European Component Oriented Architecture (ECOA®) Collaboration Programme: Architecture Specification Part 6: ECOA® Logical Interface

BAE Ref No: IAWG-ECOA-TR-006
Dassault Ref No: DGT 144481-D

Issue: 4

Prepared by
BAE Systems (Operations) Limited and Dassault Aviation

Note: *This specification represents the output of a research programme and contains mature high-level concepts, though low-level mechanisms and interfaces remain under development and are subject to change. This standard of documentation is recommended as appropriate for limited lab-based evaluation only. Product development based on this standard of documentation is not recommended.*

Contents

Figures

Tables

0 Introduction

This Architecture Specification provides the specification for creating ECOA®-based systems. It describes the standardised programming interfaces and data-model that allow a developer to construct an ECOA®-based system. The details of the other documents comprising the rest of this Architecture Specification can be found in Section 3.

This document is Part 6 of the Architecture Specification, and describes the ECOA® Logical Interface (ELI), which covers ELI messages definition and the ELI to transport binding.

1 Scope

This Architecture Specification specifies a uniform method for design, development and integration of software systems using a component oriented approach.

2 Warning

This specification represents the output of a research programme and contains mature high-level concepts, though low-level mechanisms and interfaces remain under development and are subject to change. This standard of documentation is recommended as appropriate for limited lab-based evaluation only. Product development based on this standard of documentation is not recommended.

3 Normative References

Architecture Specification Part 1	IAWG-ECOA-TR-001 / DGT 144474 Issue 4 Architecture Specification Part 1 – Concepts
Architecture Specification Part 2	IAWG-ECOA-TR-012 / DGT 144487 Issue 4 Architecture Specification Part 2 – Definitions
Architecture Specification Part 3	IAWG-ECOA-TR-007 / DGT 144482 Issue 4 Architecture Specification Part 3 – Mechanisms
Architecture Specification Part 4	IAWG-ECOA-TR-010 / DGT 144485 Issue 4 Architecture Specification Part 4 – Software Interface
Architecture Specification Part 5	IAWG-ECOA-TR-008 / DGT 144483 Issue 4 Architecture Specification Part 5 – High Level Platform Requirements
Architecture Specification Part 6	IAWG-ECOA-TR-006 / DGT 144481 Issue 4 Architecture Specification Part 6 – ECOA® Logical Interface
Architecture Specification Part 7	IAWG-ECOA-TR-011 / DGT 144486 Issue 4 Architecture Specification Part 7 – Metamodel
Architecture Specification Part 8	IAWG-ECOA-TR-004 / DGT 144477 Issue 4 Architecture Specification Part 8 – C Language Binding
Architecture Specification Part 9	IAWG-ECOA-TR-005 / DGT 144478 Issue 4 Architecture Specification Part 9 – C++ Language Binding

Architecture Specification Part 10	IAWG-ECOA-TR-003 / DGT 144476 Issue 4 Architecture Specification Part 10 – Ada Language Binding
ISO/IEC 8652:1995(E) with COR.1:2000	Ada95 Reference Manual Issue 1
ISO/IEC 9899:1999(E)	Programming Languages – C
ISO/IEC 14882:2003(E)	Programming Languages C++

4 Definitions

For the purpose of this standard, the definitions given in Architecture Specification Part 2 apply.

5 Abbreviations

DDS	Data Distribution Service
ECOA	European Component Oriented Architecture. ECOA® is a registered trademark.
ELI	ECOA® Logical Interface
EUID	ECOA® Unique Identifier (ID)
ID	Identifier
IP	Internet Protocol
NaN	Not a Number
POSIX	Portable Operating System Interface
SCA	Service Component Architecture
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
XML	eXtensible Markup Language
XSD	XML Schema Definition

6 Inter-Platform Communications

ECOA platforms communicate using the ECOA Logical Interface (ELI) message definition. This definition is generic and is independent of the underlying transport mechanism. Therefore, ELI messages can be considered as the payload of the underlying transport mechanism and the ELI will not provide mechanisms generally provided by a transport protocol.

Platforms will use a transport binding to carry ELI messages and it is responsible for transporting messages to the appropriate destinations. Several network bindings will be defined in order to support different network transport protocols (i.e. UDP, TCP, etc.). Those network bindings have been designed to be completely independent from ELI Messages. Those bindings may provide mechanisms to add robustness if the underlying transport protocol is not enough robust to meet system-level requirements (e.g. reliability, integrity, ordering, confidentiality, etc.).

ELI Messages have been defined to carry information about service operations or platform-level management data.

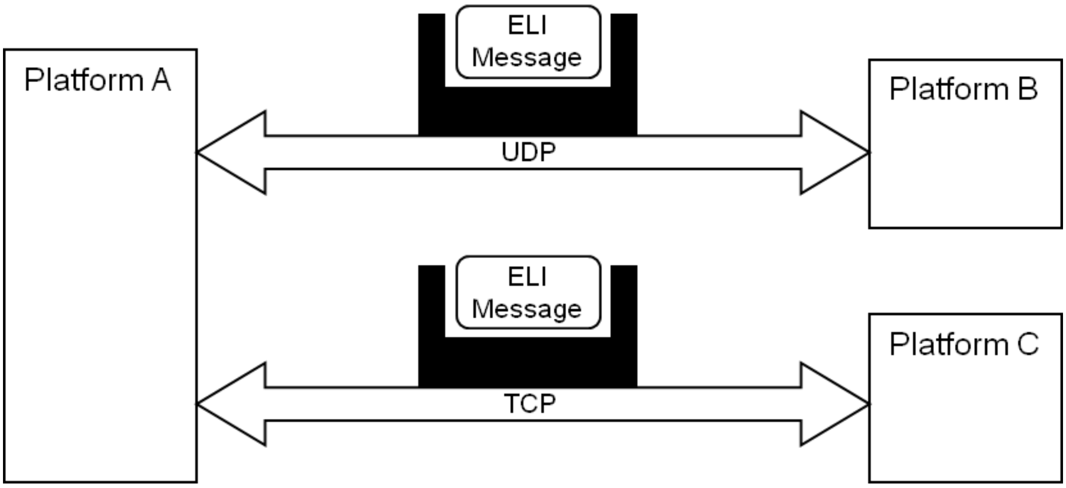

Figure 1 Example of inter-platform communications

Any ELI Message received containing a RESERVED value for any given field shall be discarded by the receiving platform. This shall be logged in the security log if required, as it may be indicative of an attempt to carry data in an undetected channel.

NOTE The ELI protocol does not cover any system requirements for time synchronisation. If this is required, then it must be provided by other means.

NOTE Data structures and data messages are independent of the binding languages (C, C++, ADA).

NOTE Having the same version of the ELI on both sides is a necessary, but not sufficient, condition for achieving a correct behaviour. To guarantee this one shall have the same version of the ECOA standard on both sides.

6.1 ELI Message Format

ELI messages have a standard structure that includes a generic message header required to route all messages, and a message specific payload that depends on the actual message type itself.

All fields within the header and payload are big endian byte order.

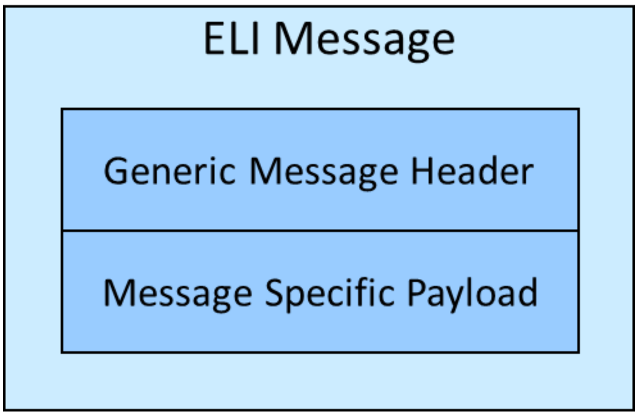

Figure 2 ELI Message Format

Table 1 ELI Message Format

Header	Value	Explanation	Length (bits)	Alignment (bits)
Generic Message Header	24 byte header	Generic message header applicable to all ELI messages	192	32
Message Specific Payload	Payload	Message specific payload dependent upon the message type	Payload Size * 8	32

6.1.1 Generic Message Header

The generic header includes:

- an ECOA mark to allow the identification of ELI messages (0xEC0A)
- a version number related to the ELI version of messages (1 in this version)
- a domain to identify the type of an ELI message (platform-level management or service operation)
- a unique ID identifying the logical platform that has sent the message
- a unique ID defining the platform-level message, or service operation message
- a timestamp
- a payload size
- a sequence number used to associate platform-level messages or request/response operations

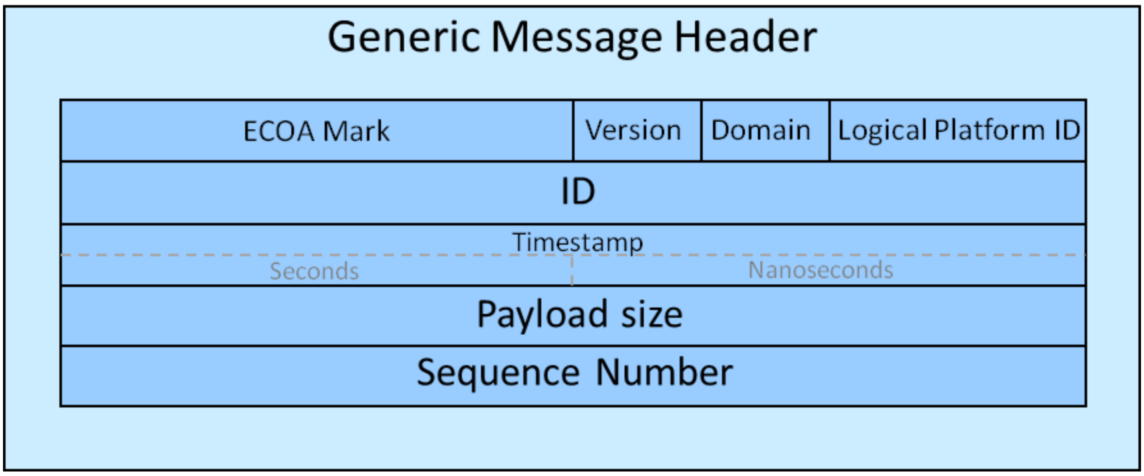

Figure 3 Generic Message Header

Each item within the generic message header is detailed in Table 2.

Table 2 Generic Message Header

Header	Value	Explanation	Length (bits)	Alignment (bits)
ECOA Mark	0xEC0A	Mark to identify the message as an ECOA message	16	16
Version	Unsigned number (1 for this version)	ELI version	4	4
Domain	0 - Platform-level Management 1 - Service Operations 2-15 - Reserved	ELI functional domain of the message : platform-level management | service operation	4	4
Logical Platform ID	Unsigned number	Sender Logical Platform ID – Unique ID within the system used to identify the sender of the message	8	8
ID	ID of the platform-level message if domain = 0 EUID of the service operation if domain = 1	Unique ID allowing routing of the message from the client to the server - and potentially the routing of a reply	32	32

Header	Value		Explanation	Length (bits)	Alignment (bits)
Timestamp	Seconds	Unsigned number	Global time of the emitter If Domain = 1 then the timestamp is set as near as possible to the time of invocation of the container operation by the module. See Table 1 in Architecture Specification Part 3. If Domain = 0 then it is the point at which the platform generates the request or response. Reference point in time : 1st January of 1970 (POSIX epoch valid until 2106)	32	32
	Nanoseconds	Unsigned number 0-999999999. 1000000000-2^32 Reserved		32	32
Payload Size	Unsigned number		Size of the payload in bytes	32	32
Sequence Number	Unsigned number		Sequence number assigned by the client container to allow association between a request and the reply. When the sequence number is used to associate a service operation or platform request response it shall take a value in the range 0x00000001..0xFFFFFFFF. When the value is unused it shall take the value 0.	32	32

Messages where the Payload Size declared in the header is different from the actual size of the payload shall be discarded by the receiving platform.

The ELI Binding Header bit format shall be as shown below:

```
       76543210
Byte 1 EEEECCCC
EEEE: 0xE (4 bits) | CCCC: 0xC (4 bits)

Byte 2 0000AAAA
0000: 0x0 (4 bits) | AAAA: 0xA (4 bits)

Byte 3 VERSDOMM
VERS: version number (4 bits) | DOMM: Domain (4 bits)

Byte 4 LOGICALP
LOGICALP: Logical Platform ID (8 bits)

Byte 5 IDMSBYTE
IDMSBYTE: ID Most significant byte (8 bits)
```

```
Byte 6 IDSSBYTE
IDSSBYTE: ID 2nd Most significant byte (8 bits)

Byte 7 IDTSBYTE
IDTSBYTE: ID 3rd Most significant byte (8 bits)

Byte 8 IDLSBYTE
IDLSBYTE: ID Least significant byte (8 bits)

Byte 9 TSMSBYTE
TSMSBYTE: Timestamp Seconds Most significant byte (8 bits)

Byte 10 TSSSBYTE
TSSSBYTE: Timestamp Seconds 2nd Most significant byte (8 bits)

Byte 11 TSTSBYTE
TSTSBYTE: Timestamp Seconds 3rd Most significant byte (8 bits)

Byte 12 TSLSBYTE
TSLSBYTE: Timestamp Seconds Least significant byte (8 bits)

Byte 13 TNMSBYTE
TNMSBYTE: Timestamp Nanoseconds Most significant byte (8 bits)

Byte 14 TNSSBYTE
TNSSBYTE: Timestamp Nanoseconds 2nd Most significant byte (8 bits)

Byte 15 TNTSBYTE
TNTSBYTE: Timestamp Nanoseconds 3rd Most significant byte (8 bits)

Byte 16 TNLSBYTE
TNLSBYTE: Timestamp Nanoseconds Least significant byte (8 bits)

Byte 17 PSMSBYTE
PSMSBYTE: Payload Size Most significant byte (8 bits)

Byte 18 PSSSBYTE
PSSSBYTE: Payload Size 2nd Most significant byte (8 bits)

Byte 19 PSTSBYTE
PSTSBYTE: Payload Size 3rd Most significant byte (8 bits)

Byte 20 PSLSBYTE
PSLSBYTE: Payload Size Least significant byte (8 bits)

Byte 21 SNMSBYTE
SNMSBYTE: Sequence Number Most significant byte (8 bits)

Byte 22 SNSSBYTE
SNSSBYTE: Sequence Number 2nd Most significant byte (8 bits)

Byte 23 SNTSBYTE
SNTSBYTE: Sequence Number 3rd Most significant byte (8 bits)

Byte 24 SNLSBYTE
SNLSBYTE: Sequence Number Least significant byte (8 bits)
```

Bytes are described from the most significant bit (7) to the least (0).

The header is sent in the following byte order: byte 1 then byte 2 then byte 3 then byte 4,,,…,byte 24.

6.1.1.1 Platform Level Message IDs

Platform-level management message IDs (Domain=0) are defined in Table 3.

Table 3 Platform-level ELI message IDs

ID	Message Type	Explanation
0x00000001	PLATFORM_STATUS	Used to push the new status of a platform or to reply to a platform status request
0x00000002	PLATFORM_STATUS_REQUEST	Used to request the status of the platform
0x00000003	AVAILABILITY_STATUS	Used to push the availability state of services provided by the platform or to reply to an availability status request
0x00000004	AVAILABILITY_STATUS_REQUEST	Used to request the availability state of one or all services provided by the platform
0x00000005	UNKNOWN_OPERATION	Used when a requested operation is not accessible on the platform
0x00000006	SERVICE_NOT_AVAILABLE	Used when a requested service is not set as available on the server platform
0x00000007	VERSIONED_DATA_PULL	Used to pull one or all versioned data available in provided service instances.
0x00000008	COMPOSITE_CHANGE_REQUEST	PROVISIONAL: Used to request the load of a new composite on the platform
0x00000009	COMPOSITE_CHANGE_REQUEST_ACK	PROVISIONAL: Used to confirm that the platform can satisfy the composite change request
0x0000000A to 0xFFFFFFFF	RESERVED	Reserved

Messages where the ID is set to a RESERVED value shall be discarded by the receiving platform.

6.1.1.2 Service Operation Message IDs

Service Operation message IDs (Domain=1) are defined by an ECOA Unique ID (EUID).

A EUID is generated from a key created by using the following string:

"[SourceComponentInstanceName]/[SourceServiceInstanceName]:[DestinationComponentInstanceName]/[DestinationServiceInstanceName]:[ServiceOperationName]"

All EUIDs need to be generated at integration time with the same method in order to have uniqueness of all IDs across the system.

The association between EUID and a specific pair of component instances / service instances / operation is defined in a dedicated table stored in an XML file who's XSD is defined by the ECOA Metamodel (Architecture Specification Part 7).

The platform will use this information to route the message from the source component instance to the target module instance of a component instance, and potentially to route any reply.

6.1.2 Message Specific Payload

6.1.2.1 Message Specific Payload for Platform-Level Management Domain

A message for platform-level management operations contains the parameters for the platform message.

The platform message is defined by the ID parameter in the generic message header when the domain=0. Table 4 defines the payload content dependent upon the ID from Table 3.

Table 4 Payload details for Platform-level Management Messages

Message type	Fields	Sub-fields	Value	Explanation	Length (bits)	Alignment (bits)
PLATFORM_STATUS	Status		0x00000000 – DOWN 0x00000001 – UP 0x00000002 to 0xFFFFFF - RESERVED	State of the platform	32	32
	Composite ID		Number	EUID of the composite loaded on the platform	32	32
PLATFORM_STATUS_REQUEST	No fields					
AVAILABILITY_STATUS	Provided Services		Number	Number of Provided Services for which this message gives the availability state	32	32
	Service Availability List			List containing pairs of elements for each service instance provided by the platform. The number of pairs is given by the previous field	64	32
		Service ID	Number	EUID of the service instance provided by a given component instance on the platform sending this message	32	32
		Availability State	0x00000000 – UNAVAILABLE 0x00000001 – AVAILABLE 0x00000002 to 0xFFFFFF - RESERVED	State of the service identified in the previous field	32	32
AVAILABILITY_STATUS_REQUEST	Service ID		Number 0xFFFFFFFF to request all service availability states	EUID of the service instance provided by a given component instance on the platform receiving the request	32	32

Message type	Fields	Sub-fields	Value	Explanation	Length (bits)	Alignment (bits)
COMPOSITE_CHANGE_REQUEST	Composite ID		Number	EUID of the composite to load on the platform	32	32
COMPOSITE_CHANGE_REQUEST_ACK	Status		0x00000000 – DISAGREE 0x00000001 – AGREE 0x00000002 to 0xFFFFFF – RESERVED	When DISAGREE is returned, the platform cannot change the requested composite. When AGREE is returned, the platform will load the requested composite	32	32
VERSIONED_DATA_PULL	EUID		Number 0xFFFFFFFF to pull all versioned data	EUID of the requested versioned data – see service operations	32	32
UNKNOWN_OPERATION	EUID		Number	EUID of the requested operation	32	32
SERVICE_NOT_AVAILABLE	EUID		Number	EUID of the requested operation	32	32

The following provides additional details related to the above table:

- At the end of a start or a reconfiguration, the platform state becomes UP once all modules are at least in IDLE state and the platform is ready to receive any ELI message, in particular versioned data.
- A platform becomes DOWN as soon as it receives a COMPOSITE_CHANGE_REQUEST or the old composite has been stopped by other means: the platform is no longer in a position to manage modules and their dependencies (local copies of versioned data). When DOWN, all services provided by the platform become unavailable.
- A platform can send a DOWN status as long as the composite is not loaded.
- The PLATFORM_STATUS can be sent periodically as a heartbeat to enable active monitoring between platforms. The composite ID provided in this message allows the receiver to check that the sender and the receiver are running the same global composite; the composite is global to an ECOA system.
- The EUID of a composite is the ID generated from a key created with the name of the composite (attribute ‘name’ of the root element of the actual implemented assembly schema file). It is up to the system integrator to adequately manage the configuration management of assembly schemas.
- If the field “Provided Services” in the AVAILABILITY_STATUS message is zero-valued, it means that the platform does not provide services to other platforms and there is no service availability data in the message. If the field “Provided Services” is non zero-valued, it indicates the number of service availability data (pair of service ID and associated service state) in the remaining part of the payload.
- The EUID of a service instance is the ID generated from a key created by the concatenation of the component instance name, the character ‘/’ and the provided service instance name: ‘component_instance_name/provided_service_instance_name’.
- When the AVAILABILITY_STATUS message is received by a platform, it may only contain information for a subset of the services provided by the remote platform. This partial information does not invalidate the locally known availability states for the other provided services. If the platform needs to know the current state of these services, it may send the remote platform a global request (for all services) or multiple requests (one per service).
- When a COMPOSITE_CHANGE_REQUEST is sent to a platform, the platform sends back a COMPOSITE_CHANGE_REQUEST_ACK with the appropriate value. Then the platform state becomes DOWN, the platform sends a PLATFORM_STATUS with the DOWN value and all the services provided outside of the platform are considered as UNAVAILABLE until the new composite has been loaded and has totally replaced the old one. When the new composite is successfully loaded, the platform becomes UP and sends a PLATFORM_STATUS with the UP value.
- When a VERSIONED_DATA_PULL is sent to a platform, the platform sends the versioned data using the normal service operation messages (see section 6.1.2.2). Only the versioned data required to be published to that platform, as defined in the assembly schema, will be sent.
- UNKNOWN_OPERATION is returned by a platform when the requested operation (pull of a given versioned data or request-response) is not available on the platform.
- If an AVAILABILITY_STATUS_REQUEST with value 0xFFFFFFFF (all services) is received by a platform, then the platform will respond only to the requester, but with the availability states of all services it can provide (irrespective of whether the requesting platform requires that service as defined in the assembly schema or not
- If a versioned data state is requested by a platform, and that state has never been published (it is uninitialized), then the platform will respond with a versioned data message whose size is zero.

6.1.2.2 Message Specific Payload for Service Operations

The message specific payload for service operations contains the operation parameters for the identified service operation.

The service operation message is identified by the ID parameter (EUID) in the generic message header when the domain=1.

Table 5 details the content of the payload based upon the type of service operation parameters.

Table 5 Payload details for Service Operations Messages

Header	Sub-header	Value	Explanation	Length (bits)	Alignment (bits)
Payload			Service operation dependent data: • Input data if operation is event or request • Output data if operation is reply (reply, deferred_reply) • data if operation is versioned data Data is in the order of the service definition from left to right	Payload Size * 8	32

The alignment is mainly used for the start of the Payload; the actual number of bytes sent onto the network is 'Payload size' bytes. It is recommended that ELI implementations zeroised possible padding in the buffers where they copy Payloads.

In order for two separate executables to marshal and unmarshal the service operation payload, each element of the message will need to conform to a standard for sizing and alignment.

Each element of the payload will be an ECOA predefined base type or a compound type constructed from one or more ECOA predefined base types.

Providing size and alignment rules for each of the predefined base types and compound types will enable two separate executables to marshal and unmarshal any service operation payload.

Table 6 identifies the byte serialization requirements for the predefined base types.

Table 6 Byte Serialization Requirements for ECOA Predefined Base Types

Header	Serialization	Length (bits)	Alignment (bits)
boolean8	endianness not applicable 0 : false, 1-255 : true	8	8
int8	endianness not applicable - two's complement notation	8	8
char8	endianness not applicable ASCII	8	8
int16	big endian - two's complement notation	16	8
int32	big endian - two's complement notation	32	8
int64	big endian - two's complement notation	64	8
uint8	endianness not applicable	8	8
byte	endianness not applicable	8	8
uint16	big endian	16	8
uint32	big endian	32	8
uint64	big endian	64	8
float32	big endian - cope with IEEE 754 - Do not transmit NaN and infinity values	32	8
double64	big endian - cope with IEEE 754 - Do not transmit NaN and infinity values	64	8

Compound types will be sized and aligned according to the rules in Table 7.

Table 7 Compound Types Sizing and Alignment Requirements

Header	Sub-header	Value	Explanation	Length (bits)	Alignment (bits)
array	size	number	Number of elements of the array	32	8
	data		Array data	array size * element type size	8
fixed array			Array data	Size of the array in bits (size in bytes * 8) according to the number of elements and their types	8
enum			low-level big endian value : ordinal value of the enum, starting at 0, if no mapping. else, transmit the mapped value	Size of the enum type	8

Header	Sub-header	Value	Explanation	Length (bits)	Alignment (bits)
record			The order of the constituents is given by the XML definition.	Sized according to its constituents	8
variant record	selector		To select the right record	Size of the selector type	8
	data		The selected record	Size of the selected record - variable	8

NOTE It is not necessary to define size fields for array data items, record fields or variant record fields as the receiving platform knows the type of all incoming data at start time. Their sizes are derived from the XML translation into the ELI binding.

6.2 Transport Bindings

It is possible to transport the generic ELI messages using a variety of different transport mechanisms. Examples of these transport mechanisms include UDP/IP, TCP/IP, MIL-Std 1553B, DDS, etc.

In term of OSI layers, a transport layer fulfils robustness requirements, such as integrity, loss of messages, confidentiality, etc. If the selected transport layer does not fulfil all system-level requirements in term of robustness, the binding shall contain mechanisms to support those requirements.

An example binding to UDP/IP is described in Annex A.

NOTE The routing of ELI messages is outside of the scope of this document, and is a system specific issue.

6.3 Platform Start-up

In order to allow platforms to start-up in any order, a defined behaviour is required which uses the Platform and Service Operation ELI messages in consistent ways across all platforms.

This section defines a set of behaviours that are an initial proposal for use when developing platforms. It is seen as a way of allowing a platform to start-up and acquire the state of any other platforms' services and versioned data, whilst providing the state of its services and versioned data to other platforms.

The following behaviours have been defined:

a) When a platform has started and is able to accept and process ELI messages (this state is known as UP) it will 'broadcast' a PLATFORM_STATUS message indicating this, where 'broadcast' in this context is that the message will be sent to all possible platforms that could exist in the system.

   NOTE. Whether this is by using an actual transport level broadcast capability is an implementation detail. E.g. for the UDP transport binding described in Annex A it would be sent to each known multicast address.

b) Any platform will view all other platforms as initially in the DOWN state, and any services that they provide would be marked as UNAVAILABLE.

c) When a platform receives a PLATFORM_STATUS message from another platform, the receiving platform will respond in the following ways:
   - If the sending platform has transitioned from DOWN to UP, then the receiving platform will send out the following Platform ELI messages only to the sending platform:
     - a PLATFORM_STATUS message with its current state (UP)
     - an AVAILABILITY_STATUS_REQUEST (for all services – 0xFFFFFFFF)
     - a VERSIONED_DATA_PULL (for all versioned data – 0xFFFFFFFF).

- If the sending platform has not change state, then the receiving platform will take no further action.
- If the sending platform has transitioned from UP to DOWN, then the receiving platform will mark all of the services provided by that sending platform as UNAVAILABLE.

These behaviours mean that all platforms will eventually receive the service availability and versioned data states from all other platforms that are UP.

If periodic publishing of PLATFORM_STATUS is being used for detecting failures, then a platform would mark all of the services provided by another platform as UNAVAILABLE if it has not had confirmation that the other platform is still UP after the system defined time period.

Figure 4 shows an example of a start-up sequence using two platforms.

Platform 1 starts-up first and 'broadcasts' its 1:PLATFORM_STATUS message. Because no other platform is available at this time the platform continues to operate without any interactions.

Once Platform 2 starts it also sends out a 2:PLATFORM_STATUS message, and this is received by Platform 1.

As a result of the 2:PLATFORM_STATUS message Platform 1 will:

- Send the 3:PLATFORM_STATUS message (as the status of Platform 2 has changed from DOWN to UP)
- Send the 4:AVAILABILITY_STATUS_REQUEST (for all services – 0xFFFFFFFF)
- Send the 5:VERSIONED_DATA_PULL (for all versioned data – 0xFFFFFFFF).

NOTE: there is no requirement about ordering of messages 3, 4 and 5.

Platform 2 will respond to the 4:AVAILABILITY_STATUS_REQUEST by sending 6:AVAILABILITY_STATUS, and respond to the 5:VERSIONED_DATA_PULL by sending 7:VERSIONED_DATA_MSGS

As a result of the 3:PLATFORM_STATUS message Platform 2 will:

- Send the 8:PLATFORM_STATUS message (as the status of Platform 1 has changed from DOWN to UP)
- Send the 9:AVAILABILITY_STATUS_REQUEST (for all services – 0xFFFFFFFF)
- Send the 10:VERSIONED_DATA_PULL (for all versioned data – 0xFFFFFFFF).

NOTE: there is no requirement about ordering of messages 6, 7, 8, 9 and 10, provided that these messages are sent in response to the reception of messages 3, 4 and 5.

NOTE: should only message 3 be received by Platform 2 (and not messages 4 and 5), Platform 2 would only send messages 8, 9 and 10.

Platform 1 will Ignore the 8:PLATFORM_STATUS message sent from Platform 2 (as the status of Platform 2 has not changed from DOWN to UP).

Platform 1 will respond to the 9:AVAILABILITY_STATUS_REQUEST by sending 11:AVAILABILITY_STATUS, and respond to the 10:VERSIONED_DATA_PULL by sending 12:VERSIONED_DATA_MSGS

Once this sequence has completed, both platforms will have (for that point in time) the service availability states for all services within the system, along with the versioned data states for all versioned data services required on each platform.

From this point onwards normal Platform ELI AVAILABILITY_STATUS messages will be used to notify other platforms of a change in state of a service, or set of services. Similarly the normal Service Operation ELI messages for VERSIONED_DATA_MSGS will be used to update versioned data state as it is re-published.

NOTE VERSIONED_DATA_MSG are service operation messages, which are defined in §6.1.2.2.

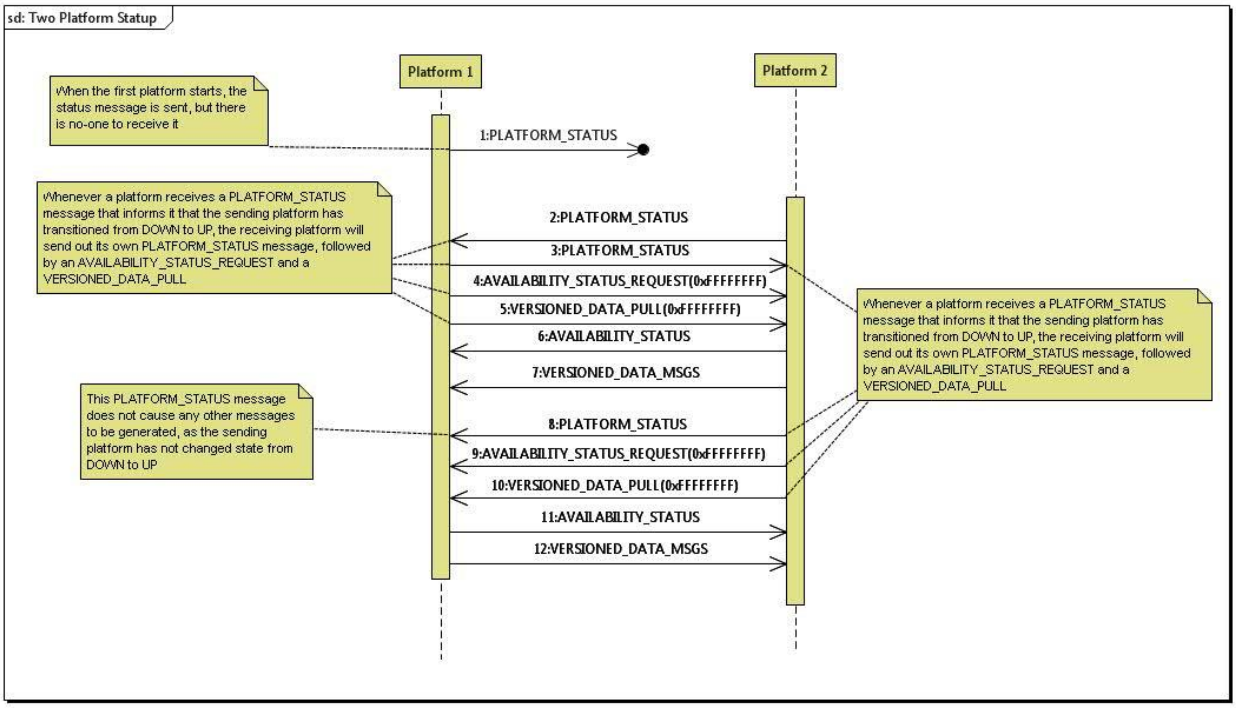

Figure 4 Two Platform Start-Up Sequence Example

Figure 5 shows an example with three platforms starting up. This example follows exactly the same rules as the two platform one, and concludes once all service availability states and versioned data state has been distributed to all platforms.

The three platform start-up example may be extended to any number of platforms, and equivalent sequences will occur.

NOTE: in the example Figure 5, Platforms can interleave messages that they send to each other. There is no ordering requirement other than sending messages in response of the reception of others, as specified in §15.

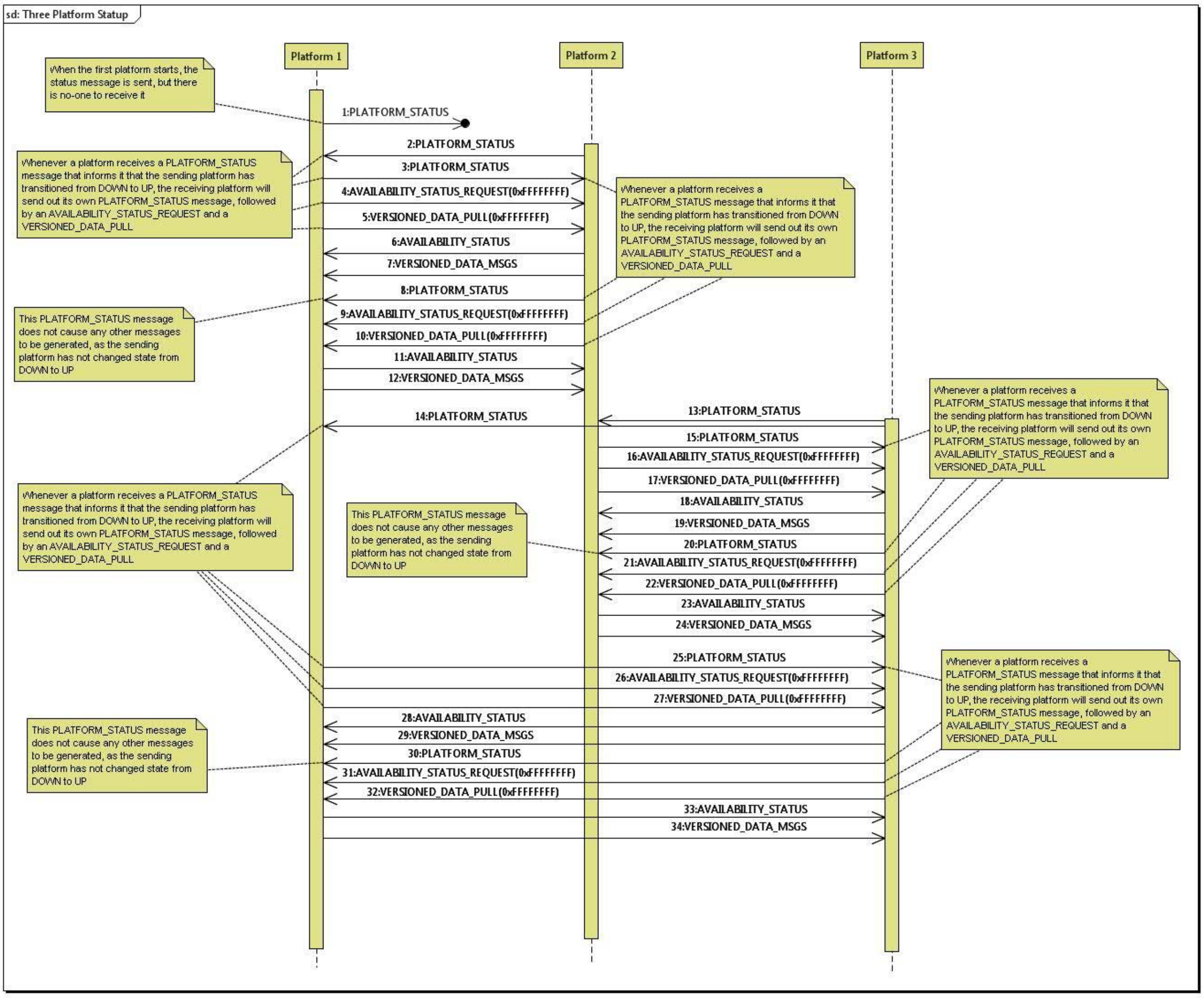

Figure 5 Three Platform Start-Up Sequence Example

Annex A. UDP Network Binding

This annex describes an example UDP Network binding that allows the transmission of an ELI Message using the UDP/IP protocol. The binding is provided to show one way that the UDP/IP protocol may be used to transport ELI messages; however it is not the only way that this may be done.

The basic principle is that ELI messages are sent from one platform to another, each platform being identified by an IP multicast address and a receiving UDP port. The following are examples of communications between platforms using this mechanism:

- When platform P1 sends an ELI message to another single platform P2, P1 sends the ELI message, through the UDP/IP protocol, to the IP multicast address of P2 on the specified UDP port.
- When platform P1 sends an ELI message to two platforms P2 and P3, P1 sends the message twice, once to the IP multicast address of P2 on the specified UDP port and once to the IP multicast address of P3 on the specified UDP port.

This section explains how to map ELI messages onto UDP/IP datagrams.

A.1 Network configuration

The network configuration is defined in an XML file dedicated for the UDP Binding configuration.

This file defines the following for each platform whose name is given by the logical system file:

- a platform ID, an integer between 0 and 15. It is used to uniquely identify one of the connected platforms.
  NOTE this range is less than that which is capable of being used in the ELI Header, and restricts the values that may be assigned. As this is an example used for demonstration purposes it is not an issue, however in a different implementation this restriction may need removing.
- the maximum number of logical channels from which ELI messages can be sent to other platforms. The maximum authorized number of channels is 256 (256 is also the default value).
- A receiving IP multicast address and a receiving UDP port number used to listen for incoming messages.

The actual identity of a sender is the composition of the platform ID and a channel ID; this allows identifying the counter associated to the sender.

The receiving multicast address and receiving port number are used by each platform (potentially at computing node level) to create one or several receiving UDP sockets and one or several sending UDP sockets. A sending socket will send ELI messages to one other platform. The receiving sockets will receive ELI messages from every platform.

The use of logical channels and the use of receiving IP multicast address allow exchanging messages between platforms without knowledge of the internal topology of these platforms (e.g. without knowledge of the number of computing nodes). For instance there could be one channel per emitting computing node. All computing nodes in a platform may receive all incoming UDP messages on the IP multicast address, and analyze the ELI header to check whether the message is relevant for them or not.

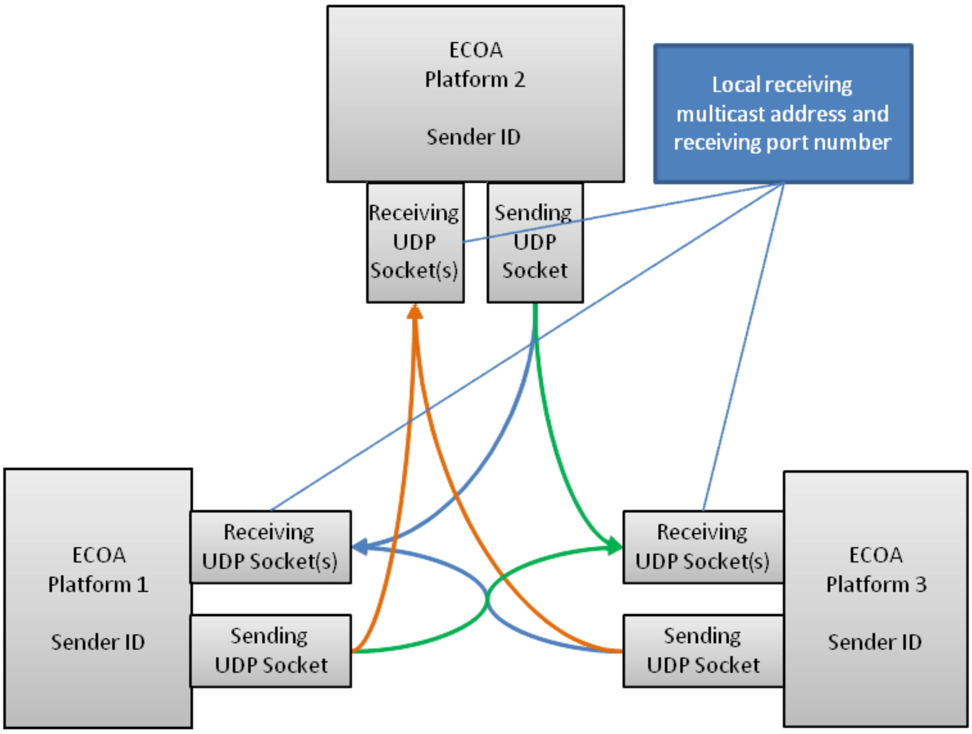

Figure 6 Example of a UDP network logical architecture

Example of a configuration file:

```
<UDPBinding xmlns="http://www.ecoa.technology/udpbinding-1.0" >
  <platform name="ECOA Platform 1" platformId="0"
            receivingPort="60426" receivingMulticastAddress="239.0.0.1"/>
  <platform name="ECOA Platform 2" platformId="2"
            receivingPort="60430" receivingMulticastAddress="239.0.0.2"/>
  <platform name="ECOA Platform 3" platformId="7" maxChannels="34"
            receivingPort="60410" receivingMulticastAddress="239.0.0.3"/>
</UDPBinding>
```

A.2 Network message definition

ECOA UDP messages are designed to:

1. transmit ELI messages between ECOA platforms with
   a. possible fragmentation of large messages
   b. lost messages detection
2. enable a receiving platform to identify the sending platform

Each ECOA UDP message contains a header and a payload containing the whole or part of an ELI message.

A.2.1 Possible fragmentation of large messages

To ensure transmission of ELI messages greater than maximum size of the UDP/IP transport, those messages are split into several fragments by the sender. Those fragments will fit the maximum size of the UDP binding payload. This can be calculated by using the following formula:

Maximum size of UDP/IP payload = Sizeof(UDP datagram) - (sizeof(IP Header) + sizeof(UDP Header))

Maximum size of UDP/IP payload = 65535-(20+8) = 65507

Size of UDP binding header = 4 bytes

Maximum UDP binding payload size is therefore 65507 - 4 = 65503 bytes (or 524024 bits).

The receiver is responsible for gathering the fragments in order to reassemble the original ELI message.

Each fragment has a “message part” attribute to define which part to of the ELI message it belongs:

- beginning of the ELI message
- middle of the ELI message
- end of the ELI message
- beginning and end of the ELI message

The message part attribute is set by the sender during the fragmentation step, and used by receiver to detect fragmented ELI messages. This information will allow the receiver to reassemble the received payloads into a complete and correct ELI message. It is assumed that the UDP network will not change the datagram sending order.

A.2.2 Detection of lost UDP messages

The ECOA UDP binding header contains a field for a counter.

This counter is related to one given channel.

This counter is incremented by the sender for each ECOA UDP message sent. This enables receivers to detect that for each sender ID, corresponding received ECOA UDP messages have consecutive counter numbers. This enables ECOA UDP message loss to be detected. As stated above, it is assumed that ECOA UDP messages are received in the same order they are sent.

A.2.3 Identification of the sending platform:

A receiving platform will be able to identify the sending platform by using the sender ID (composition of a platform ID and a channel ID) sent in ECOA UDP messages within the ECOA UDP binding header.

A.3 ECOA UDP Message Format

This section describes the global structure and details for each field of an ECOA UDP message. The payload content is a whole or part of an ELI message. ELI messages are described in section 6.1.

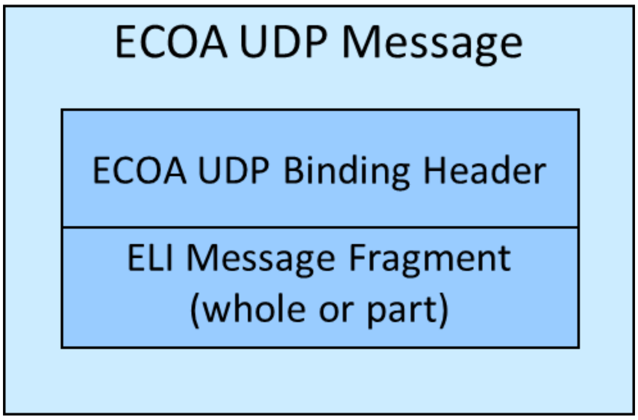

Figure 7 ELI Message Format

Table 8 ELI Message Format

Header	Value	Explanation	Length (bits)	Alignment (bits)
ECOA UDP Binding Header	4 byte header	UDP message header	32	32
ELI Message Fragment (whole or part)	ELI Message fragment, maximum 65503 bytes	Whole or fragment of an ELI message	Maximum 524024	32

A.3.1 ECOA UDP Binding Header

Figure 8 identifies the contents of the ECOA UDP Binding Header, and Table 9 contains the details of those fields.

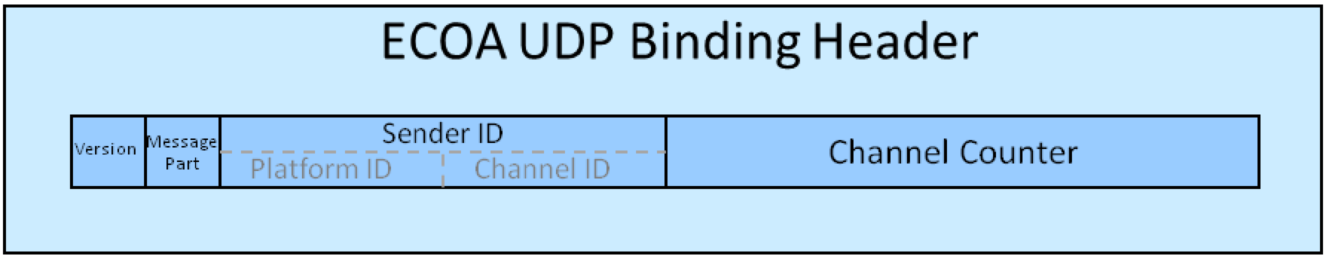

Figure 8 ECOA UDP binding header

Table 9 ECOA UDP binding header fields

Header	Subheader	Value	Explanation	Length (bits)	Alignment (bits)
Version		00b for this version 01b-11b Reserved		2	2

Header	Subheader	Value	Explanation	Length (bits)	Alignment (bits)
Message part		00b - begin 01b - middle 10b - end 11b - begin and end	Enumeration which indicates the part of the message this UDP datagram is associated with. The UDP binding can reassemble packets to create a whole ELI message.	2	2
Sender ID			Identification of the sender which broadcasts this datagram to every platform		
	Platform ID	Unsigned number between 0 and 15	Platform ID provided by the XML configuration file	4	4
	Channel ID	Unsigned number between 0 and 255	Channel ID to which the counter used for this UDP datagram is associated to. The value of the ID is set by the sending platform itself. It can rely on node ID, on module instance ID, etc.	8	8
Channel Counter		Unsigned number between 0 and 65535 transmitted in big endian	Positive counter which identifies this packet for the identified channel. The counter can loop. EXAMPLE To clarify: • Message1/Packet1 ➔ value=0 • Message1/Packet2 ➔ value=1 • Message2/Packet1 ➔ value=2 • …	16	16

The sending platform shall maintain one Channel Counter per Channel and use them accordingly.

A.3.2 ELI Message Fragment (Whole or Part)

Table 10 identifies the content of the ELI Message Fragment within the UDP message.

Table 10 ELI Message Fragment (whole or part)

Header	Subheader	Value	Explanation	Length (bits)	Alignment (bits)
ELI Message Fragment (whole or part)		ELI Message fragment (whole or part), maximum 65503 bytes	ELI Message part maximum size 65503 bytes (65535 bytes - 28 - 4 bytes)	Maximum 524024	32

A.3.3 Example ECOA UDP messages

The following sections give examples of using the UDP network binding to send various sizes of ELI messages.

A.3.3.1 Single fragment message

For an ELI message that will fit completely within the UDP binding (i.e. length <= 65503 bytes), only a single fragment will be generated. The example in Table 11 shows a single fragment that contains an ELI message of 10000 bytes. Messages of this type will contain one “begin and end” fragment.

Table 11 Single fragment message

Header	Subheader	Value	Explanation	Length (bits)	Alignment (bits)
Version		00b		2	2
Message part		11b - begin and end	Indicates this is a single fragment message	2	2
Sender ID			Identification of the sender which sends this datagram to every partner		
	Platform ID	1	Platform ID provided by the XML configuration file	4	4
	Channel ID	2	Channel ID to which the counter used for this UDP datagram is associated to.	8	8
Channel Counter		5	Positive counter which identifies this packet for the identified channel.	16	16
ELI Message Fragment (Whole)		10000 byte ELI message	ELI message comprising ELI Generic Message Header and Message Specific Payload	80000	32

A.3.3.2 Two fragment message

For an ELI message that will fit within two UDP fragments (i.e. 65503 > length <= 131006 bytes) two fragments will be generated. The example in Table 12 and Table 13 shows two fragments that contains an ELI message of 100000 bytes. Messages of this type will contain one “begin” fragment and one “end” fragment.

Table 12 First Fragment of a Two fragment message

Header	Subheader	Value	Explanation	Length (bits)	Alignment (bits)
Version		00b		2	2
Message part		00b - begin	Indicates this is the start of a multi-fragment message	2	2

Header	Subheader	Value	Explanation	Length (bits)	Alignment (bits)
Sender ID			Identification of the sender which sends this datagram to every partner		
	Platform ID	1	Platform ID provided by the XML configuration file	4	4
	Channel ID	2	Channel ID to which the counter used for this UDP datagram is associated to.	8	8
Channel Counter		8	Positive counter which identifies this packet for the identified channel.	16	16
ELI Message Fragment (Whole)		1st 65503 bytes of a 100000 byte ELI message	1st part of ELI message comprising ELI Generic Message Header and the start of the Message Specific Payload	524024	32

Table 13 Second Fragment of a Two fragment message

Header	Subheader	Value	Explanation	Length (bits)	Alignment (bits)
Version		00b		2	2
Message part		10b - end	Indicates this is the end of a multi-fragment message	2	2
Sender ID			Identification of the sender which sends this datagram to every partner		
	Platform ID	1	Platform ID provided by the XML configuration file	4	4
	Channel ID	2	Channel ID to which the counter used for this UDP datagram is associated to.	8	8
Channel Counter		9	Positive counter which identifies this packet for the identified channel.	16	16
ELI Message Fragment (Whole)		last 34497 bytes of a 100000 byte ELI message	2nd part of ELI message comprising the remainder of the Message Specific Payload	275976	32

A.3.3.3 Multiple fragment message

For an ELI message that is larger than 131006 bytes, multiple fragments will be generated. The example in Table 14, Table 15 and Table 16 shows three fragments that contains an ELI message of 150000 bytes. Messages of this type will contain one “begin” fragment, one or more “middle” fragments, and one “end” fragment.

Table 14 First Fragment of a Multi fragment message

Header	Subheader	Value	Explanation	Length (bits)	Alignment (bits)
Version		00b		2	2
Message part		00b - begin	Indicates this is the start of a multi-fragment message	2	2
Sender ID			Identification of the sender which sends this datagram to every partner		
	Platform ID	1	Platform ID provided by the XML configuration file	4	4
	Channel ID	2	Channel ID to which the counter used for this UDP datagram is associated to.	8	8
Channel Counter		302	Positive counter which identifies this packet for the identified channel.	16	16
ELI Message Fragment (Whole)		1st 65503 bytes of a 150000 byte ELI message	1st part of ELI message comprising ELI Generic Message Header and the start of the Message Specific Payload	524024	32

Table 15 Second Fragment of a Multi fragment message

Header	Subheader	Value	Explanation	Length (bits)	Alignment (bits)
Version		00b		2	2
Message part		01b - middle	Indicates this is the middle of a multi-fragment message	2	2
Sender ID			Identification of the sender which sends this datagram to every partner		
	Platform ID	1	Platform ID provided by the XML configuration file	4	4
	Channel ID	2	Channel ID to which the counter used for this UDP datagram is associated to.	8	8

Header	Subheader	Value	Explanation	Length (bits)	Alignment (bits)
Channel Counter		303	Positive counter which identifies this packet for the identified channel.	16	16
ELI Message Fragment (Whole)		2nd 65503 bytes of a 150000 byte ELI message	2nd part of ELI message comprising part of the Message Specific Payload	524024	32

Table 16 Third Fragment of a Multi fragment message

Header	Subheader	Value	Explanation	Length (bits)	Alignment (bits)
Version		00b		2	2
Message part		10b - end	Indicates this is the end of a multi-fragment message	2	2
Sender ID			Identification of the sender which sends this datagram to every partner		
	Platform ID	1	Platform ID provided by the XML configuration file	4	4
	Channel ID	2	Channel ID to which the counter used for this UDP datagram is associated to.	8	8
Channel Counter		304	Positive counter which identifies this packet for the identified channel.	16	16
ELI Message Fragment (Whole)		last 18994 bytes of a 150000 byte ELI message	last part of ELI message comprising the remainder of the Message Specific Payload	151952	32

A.4 Message Byte and Bit Order

In order to ensure complete interoperability it is required that the byte and bit order of the ECOA UDP Binding Header be defined.

The network byte order shall be as per the internet standard of big endian.

The ECOA UDP Binding Header bit format shall be as shown below:

```
          76543210
Byte 1 VEMPPLID
VE: version number (2 bits) | MP: message part (2 bits) | PLID: Platform ID (4 bits)

Byte 2 CHANNEID
CHANNEID: Channel ID (8 bits)
```

```
Byte 3 COUNTMSB
COUNTMSB: Counter Most Significant Byte

Byte 4 COUNTLSB
COUNTLSB: Counter Least Significant Byte
```

Bytes are described from the most significant bit (7) to the least (0).

The header is sent in the following byte order: byte 1 then byte 2 then byte 3 then byte 4.