LIQUID ROBOTICS
A Boeing Company

Wave Glider Management System Data Service User Guide

WDS Documentation

Revision 05
Item Number: 11675

Trademarks

Patents

The Wave Glider mechanism is patented under US Patent Number 7,371,136 and other patents pending. All aspects of the Wave Glider product have been developed at private expense by Liquid Robotics.

Warranty

The information contained herein is subject to change without notice. The only warranties for Liquid Robotics, Inc. products and services are set forth in Liquid Robotics Terms and Conditions Articles 11.1 Limited Hardware Warranties, 11.2. Limited Service Warranties, and 11.3 Limitations. Nothing herein should be construed as constituting any additional warranties or representations. Liquid Robotics, Inc. shall not be liable for technical or editorial errors or omissions contained herein.

Contact

LIQUID ROBOTICS
A Boeing Company

460 Herndon Parkway

Herndon, VA 20170

Phone: +1 408-636-4200

Customer Support: +1 888-574-4574

Fax: +1 408-747-1923

Revision History

Document Part Number	Revision	Publication Date	Changes
11675	5	Jun 2022	• Removed Logging and Data Metering • Updated Documented Workarounds
	4	Apr 2022	• Updated References • Removed Legacy Data Portal known issues
	3	Sep 2021	• Added Streaming Features & Commands, Implementation Examples • Modified Overview to include Streaming
	2	May 2021	• Include references to sensor guides for metadata information • Formatted subheadings • Update Open Oceans reference to script library and PN 11884 • Capture default report to all gliders & orgs. • Specify all requests must specify the “vid” as the unique vehicle ID. • Include pointer and definitions of formatted data types • Update to include figures and captions. • Include date range limit • Include note for JSON formatting in Excel • Including list of known bugs and workarounds
	1	March 2021	Initial Release

References

Document Part Number	Revision
05910	WGMS User Guide
11884	WGMS Data Service (WDS) Examples
05573	Wave Glider® Integrated Sensor User Guide Turner C3 Submersible Fluorometer Sensor Software
05574	Wave Glider® Integrated Sensor User Guide Sea-Bird Electronics Glider Payload CTD Sensor Software
06258	Wave Glider® Integrated Sensor User Guide GPSWaves Sensor Software
08556	Wave Glider® Integrated Sensor User Guide Vemco VR2C Sensor Software
08640	Wave Glider® Integrated Sensor User Guide Teledyne RD Instruments Workhorse Monitor ADCP Sensor Software
09262	Wave Glider® Integrated Sensor User Guide RUDICS Software

Contents

1 About this Document

The purpose of this document is to provide users guidance to operate the Wave Glider Management System (WGMS) Data Service (WDS).

2 Overview

The WDS allows users to access and export WGMS data with successful authentication via stateless programmatic requests. The primary mechanism utilizes Simple Object Access Protocol (SOAP) in order to access available data in WGMS Data Storage, including sensor data, raw data, weather data, and the Adaptable Modular Power System (AMPS) data. The secondary mechanism utilizes Remote Procedure Call (RPC) to initiate near-real-time streaming of data from a Wave Glider. Users with WGMS or Enterprise WGMS obtain access through controlled credentials and can access their specific organizations and respective Wave Glider Vehicles. Several reports can be requested and exported in JavaScript Object Notation (JSON) format by vehicle including sensor or data type and time range.

3 Features & Commands

All available commands in the data service utilize stateless, web-based requests, accessible via https://dataservice.wgms.com/WDS/WGMSData.asmx. The service description is available in the Web Service Definition Language (WSDL) format via https://dataservice.wgms.com/WDS/WGMSData.asmx?WSDL. The below are examples illustrating placeholders within SOAP requests. The placeholders shown need to be replaced with actual values.

Using applications and tools such as curl commands or 3rd party extensions like SOAP WSDL analyzers or SOAP clients may help to construct requests and use the WDS programmatically.

The following depicts an example through the use of *Boomerang*, a 3rd party *Chrome* extension. The SOAP and WSDL analyzers will list the available operations and assist in building requests through defined parameters, such as org, user, and password.

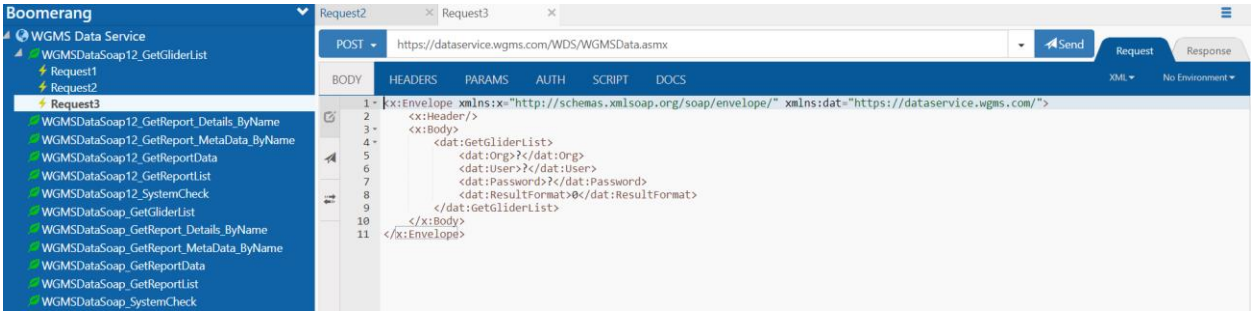


Figure 1 - Using Boomerang to access WDS

3.1 Authentication

Authentication through WDS is handled via WGMS credentials. Each stateless request will consist of user's 'org', username of type string, and password of type string. The WDS differs from Legacy Data Portal, which utilized web-based sessions for exporting and streaming of data. Each SOAP request must include user credentials to obtain access to available vehicle or sensor data.

Contact Liquid Robotics for access to WGMS and appropriate organization access.

3.2 Available Vehicles

WDS allows users to obtain the available Wave Glider Vehicles within their organization(s) in alignment with WGMS.

The following depicts the example request and expected response:

GetGliderList Request

```
POST /WDS/WGMSData.asmx HTTP/1.1
Host: dataservice.wgms.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
```

```
    <GetGliderList xmlns="https://dataservice.wgms.com/">
      <Org>string</Org>
      <User>string</User>
      <Password>string</Password>
      <ResultFormat>int</ResultFormat>
    </GetGliderList>
  </soap12:Body>
</soap12:Envelope>
```

GetGliderList Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetGliderListResponse xmlns="https://dataservice.wgms.com/">
      <GetGliderListResult>string</GetGliderListResult>
      <Error>boolean</Error>
    </GetGliderListResponse>
  </soap12:Body>
</soap12:Envelope>
```

The GetGliderList Response will follow the following metadata formatting:

```
"vehicleName": "String",
"vid": "Integer",
"created": "String"
```

Note: The ResultFormat parameter is default and does not impact the request in this initial release. It is an unused flag that may be utilized in follow-on releases.

3.3 Available Reports

WDS offers multiple reports to choose for export pending data availability in the WGMS Data Storage.

Note: Users should utilize the unique identifier “vid” for vehicle ID when requesting data. This ensures that consistent naming even when vehicles may have been moved or renamed.

All vehicles in their respected orgs will have a set of default reports. The default reports include raw data, telemetry 6 reports, and AMPS data. All other data are specific or vehicle and organization.

Note: There is a known issue in Telemetry 6 Reports when utilizing the WDS. An analysis of Telemetry 6 when compared to WGMS Exports revealed the following:

- The WDS variables of ShoreSideCreated, LastLocationFix, TemperatureSub, LightState, OceanCurrent, and OceanCurrentDirection have no comparable fields in a WGMS Export.
- WDS `GliderDistance` has no equivalent in WGMS Export although *Distance Over Ground* is close.
- WDS `GliderSpeed` has no equivalent in WGMS Export although *Speed Over Ground* is close

- WDS `HeadingFloatDegrees` has no equivalent in WGMS Export although *Desired Heading* and *Sub Heading* are close.
- WDS `SurfaceTemperature` is called *Float Temp* in WGMS Export.
- WDS `GliderHeading` is called *Path Over Ground* in WGMS Export.
- WDS `DesiredBearingDegrees` is called *Desired Heading* in WGMS Export.

Data may be requested in date ranges within 90 days but we recommend limiting it to 30 days, as large data sets can affect performance, induce time outs, and inadvertently consume your allotted data usage. To obtain more data, users must make additional requests.

A listing of data reports may be obtained through the following request and expected response:

GetReportList Request

```
POST /WDS/WGMSData.asmx HTTP/1.1
Host: dataservice.wgms.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetReportList xmlns="https://dataservice.wgms.com/">
      <Org>string</Org>
      <User>string</User>
      <Password>string</Password>
      <ResultFormat>int</ResultFormat>
    </GetReportList>
  </soap12:Body>
</soap12:Envelope>
```

GetReportList Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetReportListResponse xmlns="https://dataservice.wgms.com/">
      <GetReportListResult>string</GetReportListResult>
      <Error>boolean</Error>
    </GetReportListResponse>
  </soap12:Body>
</soap12:Envelope>
```

An example export from the training org returns the following report list:

```
"Raw Data Report","Telemetry 6 Report","Amps Power Summary
Report","Amps Output Power Status Report","Amps Solar Input Port
```

```
Report","Amps Battery Port Report","Amps Bridge Power Port
Report","Test Amps Power Summary Report","ADCP Sensor
Bathymetry","ADCP Sensor C2 Samples","ADCP Sensor C2T Samples","ADCP
Sensor C4 Samples","ADCP Sensor C4T Samples","ADCP Sensor Header
Samples","Datawell MOSE Records","GPS Waves Sensor","Aquatracka
Samples","Fluorometer Samples (C3)","Seabird CTD Records","VEMCO Vr2C
Status","VEMCO Vr2c Tags"
```

Report details are additionally available through the following request and response.

GetReport_Details_ByName Request

```
POST /WDS/WGMSData.asmx HTTP/1.1
Host: dataservice.wgms.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetReport_Details_ByName xmlns="https://dataservice.wgms.com/">
      <Org>string</Org>
      <User>string</User>
      <Password>string</Password>
      <ReportName>string</ReportName>
      <ResultFormat>int</ResultFormat>
    </GetReport_Details_ByName>
  </soap12:Body>
</soap12:Envelope>
```

GetReport_Details_ByName Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetReport_Details_ByNameResponse
xmlns="https://dataservice.wgms.com/">

<GetReport_Details_ByNameResult>string</GetReport_Details_ByNameResult
>
      <Error>boolean</Error>
    </GetReport_Details_ByNameResponse>
  </soap12:Body>
</soap12:Envelope>
```

An example response from the training org returns the following details:

```
"OrgId":0,"OrgName":"","ReportName":"Raw Data
Report","DisplayOrder":0,"MetaDataViewId":0,"ColumnXml":"","ColumnXmlD
```

efinition":"","PayloadType":0,"SensorDataFormat":0,"ReportType":1,"err
or":false

3.4 JSON Exports

Obtaining report data is executed by the following request and expected response. The Data Service serves responses in JSON Format. Depending on the tool sets, users may export the data or copy into a text file for JSON and utilize Microsoft Excel to Parse.

Using older versions of Microsoft Excel (2013), you can connect to a JSON file via selecting **Data**, navigating to **Power Query**, **From Other Sources** & Selecting **Blank Query**. Using the Advanced Editor, update the query string to point to your file & path:

```
let
    Source =
Json.Document(File.Contents("C:\Users\Name\Desktop\JSONTest.json")),
    #"Converted to Table" = Record.ToTable(Source)
in
    #"Converted to Table"
```

In newer versions of Excel, Select **Data**, **Get Data**, **From File**, **From JSON**, and choose **Import Data**. Navigate to the JSON file, and select **Open**.

GetReportData Request

```
POST /WDS/WGMSData.asmx HTTP/1.1
Host: dataservice.wgms.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetReportData xmlns="https://dataservice.wgms.com/">
      <Org>string</Org>
      <User>string</User>
      <Password>string</Password>
      <ReportName>string</ReportName>
      <SerialNo>string</SerialNo>
      <StartDate>dateTime</StartDate>
      <EndDate>dateTime</EndDate>
      <ResultFormat>int</ResultFormat>
    </GetReportData>
  </soap12:Body>
</soap12:Envelope>
```

GetReportData Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
```

```
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetReportDataResponse xmlns="https://dataservice.wgms.com/">
      <GetReportDataResult>string</GetReportDataResult>
      <Error>boolean</Error>
    </GetReportDataResponse>
  </soap12:Body>
</soap12:Envelope>
```

An example response of the *Raw Data Report* returns the following metadata:

```
 "metaData": {
    "vehicleName": "String",
    "vid": "Integer",
    "payloadLength": "Integer",
    "payload": "Byte[]",
    "gliderTimeStamp": "DateTime",
    "ascensionTime (Unix)": "Integer",
    "ascensionTime": "DateTime",
    "shoresideCreated": "DateTime",
    "structureType": "Integer",
    "source": "String",
    "satelliteLatitude": "Decimal",
    "satelliteLongitude": "Decimal",
    "satelliteCEPRadius": "Decimal"
  }
```

Timestamps

Note: When using the data service or WGMS timestamps, multiple times are reported. The a*scensionTime* is reported by Iridium and is mostly useful for diagnostics and troubleshooting. The *gliderTimestamp* maps to when the data is created onboard the vehicle. The *shoreSideCreated* time represents when the sensor measurements are instantiated into the shore side database. The *shoresideCreated* and *gliderTimeStamp* may be used in conjunction to determine end-to-end latency.

The following is an example XML request. It should be used with the GetReportData WebMethod for the WGMS Data Service.

Sample GetReportData Request

```
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetReportData xmlns="https://dataservice.wgms.com/">
      <Org>[put your org here]</Org>
      <User>[username]</User>
      <Password>[password]</Password>
      <ReportName>[reportname: e.g "Raw Data Report"]</ReportName>
      <SerialNo>[vid]</SerialNo>
      <StartDate>2020-11-01T00:00:00Z</StartDate>
      <EndDate>2020-11-30T00:00:00Z</EndDate>
```

```
      <ResultFormat>1</ResultFormat>
    </GetReportData>
  </soap12:Body>
</soap12:Envelope>
```

Users may write out the XML file to disk manually or send it via a curl command using

```
curl -d @[Request file] -H "Content-type: application/soap+xml" [url]
> [outfile.xml]
```

Example Curl Command

```
curl-d @request.xml -H "Content-type: application/soap+xml"
http://localhost:59128/WGMSData.asmx?op=GetReportData> result-1mo.json
```

Data Formatting

Note: Refer to *PN 04438* for details on message format details.

Sensor Sample Record (Unparsed)

All sensors must locate each sample in space and time using the following format. The software managing the sensor is responsible for providing the timestamp and the lat/lon. The SMC (and Payload Interface Boards (PIBs) - these are SV2 Sensor Port Payloads adapted for SV3) should provide callable routines that report these values in a form that can be directly inserted into sensor sample records. Message types 0x90 and 0x91 provide for repeat counts for multiple samples within a message.

Field Name	Bytes Used	Byte #	Format
Latitude	4	0-3	Signed 32-bit integer
Longitude	4	4-7	Signed 32-bit integer
Unix Timestamp	4	8-11	Signed 32-bit integer
Sensor Data	[variable]	12-	

Telemetry Type 148 (Payload 0x94)

Telemetry Type 148-This message format is intended to report generic minimally-parsed sensor telemetry and make the maximum space available for the payload. The payload type ID uniquely identifies the type of sensor; the board ID is intended to uniquely identify the computer on the Wave Glider that is reporting the data; the task ID is intended to identify a sub-component on the computer that is reporting the data. The combination of board ID and task ID identify the reporting instance. This type of message can be used to pass data to a custom service or external client on shore.

This message format is a truncated version of type 0x91 meant to pass data through the shore-side infrastructure to clients. Sensor data passed using type 0x94 is not parsed by shore-side systems.

Field Name	Bytes Used	Byte #	Format
PayloadType	4	0-3	See Telemetry type 0x80
Board Id	1	4	Unsigned 8-bit integer
Task Id	1	5	Unsigned 8-bit integer
Sensor Data payload	[variable]	6-	Unsigned 4-bit integer

Standard Telemetry Data 0x06 is formatted per the following:

Field Name	Byte Index	Size	Bit Index
Sub Leak Alarm	0	1	0
Float Leak Alarm	0	1	1
Sub to Float Comms Alarm	0	1	2
Payload Error Condition Alarm	0	1	3
Float Battery Low Alarm	0	1	4
Umbilical Fault Alarm	0	1	5
Sub Pressure Threshold Alarm	0	1	6
GPS Not Functioning	0	1	7
Internal Vehicle Comm Error	1	1	0
Sub Reboot Alarm	1	1	1
Float to Sub Comms Error	1	1	2
Over Current Error	1	1	3
Float Temperature Threshold Exceeded	1	1	4
Float Pressure Threshold Exceeded	1	1	5
Sub Temperature Threshold Exceeded	1	1	6
Float Reboot Alarm	1	1	7
Temp_Surface	2	2	
Latitude	4	4	
Longitude	8	4	
Last_Fix_Time_Hours	12	1	
Last_Fix_Time_Minutes	13	1	
Last_Fix_Time_Seconds	14	1	
Last_Fix_Date_Month	15	1	
Last_Fix_Date_Day	16	1	
Last_Fix_Date_Year	17	1	
File Read Transaction Timeout Error	18	1	0
File Write Fallback Image Activated	18	1	1
File Write Program Image Burned	18	1	2
File Write Fallback Image Bad or Not Found	18	1	3
File Write Selected Image Bad or Not Found	18	1	4
File Write Tentative Image Retry	18	1	5
Boot Config Corrupt	18	1	6
IPIB Not Responding	18	1	7
Temperature_Sub	19	2	
Distance_Over_Ground (m)	21	2	
Rudder Count	23	4	
Heading_Sub (degrees)	27	2	
Iridium_Signal_Strength	29	1	
Target_WayPoint	30	1	
Desired_Bearing (Degrees)	31	2	
Heading_Float (Degrees)	33	2	
Pressure_Sensor_Float	35	1	
Pressure_Sensor_Sub	36	1	

Field Name	Byte Index	Size	Bit Index
Navigation_Mode	37	1	See Nav Modes in *PN 04438*
Light_State	38	1	0
IR_State	38	1	1
Xbee_State	38	1	2
Payload 1 Power	38	1	3
Payload 2 Power	38	1	4
Sub Payload Power	38	1	5
Bubble Com Error	38	1	6
Battery Com Error	38	1	7
Reset_Flags	39	1	
Total_Power (10mWh)	40	2	
Iridium_Read_Errors	42	1	
Iridium_Write_Errors	43	1	
Iridium_Session_Errors	44	1	
Water Speed	45	2	
Water Direction	47	2	
Water Speed Std Dev	49	1	
Water Direction Std Dev	50	1	
Humidity	51	1	
Payload 3 Status	52	1	0
Payload 4 Status	52	1	1
PEP Status	52	1	2
IPIB Status	52	1	3
Modem Power	52	1	4
Increased Mailbox Checks	52	1	5
Reserved (SV3 – Thruster on/off)	52	1	6
Reserved (SV3 – Auto Evading on/off)	52	1	7

3.5 Metadata

When requesting for metadata, WDS will return a Metadata report by name. Available metadata includes but is not limited to tags, descriptions, vehicle identifiers or parameters, units, data field names, and data field descriptions.

Some metadata is specific to the sensor. Refer to the specific sensor user guides for details on metadata as captured in References.

The following depict the request and response.

GetReport_MetaData_ByName Request

```
POST /WDS/WGMSData.asmx HTTP/1.1
Host: dataservice.wgms.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
```

```
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetReport_MetaData_ByName xmlns="https://dataservice.wgms.com/">
      <Org>string</Org>
      <User>string</User>
      <Password>string</Password>
      <ReportName>string</ReportName>
      <ResultFormat>int</ResultFormat>
    </GetReport_MetaData_ByName>
  </soap12:Body>
</soap12:Envelope>
```

GetReport_MetaData_ByName Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetReport_MetaData_ByNameResponse
xmlns="https://dataservice.wgms.com/">

<GetReport_MetaData_ByNameResult>string</GetReport_MetaData_ByNameResu
lt>
      <Error>boolean</Error>
    </GetReport_MetaData_ByNameResponse>
  </soap12:Body>
</soap12:Envelope>
```

3.6 System Check & Diagnostics

The WDS offers diagnostics to assist with data access and problem troubleshooting.

The following depict system check request and response.

SystemCheck Request

```
POST /WDS/WGMSData.asmx HTTP/1.1
Host: dataservice.wgms.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <SystemCheck xmlns="https://dataservice.wgms.com/" />
  </soap12:Body>
</soap12:Envelope>
```

SystemCheck Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <SystemCheckResponse xmlns="https://dataservice.wgms.com/">
      <SystemCheckResult>string</SystemCheckResult>
    </SystemCheckResponse>
  </soap12:Body>
</soap12:Envelope>
```

3.7 Known Issues

Liquid Robotics feeds issues and bugs to the product backlog based on priority and available workaround. The following lists the known issues that are deferred until future release.

- ADCP C2T Data Bin Count = 30 but only 25 bins are served. Please contact Liquid Robotics support if this is an issue for you.
- Sensor Report Types NOT included in field definitions for AMPS Power Summary Report
- Large data pulls from queries may result in timeouts on your application

4 Example Use Cases

4.1 Connecting MATLAB to the WGMS Data Service

The purpose of this section is to provide detailed instructions on how to interface with the WGMS Data Service using Matlab. The following information was adapted from MathWorks Help Center, in particular their description of the matlab.wsdl.createWSDLClient function. This function creates an interface to SOAP-based web services. The webpage can be accessed with this link.

1. Download and install AdoptOpenJDK:
    a. From your internet Browser go to: adoptopenJDK.net.
    b. Select version and JVM, then download the file. (Note: At the time this document was written, the version used was OpenJDK 8 (LTS) and the JVM used was HotSpot.)
    c. Install AdoptOpenJDK
    d. Record the path of the installation, it will be needed later in the process.

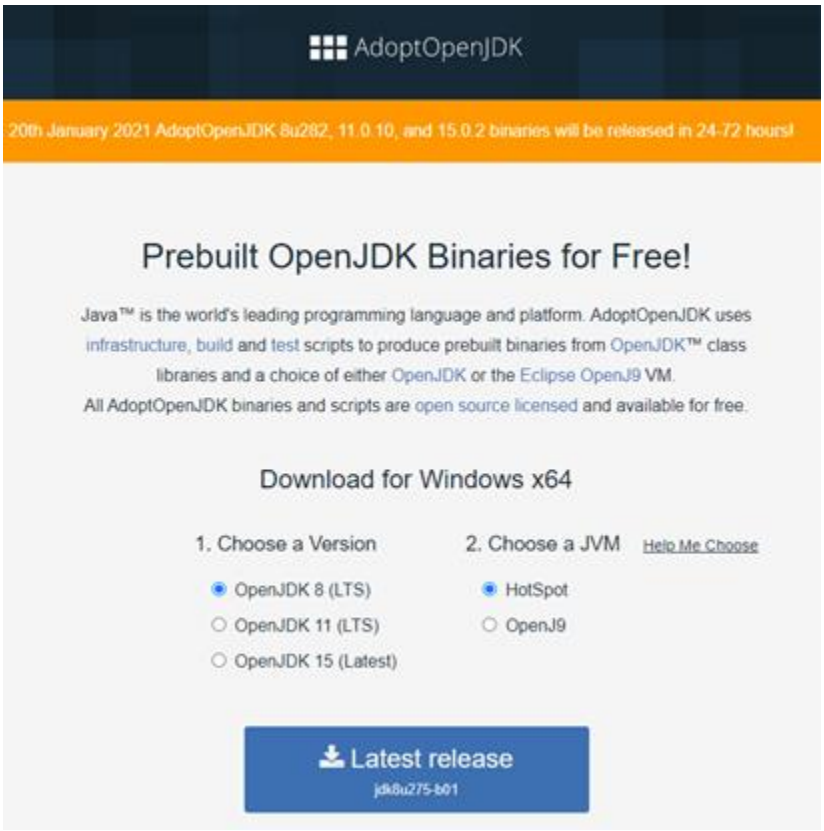


Figure 2 - AdoptOpenJDK for MATLAB & WDS

2. Download ApacheCXF:
    a. Download ApacheCXF using this link. (Note: There are newer versions of ApacheCXF but Matlab WSDL requires version 3.2.14.
    b. Unzip the compressed folder
    c. Record the path, it will be needed later in the process.

Note: The rest of this process will be executed from within the Matlab command window.

3. Assign a path to the OpenJDK software:

   Ex. If the path to the JDK software is C:\Program Files\AdoptOpenJDK\jdk-8.0.275.1-hotspot, assign a variable, jdk, in Matlab to that path.

   ```
   >>jdk = 'C:\Program Files\AdoptOpenJDK\jdk-8.0.275.1-hotspot'
   ```

4. Assign a path to the Apache CXF Software:

   Ex. If the path to Apache CXF software is C:\Program Files\apache-cxf-3.2.14 assign a variable, cxf, in Matlab to that path.

   ```
   >>cxf = 'C:\Program Files\apache-cxf-3.2.14'
   ```

5. Execute the following command:

   ```
   >>matlab.wsdl.setWSDLToolPath('JDK',jdk,'CXF',cxf)
   ```

6. Execute the following command:

   ```
   >>matlab.wsdl.createWSDLClient('https://dataservice.wgms.com/wds?wsdl')
   ```

7. Verify WGMSData.m and WGMSData1.m were created in the working directory.

   (Note: WGMSData.m is for use with SOAP 1.2 and WGMSData1.m is for use with SOAP 1.1.)

8. Add the directory where the classes were created to the Java path by executing the following command:

   ```
   >>javaaddpath('.\+wsdl\wds.jar')
   ```

The methods within WGMSData.m and WGMSData1.m can now be used to extract data from the WGMS Data Service. The methods including the inputs and outputs are described in the *.m files and are shown below. The output of GetReportList, GetGliderList, and GetReportData are JSON formatted. To convert that output to a Matlab structure, use the jsondecode.m function.

***Note**: DO NOT edit these *.m files*

9. All the methods require the WGMSData Object as an input variable. To create the WGMSData Object:
   a. Execute the following command:

      ```
      >>obj = WGMSData
      ```

   b. Verify the Endpoint and WSDLFile:

   ```
   Endpoint: '{https://dataservice.wgms.com/}WGMSData'
   WSDLFile: 'https://dataservice.wgms.com/wds?wsdl'
   ```

10. The SystemCheck is not necessary for retrieving data but can be useful for troubleshooting. To run the system check:
    a. Execute the following command:

```
>>SystemCheckResult = SystemCheck(obj)
```

    b. Verify the SystemCheckResult is Nominal:

```
SystemCheckResult =

    'WGMS Data Service System is Nominal
```

11. GetReportList: This function provides the report names for all the data available in the WGMS Org. *(Note: The report names shown in GetReportListResult are not necessarily available for <u>all</u> vehicles)*
    a. Execute the following command:
       i. `>>[GetReportListResult,Error] = GetReportList(obj,'Org','User','Password',ResultFormat)`
       ii. `Where obj = The WGMSData object (step 9)`
       iii. `Org = String representing the WGMS Org you are requesting data from`
       iv. `User = String representing your username`
       v. `Password = String representing your password`
       vi. `ResultFormat = 1`
    b. Verify the output in GetReportListResult and Error = 0. The output in GetReportListResult is used as the input variable "ReportName" in calls to the GetReportData function (step 13).
12. GetGliderList: This function provides a list of vehicle names and unique IDs from a WGMS Org.
    a. Execute the following command in the Matlab Command Window:
       i. `>> [GetGliderListResult,Error] = GetGliderList(obj,Org,User,Password,ResultFormat)`
       ii. `Where obj = The WGMSData object (step 9)`
       iii. `Org = String representing the WGMS Org you are requesting data from`
       iv. `User = String representing your username`
       v. `Password = String representing your password`
       vi. `ResultFormat = 1`
    b. Verify the Outputs, GetGliderList Result and Error = 0 in the Matlab Command Window. The output is used as the input variable "SerialNo" in calls to the GetReportData function (step 13).

13. GetReportData: This function retrieves data from the WGMS Data Service for the selected vehicle and the selected report name.
    a. Execute the following command in the Matlab Command Window
       i. `>> [GetReportDataResult,Error] = GetReportData(obj,Org,User,Password,ReportName,SerialNo,StartDate,EndDate,ResultFormat)`
       ii. `Where obj = The WGMSData object (step 9)`
       iii. `Org = String representing the WGMS Org you are requesting data from`

iv. User = String representing your username
v. Password = String representing your password
vi. ReportName = String representing the ReportName (Step 11b)
vii. SerialNo = String representing the vehicle unique ID (Step 12b)
viii. Start Date = The beginning time bound in the format dd-mmm-yyyy HH:MM:SS
ix. End Date = The end time bound in the format dd-mmm-yyyy HH:MM:SS
x. ResultFormat = 1

The format of StartDate and EndDate is explained in Table 1 of the help for the datestr.m function.

b. Verify the outputs in GetReportDataResult and Error = 0.

14. JSON to Matlab: To convert the data from JSON to a Matlab structure, run the following command:

```
Data = jsondecode(GetReportDataResult)
```

4.2 Python Script to Interface with the WGMS Data Service

Setup

If a customer has python and pip installed on their system this Python script is meant to be an easy to use interface for customers to access Data Service.

After downloading the script the user will need to install the 'zeep' module:

```
pip3 install zeep
```

The user needs to modify certain constants such as org, user, and password in the source code.

```
15  # Constants. TODO: move to a config file
16  wsdl           = "https://dataservice.wgms.com/WDS/WGMSData.asmx?wsdl" # Data Service WSDL URL
17  org            = "lriops"                                              # WGMS Org
18  user           = "XXXXX.XXXXXXXX"                                      # WGMS Org user
19  password       = "XXXXXXXXXXXXXXXXXXX"                                 # WGMS Org Password
20  reportName     = "Raw Data Report"                                     # Data to be sent back
21  resultFormat   = 1                                                     # Default Option
```

Figure 3 - Python scripting example, modified constraints

The script requires the user to add inputs to determine the commands that are to be sent to Data Service.

```
python3 DataService.py --help
```

```
theja@theja-VirtualBox:~/dev/DataService$ python3 DataService.py --help
usage: DataService.py [-h] [--getGliderList] [--getReportList] [--getReportData] [--getPeriodicData] [--vehicles [VEHICLES [VEHICLES ...]]] [--startDate STARTDATE] [--endDate ENDDATE]
                      [--interval INTERVAL] [--time TIME]

Python script to utilize the WGMS Data Service.

optional arguments:
  -h, --help            show this help message and exit
  --getGliderList       Command to get list of gliders in org.
  --getReportList       Command to get list of reports.
  --getReportData       Command to get raw data from a specified vehicle in a date range. Requires user to specify startDate and endDate.
  --getPeriodicData     Command to get data over a specified period of time at specified intervals. Requires user to specify interval and time
  --vehicles [VEHICLES [VEHICLES ...]]
                        The vehicle serial number. For multiple vehicle spaces needed between numbers.Example --vehicles 123 456 789
  --startDate STARTDATE
                        Start date and time of the requested data. Format: YYYY-mm-DDTHH:MM:SSZ
  --endDate ENDDATE     End date and time of the requested data. Format: YYYY-mm-DDTHH:MM:SSZ
  --interval INTERVAL   Length of time interval in minutes.
  --time TIME           Length of time you want the python script to run in minutes.
```

Figure 4 - Python script example, Help

GetGliderList:

To get a list of all the gliders in an Org the user would send the following command:

`python3 DataService.py --getGliderList`

Example:


```
theja@theja-VirtualBox:~/dev/DataService$ python3 DataService.py --getGliderList
{
  "metaData": {
    "vehicleName": "String",
    "unique ID": "Integer",
    "created": "String"
  },
  "recordData": [
    {
      "vehicleName": "A'a",
      "unique ID": 860481287,
      "created": "04/26/2012"
    },
    {
      "vehicleName": "Alex",
      "unique ID": 787336913,
      "created": "10/07/2011"
    },
    {
      "vehicleName": "Aquaman {dry docked}",
      "unique ID": 123456789,
      "created": "06/28/2013"
    },
    {
      "vehicleName": "Aukai (Moved to PCA Org 180328)",
      "unique ID": 282,
      "created": "08/26/2009"
    },
    {
      "vehicleName": "Benjamin (PCX_2)",
      "unique ID": 498388940,
      "created": "10/06/2011"
    },
    {
      "vehicleName": "Black Widow",
      "unique ID": 648307211,
      "created": "05/24/2013"
    },
    {
      "vehicleName": "Captain America",
      "unique ID": 1515151130,
      "created": "06/06/2013"
    },
    {
      "vehicleName": "Cosmo",
      "unique ID": 1376289541,
      "created": "07/28/2016"
    },
    {
      "vehicleName": "Deliverance (SV3-034) moved back to LRI",
      "unique ID": 16977098,
      "created": "04/25/2014"
    },
    {
      "vehicleName": "Dragon",
```


Figure 5 - Python example, getGliderList

GetReportList:

To get a list of all the reports that can be requested from Data Service the user would send the following command:

```
python3 DataService.py --getReportList
```

Example:

Figure 6 - Python example, getReportList

GetReportData:

To get data from a specified list of vehicles over a time period the user would send the following command:

```
python3 DataService.py --getReportData --vehicle [list of vehicle serial
numbers] --startDate [start date and time in YYYY-mm-DDTHH:MM:SSZ] --endDate
[end date and time in YYYY-mm-DDTHH:MM:SSZ]
```

NOTE: Requesting data from multiple specified vehicles is a workaround till milestone 3 when Data Service will natively support multiple vehicles. The script makes separate GetReportData calls for each vehicle specified.

Example for getting data from 1 vehicle:

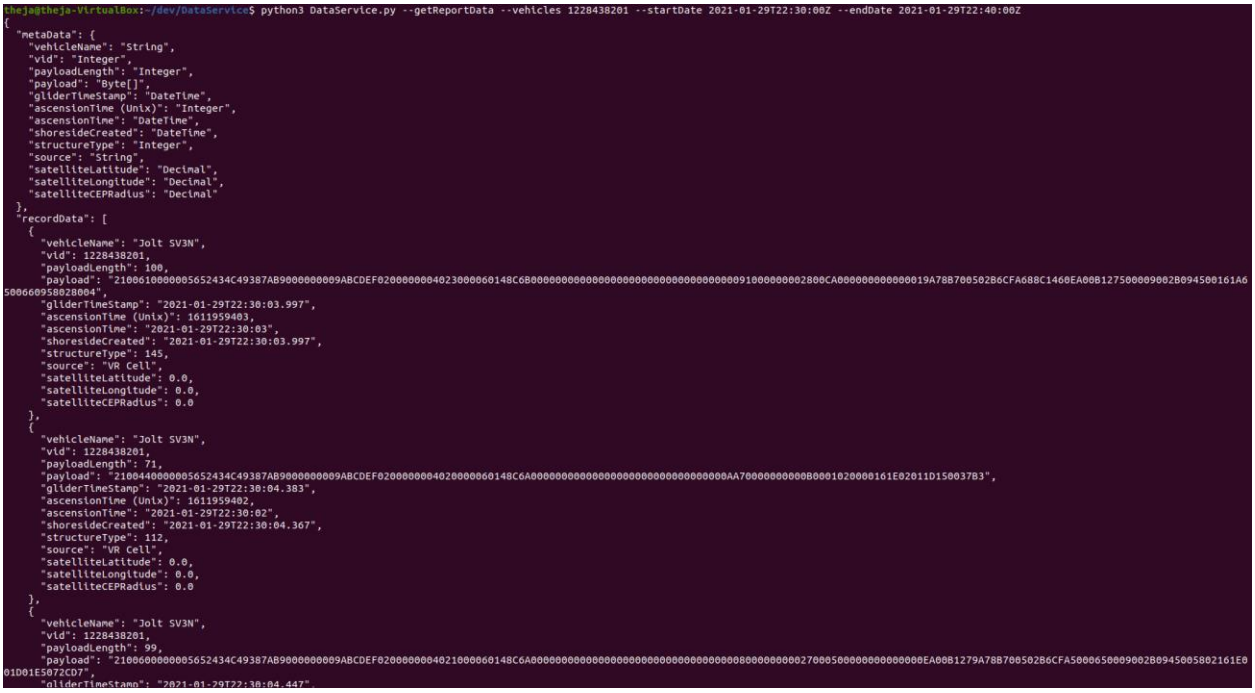


Figure 7 - Python example, getReportData 1

Example for getting data from multiple vehicles:

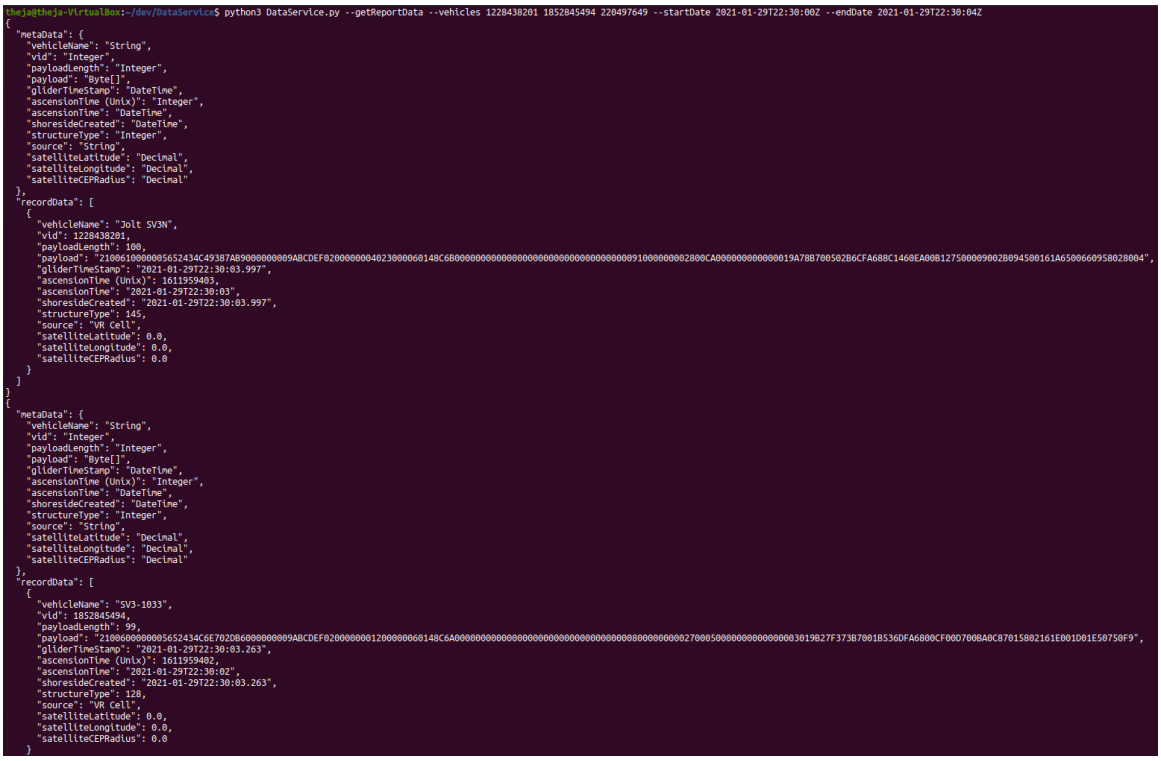


Figure 8 - Python example, getReportData 2

Figure 9 - Python example, getReportData 3

To get data from all the vehicles in the org simply omit the vehicles option.

```
python3 DataService.py --getReportData --startDate [start date and time in
YYYY-mm-DDTHH:MM:SSZ] --endDate [end date and time in YYYY-mm-DDTHH:MM:SSZ]
```

Example:

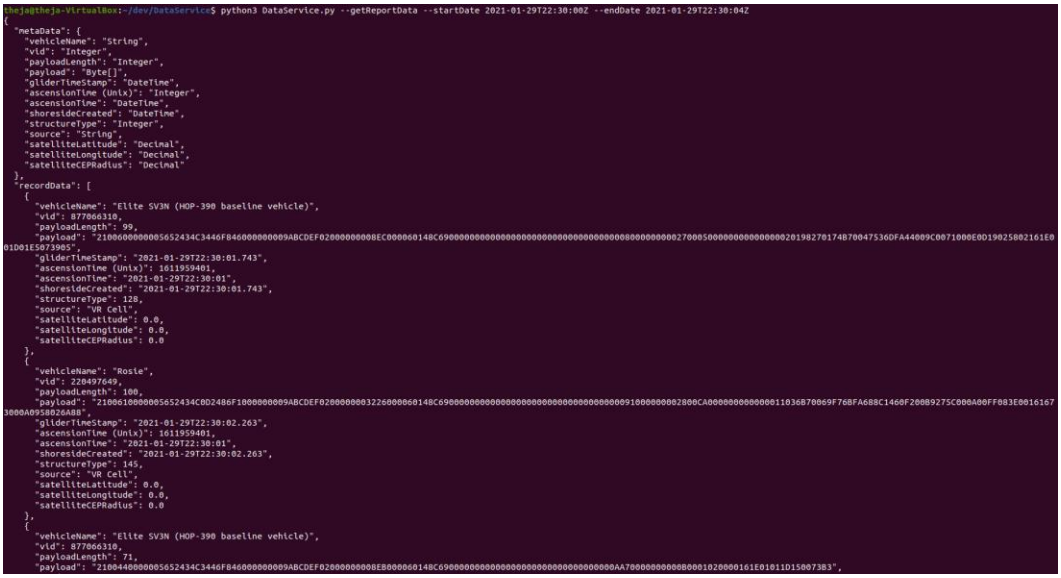


Figure 10 - Python example, getReportData 4

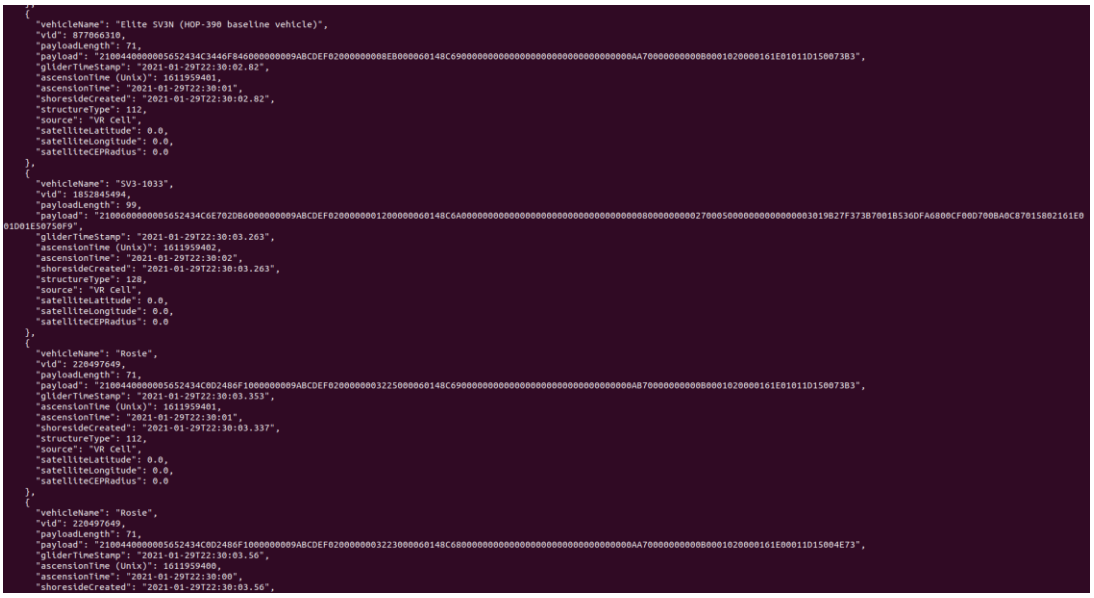


Figure 11 - Python example, getReportData 5

GetPeriodicData:

This call is to mock streaming and allows users to get data back from specified vehicles at a set interval over a period of time.

```
python3 DataService.py --getPeriodicData --vehicle [list of vehicle serial numbers] --interval [time in minutes to get data] --time [time in minutes the script will run]
```

Example:

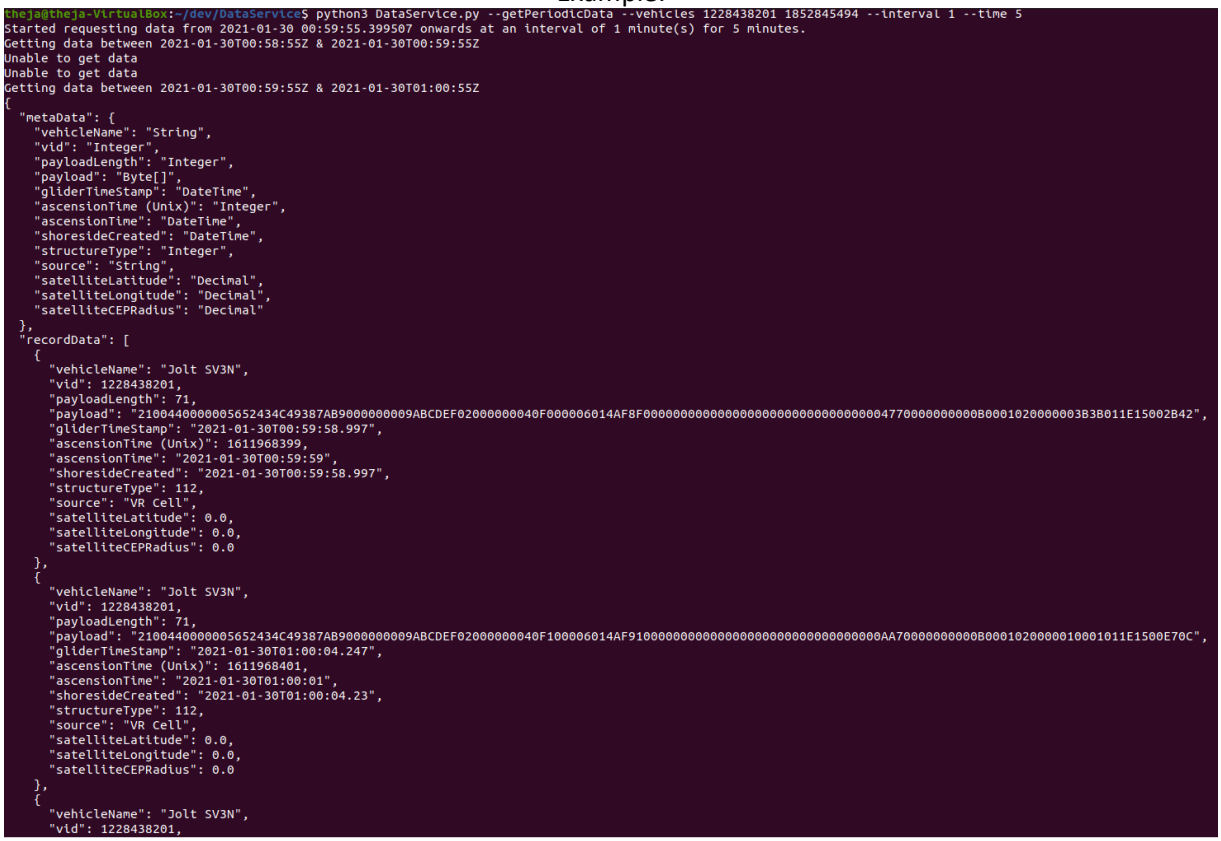
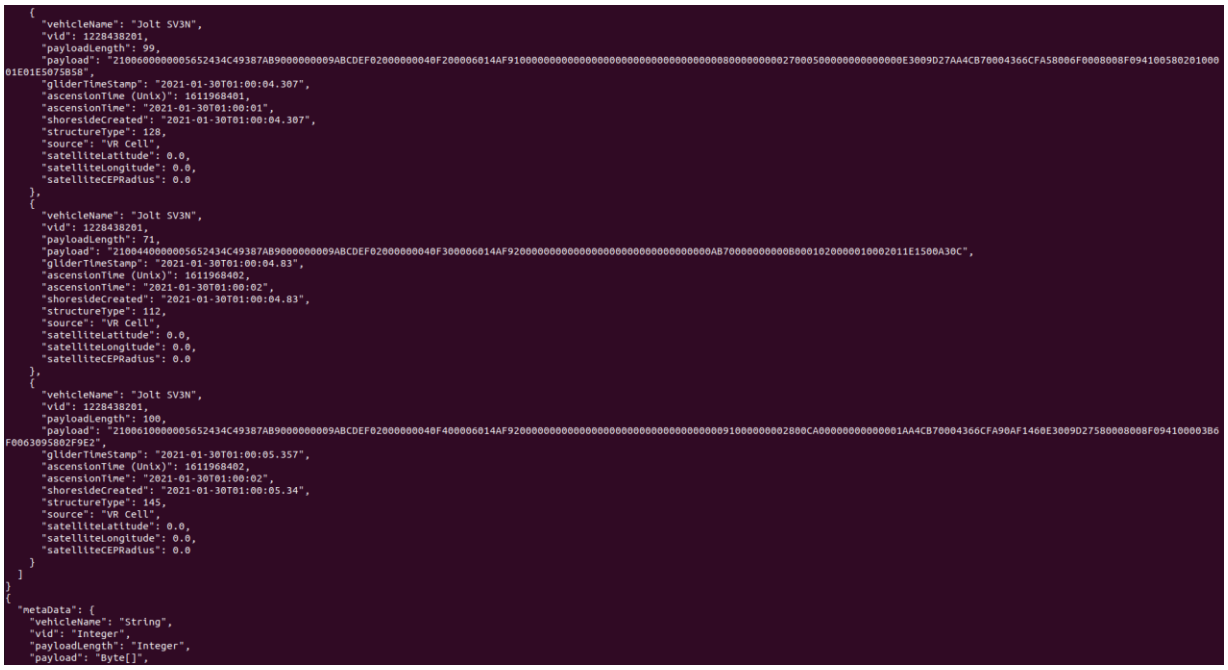
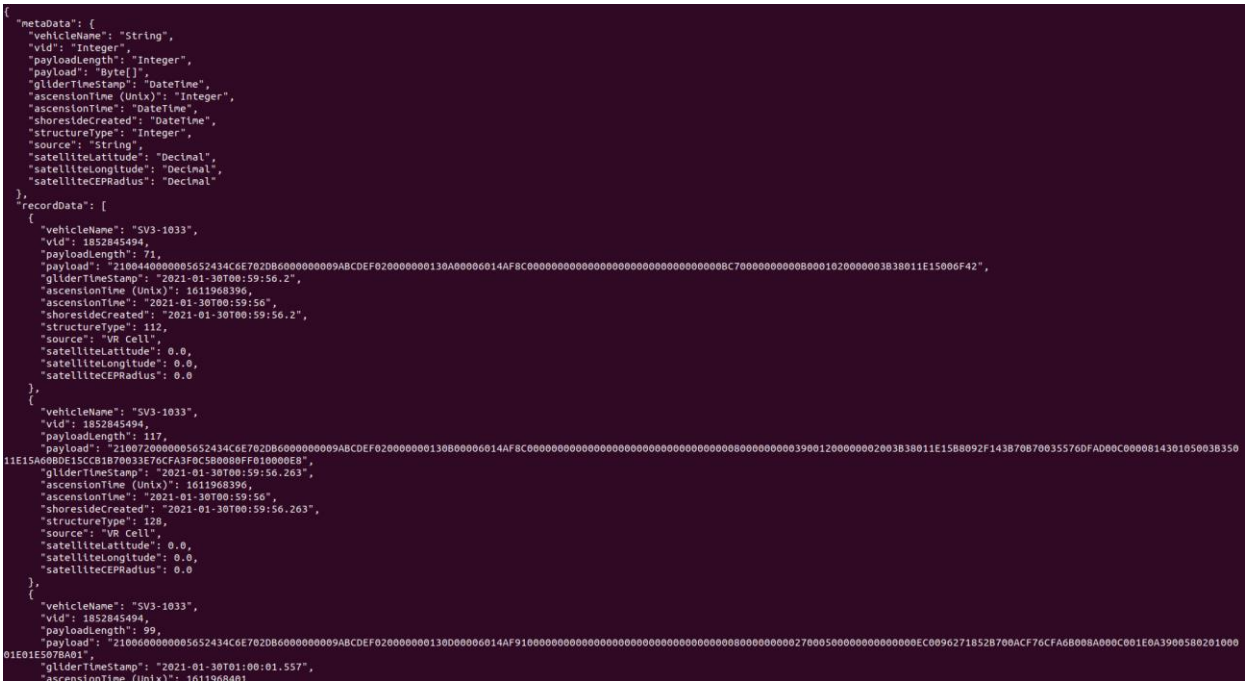


Figure 12 - Python example, getReportData, multi-vehicle 1



```
    {
      "vehicleName": "Jolt SV3N",
      "vid": 1228438201,
      "payloadLength": 99,
      "payload": "2100600000005652434C49387AB9000000009ABCDEF02000000040F200006014AF9100000000000000000000000000000000000000800000000027000500000000000000E3009D27AA4CB70004366CFA58006F0008008F094100580201000
01E01E5075B58",
      "gliderTimeStamp": "2021-01-30T01:00:04.307",
      "ascensionTime (Unix)": 1611968401,
      "ascensionTime": "2021-01-30T01:00:01",
      "shoresideCreated": "2021-01-30T01:00:04.307",
      "structureType": 128,
      "source": "VR Cell",
      "satelliteLatitude": 0.0,
      "satelliteLongitude": 0.0,
      "satelliteCEPRadius": 0.0
    },
    {
      "vehicleName": "Jolt SV3N",
      "vid": 1228438201,
      "payloadLength": 71,
      "payload": "2100440000005652434C49387AB9000000009ABCDEF02000000040F300006014AF920000000000000000000000000000000000AB700000000B000102000001000 2011E1500A30C",
      "gliderTimeStamp": "2021-01-30T01:00:04.83",
      "ascensionTime (Unix)": 1611968402,
      "ascensionTime": "2021-01-30T01:00:02",
      "shoresideCreated": "2021-01-30T01:00:04.83",
      "structureType": 112,
      "source": "VR Cell",
      "satelliteLatitude": 0.0,
      "satelliteLongitude": 0.0,
      "satelliteCEPRadius": 0.0
    },
    {
      "vehicleName": "Jolt SV3N",
      "vid": 1228438201,
      "payloadLength": 100,
      "payload": "2100610000005652434C49387AB9000000009ABCDEF02000000040F400006014AF920000000000000000000000000000000000091000000002800CA80000000000001AA4CB70004366CFA90AF1460E3009D27580008008F094100003B6
F0063095002F9E2",
      "gliderTimeStamp": "2021-01-30T01:00:05.357",
      "ascensionTime (Unix)": 1611968402,
      "ascensionTime": "2021-01-30T01:00:02",
      "shoresideCreated": "2021-01-30T01:00:05.34",
      "structureType": 145,
      "source": "VR Cell",
      "satelliteLatitude": 0.0,
      "satelliteLongitude": 0.0,
      "satelliteCEPRadius": 0.0
    }
  ]
}
{
  "metaData": {
    "vehicleName": "String",
    "vid": "Integer",
    "payloadLength": "Integer",
    "payload": "Byte[]",
```


Figure 13 - Python example, getReportData, multi-vehicle 2



```
{
  "metaData": {
    "vehicleName": "String",
    "vid": "Integer",
    "payloadLength": "Integer",
    "payload": "Byte[]",
    "gliderTimeStamp": "DateTime",
    "ascensionTime (Unix)": "Integer",
    "ascensionTime": "DateTime",
    "shoresideCreated": "DateTime",
    "structureType": "Integer",
    "source": "String",
    "satelliteLatitude": "Decimal",
    "satelliteLongitude": "Decimal",
    "satelliteCEPRadius": "Decimal"
  },
  "recordData": [
    {
      "vehicleName": "SV3-1033",
      "vid": 1852845494,
      "payloadLength": 71,
      "payload": "2100440000005652434C6E702DB6000000009ABCDEF020000000130A00006014AF8C000000000000000000000000000000000BC7000000000B000102000000 3B38011E15006F42",
      "gliderTimeStamp": "2021-01-30T00:59:56.2",
      "ascensionTime (Unix)": 1611968396,
      "ascensionTime": "2021-01-30T00:59:56",
      "shoresideCreated": "2021-01-30T00:59:56.2",
      "structureType": 112,
      "source": "VR Cell",
      "satelliteLatitude": 0.0,
      "satelliteLongitude": 0.0,
      "satelliteCEPRadius": 0.0
    },
    {
      "vehicleName": "SV3-1033",
      "vid": 1852845494,
      "payloadLength": 117,
      "payload": "2100720000005652434C6E702DB6000000009ABCDEF020000000130B00006014AF8C000000000000000000000000000000000800000000390012000000 2003B38011E15B8092F143B70B70035576DFAD00C00008143010500 3B350
11E15A60BDE15CCB1B70033E76CFA3F0C5B0080FF010000E8",
      "gliderTimeStamp": "2021-01-30T00:59:56.263",
      "ascensionTime (Unix)": 1611968396,
      "ascensionTime": "2021-01-30T00:59:56",
      "shoresideCreated": "2021-01-30T00:59:56.263",
      "structureType": 128,
      "source": "VR Cell",
      "satelliteLatitude": 0.0,
      "satelliteLongitude": 0.0,
      "satelliteCEPRadius": 0.0
    },
    {
      "vehicleName": "SV3-1033",
      "vid": 1852845494,
      "payloadLength": 99,
      "payload": "2100600000005652434C6E702DB6000000009ABCDEF020000000130D00006014AF9100000000000000000000000000000000800000000027000500000000000000EC0096271852B700ACF76CFA6B008A000C001E0A3900580201000
01E01E507BA01",
      "gliderTimeStamp": "2021-01-30T01:00:01.557",
      "ascensionTime (Unix)": 1611968401,
```

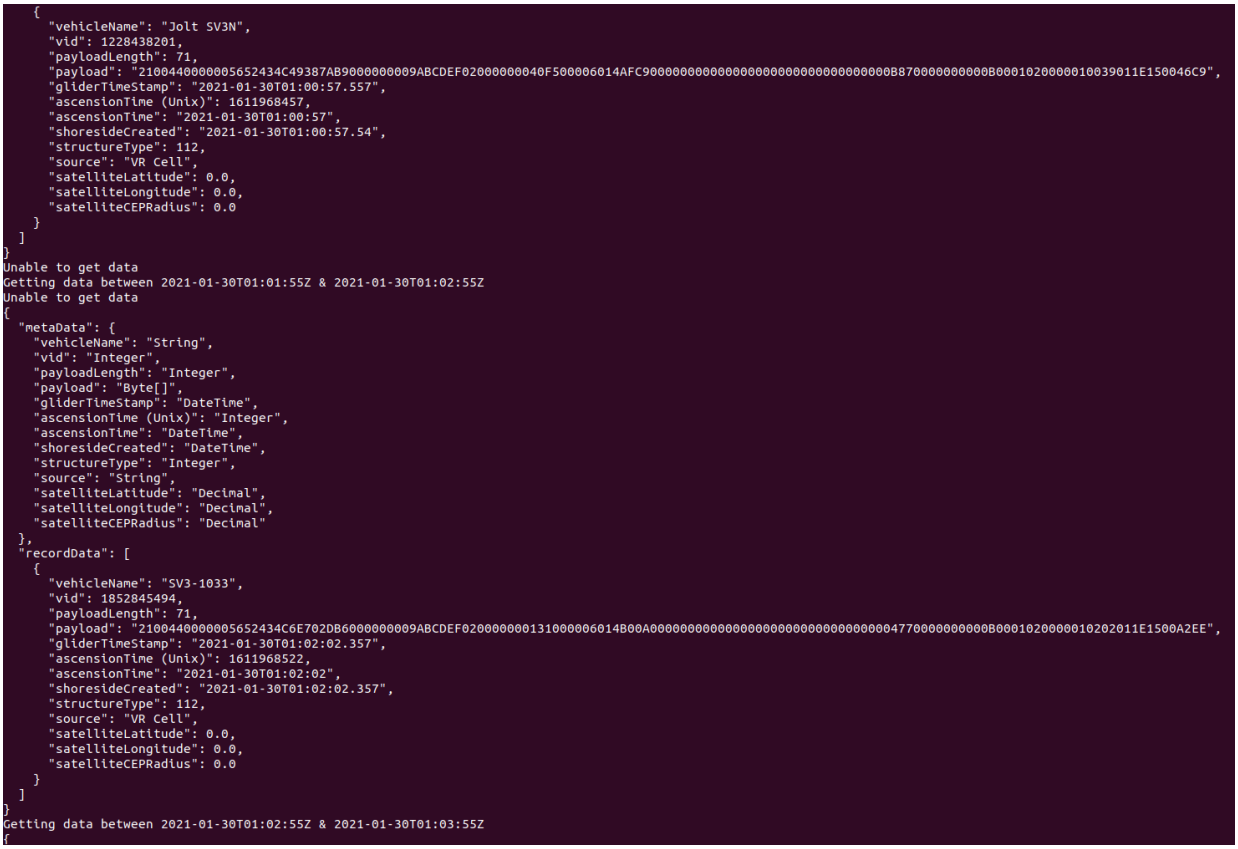

Figure 14 - Python example, getReportData, multi-vehicle 3

```
    {
      "vehicleName": "SV3-1033",
      "vid": 1852845494,
      "payloadLength": 99,
      "payload": "21006000000005652434C6E702DB6000000009ABCDEF020000000130D00006014AF910000000000000000000000000000000000000000008000000000270005000000000000000EC0096271852B700ACF76CFA6B008A000C001E0A3900580201000
01E01E507BA01",
      "gliderTimeStamp": "2021-01-30T01:00:01.557",
      "ascensionTime (Unix)": 1611968401,
      "ascensionTime": "2021-01-30T01:00:01",
      "shoresideCreated": "2021-01-30T01:00:01.557",
      "structureType": 128,
      "source": "VR Cell",
      "satelliteLatitude": 0.0,
      "satelliteLongitude": 0.0,
      "satelliteCEPRadius": 0.0
    },
    {
      "vehicleName": "SV3-1033",
      "vid": 1852845494,
      "payloadLength": 71,
      "payload": "21004400000005652434C6E702DB6000000009ABCDEF020000000130C00006014AF91000000000000000000000000000000000000AA70000000000B0001020000010001011E1500E70C",
      "gliderTimeStamp": "2021-01-30T01:00:01.737",
      "ascensionTime (Unix)": 1611968401,
      "ascensionTime": "2021-01-30T01:00:01",
      "shoresideCreated": "2021-01-30T01:00:01.737",
      "structureType": 112,
      "source": "VR Cell",
      "satelliteLatitude": 0.0,
      "satelliteLongitude": 0.0,
      "satelliteCEPRadius": 0.0
    },
    {
      "vehicleName": "SV3-1033",
      "vid": 1852845494,
      "payloadLength": 71,
      "payload": "21004400000005652434C6E702DB6000000009ABCDEF020000000130E00006014AF91000000000000000000000000000000000000AB70000000000B0001020000010001011E1500E70C",
      "gliderTimeStamp": "2021-01-30T01:00:04.32",
      "ascensionTime (Unix)": 1611968401,
      "ascensionTime": "2021-01-30T01:00:01",
      "shoresideCreated": "2021-01-30T01:00:04.32",
      "structureType": 112,
      "source": "VR Cell",
      "satelliteLatitude": 0.0,
      "satelliteLongitude": 0.0,
      "satelliteCEPRadius": 0.0
    },
    {
      "vehicleName": "SV3-1033",
      "vid": 1852845494,
      "payloadLength": 100,
      "payload": "21006100000005652434C6E702DB6000000009ABCDEF020000000130F00006014AF910000000000000000000000000000000000000091000000002800CA00000000000011852B700ACF76CFA90AF1460EC0096276B000C001E0A3900003B8
A00F70958027E8C",
      "gliderTimeStamp": "2021-01-30T01:00:04.543",
      "ascensionTime (Unix)": 1611968401,
      "ascensionTime": "2021-01-30T01:00:01",
      "shoresideCreated": "2021-01-30T01:00:04.543"
```


Figure 15 - Python example, getReportData, multi-vehicle 4


```
    },
    {
      "vehicleName": "SV3-1033",
      "vid": 1852845494,
      "payloadLength": 100,
      "payload": "21006100000005652434C6E702DB6000000009ABCDEF020000000130F00006014AF910000000000000000000000000000000000000091000000002800CA00000000000011852B700ACF76CFA90AF1460EC0096276B000C001E0A3900003B8
A00F70958027E8C",
      "gliderTimeStamp": "2021-01-30T01:00:04.543",
      "ascensionTime (Unix)": 1611968401,
      "ascensionTime": "2021-01-30T01:00:01",
      "shoresideCreated": "2021-01-30T01:00:04.543",
      "structureType": 145,
      "source": "VR Cell",
      "satelliteLatitude": 0.0,
      "satelliteLongitude": 0.0,
      "satelliteCEPRadius": 0.0
    }
  ]
}
Getting data between 2021-01-30T01:00:55Z & 2021-01-30T01:01:55Z
{
  "metaData": {
    "vehicleName": "String",
    "vid": "Integer",
    "payloadLength": "Integer",
    "payload": "Byte[]",
    "gliderTimeStamp": "DateTime",
    "ascensionTime (Unix)": "Integer",
    "ascensionTime": "DateTime",
    "shoresideCreated": "DateTime",
    "structureType": "Integer",
    "source": "String",
    "satelliteLatitude": "Decimal",
    "satelliteLongitude": "Decimal",
    "satelliteCEPRadius": "Decimal"
  },
  "recordData": [
    {
      "vehicleName": "Jolt SV3N",
      "vid": 1228438201,
      "payloadLength": 113,
      "payload": "21006E0000005652434C49387AB9000000009ABCDEF02000000040F600006014AFC900000000000000000000000000000000000000000600000000350000001E00414CB70001366CFA010038011E15001A00610000000000F4000485F400DD0
07DDB0A0100FFFF0000429200E10005000090691B",
      "gliderTimeStamp": "2021-01-30T01:00:57.38",
      "ascensionTime (Unix)": 1611968457,
      "ascensionTime": "2021-01-30T01:00:57",
      "shoresideCreated": "2021-01-30T01:00:57.367",
      "structureType": 6,
      "source": "VR Cell",
      "satelliteLatitude": 0.0,
      "satelliteLongitude": 0.0,
      "satelliteCEPRadius": 0.0
    },
    {
      "vehicleName": "Jolt SV3N",
      "vid": 1228438201,
```


Figure 16 - Python example, getReportData, multi-vehicle 5



```
    {
      "vehicleName": "Jolt SV3N",
      "vid": 1228438201,
      "payloadLength": 71,
      "payload": "2100440000005652434C49387AB9000000009ABCDEF02000000040F500006014AFC900000000000000000000000000000000B870000000000B000102000010039011E150046C9",
      "gliderTimeStamp": "2021-01-30T01:00:57.557",
      "ascensionTime (Unix)": 1611968457,
      "ascensionTime": "2021-01-30T01:00:57",
      "shoresideCreated": "2021-01-30T01:00:57.54",
      "structureType": 112,
      "source": "VR Cell",
      "satelliteLatitude": 0.0,
      "satelliteLongitude": 0.0,
      "satelliteCEPRadius": 0.0
    }
  ]
}
Unable to get data
Getting data between 2021-01-30T01:01:55Z & 2021-01-30T01:02:55Z
Unable to get data
{
  "metaData": {
    "vehicleName": "String",
    "vid": "Integer",
    "payloadLength": "Integer",
    "payload": "Byte[]",
    "gliderTimeStamp": "DateTime",
    "ascensionTime (Unix)": "Integer",
    "ascensionTime": "DateTime",
    "shoresideCreated": "DateTime",
    "structureType": "Integer",
    "source": "String",
    "satelliteLatitude": "Decimal",
    "satelliteLongitude": "Decimal",
    "satelliteCEPRadius": "Decimal"
  },
  "recordData": [
    {
      "vehicleName": "SV3-1033",
      "vid": 1852845494,
      "payloadLength": 71,
      "payload": "2100440000005652434C6E702DB6000000009ABCDEF020000000131000006014B00A000000000000000000000000000000004770000000000B000102000010202011E1500A2EE",
      "gliderTimeStamp": "2021-01-30T01:02:02.357",
      "ascensionTime (Unix)": 1611968522,
      "ascensionTime": "2021-01-30T01:02:02",
      "shoresideCreated": "2021-01-30T01:02:02.357",
      "structureType": 112,
      "source": "VR Cell",
      "satelliteLatitude": 0.0,
      "satelliteLongitude": 0.0,
      "satelliteCEPRadius": 0.0
    }
  ]
}
Getting data between 2021-01-30T01:02:55Z & 2021-01-30T01:03:55Z
{
```


Figure 17 - Python example, getReportData, multi-vehicle 6

```
Getting data between 2021-01-30T01:02:55Z & 2021-01-30T01:03:55Z
{
  "metaData": {
    "vehicleName": "String",
    "vid": "Integer",
    "payloadLength": "Integer",
    "payload": "Byte[]",
    "gliderTimeStamp": "DateTime",
    "ascensionTime (Unix)": "Integer",
    "ascensionTime": "DateTime",
    "shoresideCreated": "DateTime",
    "structureType": "Integer",
    "source": "String",
    "satelliteLatitude": "Decimal",
    "satelliteLongitude": "Decimal",
    "satelliteCEPRadius": "Decimal"
  },
  "recordData": [
    {
      "vehicleName": "Jolt SV3N",
      "vid": 1228438201,
      "payloadLength": 71,
      "payload": "2100440000005652434C49387AB9000000009ABCDEF02000000040F700006014B04200000000000000000000000000000000004770000000000B000102000001023A011E1500032B",
      "gliderTimeStamp": "2021-01-30T01:02:58.167",
      "ascensionTime (Unix)": 1611968578,
      "ascensionTime": "2021-01-30T01:02:58",
      "shoresideCreated": "2021-01-30T01:02:58.167",
      "structureType": 112,
      "source": "VR Cell",
      "satelliteLatitude": 0.0,
      "satelliteLongitude": 0.0,
      "satelliteCEPRadius": 0.0
    }
  ]
}
{
  "metaData": {
    "vehicleName": "String",
    "vid": "Integer",
    "payloadLength": "Integer",
    "payload": "Byte[]",
    "gliderTimeStamp": "DateTime",
    "ascensionTime (Unix)": "Integer",
    "ascensionTime": "DateTime",
    "shoresideCreated": "DateTime",
    "structureType": "Integer",
    "source": "String",
    "satelliteLatitude": "Decimal",
    "satelliteLongitude": "Decimal",
    "satelliteCEPRadius": "Decimal"
  },
  "recordData": [
    {
      "vehicleName": "SV3-1033",
      "vid": 1852845494,
      "payloadLength": 113
```

Figure 18 - Python example, getReportData, multi-vehicle 7

```
{
  "metaData": {
    "vehicleName": "String",
    "vid": "Integer",
    "payloadLength": "Integer",
    "payload": "Byte[]",
    "gliderTimeStamp": "DateTime",
    "ascensionTime (Unix)": "Integer",
    "ascensionTime": "DateTime",
    "shoresideCreated": "DateTime",
    "structureType": "Integer",
    "source": "String",
    "satelliteLatitude": "Decimal",
    "satelliteLongitude": "Decimal",
    "satelliteCEPRadius": "Decimal"
  },
  "recordData": [
    {
      "vehicleName": "SV3-1033",
      "vid": 1852845494,
      "payloadLength": 113,
      "payload": "21006E0000005652434C6E702DB6000000009ABCDEF020000000131200006014B05C000000000000000000000000000000000000600000000350000001F003951B70034F66CFA010318011E15001A0082000000000000020103030201F00
081EA0B0100FFFF0000002306F9005A0000D048FF",
      "gliderTimeStamp": "2021-01-30T01:03:24.93",
      "ascensionTime (Unix)": 1611968604,
      "ascensionTime": "2021-01-30T01:03:24",
      "shoresideCreated": "2021-01-30T01:03:24.93",
      "structureType": 6,
      "source": "VR Cell",
      "satelliteLatitude": 0.0,
      "satelliteLongitude": 0.0,
      "satelliteCEPRadius": 0.0
    },
    {
      "vehicleName": "SV3-1033",
      "vid": 1852845494,
      "payloadLength": 71,
      "payload": "2100440000005652434C6E702DB6000000009ABCDEF020000000131100006014B05C00000000000000000000000000000000B870000000000B000102000010318011E1500FAFD",
      "gliderTimeStamp": "2021-01-30T01:03:24.993",
      "ascensionTime (Unix)": 1611968604,
      "ascensionTime": "2021-01-30T01:03:24",
      "shoresideCreated": "2021-01-30T01:03:24.993",
      "structureType": 112,
      "source": "VR Cell",
      "satelliteLatitude": 0.0,
      "satelliteLongitude": 0.0,
      "satelliteCEPRadius": 0.0
    }
  ]
}
Getting data between 2021-01-30T01:03:55Z & 2021-01-30T01:04:55Z
Unable to get data
Unable to get data
Stopped getting data at 2021-01-30 01:04:56.450723
```

Figure 19 - Python example, getReportData, multi-vehicle 8

To get data from all the vehicles in the org simply omit the vehicles option:

```
python3 DataService.py --getPeriodicData --interval [time in minutes to get
data] --time [time in minutes the script will run]
```

4.3 Script Reference Library

Examples will become available via application notes or training modules within the Open Oceans Portal.

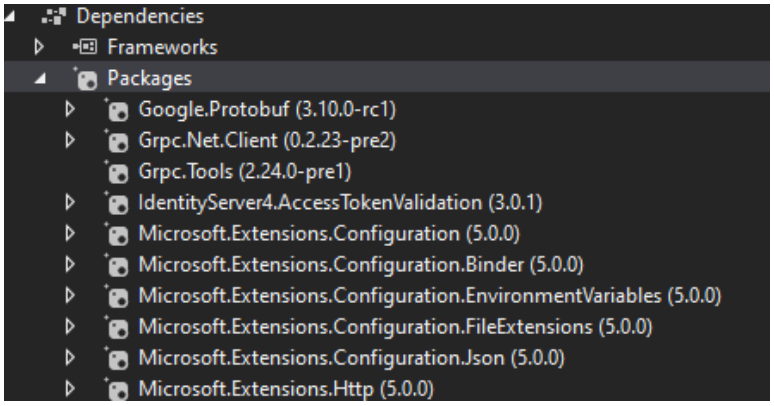
Python & Java script tools are available under PN 11884.

5 Streaming Features & Commands

Data may be streamed in near-real-time from a Wave Glider to a specified client using Data Service's Streaming capability.

5.1 Streaming Implementation Requirements

Utilizing Streaming on a Client requires the packages and libraries listed below for .NET CORE.



The Client will also require:

- Proto File
- appsettings.json
- Compiled ClientStream.proto

Streaming through Data Service is initiated through gRPC (Remote Procedure Call) messages. In gRPC, a client application can directly call a method on a server application on a different machine as if it were a local object, making it easier for you to create distributed applications and services. As in many RPC systems, gRPC is based around the idea of defining a service, specifying the methods that can be called remotely with their parameters and return types. On the server side, the server implements this interface and runs a gRPC server to handle client calls. On the client side, the client has a stub (referred to as just a client in some languages) that provides the same methods as the server.

(source: https://grpc.io/docs/what-is-grpc/introduction/)

5.2 RPC Streaming Message Variables

There are two primary message variables involved in initiating a Streaming session to a Client; one in each direction to and from a Client. These messages are used as arguments to `rpc SendMessage.`

ClientToServerMessage

This message is sent from a Client to the Server to initiate a Stream.

Variables

```
string clientToServerText = 1;
```
- VOID – used for troubleshooting(do not implement)

```
string userName = 2;
```
- Username to be used for authentication by WGMS

string password = 3;
- Password to be used for authentication by WGMS

string org = 4;
- Organization that the username and password belong to, to be used for authentication by WGMS

string reports = 5;
- Report Name. The report list can take multiple values, comma delimited

int64 VID = 6;
- Vehicle ID of the Wave Glider that the Stream should originate from. Must be within the user's Organization.

ServerToClientMessage

This message is sent from the Server to the Client in response to a request to initiate a Stream.

Variables

string serverToClientText = 1;
- VOID, used for troubleshooting(do not implement)

string validConnection = 2;
- VOID, used for troubleshooting

string reportResponse = 3;
- Report results are in JSON format.
  - Each report requested will be streamed asynchronously as they are received from the vehicle

google.protobuf.Timestamp timestamp = 4;
- Timestamp (UTC) of result coming back from server (not the time of the vehicle report)

6 Streaming Implementation Examples

This section provides general and specific examples of the code necessary to implement Below is the proto framework to implement streaming from the server to a client. This may differ according to different languages.

6.1 General Framework

```
syntax = "proto3";
option csharp_namespace = "WGMSGrpcService.Protos";
import "google/protobuf/timestamp.proto";

service ClientStream {
       rpc Login (stream LoginRequest) returns (stream LoginResponse);
       rpc SendMessage (stream ClientToServerMessage) returns (stream ServerToClientMessage);
}

message ClientToServerMessage{
    string clientToServerText = 1;
    string userName = 2;
    string password = 3;
    string org = 4;
    string reports = 5;
    int64 VID = 6;
}

message ServerToClientMessage {
    string serverToClientText = 1;
    string validConnection = 2;
    string reportResponse = 3;
    google.protobuf.Timestamp timestamp = 4;
}

message LoginRequest
{
    string userName = 1;
    string password = 2;
    int64 org = 3;
}
message LoginResponse
{
    bool loginResult = 1;
    string sessionKey = 2;
}
```

6.2 C# .NET 3.1 Implementation

Note that below, appsetttings.json is expected to be the following:

```
{
  "ServerAddresses": {
    "DevelopmentService": "[SERVERIP]",
    "ProductionService": "[SERVERIP]"
  }
}
```

The code necessary to implement streaming in C# .NET of `reports` from Wave Glider `VID` to the client is listed below.

```
using Grpc.Core;
using Grpc.Net.Client;
using Microsoft.Extensions.Configuration;
using System;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using WGMSGrpcService.Protos;

namespace GrpcClient
{
    class Program
    {
        const int PORT = 5001;
        static async Task Main(string[] args)
        {
            //Environment
            var env = Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT");
            var builder = new ConfigurationBuilder()
                .SetBasePath(Directory.GetParent(AppContext.BaseDirectory).FullName)
                .AddJsonFile("appsettings.json", true, true)
                .AddEnvironmentVariables();

            var config = builder.Build();

            //Args
            string userName = string.Empty;
            string password = string.Empty;
            string org = string.Empty;
            string VID = string.Empty;
            string reports = string.Empty;
            //---------------
            string invalidException = string.Empty;

            //Connection String
            var serverAddress = config["ServerAddresses:DevelopmentService"];


            if (args.Length == 0)
            {
                // Display message to user to provide parameters.
                System.Console.WriteLine("Please enter parameter values.");
                Console.Read();
            }
            else
            {
                try
                {
                    userName = args[0];
                    password = args[1];
                    org = args[2];
                    VID = args[3];
                    reports = args[4];
                }
                catch(Exception e)
                {
                    invalidException = "Invalid Parameters";
                    Console.WriteLine(invalidException);
```

```
                }

                var newGuid = Guid.NewGuid().ToString();
                var httpClientHandler = new HttpClientHandler();
                httpClientHandler.ServerCertificateCustomValidationCallback =
HttpClientHandler.DangerousAcceptAnyServerCertificateValidator;
                var httpClient = new HttpClient(httpClientHandler);
                var channel = GrpcChannel.ForAddress(serverAddress, new
GrpcChannelOptions { HttpClient = httpClient });
                var clientStream = new ClientStream.ClientStreamClient(channel);

                //from client
                var clientStreamVal = new ClientToServerMessage { UserName = userName,
Password = password, Org = org, VID = Convert.ToInt32(VID), Reports = reports };


                using var clientResponse = clientStream.SendMessage();
                try
                {
                    //send client message
                    await clientResponse.RequestStream.WriteAsync(clientStreamVal);

                    //await server response
                    await clientResponse.RequestStream.CompleteAsync();
                    using var response = Task.Run(async () =>
                    {
                        await foreach (var cr in
clientResponse.ResponseStream.ReadAllAsync())
                        {
                            Console.WriteLine("Results: " + cr.ReportResponse);
                        }
                    });
                    await clientResponse.RequestStream.CompleteAsync();
                    await response;
                }
                catch (RpcException e) when (e.Status.StatusCode == StatusCode.Cancelled)
                {
                    Console.WriteLine("Streaming was cancelled from the client");
                    Console.WriteLine(e.Message);
                    Console.WriteLine(e.Data);
                    Console.WriteLine(e.StackTrace);
                    Console.WriteLine(e.InnerException);
                }
                Console.ReadLine();
            }
        }
    }
}
```