NASA ARC

Heather Maliska 3/9/15
Heather Maliska – DPMf, NASA AFRC

3.9.15
Peggy S. Hayes – Deputy CSE, UAS-NAS NASA AFRC

Version #	Date	Page #	Author	Description
Baseline	May 1, 2013	All	Srba Jovic	Initial Release of Document
Rev A	Mar 28, 2014		Srba Jovic	Updates for IHITL
Rev B	Feb 26, 2015	All	Srba Jovic	Update for FT3 Requirements, including addition of Stratway+ SS Band messages and Omni Band messages.

1. Introduction

This Interface Control Document (ICD) documents and tracks the necessary information required for the Live Virtual and Constructive (LVC) system's components as well as protocols for communicating with them in order to achieve all research objectives captured by the experiment requirements. The purpose of this ICD is to clearly communicate all inputs and outputs from the subsystem components.

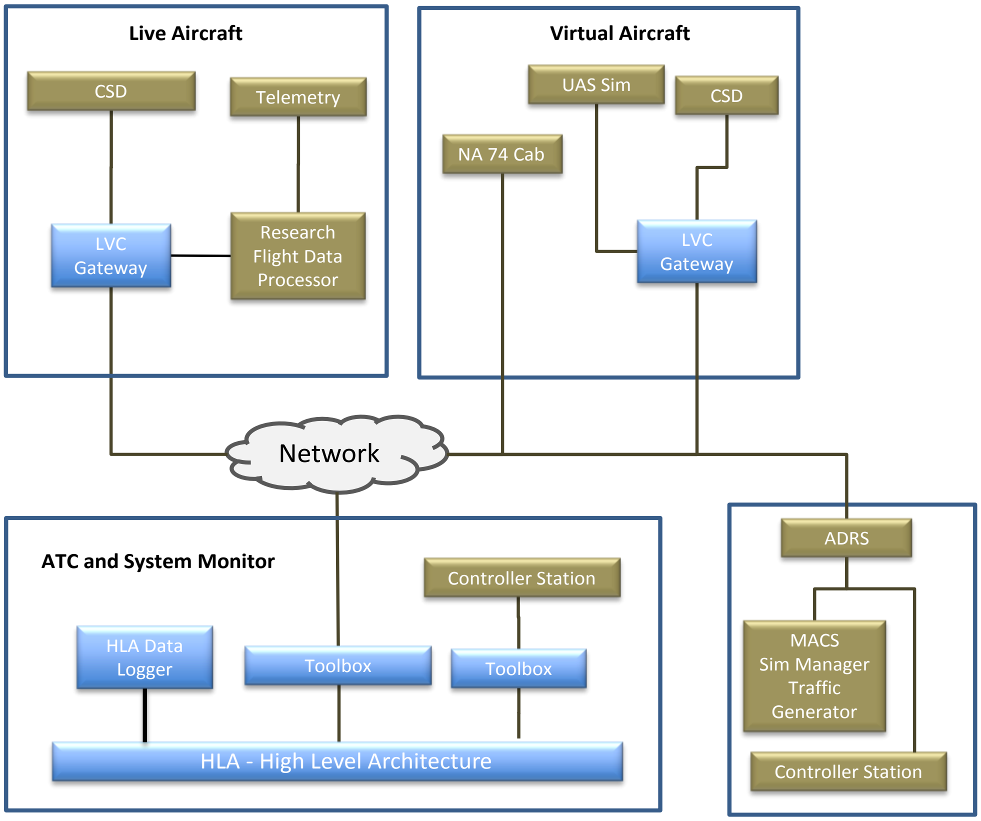


Figure 1. General System Architecture for UAS-NAS Baseline LVC Simulation

The proposed general system architecture shown in Figure 1, describes network connectivity between distributed subsystem participants for the live, virtual and constructive integrated test and evaluation in support of the UAS in the NAS Project.

The integrated LVC system configuration will connect the High Level Architecture (HLA) through the LVC Gateway Toolbox system component, the LVC Gateway, the LVC Gateway Data Logger and the SAA Processor (SaaProc). The HLA distributed environment will provide constructive traffic at the rate of 1Hz generated by the Multi-Aircraft Control System (MACS) in conjunction with Aeronautical Data link and Radar Simulator (ADRS) as depicted in Figure 1.

The Vigilant Spirit Control Station (VSCS) publishes its own simulated Flight State data to the LVC Gateway at a data rate of 10Hz. The fast rate VSCS ownship flight data will be transmitted through the LVC Gateway to the Cockpit Situation Display (CSD), SaaProc for conflict detection between the ownship and intruders, the LVC Gateway Toolbox and on to ATC display supported by MACS.

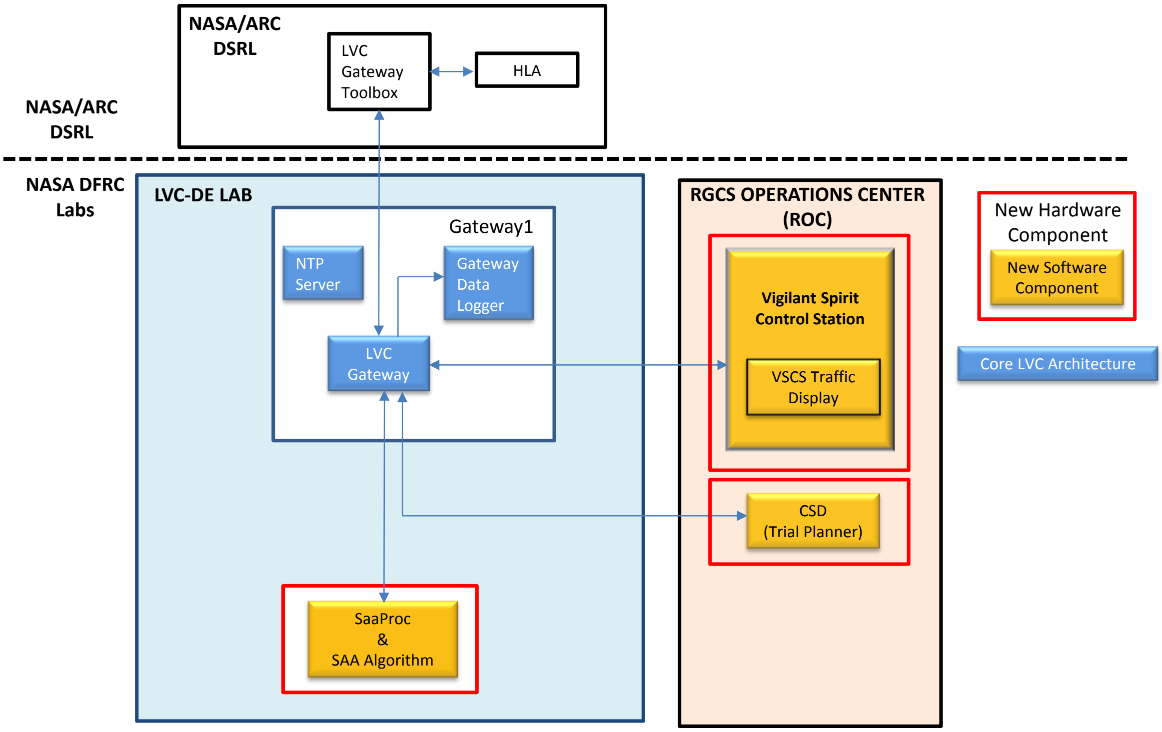


Figure 2. High Level System Architecture - Baseline LVC Gateway and Required Components

2. Applicable Documents

The following documents **(or later, earlier versions superseded)** form a part of this document to the extent specified herein. In the event of conflict between the documents referenced herein and the contents of this document, the contents of this document shall be considered a superseding requirement.

2.1. NASA Documents

IT&E CONOPS-01 Rev A	IT&E Concept of Operation Document
IT&E ORD-01 Rev A	IT&E Objective Requirements Document

2.2. References

LVC SWRD-02 Rev C	LVC Software Requirements Document
LVC SRD-01 Rev C	LVC System Requirements Document
RGCS SRD 01 Rev B	Research Ground Control Station

2.3. Standards

TCP/IP	Transmission Control Protocol / Internet Protocol (IPV4 /IPV6)

3. Definition of Messages Used in the LVC System

3.1. General Message Header

Every Multi-Purpose Interface (MPI) protocol message exchanged between different system components will have a header immediately followed by the payload of the corresponding message.

```
struct MsgHeader
{
   int   MsgTypeId;       // defined in table below
   int   MsgSize;         // header size + (payload size defined by each message structure)
   int   SrcId;           // each client is defined in Table 2
};
```

The message type defined in the header will indicate the type of message contained in the payload. This header is used for messages being passed between the LVC Gateway and the CSD, VSCS, ADRS and VIRTUAL UAS. Total size of the header structure is 12 bytes.

Table 1 identifies different message types that will be transmitted between different system components. Message data structure of corresponding message types are defined in tables below.

Table 1. Definitions of message types

Message Type	Id
MsgFlightState	5310
MsgFlightPlan	5201
MsgTrajectoryIntent	5421
MsgDeleteAc	5202
MsgHandshake	5960
MsgSetOwnship	5901
MsgFlightStateADSB	7010
MsgFlightStateTISB	7011
MsgHeartbeat	7030
MsgSaaThreatResults	5830
MsgSaaResManeuvers	5831
MsgSaaResReroute	5832
MsgSaaFlightState	5833
MsgSaaRelease	5834
MsgNavMode	5835
MsgTrialTrajectoryIntent	5454
MsgSaaTrialThreatResults	5839
MsgSaaRecapManeuver	5840
MsgTrialAccepted	5452
MsgStrwayBands	5841
MsgAcasxuRaTa	5842
MsgSaaBands	5843

For some tests or simulations, only a subset of the listed messages will be used. The data structure for each message and its representation is presented in sections below.

The client handshake message header contains Source parameter that defines the data source identification and names defined in the Table 2 below:

Table 2. Definitions of Client Names.

Source/Client Name	Id
CSD	101
IkhanaSim	102
IkhanaUAS	103
LangleyUAS	104
LVCDatalogger	105
UASRP	106
LVCGateway	107
GlennUAS	108
ADRS	109
SaaProc	111
VSCS	112
CPDS	113
StratwayGCS	114
ACASXU	115
ExelisNextGen	116

3.2 Handshake Data Structure

The Handshake Data structure is defined below. This structure represents the payload of a message sent by the client upon establishing the connection with the LVC Gateway (the server).

Note: The entire Handshake Data structure is continued on the next page.

```
struct MsgHandshake
{
  char clientName[12];            // Client name that is connecting to Gateway. See Table 2.
  char dataProviderName[12];   // callsign if ownship otherwise empty string

  bool b_publish_MsgFlightState;
  bool b_publish_MsgFlightPlan;
  bool b_publish_MsgTrajectory;
  bool b_publish_MsgFlightStateADSB;
  bool b_publish_MsgFlightStateTISB;

  bool b_subscribe_MsgFlightState;
  bool b_subscribe_MsgFlightPlan;
  bool b_subscribe_MsgTrajectory;
  bool b_subscribe_MsgFlightStateADSB;
  bool b_subscribe_MsgFlightStateTISB;
```

```
   bool b_publish_MsgDeleteAc;
   bool b_subscribe_MsgDeleteAc;
   bool b_publish_MsgSetOwnship;
   bool b_subscribe_MsgSetOwnship;

   bool b_publish_MsgSaaFlightState;
   bool b_subscribe_MsgSaaFlightState;

   bool b_publish_MsgSaaThreatResults;
   bool b_subscribe_MsgSaaThreatResults;

   bool b_publish_MsgSaaResManeuver;
   bool b_subscribe_MsgSaaResManeuver;

   bool b_publish_MsgSaaResReroute;
   bool b_subscribe_MsgSaaResReroute;

   bool b_publish_MsgSaaRelease;
   bool b_subscribe_MsgSaaRelease;

   bool b_publish_MsgTrialTrajIntent;
   bool b_subscribe_MsgTrialTrajIntent;

   bool b_publish_MsgSaaTrialThreatResults;
   bool b_subscribe_MsgSaaTrialThreatResults;

   bool b_publish_MsgSaaTrialRecapManeuver;
   bool b_subscribe_MsgSaaTrialRecapManeuver;

   bool b_publish_MsgTrialAccpeted;
   bool b_subscribe_MsgTrialAccpeted;

   bool b_publish_MsgNavigationMode;
   bool b_subscribe_MsgNavigationMode;
   bool b_publish_MsgSaaBands;
   bool b_subscribe_MsgSaaBands;

   bool b_publish_MsgAcasxu;
   bool b_subscribe_MsgAcasxu;

   bool b_publish_MsgStrwayBands;
   bool b_subscribe_MsgStrwayBands;

};
```

The role of the handshake message is twofold: 1) it is responsible for initiating the connection between the client and the server; 2) it registers the client with the server and sets up a publish/subscribe dependency.

For example, if the client that connects to LVC Gateway is CSD then the clientName is "CSD" as defined in Table 2. The dataProviderName attribute is set to a callsign of the ownship that provides ownship data for the CSD client. On the other hand, if the client is, for example a VSCS, then the clientName is set to "VSCS" while the dataProviderName is set to an empty string.

The client can publish and subscribe to certain data types specified by the Boolean attributes in the structure defined above. If the client is a VSCS entity then the first three Booleans (b_publish_MsgFlightState, b_publish_MsgFlightPlan, and b_publish_MsgTrajectory) will be set to true indicating to the LVC Gateway server that the client will publish its own flight state vector, flight plan and trajectory intent.

If the UAS is equipped with the ADS-B "In" capabilitythen the Boolean attribute b_publish_MsgFlightStateADSB will be set to true indicating that the client will publish the ADS_B state data of the surrounding traffic including its own. In that case, the flag m_equipageFlags in the MsgFlightState structure should be set by the publishing client to a value as defined in section 3.3. Note that m_equipageFlags is set to a zero for all other Flight State messages that are not generated using ADS-B and/or TIS-B tracks.

The Ikhana Sim will not consume external data. Hence all subscribe attributes will be set to false, indicating to the server that it should not send any of the traffic data to the Ikhana Sim client.

If the client is a CSD entity, then all the publish attributes should be set to false, indicating that the CSD is not publishing any data. However, the subscribe attributes will be selectively set to true or false depending upon what type of data the CSD has requested. Subscribe attributes b_subscribe_MsgFlightStateADSB and b_subscribe_MsgFlightStateTISB pertaining to FAA live traffic will be set to true, notifying the Gateway server that it should send all the Flight State ADS-B and the radar Flight State TIS-B data for background traffic. Note that m_equipageFlags structure field in the MsgFlightState structure will be set to defPasCiEqpADS_B and defPasCiEqpTIS_B (as defined in 3.3) for ADS-B and TIS-B tracks respectively. There will be cases when the two sets of targets, ADS-B and TIS-B, will contain common targets. Duplicate targets from different traffic sources will be filtered based on the criteria that will be devised in the future as needed.

3.3 Aircraft Flight State Data Structure

The Aircraft Flight State structure is defined below. This structure represents the payload of an aircraft flight state message.
Note that if some simulations do not generate some of the data fields defined in the message those values should be set to either -999999 for integers, -999999.0 for floats and doubles depending upon the variable type.

Data fields represented by strings will be published with the constant length as defined in the message interface by the ICD. If a string is shorter than the allocated space, blank spaces should be filled with "\0" (a null character). For example, "AAL123" should be represented as "AAL123\0\0\0\0\0\0" in a 12 character array.

```
struct MsgFlightState
{
   char        m_acid[ eMPI_ID_LENGTH=12];       // A/C callsign
   int         m_cid;                            // Computer generated A/C id
   double      m_timeCreated;                    // UTC time in decimal seconds decimal
   double      m_timeReceived;                   // UTC time in milliseconds
```

```
   double     m_latitude;                    // Decimal degrees signed +North/-South
   double     m_longitude;                   // Decimal degrees signed +East/-West
   float      m_pressureAltitude;            // Pressure altitude in feet
   float      m_geoAltitude;                 // not supported
   float      m_indicatedAirSpeed;           // Indicated airspeed in knots
   float      m_mach;                        // Current Mach number, non-dimensional
   float      m_bankAngle;                   // A/C bank angle in degrees
   float      m_pitchAngle;                  // A/C pitch angle in degrees
   float      m_groundSpeed;                 // A/C ground speed in knots
   float      m_verticalSpeed;               // A/C vertical speed in feet/min
   float      m_trueHeading;                 // A/C true heading in degrees based on
                                             //       true North
   float      m_magneticVariation;           // Magnetic variance degrees
   float      m_trueGroundTrack;             // A/C true ground track in degrees
   float      m_trueAirSpeed;                // Airspeed in wind frame in knots
   float      m_altitudeTarget;              // feet
   float      m_headingTarget;               // degrees
   float      m_speedTarget;                 // knots
   float      m_verticalSpeedTarget;         // not supported
   int        m_equipageFlags;               // used to set  ADS-B or TIS-B type of
                                             //  tracks
   int        m_modeFlags;                   // set to ADRS_MPI_FS_LNAV
   int        m_dlnkFlags;
   int        m_configurationFlags;
   float      m_flaps;
   float      m_speedBrakes;
   float      m_windDirection;               // degrees
   float      m_windSpeed;                   // knots
   float      m_outerAirTemperature;         // not supported
   float      m_mapRangeCaptain;
   float      m_mapRangeFo;
   float      m_headingBug;
   float      m_vhfFrequency;                // MHz
   int        m_beaconCode;                  // octal number
   int        m_geoSectorId;
   int        m_atcSectorId;
   int        m_acSectorId;
   char       m_atcSectorName[ eMPI_STRING_SECTOR=8 ];
   int        dummy4pack;
};
```

The dummy4pack field should be used to transmit ICAO code as there is no dedicated field for that attribute in any of the predefined structures.
Any int and float value that are undefined should be set to -99999.

Note that sign of longitude should follow the following convention. Westward longitude should have a negative value while Eastward should be positive~~.~~

Specifics of this message are explained below depending upon the value of the m_equipageFlag:

1. The MsgFlightState message is associated with any constructive, virtual or live non ADS-B and non TIS-B data source. Note that m_equipageFlag field in the message structure for this case will be set to zero.
2. The m_equipageFlag field will be set to a value defined in 3.3 corresponding to ADS-B track representing flight state vector for the live ADS-B equipped aircraft.
3. The m_equipageFlag field will be set to a value defined in 3.3 corresponding to TIS-B track representing flight state data for live aircraft that are not equipped with ADS-B.

The entity that is not equipped with ADS-B will publish flight state data where the m_equipageFlag field is set to zero. The entity that is equipped with ADS-B will publish flight data that map to the MsgFlightState structure with the m_equipageFlag field set to the value specified below that corresponds to the ADS-B data.

Equipage enum bit map Definitions
Two bitmaps for ADS-B and TIS-B equipage are defined below. They shall be used to set the m_equipageFlags in the **MsgFlightState** structure.

defPasCiEqpADS_B	0x00000400
defPasCiEqpTIS_B	0x00000800

The m_modeFlags field is set at least to ADRS_MPI_FS_LNAV in order for trajectory intent to show in the CSD. The m_modeFlags field should be set as a minimum to ADRS_MPI_FS_LNAV or to a value that is a result of a combination of different target flight statuses such as ADRS_MPI_FS_LNAV | ADRS_MPI_FS_VNAV | ADRS_MPI_FS_ARRIVAL | ADRS_MPI_FS_FREE_FLIGHT. The symbol "|" is a logical operation OR. The flight status types are defined below

```
typedef enum {
    ADRS_MPI_FS_UNKNOWN             = 0,
    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_IDENT_ON            = (1<<11),
    ADRS_MPI_FS_FREE_FLIGHT         = (1<<20),
    ADRS_MPI_FS_ATC_CONTROLLED = (1<<21),
    ADRS_MPI_FS_VFR                 = (1<<22),
    ADRS_MPI_FS_TFR                 = (1<<23),
    ADRS_MPI_FS_INACTIVE_INFLIGHT = (1<<24),
    ADRS_MPI_FS_PREDEPARTURE          = (1<<25),
    ADRS_MPI_FS_CDTI                  = (1<<30),
    ADRS_MPI_FS_COUNT
} adrs_mpi_flight_status_types;
```

This data type is defined in the adrs_mpi.h interface file provided to the user.

3.4 Aircraft Flight Plan Structure

The Aircraft Flight Plan structure is defined below. This structure represents the payload of an aircraft flight plan message. All messages displayed below are defined in the adrs_mpi.h interface used ADRS.

```
struct MsgFlightPlan
{
  int       m_dataSource;
  char      m_acid[ eMPI_ID_LENGTH=12];                  // aircraft callsign
  int       m_adrsProc;
  int       m_cid;                                       // computer id
  char      m_type[ eMPI_STRING_TYPE=16 ];               // aircraft type
  char      m_gateName[ eMPI_STRING_NAME=20 ];
  char      m_meterFixName[ eMPI_STRING_NAME=20 ];
  char      m_outerFixName[ eMPI_STRING_NAME=20 ];
  int       m_category;
  char      m_route[ eMPI_STRING_FILED_ROUTE=300 ];
  char      m_departureFix[ eMPI_ID_LENGTH=12 ];
  int       m_departureTime;                             // UTC time in seconds
  int       m_assignedAltitude;                          // feet
  float     m_filedSpeed;                                // knots
  int       m_timeEnroute;                               // seconds
  float     m_approachSpeed;                             // knots
  float     m_landingSpeed;                              // knots
  char      m_coordinationFrd[ eMPI_STRING_NAME=20 ];
  char      m_coordinationFix[ eMPI_STRING_NAME=20 ];
  float     m_coordinationX;                             // nautical miles
  float     m_coordinationY;                             // nautical miles
  int       m_faaCoordTime;                              // seconds
  int       m_coordinationTime;                          // UTC time in seconds
  char      m_destinationFix[ eMPI_STRING_NAME=20 ];
  char      m_destinationName[ eMPI_STRING_NAME=20 ];
  char      m_runwayName[ eMPI_STRING_NAME=20 ];
  int       m_configuration;
  int       m_beacon;                                    // A 4 digit number, each
                                                         //   digit is an octal value.
  char      m_atcType[ eMPI_STRING_TYPE=16 ];            // aircraft type
  int       m_timeReceived;                              // UTC time in seconds
  short     m_status;
  char      m_fpDataSource;
  char      m_equipmentAvailable;
  int       m_dlnkEquipped;
};
```

This is an example of a m_route field in the Flight Plan structure conforming to the standard FAA syntax: DFW.DALL7.LIT.J101.STL..CAP..BAYLI.BDF3.ORD.

Flight plans for constructive and/or live traffic will be published to LVC Gateway by the HLA via the LVC Gateway Portal component that is part of the HLA distributed environment. In addition, any constructive, virtual or live UAS entity connecting to the LVC Gateway will generate and publish

flight plan in the MsgFlightPlan format. The Gateway will transmit UAS flight plans to the LVC Gateway Portal and the HLA environment. The message field, m_adrsProc, should be set to the corresponding enum data type adrs_proc_type defined in the adrs_interface.h header file provide to the user. The m_adrsProc is set to ADRS_PROC_MPI_CLIENT_MACS = 35 if targets are generated by MACS while ADRS_PROC_MPI_CLIENT_VAST = 38 if targets are generated external to MACS, i.e. by a federate from the HLA distributed environment.
The m_status field as a minimum should be set to ADRS_MPI_FS_LNAV which corresponds to the bit field for lateral navigation management. CSD will not function nominally if m_status is set to a zero value.

3.5 Aircraft Flight Trajectory Intent Structure

The Aircraft Flight Trajectory Intent structure is defined below. It is a composite of two structures: 1) the trajectory specification structure, and 2) the waypoint structure. Both structures are defined below. The Trajectory Intent Structure represents the payload of an aircraft flight trajectory intent message.

```
struct MsgTrajectoryIntent
{
  MpiTrajSpec     m_spec;
  MpiTrajPoint    m_point[ eMPI_MAX_TRAJ_POINTS=50 ];
};
```

```
struct MpiTrajSpec
{
   char     m_acid[ eMPI_ID_LENGTH=12 ];     // aircraft sallsign
   int      m_adrsProc;
   int      m_cid;                           // computer id
   int      m_numberOfPoints;
   int      m_numberOfHorizPoints;
   float    m_climbSpeed;                    // Feet/min
   float    m_cruiseSpeed;                   // knots
   float    m_descentSpeed;                  // knots
   float    m_approachSpeed;                 // knots
   float    m_landingSpeed;                  // knots
   float    m_cruiseAltitude;                // Feet
   float    m_currentGrossWeight;            // not supported
   float    m_landingWeight;                 // not supported
   float    m_miscFloatValue;
   char     m_text[ eMPI_STRING_TRAJ=128 ];
};
```

```
struct MpiTrajPoint
{
   eMpiTrajPtType m_type;
   char     m_waypointId[ eMPI_ID_LENGTH=12 ];
   float    m_latitude;                      // decimal degrees signed +North/ -South
   float    m_longitude;                     // decimal degrees signed +East/-West
   float    m_turnRadius;                    // not supported
   int      m_miscIntValue;
   double   m_eta;                           // UTC seconds
   float    m_calibratedAirSpeed;
```

```
   float       m_altitude;                        // Feet
   float       m_fuelRemaining;                   // not supported
   float       m_outerAirTemperature;             // not supported
   float       m_windDirection;                   // TBD
   float       m_windSpeed;                       // TBD
   float       m_trueAirSpeed;                    // knots
   float       m_trueCourseIntoPoint;             // not supported by MACS
                                   // Note: used for "heading" Trial Planner
                                   // set to -999999.0 when not used
                                   // otherwise, set to the trial angle
   float       m_distanceToPoint;                 // (TBD: subject to computation:
                                                  //  MACS uses EntryTime)
   float       m_predictedGrossWeight;            // not supported by MACS
   float       m_x;                               // TBD
   float       m_y;                               // TBD
   int         m_constraint;
   float       m_miscFloatValue;
};
```

The MpiTrajectory of the constructive and/or live traffic is published by the HLA via the LVC Gateway Portal component to the LVC Gateway. The Gateway will publish the trajectory intent of any constructive, virtual or live UAS entity connecting to the LVC Gateway. Subsequently, MsgTrajectory messages associated with UAS entities will be transmitted to the LVC Gateway Portal and HLA environment. The message filed, m_adrsProc, should be set to the corresponding data source value defined in Table 2.

Note that sign of longitude should follow the following convention. Westward longitude should have a negative value while Eastward should be positive.

Enumeration below defines waypoint types in the **MpiTrajPoint** structure**.** The size of the enumeration field is 4 bytes.

```
enum eMpiTrajPtType
{
     eMPI_TRAJ_TYPE_WP = 0,     /* waypoint*/
     eMPI_TRAJ_TYPE_HP = 1,     /* holding pattern*/
     eMPI_TRAJ_TYPE_PH = 2,     /* proc hold*/
     eMPI_TRAJ_TYPE_PT = 3,     /* proc turn*/
     eMPI_TRAJ_TYPE_RF = 4,     /* rf leg*/
     eMPI_TRAJ_TYPE_TC = 5,     /* TOC*/
     eMPI_TRAJ_TYPE_TD = 6,     /* TOD   */
     eMPI_TRAJ_TYPE_SL = 7,     /* start of level*/
     eMPI_TRAJ_TYPE_CA = 8,     /* crossover altitude*/
     eMPI_TRAJ_TYPE_TA = 9,     /* transition altitude*/
     eMPI_TRAJ_TYPE_AC = 10,    /* Aircraft position  */
     eMPI_TRAJ_TYPE_CS = 11,    /* only constraint */
     eMPI_TRAJ_TYPE_RT = 12,    /* part of current rte*/
     eMPI_TRAJ_TYPE_AP = 13,    /* Airport DATA       */
     eMPI_TRAJ_TYPE_SC = 14     /* Speed Change Point */
};
```

This data type is defined in the adrs_trajectory.h interface file provided to the user.

3.6 Aircraft Delete Structure

The Aircraft Delete structure is defined below. This structure represents the payload of an aircraft delete message.

```
struct MsgDeleteAc
{
        char         m_acid[ eMPI_ID_LENGTH=12 ];
        int          m_adrsProc;
        int          m_cid;
};
```

This message is initiated by the Gateway clients Ikhana GCS, Ikhana Sim, Langley UAS or VIRTUAL UAS and will be sent to the LVC Gateway. An aircraft may drop out of the simulation environment due to a process crash, operational reasons (intentional shut down of the process) or during the debugging process. The LVC Gateway will send the MsgDeleteAc message to all clients that subscribe to the delete message. After the problem is addressed, the Ikhana, Ikhana Sim, Langley UAS, or VIRTUAL UAS can reconnect during the run time and continue participating in the simulation.

In addition, the delete message can be initiated by the HLA distributed environment when the aircraft from the background traffic drops out of the simulation. This event will generate delete message in the HLA environment which will be propagated throughout the entire distributed system informing the system components that the HLA aircraft is no longer active and that the local instance of the aircraft should be removed.

In case of the lost link between the Ikhana GCS and the Ikhana aircraft the delete message will be sent from the RFDP to the Gateway. During the lost link event either data sources, telemetry data provided by the GCS and ADS-B/TIS-B provided by the laptop will stop supplying data.

3.7 Set Ownship Structure

The Aircraft Ownship structure is defined below. This structure represents the payload of an aircraft set-ownship message.

```
struct MsgSetOwnship
{
        char            m_acid[ eMPI_ID_LENGTH=12 ];
        char            m_host[ eMPI_STRING_HOST=24 ];
        int             m_cid;
        int             m_control;
};
```

This message is used to inform a CSD system component about the target it is associated with. The CSD will initially provide the ownship callsign by the handshake message sent to the LVC Gateway. The ownship callsign is specified by the dataProividerName data field. Upon receiving handshake message, the Gateway will generate MsgSetOwnship message using the received callsign and the cid corresponding to the target with the specified callsign and will send it back to CSD.

3.8 Sense and Avoid (Saa) Aircraft Flight State Data Structure

Note that the new terminology for Sense and Avoid (SAA) has been introduced recently. SAA has been replaced by the Detect and Avoid (DAA) term. However, it has been decided to retain all the legacy references to SAA in all of the pertinent messages in this ICD. This preserves and maintains consistent terminology between the current ICD and the software that had been developed using the previous version of the ICD for the earlier phases of the UAS-in-the-NAS project.

The Saa Aircraft Flight State structure, **MsgSaaFlightState**, is defined in section 3.3. The Saa Aircraft Flight State message is a result of the sensor surveillance range filtering (part of the sensor model) applied to the entire simulated traffic (defined by the MsgFlightState message) that is received by the Sense and Avoid Process (SaaProc) from the LVC Gateway. Only the filtered traffic is visible by the surveillance system of the ownship aircraft. The MsgSaaFlightState is then published back to the LVC Gateway which in turn sends the data to the subscribing clients such as the Cockpit Situation Display (CSD) or the VSCS traffic display, depending upon which traffic display is active during the test event, to be displayed for the pilot’s situation awareness.

Note that if some simulations do not generate some of the data fields defined in the message those values should be set to either -99999 or to an empty string, depending upon the variable type. Message type is defined in Table 1.

3.9 Sense and Avoid (Saa) Threat Results Message

The Saa Threat Results Message data structure is defined below. It is a composite of two structures: 1) the threat specification data structure, and 2) the threat data structure. Both structures are defined below.

The Saa Threat Results Message Structure represents the payload comprised of array of SaaThreat data structures defined below.

```
struct MsgSaaThreatResults
{
    SaaThreatSpec   m_spec;
    SaaThreat       m_threats[SAA_MAX_THREATS=50];  // arbitrary, feel free to change
};
```

```
struct SaaThreatSpec
{
    char        m_acid[eMPI_ID_LENGTH=12];         // ownship callsign
    int         m_cid;                             // ownship flight number
    int         m_numberOfThreats;
};
```

Note that the eSaaType is type defined as an int, i.e. typedef int eSaaType.

```
struct SaaThreat
{
    eSaaType      m_saaType;                       // int - alert level
    int           m_intruderCid;
    double        m_conflictStartTime;             // UTC seconds
    double        m_conflictEndTime;               // UTC seconds
```

```
    double          m_conflictDuration;             // seconds
    double          m_timeToCpa;                    // seconds
    double          m_timeToFirstLoss;              // seconds
    double          m_dTauSimple;                   // range divided by range rate
    double          m_dTauModified;                 // range divided by range rate
    float           m_minHorzSep;                   // nm
    float           m_minVertSep;                   // feet

    double          m_ownshipCpaLat;                // degrees
    double          m_ownshipCpaLon;                // degrees
    double          m_intruderCpaLat;
    double          m_intruderCpaLon;
    double          m_ownshipFirstLossLat;
    double          m_ownshipFirstLossLon;
    double          m_intruderFirstLossLat;
    double          m_intruderFirstLossLon;

    float           m_ownshipCpaAlt;                // feet
    float           m_ownshipCpaGroundSpeed;        // knots
    float           m_ownshipCpaCalibratedAirSpeed; // indicated airspeed in knots
    float           m_ownshipCpaVerticalSpeed;      // feet/min
    float           m_ownshipCpaHeading;            // degrees

    float           m_intruderCpaAlt;
    float           m_intruderCpaGroundSpeed;
    float           m_intruderCpaCalibratedAirSpeed;
    float           m_intruderCpaVerticalSpeed;
    float           m_intruderCpaHeading;

    float           m_ownshipFirstLossAlt;          // feet
    float           m_ownshipFirstLossGroundSpeed;  // knots
    float           m_ownshipFirstLossCalibratedAirSpeed;// indicated airspeed in knots
    float           m_ownshipFirstLossVerticalSpeed; // feet/min
    float           m_ownshipFirstLossHeading;      // degrees

    float           m_intruderFirstLossAlt;         // feet
    float           m_intruderFirstLossGroundSpeed; // knots
    float           m_intruderFirstLossCalibratedAirSpeed;// indicated airspeed in knots
    float           m_intruderFirstLossVerticalSpeed; // feet/min
    float           m_intruderFirstLossHeading;     // degrees
   bool             m_isPredictedStricter           //true if alert is predicted to be
                                                    //stricter later in time, i.e. if SS
                                                    //alert and predicted to be CA later
    char            pad[7];
};
```

The definition of alert levels in the previous version of the ICD, LVC_ICD-03_REV_A, has been replaced by values defined in the table below.

Table 3. Definitions of Alert Levels.

0	no alert
1	proximate alert
2	self–separation preventive alert
3	self–separation alert
4	self–separation warning alert

The color scheme, symbology, and the threshold levels associated with the alert levels are presented in Appendix B.

3.10 Saa Release Structure

The SaaProc sends the SaaRelease message when the Sense And Avoid algorithm returns RELEASE as the threat state for the ownship. This indicates that the conflict has been cleared as a result of executing a previously advised maneuver.

```
typedef int eSaaType;   // 4 bytes long

struct MsgSaaRelease
{
    char        m_acid[eMPI_ID_LENGTH=12];          // ownship callsign
    int         m_cid;                              // ownship flight number
    eSaaType    m_saaType;                          // int - alert level
}
```

3.11 Sense and Avoid (Saa) Resolution Maneuver

The Saa Resolution Maneuver data structure is defined below. It is a composite of two structures: 1) the maneuver specification data structure, and 2) the maneuver data structure. Both structures are defined below.
The **MsgSaaResManeuver** structure describes the payload of a Saa Resolution Maneuver message.

```
struct MsgSaaResManeuver
{
  SaaResManeuverSpec          m_maneuverSpec,
  SaaManeuver                 m_maneuvers[SAA_MAX_MANEUVERS=20]
};

struct SaaResManeuverSpec
{
  char                  m_acid[eMPI_ID_LENGTH=12];         // ownship  callsign
  int                   m_ownshipCid;
  int                   m_numberOfManeuvers;
};

struct SaaManeuver
{
  eManeuverType         m_maneuverType;          // enum
  eSaaType              m_saaType;               // int – alert level
  double                m_startTime;             // UTC seconds
```

```
  double                 m_endTime;             // UTC seconds
  float                  m_altitude;            // not set if maneuver type != altitude
  float                  m_headingAbs;          // absolute heading in deg (0-359);
                                                // not set if maneuver type != heading
  float                  m_headingRel;          // relative heading in deg where +30
                                                // means 30 degrees right turn;not set  if
                                                // maneuver type != heading
  float                  m_speed;               // not set if maneuver type != speed
};

enum eManeuverType
{
  eSAA_MANEUVER_TYPE_REROUTE            = 0,
  eSAA_MANEUVER_TYPE_HEADING            = 1,
  eSAA_MANEUVER_TYPE_SPEED              = 2,
  eSAA_MANEUVER_TYPE_ALTITUDE           = 3,
  eSAA_MANEUVER_TYPE_COMPOUND           = 4,
  eSAA_MANEUVER_TYPE_APP_REFINED        = 5,
  eSAA_MANEUVER_TYPE_NOT_SET            = -999999 // same as INT_NOT_SET
};
```

3.12 Sense and Avoid (Saa) Resolution Reroute

The Saa Resolution Reroute data structure is defined below. It is a composite of two structures: 1) the resolution reroute specification data structure, and 2) the resolution waypoints data structure. Both structures are defined below. The **MsgSaaResReroute** structure represents the payload of an aircraft flight trajectory intent message.

```
struct MsgSaaResReroute
{
    SaaResRerouteSpec   m_rerouteSpec;
    SaaResWayponts      m_waypoints[MPI_MAX_NUM_OF_WAYPOINTS=50] ;
};

struct SaaResRerouteSpec
{
    char                 m_acid[eMPI_ID_LENGTH=12];      // ownship callsign
    int                  m_ownshipCid;
    eSaaType             m_saaType;                      // int – alert level
    double               m_startTime;                    // UTC seconds
    double               m_endTime      ;                // UTC seconds
    int                  m_numberOfWaypoints;
    double               m_turnOutAngle;                 // turn angle to the next
                                                         // fix from the current
                                // location; + right turn, - left turn in degrees
};

struct SaaResWaypoint
{
    char m_name[eMPI_ID_LENGTH=16];  // nav wpt name, or arbitrary if not
                                    // available
```

```
      double m_latitude;                    // decimal degrees
      double m_longitude;                   // decimal degrees
      float altitude;                       // above sea level in ft
      float speed;                          // true air speed in knots
};
```

3.13 NavigationMode Message Structure

The Navigation Mode Message is used whenever the ownship flight control system executes a maneuver or when the ownship consumes a waypoint on the route. The purpose is to send the SAA system intent information, so it can build an accurate trajectory prediction while detecting threats.

```
struct MsgNavMode
{
// Note: the three fields in the first group below are mandatory for
// all four Nav Modes including Flightplan, Autopilot mode, Override and Manual mode

// Flighthplan mode and Manual mode have only three fields shown below
//
      eNavMode      m_eNavMode;                       // enum
      char          m_acid[eMPI_ID_LENGTH=12];        // ownship callsign
      int           m_ownshipCid;

// Autopilot mode – set -999999.0 to the two fields if not autopilot mode
      float         m_heading;                        //degs. True North (absolute)
      float         m_altitude                        //feet

// Override mode - set -999999.0 to the four fields if not Override mode
      float         m_overrideAltitude                //feet
      float         m_tas                             //true airspeed in knots
      float         m_cas                             //calibrated airspeed in knots
      float         m_mach                      //NOTE: at least one speed must be set
};

enum eNavMode
{
   eNAV_MODE_FLIGHT_PLAN      = 0,
   eNAV_MODE_AUTO_PILOT       = 1,
   eNAV_MODE_OVERRIDE         = 2,
   eNAV_MODE_MANUAL           = 3,
   eNAV_MODE_NOT_SET          = -999999 // same as INT_NOT_SET
};
```

3.14 Trial Trajectory Intent Message

The Trial Trajectory Intent Message, **MsgTrialTrajectoryIntent**, is sent across the LVC system by CSD during the trial planning operation. Alternately, the VSCS traffic display contains a trial planning function that provides the same capability as the CSD. Only one traffic display may perform trial planning function during a simulation on one gateway. The interface between the VSCS trail planning function and the LVC Gateway utilizes the same MsgTrialTrajectoryIntent message. Trial planning messages will be sent at a 15Hz rate to LVC Gateway that will transmit those messages to SaaProc component for conflict assessment with intruders. The payload of the Saa Trial Trajectory Intent Message is the same data structure as the one defined by the Aircraft Flight Trajectory Intent Structure in section 3.5.

For clarity, the MsgTrialTrajectoryIntent message is shown below

```
struct MsgTrialTrajectoryIntent
{
  MpiTrajSpec       m_spec;
  MpiTrajPoint      m_point[ eMPI_MAX_TRAJ_POINTS=50 ];
};
```

where MpiTrajSpec and MpiTrajPoint are defined in section 3.5.

3.15 Trial Threat Results Message

Upon receiving the Trial Trajectory Intent message, the Saa algorithm will assess whether the well clear state of the ownship is violated against the surrounding traffic. If the well clear is violated the pilot will receive **MsgSaaTrialThreatResults** message which is the same data structure as the one defined by MsgSaaThreatResult message data structure as defined in section 3.9.

For clarity, the MsgSaaThreatResult message is defined below

```
struct MsgSaaTrialThreatResults
{
    SaaThreatSpec   m_spec;
    SaaThreat       m_threats[SAA_MAX_THREATS=50];  // arbitrary, feel free to change
};
```

where SaaTrajSpec and SaaThreat are defined in section 3.9.

3.16 Trial Recap Maneuver Message

TBD.

3.17 Trial Accepted Message

Pilot evaluates trial planned ownship trajectory by rubber-banding it across the CSD display. He selects trajectory that provides well clear condition for the ownship. After he negotiates heading and/or altitude maneuver to the first way point in the trajectory with the ATC controller, he presses the RAT (Route Assessment Tool) button on the CSD to send the Saa Trial Accept Message, **MsgTrialAccepted** data structure, to the VSCS via the LVC Gateway. The message payload is the selected trialed Trajectory Intent defined in section 3.14. It has been determined that this message will not be used at this time.

3.18 Release Message

When the SAA algorithm determines that SS or CA threat no longer exists, SaaProc generates the MpiReleaseMsg and sends it out to the LVC Gateway. The threat symbology is subsequently removed from the CSD or VSCS displays.

3.19 ACAS_Xu Data Structures

ACAS-Xu algorithm combining STM (Surveillance and Tracking Module) and TRM (Threat Resolution Module) modules will produce a TRM output given the intruder inputs in a prescribed format. The pertinent data structures and input requirements are defined in the ACAS-Xu documentation that is handled by the ACAS-Xu team. The AcasxuProc is a process that wraps the ACAS-Xu STM and TRM libraries and by utilizing the STM and TRM API calls the traffic input data generates the MsgAcasxu Ta and RA output message defined below.

```
#define ACASXU_MAX_INTRUDERS   20    // As defined by IT&E  team
#define PADSIZE_7BYTES          7    // for DOUBLEWORD alignment

typedef struct AcasxuTrmIntruderSpec
{
   int            m_numOfIntruders;          //      | 004 bytes | 004 bytes |
   int            m_numOfExpiredIntruders;   //      | 004 bytes | 008 bytes |
} AcasxuTrmIntruderSpecType;                 //      | total ----> 008 bytes |

typedef struct AcasxuTrmIntruderOut
{
   double         m_tds;        // track display score          | 008 bytes | 008 bytes |
   unsigned int   m_id;         // id of the intruder           | 004 bytes | 012 bytes |
   uint8          m_cvs;        // cancel vert complement       | 001 bytes | 013 bytes |
  uint8           m_vrc;        // vert resolution complement   | 001 bytes | 014 bytes |
  uint8           m_vsb;        // vert sense bit               | 001 bytes | 015 bytes |
  uint8           m_code;       // track code;                  | 001 bytes | 016 bytes |
} AcasxuTrmIntruderOutType;                                     | total ----> 016 bytes |

typedef struct MsgAcasxuTrmOut
{
   char       m_callsign[eMPI_ID_LENGTH]; // ownships allsign     | 012 bytes | 012 bytes |
   int        m_cid;                      // ownship cid          | 004 bytes | 016 bytes |
   TrmIntruderSpecType   m_intruderSpec;                          | 008 bytes | 024 bytes |
   double           m_target_rate;        // ft/s                 | 008 bytes | 032 bytes |
   double           m_dh_min;             // ft/s                 | 008 bytes | 040 bytes |
   double           m_ddh;                //                      | 008 bytes | 048 bytes |
   double           m_dh_max;             // ft/s                 | 008 bytes | 056 bytes |
   uint8            m_combined_control;          //               | 001 bytes | 057 bytes |
   uint8             m_vertical_control;         //               | 001 bytes | 058 bytes |
   uint8             m_up_advisory;              //               | 001 bytes | 059 bytes |
   uint8             m_down_advisory;            //               | 001 bytes | 060   bytes |
   bool            m_turn_off_aurals;            //               | 001 bytes | 061 bytes |
   bool            m_crossing;                   //               | 001 bytes | 062 bytes |
```

```
  bool                m_alarm;                        //      | 001 bytes | 063 bytes |
  bool                m_alert;            // TA active        | 001 bytes | 064 bytes |
  char                m_sensitivity_index;            //      | 001 bytes | 065 bytes |
  char                m_pad[PADSIZE_7BYTES];          //      |007 bytes | 072 bytes |
  TrmIntruderOutputType m_intruders[MAX_INTRUDERS];           | 160 bytes | 232 bytes |
  TrmIntruderOutputType m_expiredIntruders[MAX_INTRUDERS];//  | 160 bytes | 392 bytes |
}  MsgAcasxuTrmOutType;                                //     | total ----> 392 bytes|
```

The four fields m_combined_control, m_vertical_control, m_up_advisory, and m_down_advisory are described in the resolution advisory, RA, as defined in the ARINC 270 labels document attached in the Appendix A.

3.20 Stratway Bands Data Structure

The original Stratway Bands message is defined in the Stratway+ External Interface (Stratway+ ExternalInterface_Dec_22) ICD provided by LaRC team. The Stratway ICD is shown in Appendix C.

The LVC Gateway will receive the Stratway Bands message from the Stratway+ GCS. A UDP client/server multicast protocol is used to send/receive Stratway+ bands data. In this configuration, Stratway+ GCS socket is a server while LVC Gateway socket is a client. The detailed ICD for this interface is specified in the Stratway+ Interface Specification Document published by the NASA LaRC SSI team.

LVC Gateway will transmit the Stratway+ bands data to the subscribing clients based on the following Stratway+ Bands Message definitions.

```
#define STRWAY_MAX_INTERVALS         10
#define STRWAY_MAX_INTRUDERS         10
typedef char CharString8Type[8];   // 8 bytes long

enum eIntervalType{
    eSTRWAY_INTERVAL_TYPE_UNKNKOWN = 0,
    eSTRWAY_INTERVAL_TYPE_NONE = 1,
    eSTRWAY_INTERVAL_TYPE_NEAR = 2,
    eSTRWAY_INTERVAL_TYPE_RECOVERY = 3
 }

typedef struct StratwayInterval
{
   eIntervalType m_eIntervalType;          //       | 004 bytes | 004 bytes
   double m_low_interval;                  //       | 008 bytes | 012 bytes
   double m_up_interval;                   //       | 008 bytes | 020 bytes
} stStratwayIntervalType;                  //        total--->   | 020 bytes

typedef struct StratwayIntruder {
   CharString8Type m_callSign;                      | 008 bytes | 008 bytes
   eSaaType m_alertLevel;   // int - alert level    | 004 bytes | 012 bytes
} stStratwayIntruderType;                           | total--->  | 012 bytes
```

The alert level, m_eSaaType, can have any value between 0 and 4 as defined in Table 3. in section 3.9.

```
typedef stStratwayIntervalType
        StratwayIntervalListType [ STRWAY_MAX_INTERVALS ];// | 24 * 10 = 240 bytes
typedef stStratwayIntruderType
        StratwayIntruderListType [STRWAY_ MAX_INTRUDERS ];// | 12  * 10 = 120  bytes
```

The Stratway+ bands data message that is sent from MACS's External Interface Communications Thread consists of the following data members:

```
typedef struct MsgStrwayBandsMessage
{
  CharString8Type m_callSign;                          //  008 bytes | 008 bytes |
  double          m_timeSeconds;                       //  008 bytes | 016 bytes  |
  int             m_participantAddress;                //  004 bytes | 020 bytes  |
  int             m_numberOfHeadingIntervals;          //  004 bytes | 024 bytes  |
  StratwayIntervalListType m_headingIntervalList;      //  160 bytes | 184 bytes  |
  int             m_numberOfTrueAirSpeedIntervals;     //  004 bytes | 188 bytes  |
  int             m_pad1;                              //  004 bytes | 192 bytes  |
  StratwayIntervalListType m_trueAirSpeedIntervalList; //  160 bytes | 352 bytes  |
  int           m_numberOfVerticalSpeedIntervals;      //  004 bytes | 356 bytes  |
  int           m_pad2;                                //  004 bytes | 360 bytes  |
  StratwayIntervalListType m_verticalSpeedIntervalList; // 160 bytes | 520 bytes  |
  int            m_numberOfAltitudeIntervals;          // 004 bytes  | 524 bytes  |
  int            m_pad3;                               // 004 bytes  | 528 bytes  |
  StratwayIntervalListType m_altitudeIntervalList;     // 160 bytes  | 688 bytes  |
  int           m_numberOfIntruders;                   //  004 bytes | 692 bytes  |
  int           m_pad4;                                //  004 bytes | 696 bytes  |
  StratwayIntruderListType m_intrudersList;            //  120 bytes | 816 bytes  |
} MsgStrwayBandsType;                                  //  total---> | 816 bytes    |
```

The data structure presented above is applicable for both 32 bit and 64 bit applications since alignment is 8-byte double-word aligned. However, padding has to be introduced in the structure to enforce the alignment.

3.21 Stratway Clear Bands Data Structure

TBD

3.22 Omni (SAA) Band Message

Defines an interval with the same alert levels (i.e. PROX, SS, CA, or NONE as defined by eSaaType) throughout. The interval is defined as [min, max] inclusive.

The same structure (OmniBandInterval) will be used to represent heading and altitude bands in the OmniBand concept (see MsgSaaOmniBands). For heading OmniBandInterval, min and max indicate heading in degrees relative to current ownship heading, e.g. -30 is 30 degrees left of ownship's current heading, and +30 is 30 degrees right of ownship's current heading. For altitude bands, min and max values will always be set to the same value as altitude bands represent a single altitude level in feet above MSL.

A heading OmniBandInterval indicates that between min and max the band should be colored according to the associated alertLevel. For example, if min = -45, max = 0, and alertLevel=0, then the interval from 45 degrees left of ownship to its current heading should be painted green.

For altitude OmniBandInterval, min = max, so either can represent the altitude level to be shown in the altitude menu and the alertLevel describes the color of its outline in the menu. For example, if min and max=15000 and alertLevel=3, this means that the altitude menu will include 15000 feet entry, whose outline should be red meaning ownship would cause a loss of well-clear if it maneuvers to 15,000 ft.

```
#define JADEM_MAX_BAND_INTERVALS                    20

typedef struct OmniBandInterval
{
    eSaaType m_alertLevel;    // int - alertLevel;                   | 004 bytes | 004 bytes |
    int m_min;                //for heading – relative degrees,      | 004 bytes | 008 bytes |
                              // for altitude - feet above MSL
    int m_max;                //for heading – relative degrees,      | 004 bytes | 012 bytes |
                              // for altitude - feet above MSL
} OmniBandIntervalType;                                              | total = 20 bytes      |

typedef struct MsgSaaOmniBands
{
    char m_callsign[eMPI_ID_LENGTH];          // Onwship callsign   | 012 bytes | 012 |
     int   m_pad;                                                   | 004 bytes | 016 |
    double m_timeCreated;   // time msg created                     | 008bytes | 024 |

    // Number of heading and alt band intervals
     int numberOfHeadingIntervals;                                  | 004 bytes | 028 |
     int numberOfAltitudeIntervals;                                 | 004 bytes | 032 |

    // List of alerted/non-alerted intervals
    OmniBandInterval  headingIntervals[SAA_MAX_BAND_INTERVALS]; //| 400 bytes | 432 |
    OmniBandInterval  altitudeIntervals [SAA_MAX_BAND_INTERVALS]; //| 400 bytes | 832 |
} MsgSaaOmniBandsType;                                              | total = 832 bytes   |
```

3.23 CSD and VSCS Displays

CSD has the Basic and the Advanced mode for displaying SAA threat and resolution advisories. The Basic mode is set by entering values 0, 0 in the two text fields in the primary CSD UI display. Consequently, CSD publishes the Handshake message to LVC Gateway specifying the data to which it publishes/subscribes. In the Basic mode, CSD subscribes to: SAA Flight State of the background traffic (1Hz update rate), Flight State of the ownship (10Hz update rate), ownship Trajectory Intent (published when changed), SAA Threat Results (1Hz update rate), and SAA Release message (published when the threat is cleared). In Basic mode, CSD displays traffic icons but does not show any special alerting symbology beyond the imminent severity levels of traffic conflicts using white, yellow, and red colors. The Trial Planning tool is not enabled for the Basic mode.

The Advanced mode is set by entering 0, 2 in the same text fields. The Trial Planning tool is enabled for the Advanced mode. CSD publishes the Handshake message to LVC Gateway specifying the data to which it publishes/subscribes. In addition to what it subscribes to in Basic mode, CSD subscribes to: the SAA Resolution Maneuvers, SAA Resolution Reroute, and SAA Trial Threat Results, while it publishes the Trial Trajectory Intent message (15Hz update rate). Pilots may use the CSD trial planner, formally called the Route Assessment Tool (RAT), to make further refinements to route resolutions or provide a manual one from scratch. By pushing the RAT button and rubber-banding the current ownship trajectory, the Trial Trajectory Intent message (15Hz update rate) is published to LVC Gateway for processing by the SAA algorithm.

VSCS has three modes: None, Basic and Advanced. In the None mode, VSCS publishes ownship messages containing Flight Plan, Flight State and Trajectory Intent. VSCS does not subscribe to any messages form LVC Gateway. In the Basic mode, in addition to the publishing the same messages as in the None mode, VSCS subscribes to intruder SAA Flight State and following SAA related messages: SAA Threat Results, SAA Res Maneuvers, SAA Res Reroute, SAA Trial Threat Results, and SAA Release messages. The Trial Planning tool is not available while VSCS is in Basic mode but it is coupled with trial planning performed in CSD. In the Advanced mode, VSCS subscribes to intruder SAA Flight State, while it publishes Nav Mode message in addition to its Flight State, Flight Plan and Trajectory Intent messages. The Trial Planning tool is enabled when Trial Trajectory Intent messages (15Hz update rate) are published to LVC Gateway.

2.23.1 CSD Display

When the VSCS display is in the Basic mode, Self Separation (SS) alerts are accompanied with visual and aural alerts. Ownship and Intruder pop-up data tags will be displayed underneath the baseball card during a traffic alert, and a yellow halo will be displayed around the ownship. An aural alert “traffic, traffic” will be provided. When a Collision Avoidance (CA) alert is received, visual and aural alerts are provided to the pilot. Ownship and Intruder data tags will pop up (or stay up if already active) while a traffic alert will be displayed underneath the baseball card. A red halo will be displayed around the ownship and at the same time a directive aural alert will be given, e.g. “Climb, Climb”.

When the CSD display is in the Advanced mode, SS alerts are accompanied with visual and aural alerts. The recommended maneuver is shown in upper right corner**.** The *RES* button on the primary CSD UI will be highlighted if a new maneuver is available. Both the lateral and vertical trial planning tools are available for use at that time. The pilot will verify maneuver with the controller. After receiving ATC clearance, the pilot will execute the maneuver. When a Collision Avoidance (CA) threat is received, visual and aural alerts are provided to the pilot. The CA maneuver is shown in upper right corner and the trial planning tools are no longer enabled for use. The pilot will fly the first CA maneuver that is displayed.

2.23.2 VSCS Display

When the VSCS display is in the **Basic mode,** Self Separation alerts are accompanied with visual and aural alerts. Ownship and Intruder data tags will pop up when a traffic alert will be displayed underneath the baseball card and a yellow halo will be displayed around the ownship. An aural alert “traffic, traffic” will be given. When Collision Avoidance (CA) alert is received, visual and aural alerts are provided to the pilot. Ownship and Intruder data tags will pop up (or stay up if already active) while a traffic alert will be displayed underneath the baseball card. A red halo will be displayed around the ownship and at the same time a directive aural alert will be given, e.g. “traffic, traffic”.

When the VSCS display is in the **Advanced mode,** during Self Separation alerts pilots are provided with visual and aural alerts. The recommended maneuver is shown to the right of the baseball card. If multiple maneuvers are provided for the encounter, pilot will press the REFRESH button to view maneuvers**.** Both the lateral and vertical trial planning tools are available for use**.** Once the pilot has decided on an appropriate maneuver, he will negotiate the maneuver with the controller and if cleared he will press send button in the steering window. If the VSCS receives a Collision Avoidance alert, visual and aural alerts will be provided. The CA maneuver is shown to the right of the baseball card and the green arrow on the compass rose graphically depicts the CA maneuver**.** At that time, the trial planning tools are not available for use. The pilot must execute the CA maneuver by clicking the ‘Execute’ button.

2.24 Note about Heartbeat Message

Optionally, the Gateway shall periodically send heartbeat message to the clients with enumeration defined below.

The LVC Gateway will send a periodic heartbeat message at a configurable time interval to every client for the sole purpose of detecting whether the client socket port has been shut down, or closed. This infrastructure will detect a process that crashed and was running on the client connected to the Gateway. Upon detecting the closed socket, the Gateway will send MsgHeader message to every client unconditionally. The MsgHeader message requires no action, i.e. no response by recipients. A message of type MsgHeader of size 12 bytes shall contain the value of 7030 in the MsgType field according to the definitions in Table 1. The MsgSize field (i.e. sizeof(MsgHeader)) shall be set to 12 bytes which essentially means there is no subsequent payload. Therefore, there is no need for the recipient to read the socket port of any further payload data.

Clients that receive the MsgHeader message shall be expected to consume this message nominally. Consequently, if the LVC Gateway does in fact detect a closed socket port, then it will forward a delete aircraft message to all other active and valid subscribers. A MsgDeleteAc message shall be sent for each aircraft that was owned by the closed client.

2.25 Primitive Data Type Definitions and Sizes in Bytes

The "C" structures displayed above are used on a Windows platform using x86 or x86-64 architecture. The byte order for Windows platforms is little endian (the least significant byte is stored first) and the sizes of the primitive data types are given below:

- long: 4 bytes
- unsigned long: 4 bytes
- int: 4 bytes
- unsigned int: 4 bytes
- short int: 2 bytes
- unsigned short int: 2 bytes
- char: 1 byte
- float: 4 bytes

- double: 8 bytes

2.26 Byte Order and Need for Byte Swapping

All clients will publish messages in network byte order as computer networks transmit multi-byte numbers in this particular byte order. The most significant byte of a multi-byte number that is transmitted first over a network constitutes network byte order. Generally, different hosts (different CPUs) in the distributed environment can be little-endian or big-endian depending upon how bytes are ordered within a single word in the host memory. Therefore, when the little-endian host sends messages over the network it needs to convert (byte swap) them to network byte order before sending the messages out. Consequently, when the little-endian host receives a message over the network, it needs to convert the message back to host native byte representation, i.e. little-endian byte order.

Acronym List

ADRS – Aeronautical Data link and Radar Simulator
ADS-B - Automatic Dependent Surveillance-Broadcast
ATC – Air Traffic Control
FAA – Federal Aviation Administration
FIS-B - Flight Information Services-Broadcast
FLAPS – Flexible Acquisition Processing System
CSD - Cockpit Situation Display
GCS - Ground Control Station
HLA - High Level Architecture
LMA – Link Management Assembly
LVC - Live Virtual Constructive-Distributed Environment
MACS - Multi-Aircraft Control System
MPI –Multipurpose Protocol Interface
VIRTUAL UAS - Multi-UAS Simulator
NASA – National Aeronautics and Space Administration
SAA – Sense and Avoid
SaaProc – Sense And Avoidance Processor
TCP/IP - Transmission Control Protocol/Internet Protocol
TIS-B - Traffic Information Service-Broadcast
UAS-NAS - Unmanned Aircraft System-National Airspace System
UAT – Universal Access Transceiver
UTC - Coordinated Universal Time
VAST- HLA Virtual Airspace simulation Technology-High Level Architecture
VSCS - Vigilant Spirit Computer System

Appendix A

ARINC CHARACTERISTIC 735B - Page 103
ATTACHMENT 6
DATA WORDS APPLICABLE TO TRAFFIC COMPUTER (TCAS WITH ADS-B)

PART 6E
ARINC 429 CONTROL WORD –TCAS TO DISPLAY
TCAS Vertical Resolution Advisory RA Data Output Word
LABEL 270

BIT FUNCTION CODING NOTES
1 Label 1st Digit MSB 2 1
2 Label 1st Digit 0
3 Label 2nd Digit MSB 7 1
4 Label 2nd Digit 1
5 Label 2nd Digit 1
6 Label 3rd Digit MSB 0 0
7 Label 3rd Digit 0
8 Label 3rd Digit 0
9 SDI BIT 0
10 SDI BIT 1
11 Advisory 100 ft/min [9]
12 Rate to 200 ft/min
13 Maintain 400 ft/min
14 Binary Two's 800 ft/min
15 Complement 1600 ft/min
16 3200 ft/min
17 Sign
18 Combined Control
19 Combined Control [1]
20 Combined Control
21 Vertical Control
22 Vertical Control
23 Vertical Control [2]
24 Up Advisory
25 Up Advisory [3] [8]
26 Up Advisory
27 Down Advisory
28 Down Advisory [4] [8]
29 Down Advisory
30 SSM
31 SSM [5] [6] [7]
32 Parity (Odd)

1. Combined Control

BITS
20 19 18
MEANING
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
No Advisory

Clear of Conflict
Spare
Spare
Up Advisory Corrective
Down Advisory Corrective
Preventive
Not Used

2. Vertical Control

ARINC CHARACTERISTIC 735B - Page 104
ATTACHMENT 6
DATA WORDS APPLICABLE TO TRAFFIC COMPUTER (TCAS WITH ADS-B)
BITS
23 22 21
MEANING
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
Adv is not one of the
following types:
Crossing
Reversal
Increase
Maintain
Not Used
Not Used
Not Used

3. Up Advisory

BITS
26 25 24
MEANING
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
No Up Advisory
Climb
Don't Descend
Don't Descend >500
Don't Descend >1000
Don't Descend >2000
Not Used
Not Used

4. Down Advisory

BITS
29 28 27
MEANING
0 0 0

0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
No Down Advisory
Descend
Don't Climb
Don't Climb >500
Don't Climb >1000
Don't Climb >2000
Not Used
Not Used

5. Sign Status Matrix (SSM)(DISC)

BITS
31 30
MEANING
0 0
0 1
1 0
1 1
Normal Operation
No Computed Data
Functional Test
Failure Warning

6. The presence of a No Computed Data report in the SSM field indicates that the information in bits 11 through 29 is unreliable. Therefore, no RA should be issued by the Display.

7. The TCAS Computer should also set the SSM of this word to NCD when it is in STBY or TA Only mode (as reflected in the SL and RI fields of TX Word 2, label 274). Failure Warning should be reported in the SSM field only if the TCAS computer itself has failed. The presence of a Functional Test report in the SSM field of this word indicates that a TCAS Functional Test sequence should be performed by the displays. Refer to Section 4.2.

ARINC CHARACTERISTIC 735B - Page 105
ATTACHMENT 6
DATA WORDS APPLICABLE TO TRAFFIC COMPUTER (TCAS WITH ADS-B)

[8] Whenever “Climb” (Bits 24-26 = 1,0,0) or “Descend” (Bits 27-29 = 1,0,0) are set in Word 270, the TCAS computer sets the Advisory Rate Field (Bits 11-17) to the desired Climb/Descend value.

[9] If no RA is present, bits 11-17 should be set to zero.

ARINC CHARACTERISTIC 735B - Page 106
ATTACHMENT 6
DATA WORDS APPLICABLE TO TRAFFIC COMPUTER (TCAS WITH ADS-B)

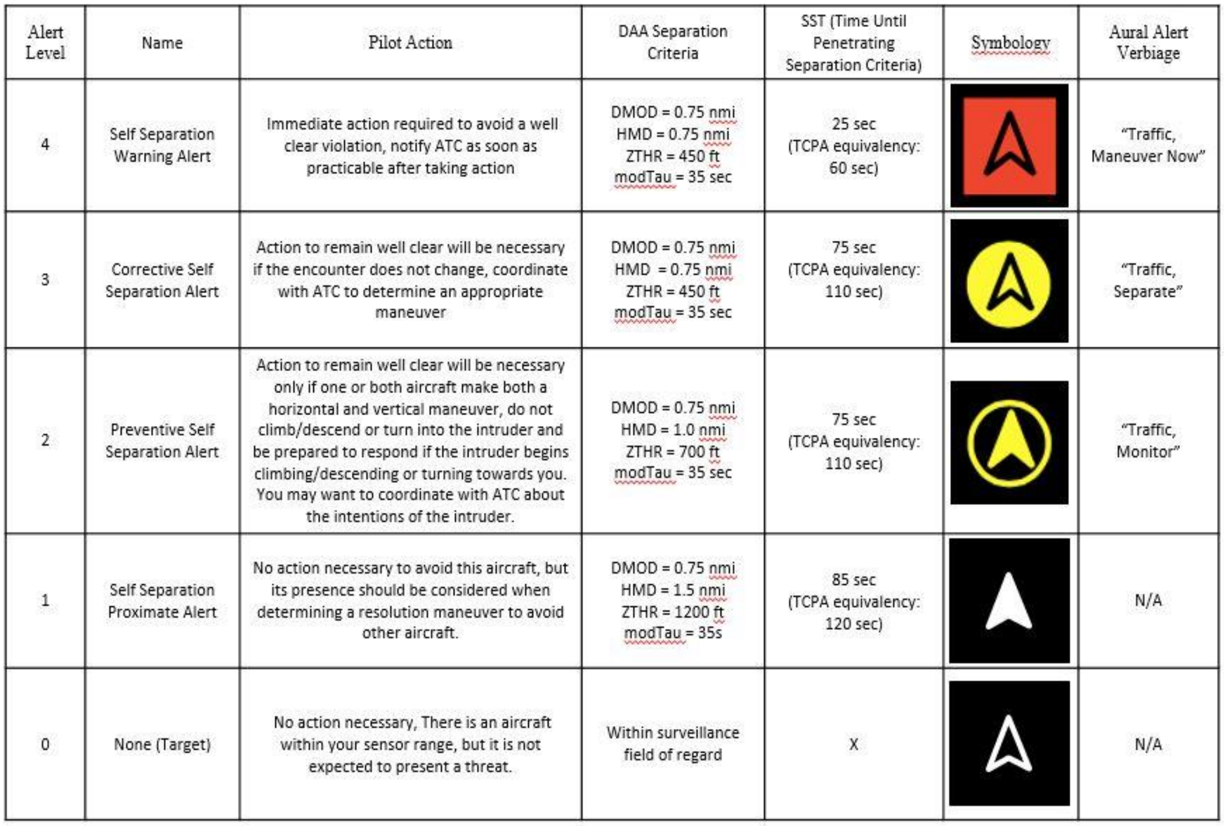
PART 6F

ARINC 429 CONTROL WORD –TCAS TO DISPLAY

TCAS Horizontal RA Data Output Word

INTENTIONALLY LEFT BLANK

Appendix B



Alert Level	Name	Pilot Action	DAA Separation Criteria	SST (Time Until Penetrating Separation Criteria)	Symbology	Aural Alert Verbiage
4	Self Separation Warning Alert	Immediate action required to avoid a well clear violation, notify ATC as soon as practicable after taking action	DMOD = 0.75 nmi HMD = 0.75 nmi ZTHR = 450 ft modTau = 35 sec	25 sec (TCPA equivalency: 60 sec)	🟥	"Traffic, Maneuver Now"
3	Corrective Self Separation Alert	Action to remain well clear will be necessary if the encounter does not change, coordinate with ATC to determine an appropriate maneuver	DMOD = 0.75 nmi HMD = 0.75 nmi ZTHR = 450 ft modTau = 35 sec	75 sec (TCPA equivalency: 110 sec)	🟡	"Traffic, Separate"
2	Preventive Self Separation Alert	Action to remain well clear will be necessary only if one or both aircraft make both a horizontal and vertical maneuver, do not climb/descend or turn into the intruder and be prepared to respond if the intruder begins climbing/descending or turning towards you. You may want to coordinate with ATC about the intentions of the intruder.	DMOD = 0.75 nmi HMD = 1.0 nmi ZTHR = 700 ft modTau = 35 sec	75 sec (TCPA equivalency: 110 sec)	◎	"Traffic, Monitor"
1	Self Separation Proximate Alert	No action necessary to avoid this aircraft, but its presence should be considered when determining a resolution maneuver to avoid other aircraft.	DMOD = 0.75 nmi HMD = 1.5 nmi ZTHR = 1200 ft modTau = 35s	85 sec (TCPA equivalency: 120 sec)	▲	N/A
0	None (Target)	No action necessary, There is an aircraft within your sensor range, but it is not expected to present a threat.	Within surveillance field of regard	X	△	N/A

THIS PAGE INTENTIONALLY LEFT BLANK