ONS JupyterHub Notebook Query Guide

Table of Contents

Service Connection Information

The ONS JupyterHub service can be accessed by navigating to https://location.officialstatistics.org/hub.

Using this service requires account credentials granted by support@officialstatistics.org.

Available Data Sources

Currently there exists 3 data sources available for users:

Data Source	Description	Dates Available
exactEarth	AIS provider for maritime vessels	2019/04/01--> Present
ORBCOMM	AIS provider for maritime vessels	2018/10/01--> Present
ADS-B Exchange	ADSB provider for aircraft flight data	2018/10/01--> Present

For each of these data sources, Optix enriches received positional observations with static message data from the live data stream. However, enrichment does not occur whenever a backfill of data is performed.

For AIS data, here is the international standards document which covers the detail of the AIS protocol and messages. Page 109 of the document (111 of the PDF) includes TABLE 48 which has some additional descriptions and possible values of the fields.

exactEarth

Attribute	Type	Description	Index
mmsi	Long	Maritime Mobile Service Identity (MMSI)	X
position	Point	WGS84 Point, Geographic Location [Geometry]	X
dtg	Date	Observation Date - yyyy-MM-dd'T'HH:mm:ssZ	X
imo	Int	International Maritime Organization (IMO)	
vessel_name	String	Vessel Name	
callsign	String	Vessel Call Sign	
vessel_type	String	Vessel Type	
vessel_type_code	Int	Vessel Type Code	
vessel_type_cargo	String	Vessel Type Cargo	
vessel_class	String	Class of Vessel (A/B)	
length	Int	Length of Bow to Main Tower and Main Tower to Stern [Meters]	
width	Int	Length of Port to Main Tower and Main Tower to Starboard [Meters]	
flag_country	String	Country of Registration	
flag_code	Int	Country of Registration Code	
destination	String	Port of Destination	
eta	String	Month, Day, Hour, and Minute of Estimated Time of Arrival in UTC [MMDDHHmm]	
draught	Double	Vessel Draught [Metres]	
longitude	Double	WGS 84 Longitude Coordinate [Decimal Degrees]	
latitude	Double	WGS 84 Latitude Coordinate [Decimal Degrees]	
sog	Double	Speed over Ground [Knots]	
cog	Double	Course over Ground [Degrees]	
rot	Double	Rate of Turn [Degrees / Min]	
heading	Double	Heading [Degrees]	
nav_status	String	Navigational Status	
nav_status_code	Int	Navigational Status Code	
source	String	Source of Position Report (S-AIS or T-AIS)	
ts_pos_utc	String	Date and Time of Last Position AIS Message in UTC [YYYYMMDDHHmmSS]	
ts_static_utc	String	Date and Time of Last Static AIS Message in UTC [YYYYMMDDHHmmSS]	
dt_pos_utc	String	Date and Time of Last Position AIS Message in UTC [YYYY-MM-DD HH24:MI:SS]	
dt_static_utc	String	Date and Time of Last Static AIS Message in UTC [YYYY-MM-DD HH24:MI:SS]	
vessel_type_main	String	Vessel Type Main	
vessel_type_sub	String	Vessel Type Sub-Category	
message_type	Int	AIS Position Message type (1,2,3,4,18,19,27)	
eeid	Long	exactEarth Identifier (eEID)	

ORBCOMM

Attribute	Type	Index
mmsi	String	X
geom	Point	X
dtg	Date	X
latitude	Double	
longitude	Double	
rawLatitude	Double	
rawLongitude	Double	
navStatus	Integer	
rot	Integer	
sog	Integer	
posAcc	Integer	
cog	Integer	
trueHeading	Integer	
specialManeuverIndicator	Integer	
utcSec	Integer	
spare	Integer	
raim	Integer	
syncState	Integer	
slotTimeout	Integer	
subMessage	Integer	
slotIncrement	Integer	
numSlots	Integer	
keep	Integer	
version	Integer	
imo	Integer	
posType	Integer	
eta	Long	
draught	Integer	
dest	String	
dte	Integer	
classBUnitFlag	Integer	
classBDisplayFlag	Integer	
classBDscFlag	Integer	
classBDandFlag	Integer	
classBMsg22Flag	Integer	
modeFlag	Integer	
commStateSelectorFlag	Integer	

commState	Integer	
raimFlag	Integer	
spare1	Integer	
spare2	Integer	
spare3	Integer	
atonType	Integer	
name	String	
dimBow	Integer	
dimStern	Integer	
dimPort	Integer	
dimStarboard	Integer	
offPosition	Integer	
regional	Integer	
virtual	Integer	
assigned	Integer	
nameExt	String	
partNumber	Integer	
vendorId	String	

ADS-B Exchange

ADS-B Exchange attribute descriptors can be found here: https://www.adsbexchange.com/datafields/

Attribute	Type	Index
Icao	String	X
geom	Point	X
dtg	Date	X
aircraft_Id	String	
Rcvr	Integer	
HasSig	Boolean	
Sig	Integer	
Reg	String	
Fseen	Date	
Tsecs	Integer	
Cmsgs	Integer	
Alt	Integer	
Galt	Integer	
InHG	Double	
AltT	Boolean	
PosTime	Date	
Mlat	Boolean	
TisB	Boolean	
Spd	Double	
SpdTyp	String	
Trak	Double	
TrkH	Boolean	
Type	String	
Mdl	String	
Man	String	
Year	Integer	
Cnum	String	
Op	String	
OpIcao	String	
Sqk	Integer	
Vsi	Integer	
VsiT	String	
WTC	String	
Species	String	
EngType	String	
EngMount	String	

Engines	String	
Mil	Boolean	
Cou	String	
From	String	
To	String	
Gnd	Boolean	
Call	String	
CallSus	Boolean	
HasPic	Boolean	
FlightsCount	Integer	
Interested	Boolean	
Help	Boolean	
Trt	String	
TT	String	
Talt	Integer	
Ttrk	Integer	
Sat	Boolean	
PosStale	Boolean	

General Query Tips

1. Whenever possible, queries should be designed to utilize one of the defined indices and additional analysis can be performed on the returned subset of data. The following are some best practices when using indices:
    a. Bounding box (BBox) + dtg queries should contain a reasonably small BBox for a reasonable time frame. "Reasonable" is kind of a loaded word as there are many factors which affect the success of this query. As a general rule of thumb, you should be able to quickly receive historical results for a BBox roughly 150km^2 for 12 hours. From this baseline, adjust the size of your BBox and time frame to obtain desired results. When making a historical query, always combine both positional and dtg fields to ensure that the index is properly utilized
    b. ID (MMSI or Icao) full historical data for a single vessel or aircraft is an acceptable query.
2. JupyterHub users share resources on a first come / first serve basis. If the Spark cluster is saturated this will adversely affect user queries.

Sample Notebooks

Sample notebooks have been provided within each user's JupyterHub directory and provide basic examples for how one could query the system. The connection parameters may be out of date, so please see below for our most recent query examples.

You will need the following 3 pieces of information to query a particular data source:

HBase Zookeeper = hbase.optix-ons-local:2181

HBase catalog = ons-historical

GeoMesa feature = <data source dependent>

The following values should be used for the corresponding data source feature:

Data Source	GeoMesa Feature
exactEarth	ee
ORBCOMM	orbcomm
ADS-B Exchange	adsbx

With this information, queries can be made using either Python or Scala.

For the provided examples in this document, please note that each code block maps to a notebook cell and are meant to be run sequentially.

Python - ADS-B Exchange

1. Setup Spark Context and Configuration

```
import geomesa_pyspark
conf = geomesa_pyspark.configure(
    jars=['/usr/lib/spark/jars/geomesa-hbase-spark-runtime_2.11-2.1.0-m.2.jar'],
    packages=['geomesa_pyspark','pytz'],
    spark_home='/usr/lib/spark/').\
    setAppName('MyTestApp')

conf.get('spark.master')
# u'yarn'

from pyspark.sql import SparkSession

spark = ( SparkSession
    .builder
    .config(conf=conf)
    .getOrCreate()
)
```

2. Connect to GeoMesa and retrieve a Dataframe backed by a GeoMesa Table

```
params = {
    "hbase.zookeepers": "hbase.optix-ons-local:2181",
    "hbase.catalog": "ons-historical"
}
feature = "adsbx"
adsbx = ( spark
    .read
    .format("geomesa")
    .options(**params)
    .option("geomesa.feature", feature)
    .load()
)

adsbx.createOrReplaceTempView("adsbx")
```

3. Number of planes that visited Dubai over time period

```
spark.sql("""
select
    count(distinct Icao) as num_planes
from adsbx
where st_contains(st_bufferPoint(st_makePoint(55.31, 25.26), 50000), geom)
    and dtg > cast('2018-10-13' as timestamp)
    and dtg < cast('2018-10-20' as timestamp)
""").show()
```

4. Query for the distinct planes around the Dubai Airport and group by day

```
spark.sql("""
select
    num_planes,
    date_format(dtg_sub, "YYYY-MM-dd") as day
from (
    select
        count(distinct Icao) as num_planes,
        /* Convert to 00:00:00 */
        date_sub(dtg, 0) as dtg_sub
    FROM (
        select Icao, dtg
        from adsbx
        where st_contains(st_bufferPoint(st_makePoint(55.31, 25.26), 50000), geom)
            and dtg > cast('2018-10-13' as timestamp)
            and dtg < cast('2018-10-20' as timestamp)
        )
    group by
        dtg_sub
    order by
        dtg_sub
)
""").show()
```

Python - exactEarth

1. Setup Spark Context and Configuration

```
import geomesa_pyspark
conf = geomesa_pyspark.configure(
    jars=['/usr/lib/spark/jars/geomesa-hbase-spark-runtime_2.11-2.1.0-m.2.jar'],
    packages=['geomesa_pyspark','pytz'],
    spark_home='/usr/lib/spark/').\
    setAppName('MyTestApp')

conf.get('spark.master')
# u'yarn'

from pyspark.sql import SparkSession

spark = ( SparkSession
    .builder
    .config(conf=conf)
    .getOrCreate()
)
```

2. Connect to GeoMesa and retrieve a Dataframe backed by a GeoMesa Table

```
params = {
    "hbase.zookeepers": "hbase.optix-ons-local:2181",
    "hbase.catalog": "ons-historical"
}
feature = "ee"
ee = ( spark
    .read
    .format("geomesa")
    .options(**params)
    .option("geomesa.feature", feature)
    .load()
)

ee.createOrReplaceTempView("ee")
```

3. Query for the last day of observations for a particular vessel mmsi

```
#result set will be very large, show only 10 results for this example
spark.sql("""
select
        dt_pos_utc, mmsi, vessel_name, longitude, latitude
FROM
        ee
WHERE
        mmsi = 366206000
    AND dt_pos_utc > date_add(current_timestamp(), -1)
""").show(10)
```

Scala - ORBCOMM

1. Import dependencies and create datastore connection

```
import org.locationtech.jts.geom._
import org.apache.spark.sql.types._
import org.locationtech.geomesa.spark.jts._

val dataFrame = spark.read.format("geomesa").option("hbase.zookeepers","hbase.optix-ons-local:2181").
option("hbase.catalog", "ons-historical").option("geomesa.feature", "orbcomm").load()
dataFrame.createOrReplaceTempView("ais")
```

2. Create temp view with all observations for a week within 100km of Dubai. Calculate the distance from that ship to the Dubai port

```
%%sql
create or replace temp view travel_log as (
    select
        mmsi,
        dtg,
        geom,
        st_distance(geom, st_makePoint(55.31, 25.26)) as distance
    from ais
    where st_contains(st_bufferPoint(st_makePoint(55.31, 25.26), 100000), geom)
        and dtg > cast('2018-11-13' as timestamp)
        and dtg < cast('2018-11-20' as timestamp)
    order by
        mmsi,
        dtg
)
```

3. Show results

```
spark.sql("""
select * from travel_log order by distance desc
""").show()
```

4. If we consider 50km as "Arriving" at the port we can look for ships that have moved from outside this radius and disregard passing ships

```
spark.sql("""
select count(*)
from (
    select
        *,
        lag(distance, 1) over (order by mmsi, dtg) as last_distance
    from travel_log
)
where
    last_distance > .5
    and distance < .5
""").show()
```