EMSA
European Maritime Safety Agency

Star Streaming Remote Hub Interface Control Document

Document version: 1.02

Document release date: 07/10/2019

Subject		
Star Streaming Message Distribution Reference Guide **[1.00]**		
Writer	Sign off and Date	Deadline for Comments
XpandIT	27/09/2019	

RECIPIENTS		
Name	Organization	Sign-off
Oscar	EMSA	07/10/2019

TABLE OF CONTENTS

1 INTRODUCTION 4

1.1 PURPOSE 4
1.2 SCOPE 4
1.3 ABBREVIATIONS AND ACRONYMS 4

2 GENERAL OPERATIONAL CONCEPT 5

3 PROVIDER INTERFACES 7

3.1 AIS TCP PROVIDER INTERFACE 7
3.1.1 Transport protocol 7
3.1.2 Message Format 7
3.1.3 Example message 8
3.2 AIS KAFKA PROVIDER INTERFACE 8
3.2.1 Transport protocol 8
3.2.2 Message Format 9
3.2.3 Examples 9

4 CONSUMER INTERFACES 10

4.1 AIS TCP CONSUMER INTERFACE 10
4.1.1 Transport protocol 10
4.1.2 Message Format 10
4.1.3 Example message (TCP consumer) 10
4.2 AIS KAFKA CONSUMER INTERFACE 10
4.2.1 Transport protocol 10
4.2.2 Enriched position stream kafka consumer interface 11
4.2.3 Position Report Avro 11
4.2.4 Example Avro Message 14

5 INTERFACING WITH KAFKA TOPICS 18

5.1 PRODUCER 18
5.1.1 Functionality 18
5.1.2 Produced messages 18
5.1.3 Message flow 18
5.1.4 Implementation 18
5.1.5 Properties 18
5.1.6 Implementation 20
5.2 CONSUMER 22
5.2.1 Functionality 22
5.2.2 Consumed messages 22
5.2.3 Message flow 22
5.2.4 Implementation 22
5.2.5 Properties 22
5.2.6 Implementation 24

1 INTRODUCTION

1.1 Purpose

This document is the Interface Control Document of the EMSA Star Streaming Remote Hub (SSTRH). The SSTRH is the integration software provided by EMSA to its partners, allowing them to efficiently exchange streaming information, mainly AIS and other ship position sources.

1.2 Scope

This document specifies the system interfaces of SSTRH. All the inbound and outbound messages interfaces are described in the document.

1.3 Abbreviations and acronyms

A list of the principal abbreviations and acronyms used in the document is provided here for a better understanding of this document.

Abbreviation	Definition
AIS	Automatic Identification Systems
EMSA	European Maritime Safety Agency
JMS	Java Message Service
JSON	JavaScript Object Notation is a lightweight data-interchange format
MM	Mirror Maker
N/A	Not Applicable or Not Available
S-AIS	Satellite Automatic Identification Systems
SST	Star Streaming
SSTRH	Star Streaming Remote Hub
T-AIS	Terrestrial Automatic Identification Systems
TCP	Transmission Control Protocol
URL	Unified Resource Locator
XML	eXtensible Markup Language

2 GENERAL OPERATIONAL CONCEPT

The EMSA SSTRH is designed to exchange-provide and consume- AIS streams and other ship position message streams with EMSA SST platform.

The SSTRH component is hosted at the EMSA partner premises and providers several interfaces for integration with the partner systems to exchange AIS and other ship position sources with EMSA efficiently.

The current document describes two distinct types of interfaces: producer and consumer.

Producer interfaces allow for data providers to send AIS and other ship position streams to EMSA. The SSTRH performs the validation and filtering of incoming messages. This processing is implemented using Apache Kafka technology.

The validation process starts with messages, sent by the provider, via the SSTRH producer interface, e.g. the TCP Connector in the provider nodes, these messages are in raw format. The received messages are stored in Kafka Topics and later validated with Kafka Streams. After the validation process is finished messages are replicated to the Central EMSA Cluster via Kafka Mirror Maker.

In the EMSA Central cluster messages can be integrated in a common stream, to be filtered and distributed to EMSA applications and EMSA partners for further processing.

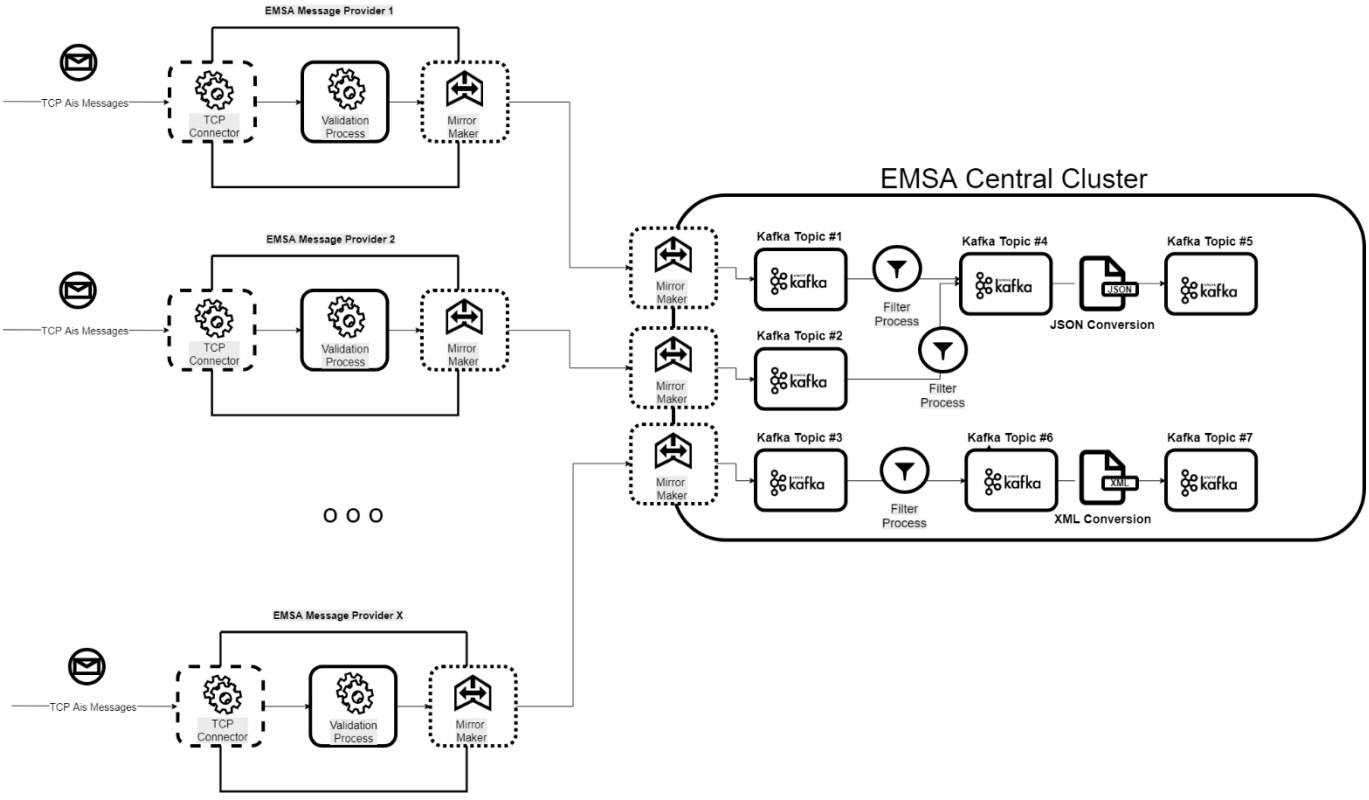


Figure 1 – EMSA Message Providers interaction with the EMSA Central Cluster

In the case of data consumers, the integrated message stream is filtered to include the subset of messages to be delivered to that consumer (e.g. Satellite AIS messages in a given geographic bounding box). These messages are filtered into a dedicated Kafka topic, which is then replicated to the corresponding SSTRH for that data consumer. The consumer partner can then use the data consumer interface of choice to read messages from SSTRH into their system. For

example, the partner system can connect into the TCP socket consumer interface and read AIS messages.

The following chapters provide detailed explanation on available SSTRH data producer and consumer interfaces.

3 PROVIDER INTERFACES

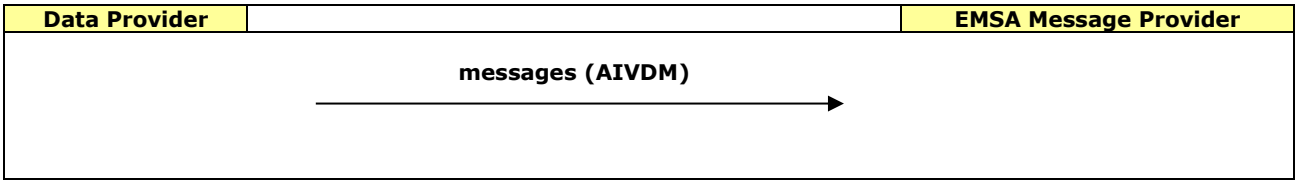
3.1 AIS TCP provider interface

The AIS provider connects to the AIS TCP provider interface and sends AIVDM messages for transmission to EMSA.

3.1.1 Transport protocol

The data provider system establishes a TCP socket connection to the SSTRH and starts sending messages to the SSTRH via TCP socket stream.

The following figure outlines the expected synchronous flow of the raw message.



3.1.2 Message Format

The data provider sends messages to the SSTRH via TCP Connector in the AIVDM format. The comment block extension defined in the current document allows providing additional information not encodable in NMEA format.

3.1.2.1 Information Comment Block extension

The information "i:" tag in the comment block has been extended in order to allow specifying additional information

The following table describes the raw message received by the TCP Connector.

Array	*Value*			*Description*	*Occ*
vdmSentence					1
commentBlock					0-1
	time				0-1
	information				0-1
		xml tag O		Message originator (typically a 2- , 3 letter ISO Country Code)	0-1
		xml tag S		Message source (T-AIS, Sat-AIS, Ship-AIS, RPAS-AIS)	0-1
		satInfo			0-1
			groundStationAcquisitionTs	Timestamp of acquisition at fround station	0-1
			dataCentreIngestionTs	Timestamp of delivery to the data centre	0-1
			dataCentreDeliveryTs	Timestamp of delivery by the data centre to EMSA	0-1

			satelliteId	Satellite identified	0-1
			groundStationId		0-1
			foa	Frequency of arrival, for doppler position validation	0-1
			toa	Timo of arrival, for doppler position validation	0-1

Code	Source type	Description
A	T-AIS	Terrestrial AIS
D	RPAS-AIS	AIS from RPAS
P	Ship-AIS	Ship AIS
S	S-AIS	Satellite AIS

Table 1Possible values of Tag S

3.1.3 Example message

Container includes raw AIS Message.

```
!AIVDM,1,1,,B,7050QlAD:Go<E2Ush5@aPEQu;;;@,0*56
```

Container includes raw multiline AIS Message.

```
!AIVDM,2,1,3,A,577NrV02CT9QI8l;B21<P4v1<P4r3N2222222216H0Q:F6O80B5Dp1k4p888,0*5D
!AIVDM,2,2,3,A,88888888880,2*27
```

Container includes AIVDM multiline AIS Message with comment block.

```
\\g:1-2-
1932,c:1474364472,s:106,i:<S>S</S><Q>12</Q><O>XLS</O>*31\\!AIVDM,2,1,3,A,577NrV02CT9QI8l;B21<P4v
1<P4r3N2222222216H0Q:F6O80B5Dp1k4p888,0*5D
\\g:2-2-1932*54\\!AIVDM,2,2,3,A,88888888880,2*27
```

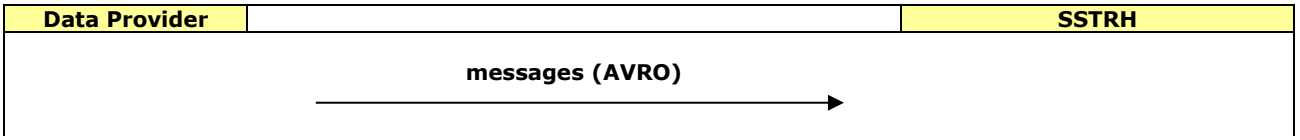
3.2 AIS kafka provider interface

The AIS provider connects to the AIS KAFKA provider interface and sends AVRO messages for transmission to EMSA. Please refer to chapter 5 to detailed information in how to create a Kafka Producer.

3.2.1 Transport protocol

The data provider connects to the SSTRH kafka broker specifying the destination topic and starts writing AVRO messages into that topic.

The following figure outlines the expected synchronous flow of the raw message.



3.2.2 Message Format

The raw AIS messages shall be written in AVRO format following the Generic Message AVRO schema.

The table below describes the fields in the Generic Message AVRO schema.

Field	Mandatory	
uniqueId	Y	uniqueID **must** obey the following format: `raw_message_hashcode@timestamp`
source	Y	
originator	Y	
timestamp	Y	
raw_message	Y	Raw AIS messsage

3.2.3 Examples

Container includes Avro Generic Message.

```
{"uniqueID":"861919401@1546970330",
"originator":"RUS",
"source":"source-xpa",
"timestamp":1546970330,
"raw_message":"\\i:<O>RUS</O>*28\\!AIVDM,1,1,,1,15CV4N001jCCbGLAJO22WR220H0e,0*63"}
```

4 CONSUMER INTERFACES

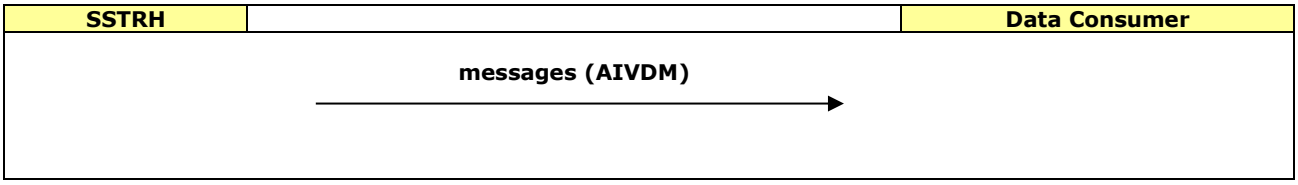
4.1 AIS TCP consumer interface

The AIS consumer connects to the AIS TCP consumer interface and receives AIVDM messages from the EMSA SSTRH.

4.1.1 Transport protocol

The data consumer system establishes a TCP socket connection to the SSTRH and starts reading messages from the SSTRH via TCP socket stream.

The following figure outlines the expected synchronous flow of the raw message.



4.1.2 Message Format

The following table describes the Message that is sent to the TCP Client.

Value	*Occ*
raw_message	1

4.1.3 Example message (TCP consumer)

Container includes a sample Message of the message sent to the TCP Client.
\\i:<O>RUS</O>*28\\!AIVDM,1,1,,1,15CV4N001jCCbGLAJO22WR220H0e,0*63

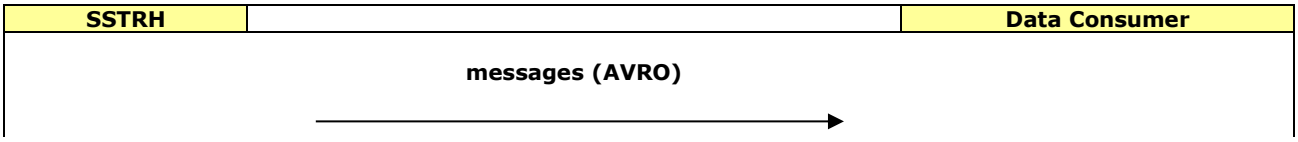
4.2 AIS kafka consumer interface

The AIS provider connects to the AIS kafka provider interface and receives AVRO messages from the EMSA SSTRH.

4.2.1 Transport protocol

The data consumer connects to the SSTRH kafka broker specifying the origin topic and starts reading AVRO messages from that topic.

The following figure outlines the expected synchronous flow of the raw message.



4.2.2 Enriched position stream kafka consumer interface

The enrichment stream is a Kafka Streams application that is designed to consume the messages obtained from the Kafka topic with the purpose of:

- validate the messages and convert them into the defined EMSA Avro format, if the messages are not already in the desired format.
- enrich the messages with information obtained from different endpoints.
- filter the messages into another Kafka topic.

The end result of these transformation is a topic that can be consumed and has the following format:

4.2.3 Position Report Avro

Value			*Occ*
message_id			1
position_id			1
source			1
timestamp			1
originator			1
requestor			0-1
authorized_roles			0-1
longitude			1
latitude			1
speed_over_ground			0-1
courser_over_ground			0-1
navigational_status			0-1
true_heading			0-1
heading			0-1
rate_of_turn			0-1
validity_flag			1
source_specific			0-1
	AisSpecific		0-1
		message_type	1
		position_accuracy	1
		raim_flag	1
		nmea	1
	SatAisSpecific		0-1
		message_type	1
		position_accuracy	1
		raim_flag	1
		nmea	1
		satellite_id	0-1
		ground_station_identifier	0-1
		ground_station_acquisition_ts	0-1
		data_centre_ingestion_ts	0-1

		data_centre_delivery_ts		0-1
		frequency_of_arrival		0-1
		time_of_arrival		0-1
		doppler_signal_level		0-1
		doppler_signal_noise_ratio		0-1
		doppler_channel_id		0-1
		data_flow_id		0-1
	LritSpecific			0-1
		message_type		1
		response_type		1
		reference_id		1
		message_id		1
	VmsSpecific			0-1
		naf		1
		fishing_serial_trip_number		0-1
	particulars			1
		csd_id		0-1
		emsa_id		0-1
		imo		0-1
		mmsi		0-1
		ir		0-1
		name		0-1
		call_sign		0-1
		flag_state		0-1
	ship_enrichment			0-1
		source		1
		timestamp		1
		particulars		0-1
			csd_id	0-1
			emsa_id	0-1
			imo	0-1
			mmsi	0-1
			ir	0-1
			name	0-1
			call_sign	0-1
			flag_state	0-1
		ship_type		0-1
		banned		0-1
		single_hull_tanker		0-1
		position_fixing_device		0-1
		detained		0-1
		ship_risk_profile		0-1
		priority_of_inspection		0-1

		eligible_for_esp_inspection			0-1
		eligible_for_banning			0-1
		dimensions			0-1
			BeamAndLength		0-1
				beam	1
				length_overal	1
			ShipDimensions		0-1
				a	1
				b	1
				c	1
				d	1
	voyage_enrichment				0-1
		source			1
		timestamp			1
		ssn_voyage_id			1
		ship_call_id			1
		last_port			0-1
			PortInfo		
				name	0-1
				locode	1
				eta	0-1
				ata	0-1
				atd	0-1
				location_in_port	0-1
		port_of_call			1
			PortInfo		
				name	0-1
				locode	1
				eta	0-1
				ata	0-1
				atd	0-1
				location_in_port	0-1
		next_port			0-1
			PortInfo		
				name	0-1
				locode	1
				eta	0-1
				ata	0-1
				atd	0-1
				location_in_port	0-1
		pob			0-1
		hazardous_materials			0-1
		waste_delivery_status			0-1

		current_security_level			0-1
		bunkers_reported			0-1
	incident_enrichment				0-1
		source			1
		timestamp			1
		type			1
		incident_id			1
	mrs_enrichment				0-1
		source			1
		timestamp			1
		mrs_notification			1
		cst_identification			0-1
		next_port			0-1
			PortInfo		
				name	0-1
				locode	1
				eta	0-1
				ata	0-1
				atd	0-1
				location_in_port	0-1
		pob			0-1
		any_dg			1
		longitude			1
		latitude			1
		reporting_date_and_time			1
	exemption_enrichment				0-1
		source			1
		timestamp			1
		type			1
		company_name			1
		from			1
		to			1
		route_port			1
		exemption_applies_to			0-1
		authority			1
		locode			0-1

4.2.4 Example Avro Message

An example of a message in the PositionReport avro format after full enrichment process.

```
{
    "authorised_roles": {
        "array": [
```

```
            "ROL_VIEW_TAIS_HELCOM",
            "ROL_VIEW_TAIS_EU",
            "ROL_VIEW_TAIS_OWN",
            "ROL_VIEW_POSITIONS_ALL"
        ]
    },
    "course_over_ground": {
        "float": 39.0
    },
    "exemption_enrichment": {
        "eu.europa.emsa.cdf.avro.ExemptionEnrichment": {
            "authority": "GB",
            "company_name": "Stena",
            "exemption_applies_to": {
                "array": [
                    "GBKGH"
                ]
            },
            "from": 17757,
            "locode": {
                "eu.europa.emsa.cdf.avro.LocationCode": "GBSOU"
            },
            "route_port": [
                "GBKGH"
            ],
            "source": "test",
            "timestamp": 1564499050615,
            "to": 19582,
            "type": "PRE_ARRIVAL"
        }
    },
    "heading": null,
    "incident_enrichment": {
        "array": [
            {
                "incident_id": "IT5SYpcQOIQpN7v7d",
                "source": "test",
                "timestamp": 1564499050608,
                "type": "SITREP"
            },
            {
                "incident_id": "IT8FQaXyN5lcFfCtg",
                "source": "test",
                "timestamp": 1564499050608,
                "type": "OTHERS"
            }
        ]
    },
    "latitude": 63.447964,
    "longitude": -20.031631,
    "message_id": "test",
    "mrs_enrichment": {
        "eu.europa.emsa.cdf.avro.MrsEnrichment": {
            "any_dg": false,
            "cst_identification": null,
            "latitude": 33789000.0,
            "longitude": 7395999.0,
            "mrs_notification": "SOUNDREP",
            "next_port": {
                "eu.europa.emsa.cdf.avro.PortInfo": {
                    "ata": null,
                    "atd": null,
                    "eta": null,
                    "location_in_port": null,
                    "locode": "SEMMA",
                    "name": null
                }
            },
            "pob": {
                "int": 17
            },
            "reporting_date_and_time": 1547349120000,
            "source": "test",
            "timestamp": 1564499050612
```

```
        }
    },
    "navigational_status": null,
    "originator": "ISL",
    "particulars": {
        "call_sign": null,
        "csd_id": null,
        "emsa_id": {
            "string": "23342334"
        },
        "flag_state": null,
        "imo": null,
        "ir": null,
        "mmsi": {
            "long": 251011000
        },
        "name": null
    },
    "position_id": 295170634794987646,
    "rate_of_turn": {
        "double": 720.0
    },
    "requestor": null,
    "ship_enrichment": {
        "eu.europa.emsa.cdf.avro.ShipEnrichment": {
            "banned": null,
            "detained": null,
            "dimensions": {
                "eu.europa.emsa.cdf.avro.ShipDimensions": {
                    "a": 12.0,
                    "b": 30.0,
                    "c": 4.0,
                    "d": 4.0
                }
            },
            "eligible_for_banning": null,
            "eligible_for_esp_inspection": null,
            "particulars": {
                "eu.europa.emsa.cdf.avro.ShipParticulars": {
                    "call_sign": {
                        "string": "TFTV"
                    },
                    "csd_id": {
                        "long": 163606
                    },
                    "emsa_id": {
                        "string": "1585292"
                    },
                    "flag_state": null,
                    "imo": {
                        "eu.europa.emsa.cdf.avro.ImoNumber": "8303410"
                    },
                    "ir": null,
                    "mmsi": {
                        "long": 251011000
                    },
                    "name": {
                        "string": "BRYNJOLFUR"
                    }
                }
            },
            "position_fixing_device": null,
            "priority_of_inspection": null,
            "ship_risk_profile": null,
            "ship_type": {
                "eu.europa.emsa.cdf.avro.ShipType": "315"
            },
            "single_hull_tanker": null,
            "source": "T-AIS",
            "timestamp": 1553602963000
        }
    },
    "source": "T-AIS",
    "source_specific": {
```

```
        "eu.europa.emsa.cdf.avro.AisSpecific": {
            "message_type": "CLASS_A",
            "nmea":
"\\i:&lt;O>ISL&lt;/O>,c:1553602690*56\\!AIVDM,1,1,,A,13gHOf?Oh8NTCFrTCR:QQbBH00S?,0*14",
            "position_accuracy": "LOW",
            "raim_flag": false
        }
    },
    "speed_over_ground": {
        "float": 0.8
    },
    "timestamp": 1553602690000,
    "true_heading": {
        "float": 329.0
    },
    "validity_flag": "U",
    "voyage_enrichment": {
        "eu.europa.emsa.cdf.avro.VoyageEnrichment": {
            "bunkers_reported": null,
            "current_security_level": null,
            "hazardous_materials": {
                "boolean": false
            },
            "last_port": {
                "eu.europa.emsa.cdf.avro.PortInfo": {
                    "ata": null,
                    "atd": null,
                    "eta": null,
                    "location_in_port": null,
                    "locode": "SEMMA",
                    "name": null
                }
            },
            "next_port": {
                "eu.europa.emsa.cdf.avro.PortInfo": {
                    "ata": null,
                    "atd": null,
                    "eta": {
                        "long": 1547708400000
                    },
                    "location_in_port": null,
                    "locode": "RULED",
                    "name": null
                }
            },
            "pob": {
                "int": 18
            },
            "port_of_call": {
                "ata": null,
                "atd": null,
                "eta": {
                    "long": 1547594400000
                },
                "location_in_port": null,
                "locode": "EEPLS",
                "name": null
            },
            "ship_call_id": "19LK-00049",
            "source": "test",
            "ssn_voyage_id": "19LK-00049#EE",
            "timestamp": 1564499050519,
            "waste_delivery_status": {
                "eu.europa.emsa.cdf.avro.WasteDeliveryStatus": "SOME"
            }
        }
    }
}
```

5 INTERFACING WITH KAFKA TOPICS

The following sections aim at providing a technical guide on interfacing programmatically with a kafka topic from a Java Virtual Machine (JVM). They information hereafter is only relevant for interfaces described in 3.2 and 4.2.

5.1 Producer

5.1.1 Functionality

A JVM Kafka Producer that is able to connect to a topic and write messages to it.

The producer can be a simple type key/value producer and use the apache.kafka.common serializers. This type of producer can be used in most platforms that support kafka.

It can also be a more complex producer and use avro classes, in that case the producer can only be used in the confluent platform (uses the confluent avro serializers).

5.1.2 Produced messages

The produced messages' format depends on the implementation of the producer; it can be a plaintext key/value pair or it can be an avro key/value pair.

5.1.3 Message flow

The following figure outlines the expected synchronous flow of the produced messages.

5.1.4 Implementation

5.1.5 Properties

For the producer to be able to produce messages to a desired topic a few properties have to be passed, either implicitly in the java code or via a properties file.

5.1.5.1 Required Properties

bootstrap.servers

Is a comma-separated list of host and port pairs that are the addresses of the Kafka brokers.

key.serializer

The java class for serialization of the message key. For example if the key is defined as a String the class passed should be "org.apache.kafka.common.serialization.StringSerializer".

(This value is defined in the java code as it should not change)

value.serializer

The java class for serialization of the message value. For example if the value is defined as an Integer the class passed should be "org.apache.kafka.common.serialization.IntegerSerializer".

(This value is defined in the java code as it should not change)

5.1.5.2 Optional Properties

client.id

An id string to pass to the server when making requests so that the source of the requests can be tracked beyond just ip/port by allowing a logical application name to be included in server-side request logging.

acks

The number of acknowledgments the producer requires the leader to have received before considering a request complete.

0: The producer will not wait for any acknowledgment from the server at all. The record will be immediately added to the socket buffer and considered sent. No guarantee can be made that the server has received the record in this case.

1: The leader will write the record to its local log but will respond without awaiting full acknowledgement from all followers. In this case should the leader fail immediately after acknowledging the record but before the followers have replicated it then the record will be lost.

all or -1: The leader will wait for the full set of in-sync replicas to acknowledge the record. This guarantees that the record will not be lost as long as at least one in-sync replica remains alive. This is the strongest available guarantee.

retries

Setting a value greater than zero will cause the client to try to resend up to n times any record whose send fails with a potentially transient error.

5.1.5.3 Security Properties

This is also considered an optional set of properties and should only be used if the brokers are using the SSL security protocol to access the topics.

security.protocol

Should be filled with "ssl" if it is indeed the security protocol being used, otherwise ignore the property or set it as "plaintext".

ssl.truststore.location

The location path to the ssl truststore (should only be filled when the security.protocol=ssl).

ssl.truststore.password

The ssl truststore password (should only be filled when the security.protocol=ssl).

ssl.keystore.location

The location path to the ssl keystore (should only be filled when the security.protocol=ssl).

ssl.keystore.password

The ssl keystore password (should only be filled when the security.protocol=ssl).

ssl.key.password

The ssl key password (should only be filled when the security.protocol=ssl).

5.1.5.4 Example of a properties file

Properties file passed to the Java Kafka Producer.

```
#application
client.id=test-client

#services
bootstrap.servers=remote-hub.emsa.europa.eu:9092
schema.registry.url=http://remote-hub.emsa.europa.eu:8081

#channels
topic=test-topic

#options
aks=all

#Security
security.protocol=ssl
ssl.truststore.location=C:/Users/Xpand-it/IdeaProjects/main/ais-kafka-
producer/src/main/resources/kafka.truststore.jks
ssl.truststore.password=6qVcjL8TeKraQFRH
ssl.keystore.location=C:/Users/Xpand-it/IdeaProjects/main/ais-kafka-
producer/src/main/resources/kafka.keystore.jks
ssl.keystore.password=6qVcjL8TeKraQFRH
ssl.key.password=6qVcjL8TeKraQFRH
```

5.1.6 Implementation

5.1.6.1 Creating the Java Producer

To create a java kafka producer we can pass as an argument to the JVM the properties file and with this file build a Configuration class that the consumer will use.

Then we can simply access the properties in the Configuration file to use as the properties of the consumer.

Creating a Java Kafka Producer with the provided properties.

```
public AvroProducer(Configuration configuration){
    this.configuration = configuration;

}

public void run(){

    //producer properties
    Properties props = new Properties();
    props.put("bootstrap.servers", this.configuration.bootstrapServers);
    props.put("schema.registry.url", this.configuration.schemaRegistryUrl);
    props.put("client.id", this.configuration.groupId);
    //key and value serializers
    props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
    props.put("value.serializer", "io.confluent.kafka.serializers.KafkaAvroSerializer");
    props.put("specific.avro.reader", true);

    ...

    //kafka producer object
```

```
    KafkaProducer<String, ValidMessage> producer = new KafkaProducer<String,
ValidMessage>(props);
```

With the configurations of the provided snippet, the Kafka Producer will produce messages with a key with a common type (String) and a value as a complex type (Valid Message Avro Class). The serialization properties were set accordingly.

Note: The ValidMessage class can be generated via the maven avro plugin, the avsc schema is placed in a pre-defined directory and the maven plugin will generate the java classes from the schema.

5.1.6.2 Producing to the Kafka Topic

The above implementation creates a Kafka producer object with pre-defined key and value types.

The next step is to create ProducerRecord objects that contain the key and value data to be sent as well as the topic name where the data should be sent to.

Finally, the KafkaProducer method send() can be used to deliver the record to the specified kafka topic.

A kafka producer sending records to a kafka topic.

```
public void run(){

    ...


   //kafka producer object
   KafkaProducer<String, ValidMessage> producer = new KafkaProducer<String,
ValidMessage>(props);

   final ProducerRecord<String, ValidMessage> record = new ProducerRecord<>(topic, key, value);
   Future<RecordMetadata> future = producer.send(record);
```

Note: All writes are asynchronous by default. KafkaProducer method send() returns a future which can be polled to get the result of the send.

5.1.6.3 Running the Kafka Producer JVM

The main class of the JVM Kafka Producer.

```
public static void main(String[] args) {

   Configuration config = loadConfiguration(args[0]);

   AvroProducer producerClient = new AvroProducer(config);
   producerClient.run();

}
```

The load configurations method should receive a properties file as an argument and produce a Configurations object where each property is mapped to the corresponding value.

The ".run()" method will then start the producer and send the specified records to the specified kafka topics.

Running the JVM Kafka Producer on a terminal.
`$ java -jar ais-kafka-producer-1.0.0.jar producer.properties`

5.2 Consumer

5.2.1 Functionality

A JVM Kafka Consumer that is able to connect to a topic and consume messages from it.

The consumer can be a simple type key/value consumer and use the apache.kafka.common deserializers. This type of consumer can be used in most platforms that support kafka.

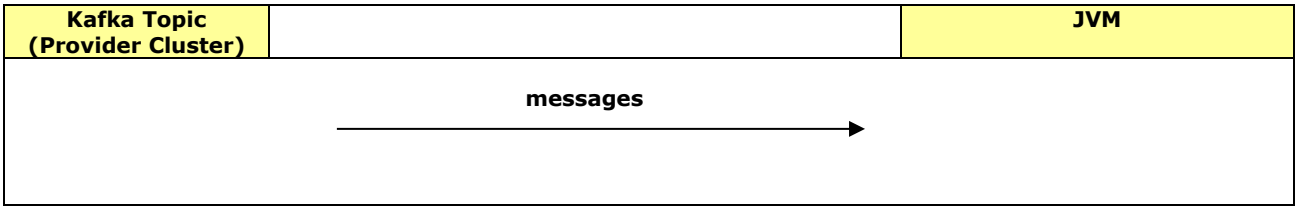
It can be a more complex consumer and use avro classes, in that case the consumer can only be used in the confluent platform (needs access to the confluent schema registry and also uses the confluent avro deserializers).

5.2.2 Consumed messages

The consumed messages' format depends on the implementation of the consumer, it can be a plaintext key/value pair or it can be an avro key/value pair.

5.2.3 Message flow

The following figure outlines the expected synchronous flow of the consumed messages.



5.2.4 Implementation

5.2.5 Properties

For the consumer to be able to consume the messages on a desired topic a few properties have to be passed, either implicitly in the java code or via a properties file.

5.2.5.1 Required Properties

bootstrap.servers

Is a comma-separated list of host and port pairs that are the addresses of the Kafka brokers.

group.id

This property defines the consumer group id, multiple consumers can be combined in a group and consume messages in a parallelized manner. The level of parallelization depends on the partitions of the Kafka topic consumed.

key.deserializer

The java class for deserialization of the message key. For example if the key is defined as a String the class passed should be "org.apache.kafka.common.serialization.StringDeserializer".

(This value is defined in the java code as it should not change)

value.deserializer

The java class for deserialization of the message value. For example if the value is defined as an Integer the class passed should be "org.apache.kafka.common.serialization.IntegerDeserializer".

(This value is defined in the java code as it should not change)

5.2.5.2 Optional Properties

schema.registry.url

The host and port address of the confluent schema registry. This property is used when consuming from a topic with a schema associated (in the confluent platform), the consumer will not be able to consume messages from the topic if this property is not passed.

specific.avro.reader

If we are consuming from a topic with an avro schema associated this property should be set as true. In that case the deserializer class for the key/value should be set accordingly. For example if the value of the message is an avro class the "value.deserializer" should be set to "io.confluent.kafka.serializers.KafkaAvroDeserializer".

(This value is defined in the java code as it should not change)

enable.auto.commit

If this property is set as true the consumer will automatically commit the offsets of the messages consumed to the Kafka brokers, otherwise the offsets must be committed manually.

auto.offset.reset

This property enable the consumer to read either from a pre-defined offset:

- The "latest" committed offset will enable the consumer to read only "new" messages (default value).
- The "earliest" offset committed will read all the available messages in the topic from the oldest committed offset.
- "none" will throw exception to the consumer if no previous offset is found for the consumer's group.

5.2.5.3 Security Properties

This is also considered an optional set of properties and should only be used if the brokers are using the SSL security protocol to access the topics.

security.protocol

Should be filled with "ssl" if it is indeed the security protocol being used, otherwise ignore the property or set it as "plaintext".

ssl.truststore.location

The location path to the ssl truststore (should only be filled when the security.protocol=ssl).

ssl.truststore.password

The ssl truststore password (should only be filled when the security.protocol=ssl).

ssl.keystore.location

The location path to the ssl keystore (should only be filled when the security.protocol=ssl).

ssl. keystore.password

The ssl keystore password (should only be filled when the security.protocol=ssl).

ssl. key.password

The ssl key password (should only be filled when the security.protocol=ssl).

5.2.5.4 Example of a properties file

Properties file passed to the Java Kafka Consumer.

```
#application
group.id=test-group

#services
bootstrap.servers=remote-hub.emsa.europa.eu:9092
schema.registry.url=http://remote-hub.emsa.europa.eu:8081

#channels
topic=test-topic

#options
enable.auto.commit=true
auto.offset.reset=latest

#Security
security.protocol=ssl
ssl.truststore.location=C:/Users/Xpand-it/IdeaProjects/main/ais-kafka-
consumer/src/main/resources/kafka.truststore.jks
ssl.truststore.password=6qVcjL8TeKraQFRH
ssl.keystore.location=C:/Users/Xpand-it/IdeaProjects/main/ais-kafka-
consumer/src/main/resources/kafka.keystore.jks
ssl.keystore.password=6qVcjL8TeKraQFRH
ssl.key.password=6qVcjL8TeKraQFRH
```

5.2.6 Implementation

5.2.6.1 Creating the Java Kafka Consumer

To create a java kafka consumer we can pass as an argument to the JVM the properties file and with this file construct a Configuration class that the consumer will use.

Then we can simply access the properties in the Configuration file to use as the properties of the consumer.

Creating a Java Kafka Consumer with the provided properties.

```
public AvroConsumer(Configuration configuration){
    this.configuration = configuration;

}

public void run(){
```

```
    //consumer properties
    Properties props = new Properties();
    props.put("bootstrap.servers", this.configuration.bootstrapServers);
    props.put("schema.registry.url", this.configuration.schemaRegistryUrl);
    props.put("group.id", this.configuration.groupId);
    //string inputs and outputs deserializers
    props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
    props.put("value.deserializer", "io.confluent.kafka.serializers.KafkaAvroDeserializer");
    props.put("specific.avro.reader", true);

    ...

    //kafka consumer object
    KafkaConsumer<String, ValidMessage> consumer = new KafkaConsumer<String, ValidMessage>(props);
```

In the provided snippet the Kafka Consumer will consume message with a key with a common type (String) and a value as a complex type (Valid Message Avro Class). The deserialization properties were set accordingly.

Note: The ValidMessage class was generated via the maven avro plugin, the avsc schema is placed in a pre-defined directory and the maven plugin will generate the java classes from the schema.

5.2.6.2 Subscribing to the Kafka Topic

The above implementation only creates a Kafka Consumer class with pre-defined key and value types.

The consumer will still have to subscribe to a kafka topic.

A Java Kafka Consumer subscribing to a Kafka Topic.

```
public void run(){

    ...


    //kafka consumer object
    KafkaConsumer<String, ValidMessage> consumer = new KafkaConsumer<>(props);


    Consumer.subscribe(Arrays.asList(this.configuration.inputTopic));
```

The subscribe method can take several topics and when subscribed to a particular set of topics each time the poll method is called the consumer will obtain the available records in each of these topics.

The signature of the consumer must be specified taking into consideration the topics being consumed. If the defined type for the consumer key/value is not

5.2.6.3 Consuming from the Kafka Topics

A Java Kafka Consumer doing poll to the subscribed topics (infinite loop)

```
while (true) {
        ConsumerRecords<String, ValidMessage> records = consumer.poll(100);
        for (ConsumerRecord<String, ValidMessage> record : records){
                String key = record.key();
                ValidMessage value = record.value();

                ...

        }
```

```
}
```

The poll method will fetch all the available messages in the subscribed topics, these messages are defined as a ConsumerRecord and each one of these is composed by a key (accessed via the ".key()" method) and a value (accessed via the ".value()" method).

If either the key or value is defined as an avro class the return values can be directly converted to the correspondent java class.

The process done to the received records at each call of poll is defined by the developer.

5.2.6.4 Running the Kafka Consumer JVM

The main class of the JVM Kafka Consumer.

```
public static void main(String[] args) {

   Configuration config = loadConfiguration(args[0]);

   AvroConsumer consumer = new AvroConsumer(config);
   consumer.run();

}
```

The load configurations method should receive a properties file as an argument and produce a Configurations object where each property is mapped to the corresponding value.

The ".run()" method will then start the connector and poll the topics until the JVM process is stopped.

Running the JVM Kafka Consumer on a terminal.

```
$ java -jar ais-kafka-consumer-1.0.0.jar consumer.properties
```