1 Installation

1.1 System Requirements

The current version of the software has been tested to run on Linux based systems. Please go through the README file for further details.

1.1.1 NATS Server Requirements

To run the NATS server, you will need Java™ Platform, Standard Edition Development Kit (JDK™) 7 or above, which can be obtained from the ORACLE® website. You will also need grib-api, jasper and hdf5, which are already available through the current NATS distribution, and will be installed later.

1.1.2 NATS Client Requirements

The NATS client can be developed using MATLAB®, Python, Java and C++ environments. While there are no specific requirements for the NATS Client, it is imperative that you have MATLAB (for Linux), Python, JDK™.

1.2 Installing the Software

To begin with you will have two files: `NATS_Client.zip` and `NATS_Server.zip`. We will first install the server and the dependencies and then install the client.

1.2.1 Installing NATS Server

To install NATS Server, open the shell script/ terminal window in UNIX and go to the directory where the zip files are. We will refer to this directory as `NATS_dir`. Then unzip the `NATS_Server.zip` file into a newly created `NATS_Server` directory.

```
$ cd NATS_dir
~/NATS_dir$ ls
NATS_Client.zip NATS_Server.zip
~/NATS_dir$ unzip NATS_Server.zip –d NATS_Server
```

Then change the directory to `NATS_Server`. Create a new directory called `lib`, and then go into `dependency_library` directory. The directory `lib` is where all the dependencies for `NATS_Server` will be installed.

```
~/NATS_dir$ cd NATS_Server
~/NATS_dir/NATS_Server$ ls
dependency_library dist lib_precomp README run share
~/NATS_dir/NATS_Server$ mkdir lib
~/NATS_dir/NATS_Server$ ls
dependency_library dist lib lib_precomp README run share
~/NATS_dir/NATS_Server$ cd dependency_library
```

Upon entering the `dependency_library` you would notice shell scripts such as `install_grib.sh`, `install_hdf5.sh` and `install_jasper.sh`. These scripts will install the sever dependencies mentioned in the previous section. First you have to modify the permissions of these files using `chmod`. Next each dependency will be installed, in the same sequence as shown below.

```
~/NATS_dir/NATS_Server/dependency_library $ ls
grib_api-1.11.0.tar.gz  hdf5-1.8.11.tar.gz
install_grib.sh  install_hdf5.sh  install_jasper.sh
jasper-1.900.1.zip

~/NATS_dir/NATS_Server/dependency_library $ chmod +x
install_grib.sh install_hdf5.sh install_jasper.sh

~/NATS_dir/NATS_Server/dependency_library
$./install_jasper.sh
```

Once jasper is installed, you have to put the jasper libraries in your `LD_LIBRARY_PATH`. To do that you need to execute the following,

```
~/NATS_dir/NATS_Server/dependency_library export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${PWD}../lib/jasper/lib
```

Next you need to install grib_api and put the corresponding libraries in your `LD_LIBRARY_PATH` variable. To do that you need to execute

```
~/NATS_dir/NATS_Server/dependency_library $./install_grib.sh

~/NATS_dir/NATS_Server/dependency_library export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${PWD}../lib/grib_api/lib
```

Next you can install hdf5 using the command

```
~/NATS_dir/NATS_Server/dependency_library $./install_hdf5.sh
```

The above processes will install jasper, grib-api and hdf5 in your local directory. Advanced users can manually install these packages in the directory of their choice.

In case you obtain an error message and are not able to install these software packages, you need to go to jasper website (https://jasperproject.github.io/), grib-api website (https://software.ecmwf.int/wiki/display/GRIB/Home) and hdf5 website (https://support.hdfgroup.org/HDF5/) and download the proper version for your system and follow the instructions to install the files. Standard installation is pretty straight forward and involves the following steps,

```
~/NATS_dir/NATS_Server/dependency_library $ ./configure
--prefix=${PWD}../lib/(PACKAGE NAME); make; make
install
```

Upon installation, come out of the `dependency_library` directory, go inside `lib` directory, and add the `hdf5` library it to your `LD_LIBRARY_PATH`. You also have to add `hdf5/bin` to your `PATH` variable. These can be done as follows,

```
~/NATS_dir/NATS_Server/dependency_library $ cd ..

~/NATS_dir/NATS_Server $ cd lib

~/NATS_dir/NATS_Server/lib $ cd hdf5/lib

~/NATS_dir/NATS_Server/lib/hdf5/lib $ export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${PWD}

~/NATS_dir/NATS_Server/lib/hdf5/lib $ cd ../bin

~/NATS_dir/NATS_Server/lib/hdf5/bin $ export
PATH=$PATH:${PWD}
```

Upon successful completion of these steps, you are ready to run the NATS server.

1.2.2 Installing NATS Client

To install NATS Client, open another shell script/ terminal window in UNIX and go to `NATS_dir` directory. Then unzip the `NATS_Client.zip` file into a newly created `NATS_Client` directory.

```
$ cd NATS_dir
~/NATS_dir$ ls
NATS_Client.zip NATS_Server NATS_Server.zip
~/NATS_dir$ unzip NATS_Client.zip –d NATS_Client
```

Enter the `NATS_Client` directory and you can find Java™, MATLAB and Python examples for clients. Open the `README` file and follow the readme file to run the examples and to install and required dependencies for the examples. ^^^

```
~/NATS_dir/NATS_Client$ ls
data  dist  README  runSample_Java  Sample_Matlab.m
Sample_Python_Jpype.py
```

Once that is done you are ready to run the NATS client.

2 Getting Started with NATS

2.1 Running the Server

Before running the NATS server, return to the `NATS_Server` directory level and open the `run` file.

```
~/NATS_dir/NATS_Server/lib/hdf5/bin $ cd ../../..
~/NATS_dir/NATS_Server $ gedit run&
```

The `run` file looks like,

```
#!/bin/sh

export NATS_HOME=/your_path/NATS_Server

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$NATS_HOME/dist

/opt/java/64/jdk1.7.0_80/bin/java -cp $NATS_HOME/dist/nats-server.jar:$NATS_HOME/dist/nats-shared.jar -Djava.library.path=$NATS_HOME/dist -Xmx768m NATSServer 2017
```

Replace the text "`/your_path/NATS_Server`" with current directory path or `${PWD}` and "`/opt/java/64/jdk1.7.0_80/bin/java`" with location of your java binary which can be found by using the `which` function,

```
~/NATS_dir/NATS_Server $ which java
/usr/bin/java
```

Let us suppose that the java binary path is `/usr/bin/java`, the modified run file looks like

```
#!/bin/sh

export NATS_HOME= ${PWD}

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$NATS_HOME/dist

/usr/bin/java -cp $NATS_HOME/dist/nats-server.jar:$NATS_HOME/dist/nats-shared.jar -Djava.library.path=$NATS_HOME/dist -Xmx768m NATSServer 2017
```

Close the run file and change its permissions. Then you can run the NATS server. The following commands need to be executed.

```
~/NATS_dir/NATS_Server $ chmod +x run
~/NATS dir/NATS Server $ ./run
```

If the NATS server is running properly the following would be observed in the terminal,

```
~/NATS_dir/NATS_Server $ chmod +x run
~/NATS_dir/NATS_Server $ ./run
============================================================
  National Airspace Trajectory-Prediction System(NATS) Server

  Optimal Synthesis Inc.
============================================================

  Loading airport data
  Loading waypoint data
  Loading BADA data
  Loading airway data
  Loading sector data
  Loading PAR data
  Loading SID/STAR data

NATS Server started on port 2017.
```

2.2 Running the Client

Open the terminal window where NATS client is running and you would be able to run the client examples from there. **Make sure that the server is up and running before running the client examples.** The client examples will produce error message if server is not running.

To run the Python example type

```
~/NATS_dir/NATS_Client $ python basic_python_example.py
```

For the Java™ example you first need to change the `runSample_Java` file. Open the file using `gedit,` it looks like

```
#!/bin/sh

export NATS_HOME=/your_path/NATS_Client

/opt/java/64/jdk1.7.0_80/bin/java -cp $NATS_HOME/dist/nats-
client.jar:$NATS_HOME/dist/nats-shared.jar -Xmx768m Sample_Java
```

Similar to the NATS server, replace the text "`/your_path/NATS_Client`" with current directory path or `${PWD}` and "`/opt/java/64/jdk1.7.0_80/bin/java`" with location of your java binary which can be found by using the `which` function. The modified file would look like,

```
#!/bin/sh

export NATS_HOME=${PWD}

/usr/bin/java -cp $NATS_HOME/dist/nats-client.jar:$NATS_HOME/dist/nats-shared.jar
-Xmx768m Sample_Java
```

Upon changing `runSample_Java` file, you need to change its permissions and then you can run the java example from the terminal.

```
~/NATS_dir/NATS_Client $ chmod +x runSample_Java
~/NATS_dir/NATS_Client $./runSample_Java
```

To run the MATLAB example one needs to open MATLAB and run the `Sample_Matlab.m` file.

2.2.1 Run the Client Examples

We will discuss a Python example here. We will run the file `basic_python_example.py` and explain what it does. Before moving forward you must make sure that the server is running. Once server is running you can run the Python example from command line as

```
~/NATS_dir/NATS_Client $ python basic_python_example.py
```

The program runs the NATS simulation for 2 hours with a step size of 1 second. It simulates all the flights whose flight plans can be found in the file with path '`share/tg/trx/TRX_07132005_noduplicates_cut_crypted_2Flights.trx`'. Further it writes the trajectories of each aircraft in the in file `Sample_output_trajectory.csv`.

```
classpath = "dist/nats-client.jar:dist/nats-shared.jar"

startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path=%s" %
classpath)
FLIGHT_MODE_PREDEPARTURE =
JPackage('com').osi.util.Constants.FLIGHT_MODE_PREDEPARTURE
FLIGHT_MODE_CLIMB =
JPackage('com').osi.util.Constants.FLIGHT_MODE_CLIMB
FLIGHT_MODE_CRUISE =
JPackage('com').osi.util.Constants.FLIGHT_MODE_CRUISE
FLIGHT_MODE_DESCENT =
JPackage('com').osi.util.Constants.FLIGHT_MODE_DESCENT
FLIGHT_MODE_LANDED =
JPackage('com').osi.util.Constants.FLIGHT_MODE_LANDED
FLIGHT_MODE_HOLDING =
JPackage('com').osi.util.Constants.FLIGHT_MODE_HOLDING

# NATS simulation status definition
# You can get simulation status from the server and know what it
refers to
NATS_SIMULATION_STATUS_READY =
JPackage('com').osi.util.Constants.NATS_SIMULATION_STATUS_READY
NATS_SIMULATION_STATUS_START =
JPackage('com').osi.util.Constants.NATS_SIMULATION_STATUS_START
NATS_SIMULATION_STATUS_PAUSE =
JPackage('com').osi.util.Constants.NATS_SIMULATION_STATUS_PAUSE
NATS_SIMULATION_STATUS_RESUME =
JPackage('com').osi.util.Constants.NATS_SIMULATION_STATUS_RESUME
NATS_SIMULATION_STATUS_STOP =
JPackage('com').osi.util.Constants.NATS_SIMULATION_STATUS_STOP
NATS_SIMULATION_STATUS_ENDED =
JPackage('com').osi.util.Constants.NATS_SIMULATION_STATUS_ENDED

NATSClientFactory = JClass('NATSClientFactory')
natsClient = NATSClientFactory.getNATSClient()
sim = natsClient.getSimulationInterface();
# Get EquipmentInterface
equipmentInterface = natsClient.getEquipmentInterface();
# Get AircraftInterface
aircraftInterface = equipmentInterface.getAircraftInterface();

# Get EnvironmentInterface
environmentInterface = natsClient.getEnvironmentInterface();
# Get AirportInterface
airportInterface = environmentInterface.getAirportInterface()
# Get TerminalAreaInterface
terminalAreaInterface =
environmentInterface.getTerminalAreaInterface()
```

The program needs jpype to connect to the Java™ server. The `classpath` variable defines the path for NATS libraries and the startJVM module starts the threads to the server. Further the

`get` functions gets the handles to each interfaces. **This is the part of the code you would have in all of you programs.**

The next part of the program loads wind data and the flight plans. At first the memory is cleared. The wind files are stored in hdf5 format in the folder "`share/tg/rap`". The flight plans are in '`share/tg/trx/TRX_07132005_noduplicates_cut_crypted_2Flights.trx`'. The maximum flight levels corresponding to each flight in this file are given in '`share/tg/trx/TRX_07132005_noduplicatesOSIMaxfltLvl_cut_crypted_2Flights.trx`'.

```
sim.clear_trajectory()
environmentInterface.load_rap("share/tg/rap")
aircraftInterface.load_aircraft("share/tg/trx/TRX_07132005_noduplicates_cut_crypted_2Flights.trx",
"share/tg/trx/TRX_07132005_noduplicatesOSIMaxFltLvl_cut_crypted_2Flights.trx")
```

Once the flight are loaded we are ready to simulate. The `sim.setupSimulation (7200, 1)` command sets up the simulation for 7200 seconds (2 Hours) with the time step of 1 sec. The `sim.start()` command starts the NATS simulation. Further there is a `while loop` which constantly checks if the simulation has ended. **This loop should be part of every program and should be added just before output file generation.**

```
sim.setupSimulation(7200, 1)
sim.start()
time.sleep(2)
while True:
   server_runtime_sim_status = sim.get_runtime_sim_status()
   if (server_runtime_sim_status == NATS_SIMULATION_STATUS_ENDED) :
      break
   else:
      time.sleep(1)
```

Once the simulation has ended the following commands outputs the trajectories and frees the memory assigned to aircraft specific files and wind files. Subsequently, the client thread to the server is terminated.

```
print "Outputting trajectory data.  Please wait...."
sim.write_trajectories("Sample_output_trajectory.xml")

aircraftInterface.release_aircraft()
environmentInterface.release_rap()

shutdownJVM()
```

1.1 Output File Generation

Currently output of the NATS can be generated in `.xml`, `.h5` and `.csv` formats. The `.h5` file represents the output in hdf5 format. You can specify the output format by from the filename used to output trajectories in client program.

```
sim.write_trajectories("Sample_output_trajectory.xml")
sim.write_trajectories("Sample_output_trajectory.csv")
sim.write_trajectories("Sample_output_trajectory.h5")
```

1.1.1 Output Description

As mentioned before, the output trajectories can be generated in xml, csv format as well as hdf5 format. As hdf5 is only machine readable we will describe the xml format. Sample xml output looks something as follows.

```
<trajectories simulation_start_time="1121238067">
    <trajectory flight_index="0" callsign="ULI13-239120" actype="B752"
origin_airport="ATL" destination_airport="KSEA" start_time="0" interval="1">
        <trajectory_point timestamp="0">
            <latitude_deg>41.205</latitude_deg>
            <longitude_deg>-96.1724</longitude_deg>
            <altitude_ft>36000</altitude_ft>
            <rocd_fps>0</rocd_fps>
            <tas_knots>460</tas_knots>
            <heading_deg>-59.1495</heading_deg>
            <fpa_deg>0</fpa_deg>
            <sector_index>2810</sector_index>
            <sector_name>ZMP4201M1</sector_name>
            <flight_mode>CRUISE</flight_mode>
        </trajectory_point>
```

A sample csv output looks something like this,

********* TRAJECTORY OUTPUT DATA *********														
** Output Format:														
** simulation_start_time														
**														
** AC	flight_inde	callsign	actype	origin_airp	destinatio	start_time	simulation	simulation	cruise_alti	cruise_tas	origin_airp	destinatio	number_of_trajectory_rec	
** timesta	latitude_d	longitude_	altitude_ft	rocd_fps	tas_knots	course_de	fpa_deg	sector_ind	sector_na	flight_mode				
1.12E+09														
AC	0	ULI7-1883	ALOR1	PHX	EWR	0	1	30	37000	413	1135	17	414	
0	36.98609	-103.248	36011	11	402	69.10494	0.928934	2745	ZAB7271M	CLIMB				
30	37.00589	-103.183	36326.09	10.05187	401.684	69.14406	0.849527	2745	ZAB7271M	CLIMB				
60	37.02568	-103.118	36614.03	9.185469	401.3951	69.1833	0.776858	2745	ZAB7271M	CLIMB				
90	37.04536	-103.053	36877.13	8.393785	401.1313	69.2224	0.710365	2745	ZAB7271M	CLIMB				
120	37.06533	-102.987	37000	0	413	69.26213	0	2745	ZAB7271M	CRUISE				
150	37.08559	-102.92	37000	0	413	69.30262	0	2745	ZAB7271M	CRUISE				
180	37.10574	-102.853	37000	0	413	69.34296	0	2745	ZAB7271M	CRUISE				
210	37.12588	-102.786	37000	0	413	69.38345	0	2745	ZAB7271M	CRUISE				
240	37.14602	-102.718	37000	0	413	69.42405	0	2745	ZAB7271M	CRUISE				
270	37.16605	-102.651	37000	0	413	69.46454	0	2745	ZAB7271M	CRUISE				
300	37.18608	-102.584	37000	0	413	69.50513	0	2745	ZAB7271M	CRUISE				
330	37.20611	-102.517	37000	0	413	69.54584	0	2745	ZAB7271M	CRUISE				
360	37.22603	-102.45	37000	0	413	69.58643	0	2745	ZAB7271M	CRUISE				
390	37.24594	-102.382	37000	0	413	69.62714	0	2745	ZAB7271M	CRUISE				
420	37.26586	-102.315	37000	0	413	69.66799	0	2897	ZKC2201M	CRUISE				
450	37.28567	-102.248	37000	0	413	69.70873	0	2897	ZKC2201M	CRUISE				
480	37.30547	-102.18	37000	0	413	69.74957	0	2897	ZKC2201M	CRUISE				
510	37.32526	-102.113	37000	0	413	69.79053	0	2897	ZKC2201M	CRUISE				
540	37.34496	-102.045	37000	0	413	69.83135	0	2897	ZKC2201M	CRUISE				
570	37.36464	-101.978	37000	0	413	69.8723	0	2897	ZKC2201M	CRUISE				

For each simulation start time, we have list of states for each aircraft. We also list the aircraft type, origin, destination and simulation interval. The xml or csv file then can be read with available parsers in Python and MATLAB.