Open SDSE User Guide

Draft version - September 2017

Jean-Baptiste Chaudron

Summary

1 Introduction

OpenSDSE means Open Source Simulation Distribuee de Systemes Embarques (french accronym for Distributed Simulation for Embedded System). This is a distributed aircraft simulator compliant with the High Level Architecture (HLA) standard, the whole code has been implemented in C/C++. Historically, this project started in 2011 as a test case to allow real-time distributed simulation with CERTI and now we want to distribute it as an open source project under GPL license. This document might help users to properly install, use and understand the OpenSDSE application and is structured as follows:

- Chapter 2 presents a general overview presenting the architecture and its dependencies;
- Chapter 3 describes the installation procedure for Linux;
- Chapter 4.2 explains how to configure the simulation for a proper execution according to you need.

Note: This is a draft version which is incomplete and might contain some errors, inconsistencies and issues.

2 General Overview

2.1 Platforms and dependencies

OpenSDSE can be built and installed on both Linux and OS X but there is no version compatible with windows yet.
Linux was the trageted in the beginning of the project and it has the full support of the simulation and its functionalities. OpenSDSE was successfully installed and tested with Fedora, CentOS, Mint and Ubuntu operating systems. Under OS X, everything works except the joystick api not compatible yet with OS X API (it might be migrated in future).

OpenSDSE depends on several tools and libraries which must be installed before trying to build the sources:

- CMake, the building tool used on the project. Both CERTI and OpenSDSE use CMake as the main building tool.
- CERTI an open source Run-Time Infrastructure (RTI) runtime environment based on HLA. We are using this HLA middleware since the beginning of the project, however, the project might be compatible with some others RTIs which are providing C++ bindings.
- Qt5 required for the GUI compilation (Last versions tested were Qt5.2.0 and Qt5.5.1).
- FlightGear (optional) used for 3D view of the simulation (Last versions tested were 2016.2.1 and 2016.4.4).

OpenSDSE contains parts of others open source projects integrated in the source code:

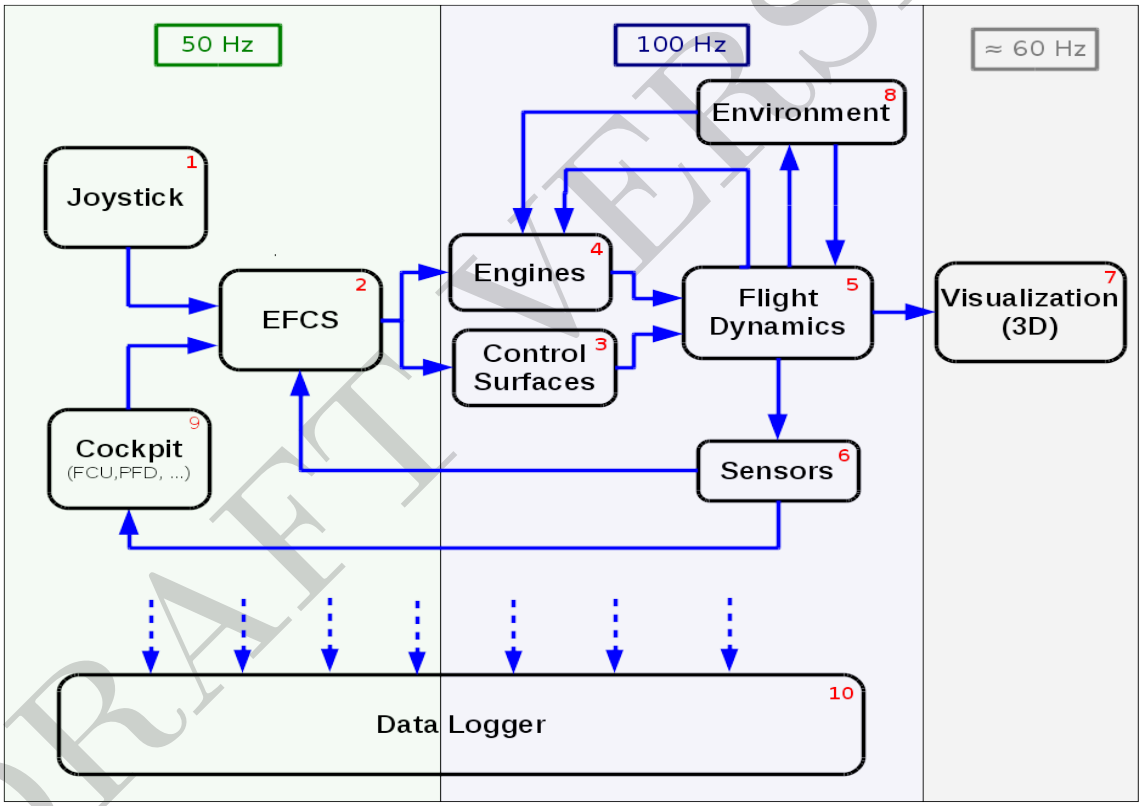
- JsBSim
- Flightgear
- QtFlightInstruments
- TinyXML2

- GnuPlot

We tried our best to properly reference the link to each open source project we were/are using. However, we may have missed some references thus if you find any inconsistency in the references, please contact the project head at: jean-baptiste.chaudron@isae.fr .

2.2 Simulation Architecture

The OpenSDSE HLA Federation is composed of ten federates, each representing a specific part of the aircraft or the environment itself.



2.2.1 Federate 1: Joystick

The Joystick federate acquires the pilot orders transmitted by a joystick system. The elevator, aileron, and rudder axes can then be commanded through the flight control laws. The engine thrust is not directly controlled as the auto-throttle implemented in the flight controller federate alleviates the pilot workload.

2.2.2 Federate 2: EFCS

The Electrical Flight Controller System (EFCS) federate is in charge of the aircraft control; it implements the classical autopilot functions (e.g. speed and altitude hold control systems). Via the Flight Control Unit (FCU) interface he aircraft can then be turned in automatic mode where no pilot is needed or in manual mode where the pilot interacts through the joystick federate (Federate 1).

Note: The design of the flight control is complex task and, for the current version, only essential functions for the autopilot were integrated to operate the aircraft. One way of improvement of this project is to extend the current implementation of the flight controller (for the A320 aircraft used). For example, extend it to a full fly-by-wire component would be a plus.

2.2.3 Federate 3: Control Surfaces

The Control Surfaces federate gathers all the control surface actuators whose deflections change the aerodynamic forces and thus influence the aircraft motion. We consider here left and right ailerons, left and right elevators and rudder. Each control surface is modeled by a second-order system with position and rate saturation to enforce realism. The federate receives control surfaces deflection order from the EFCS federate.

Note: Delay, bias, and hysteresis phenomena can also be considered in the model but this needs to be further investigated.

2.2.4 Federate 4: Engines

The Engines federate simulates two high-bypass turbofan engines whose characteristics change with the atmospheric conditions (as the temperature) and the aircraft Mach number. The federate receive throttle commands from the EFCS federate.

2.2.5 Federate 5: Flight Dynamics

The Flight Dynamics federate represents the core of the simulation model as it computes the equations of motion. Under the action of aerodynamic, gravity, and propulsion forces, the aircraft state evolves accordingly. The aerodynamic coefficients are implemented in the form of look-up tables with many entries as in JsBSim. Computing the aerodynamic forces can require some computational power to complete and it has to be considered for real-time execution.

Note: In the current version, the aerodynamic and mechanical models are based on an A320 aircraft.

2.2.6 Federate 6: Sensors

To perform control, the flight controller needs measurements to evaluate the current aircraft state. these measurements are available through sensors. The Sensors federate simulates a sensor fusion unit as an ADIRU (air data inertial reference unit). The federate receives the relevant parameters from the Flight Dynamics federate. As sensors elements have their own dynamics, the measurements from the Flight dynamics are given to low-pass Butterworth filters (first- or second-order) whose cutoff frequency depends on the nature of the measurement (e.g. rates, accelerations).

Note: Delay, bias, and drift phenomena can also be considered in the model but this needs to be further investigated to ensure proper behavior of the simulation.

2.2.7 Federate 7: 3D Visualization

The actual aircraft position and attitude are fed to the 3D Visualization federate that takes the input from the simulation and package it to as standard Flightgear UDP packet which can be used by the Flightgear Simulator to display the aircraft in a 3D environment.

2.2.8 Federate 8: Environment

The Environment federate simulates the US Standard Atmosphere 1976. Based on the altitude received from the Flight Dynamics federate, it calculates the corresponding atmospheric variables (temperature, pressure, air density, and sound speed) and feeds them back to other federates. Different types of wind and turbulence can also be added (i.e. wind shear, wind gust, Dryden and Von Karman turbulences) to reproduce realistic weather conditions.

2.2.9 Federate 9: Cockpit

The cockpit federate reproduces some pilot graphical interfaces such as in an aircraft cockpit. The OpenSDSE Cockpit, illustrated in Figure 2.1), contains:

- a Primary Flight Display (PFD) (*Item 2 in the figure*)
- a small Electronic Centralized Aircraft Monitor (ECAM) to display Engines information (*Item 3 in the figure*).
- a Navigation Display (ND) (*Item 4 in the figure*)

These interfaces provide visual cues to the pilot from the information sent by the Sensors federate. Last but not least, the Cockpit federate interface also provides a Flight Control Unit graphical interface (*Item 1 in the figure*) which can be use by the user to

set the parameters for the autopilot (as a reference heading). For instance, a reference heading or a reference altitude can be selected via this interface.

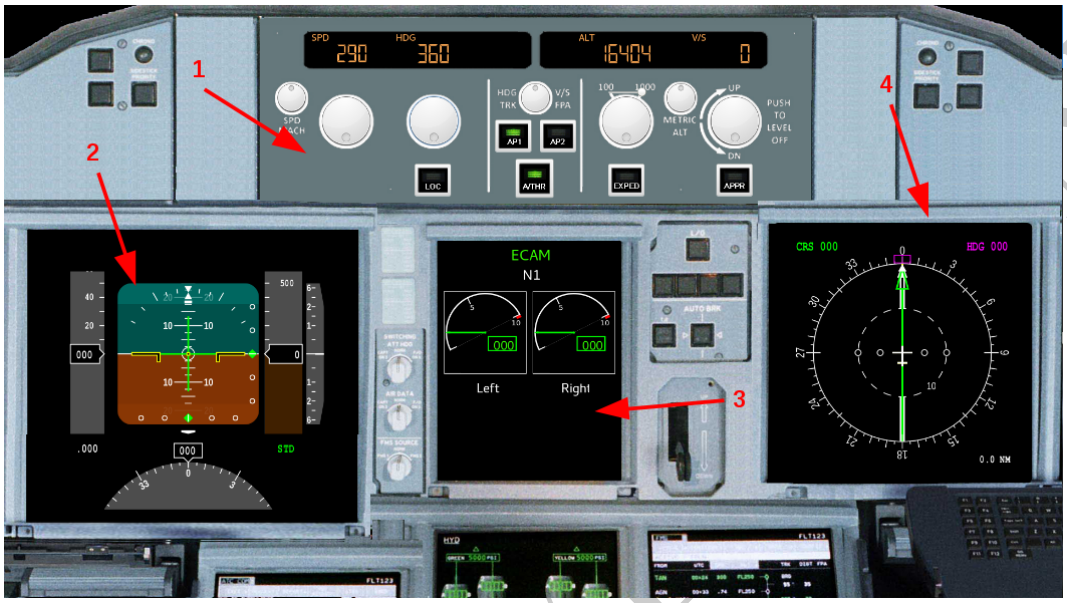


Figure 2.1: OpenSDSE Cockpit Overview

2.2.10 Federate 10: Data Logger

The data logger is a federate which receives data coming from the whole federation (i.e. the global simulation) and can store all values in a csv file. Note that, user might select wisely the parameters to avoid dumping an enormous csv file from a long federation execution.

2.3 Configuration Tool

This program allow to change some configuration parameters for the OpenSDSE HLA federation via a graphical interface (Cf. Figure 2.2). The relevant parameters are stored in an XML file named **sdse_init_parameters.xml** which is then read by all the federates.

The program doesn't have to be modified to take into account some new parameters, these must be added to the xml file directly. However, the xml file should only be hand modified by progammers of the simulator who are required to provide documentation and possible values in order to help an end user.

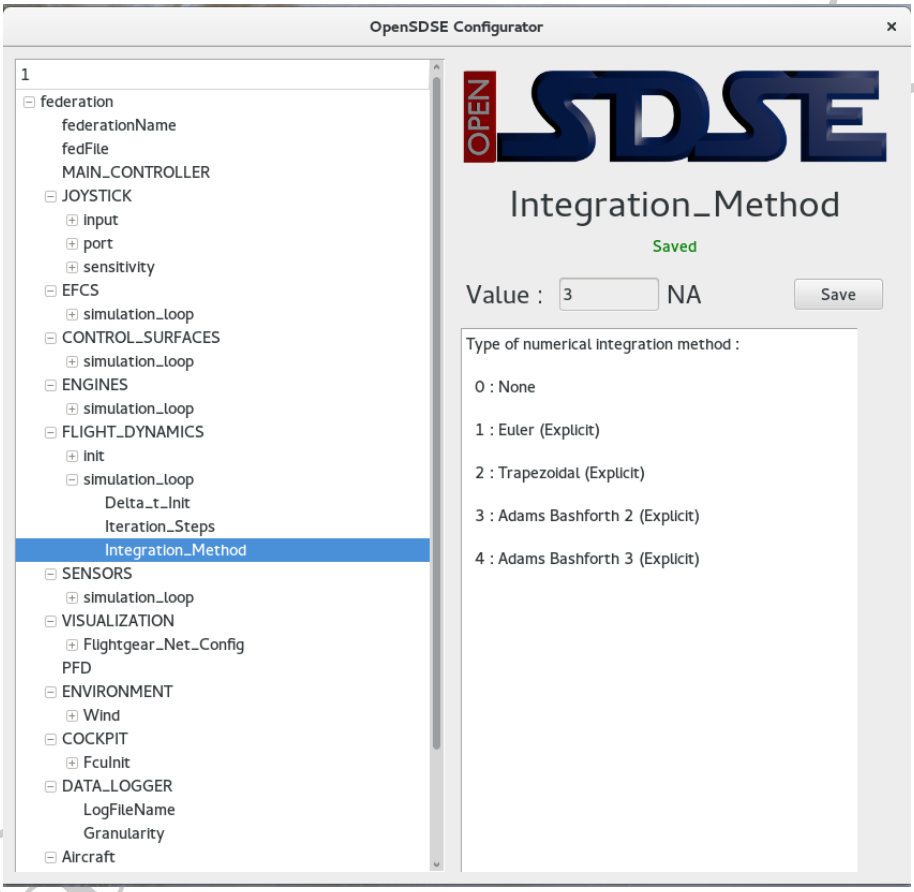


Figure 2.2: OpenSDSE Configurator Overview

3 Installation Procedure

This section will guide you step by step in the installation process to get a running version of OpenSDSE. Please make sure that you installed all required dependencies before building the sources.

3.1 Install the required Dependencies

3.1.1 step 0: Compiler

Before trying to build anything, you must first install a working compiler. For Linux users, we recommend the GNU C Compiler (GCC). For OS X users, we recommend theClang for OS X users. You can easily install these compilers from your package manager.

3.1.2 step 1: CMake

CMake is a very well known cross platform build system. You can install it with your package manager or directly from the website http://www.cmake.org

3.1.3 step 2: CERTI

The best way to install CERTI is to download the sources of the latest version of CERTI.

You can either download a tarball from the release repository or clone a version directly from the git repository. OpenSDSE has been mainly tested with the release version CERTI 3.5.1. Certi has to be compiled from source.

Note: Administrator privileges will be asked if you want to to install the CERTI software in /usr/local.

3.1.4 step 3: Qt5

The SdseConfigurator tool and the Cockpit interface window require Qt5 library (and headers) to be properly compiled and installed. You have to install a proper Qt5 version

on your system, using a package manager or the installer provided on the website https://download.qt.io/archive/qt/. Currently, the versions tested with OpenSDSE are Qt 5.2.0 and Qt 5.5.1.

3.1.5 step 4: FlightGear (optional)

OpenSDSE can use FlightGear as a visualization tool to represent the simulated aircraft in a 3D graphical environment. Depending on your OS, you may find the right package to install FlightGear here. Currently, the versions tested with OpenSDSE are with Flightgear 2016.2.1 and Flightgear 2016.4.4.

3.2 Install OpenSDSE

Once all the required dependencies are installed, you can install OpenSDSE following those steps.

3.2.1 step 0: Downloading the sources

Skip this step if you already have the source folder. You can get the sources from the git repository using this command from the directory where you wish to install OpenSDSE:

```
>> git clone https://openforge.isae.fr/git/opensdse
```

The open_sdse folder contains:

- the *AUTHORS* file lists all the authors and contributors to this project;
- the *README.txt* file provides very short notes about this project;
- the *OpenSDSE_user_guide.pdf* file (this document) is the user documentation;
- the *src/* folder is containing the whole source code package and configuration files.

Note: As you may notice, git is required thus you have to install git using your package manager.

3.2.2 step 1: Configure the compilation script

Under the root folder, we already prepared a script which configure and compile the application (Linux user): **builNInstall.sh**.

In this script, you must set some variables depending on your OS:

- BUILD_TOOL: you can use "make" or "ninja" here with no significant impact on the compilation here.

- QT_DIR: this is the path where is located your Qt5 environment (binaries, libraries and headers). Note that if Qt5 is not installed and configured properly, OpenSDSE will NOT work.
- CERTI_RUN_DIR: this is the path where is located your CERTI HLA environment (binaries, libraries and headers).

Note: OpenSDSE is a cmake project thus, it is not required to use the builNInstall.sh script and you can configure and compile it as any others cmake project with proper configuration.

3.2.3 step 2: Build the sources

In a second step, you may run the **builNInstall.sh** script. If you get any error during the compilation, check the installation of the dependencies or the paths set in the script (or in your cmake environment). Running the script properly will create two additional folders:

- the *build/* folder is containing the cmake environment. Normally, you don't have to play too much in this folder except to check specific configuration variable using ccmake command.
- the *run/* folder contains all the the OpenSDSE federation installation package:
  - In the *bin/* folder, you will found all the executables (i.e. binaries);
  - In the *fom/* folder, you will found the HLA 1.3 and 1516 federation object model files;
  - In the *logs/* folder, you will found the log of the federation as csv file (generated per the Data Logger federate while running) and some example scripts to use GnuPlot;
  - In the *scripts/* folder, you will found some examples scripts to run the federation.

4 Simulation Execution and Configuration

4.1 How to run the federation

4.1.1 Configuring the execution script

At this step, you should now be able to run the OpenSDSE federation. In the scripts/ subfolder, you will find a **Local_Run_bash.sh** which to need to configure to properly run the simulation;

In this script also, you must set some variables depending on your OS:

- CERTI_RUN_DIR: this is the path where is located your CERTI HLA environment (binaries, libraries and headers).
- PATH and LD_LIBRARY_PATH have to be updated according to your Qt5 environment (binaries, libraries and headers). Note that if Qt5 configured properly, OpenSDSE will NOT work.

4.1.2 Launch the whole federation

Then you can try to launchthe simulation manually from a terminal using this command in the script/ directory:

```
>> cd script
>> source Local_Run_bash.sh
```

4.1.3 Launch Flightgear

If you want to launch FlightGear, run this command in a terminal *before* the simulation.

```
>> fgfs --native-fdm=socket,in,50,127.0.0.1,5500,udp --fdm=null
 --callsign=OpenSDSE --timeofday=noon --aircraft=747-100 --airport=KSFO
```

*Note: You need to tune the command line according to your settings. In particular, –native-fdm option must be configured to match with the settings of the Visualization Federate (*Cf. *subsection 4.3.8). For the –aircraft option, it should match with an existing aircraft in your flightgear configuration.*

4.2 How to configure the simulation

4.3 Use the configuration Tool

To change any parameter of OpenSDSE, you need to run the Open SDSE Configuration Tool manually.

```
>> cd bin
>> ./OpenSdseConfigurator
```

You have access, via the graphical interface (Cf. Figure 2.2), to the whole structure of XML file named **sdse_init_parameters.xml**. You can then tune some parameters and you must save any change before running the simulation as the configuration file will be loaded only once in the process.

Note: If you run in a distributed scenario, all federates are expecting to get a valid sdse_init_parameters.xml file in the bin folder. Please make sure that all theses files (all the distributed onces are the sames.

4.3.1 Federation configuration

The federation can be configured per two items:

- **federationName**: this is the name of the federation (whole simulation). By default, it's "sdse_hla13" as OpenSDSE is only supporting HLA 1.3 standard for now.
- **fedFile**: This is the name of the federation file loaded by CERTI to describe the federation.

4.3.2 Joystick federate configuration

You can set some parameters (relevant for the control surfaces) for the calculation of the Joystick federate through different items:

- **input**: Each signal emitted from the joystick is assigned to a number. Aileron, Elevator, Rudder and Throttle must be assigned to an axis number and Flaps, Spoilers, Gears and Brakes must be assigned to a button number. The easiest way to find the right number for each input is by using the jstest-gtk package on Linux. This api, when connected to the joystick, shows in real time the position of the inputs determined by their numbers. It's very easy to identify each one by moving the stick or pressing the buttons.

- **port**: OpenSDSE can manage two joysticks simultaneously. You can set here which joystick will emit each signal. If one joystick is connected, it will be assigned to 0. Two joysticks are assigned to 0 and 1.

- **sensitivity**: Depending on your hardware and the aircraft, you may want to change the effectiveness of the joystick. Setting a low value will result in a lower efficiency. Basically, this is a multiplicative factor added to the value of the input. A negative value will reverse the input.

4.3.3 EFCS federate configuration

In the **simulation_loop** item, you can set some parameters for the calculation of the EFCS federate:

- *Sample time*: the time step of the calculation

- *Discretization type*: choose between several methods of discretization (unused for the moment)

4.3.4 Control Surfaces federate configuration

In the **simulation_loop** item, you can set some parameters for the calculation of the Control Surfaces federate:

- *Delta_t_Init*: the time step of the calculation

- *Iteration_Steps*: the number of sub-loops to enhance the calculation

- *Integration_Method*: Choose between several methods of integration

- *Saturation*: Use the control surfaces limitations of the model in autopilot (unused for the moment).

4.3.5 Engines federate configuration

In the **simulation_loop** item, you can set some parameters for the calculation of the Engines federate:

- *Delta_t_Init*: the time step of the calculation

- *Iteration_Steps*: the number of sub-loops to enhance the calculation

- *Integration_Method*: Choose between several methods of integration

4.3.6 Flight Dynamics federate configuration

In the **init** item, you can set the Initial values for the flight dynamics federate:

- *AltitudeInit*: the starting altitude
- *VitesseInit*: the starting Airspeed
- *LogitudeInit*: the starting longitude
- *HeadingInit*: the starting heading
- *TankFilling*: the tank filling at the beginning of the simulation

In the **simulation_loop** item, you can set some parameters for the calculation of the flight dynamics federate:

- *Delta_t_Init*: the time step of the calculation
- *Iteration_Steps*: the number of sub-loops to enhance the calculation
- *Integration_Method*: Choose between several methods of integration

4.3.7 Sensors federate configuration

In the **simulation_loop** item, you can set some parameters for the calculation of the Sensors federate:

- *Delta_t_Init*: the time step of the calculation
- *Iteration_Steps*: the number of sub-loops to enhance the calculation
- *Integration_Method*: Choose between several methods of integration

4.3.8 Visualization federate configuration

In the **Flightgear_Net_Config** item, you can set the parameters for proper connection with the Flightgear Simulator:

- *UDP_Port*: This is the UDP port for communication with Flighgear using the native UDP protocol
- *IP_Address*: This is the IP Address of the Flighgear interface. Note that for localhost IP address is 127.0.0.1

4.3.9 Environment federate configuration

In the **Wind** item, you can configure the parameters for proper wind simulation

- *WindSpeed*: The constant speed of the wind.
- *WindPsi*: The orientation of the wind.
- *W20* Wingspeed at 6 meters (20ft) This value is used to calculate the Wind Shear as well as the turbulence. 7.72 m/s (25.31 ft/s) or 15 knots for light turbulence 15.43 m/s (50.63 ft/s) or 30 knots for moderate turbulence 23.15 m/s (75.95 ft/s) or 45 knots for severe turbulence
- *POEindex*: Determine the strength of the turbulence : Probability Of Exceedence index (Index 1 : 2E-1, Index 2 : 1E-1, Index 3 : 1E-2 (Light), Index 4 : 1E-3 (Moderate), Index 5 : 1E-4, Index 6 : 1E-5 (Severe), Index 7 : 1E-6)
- *TurbulenceType*: Type of filter used to generate the turbulence. Set "None" for no turbulence.

4.3.10 Cockpit federate configuration

In the **FcuInit** item, you can set the Initial values for the Flight Control Unit (FCU) of the Cockpit federate:

- *HeadingRef*: the required reference heading for the FCU
- *AltitudeRef*: the required reference altitude for the FCU
- *IasRef*: the required reference speed for the FCU

4.3.11 Data Logger federate configuration

The Data Logger federate can be configured per two items:

- **LogFileName**: this is the name of the csv file which will be created per the federate and stored in the logs/ folder. Per default, the name is "sdse_data.csv" and if you change it you might need to adapt the gnuplot_launcher script.
- **Granularity**: This is how often the logger will write in the text file. Note that the data logger runs at 20 milliseconds loop cycle, thus a value of 1 means that the data logger will write every 20 ms cycle in the file which can take some cpu ressources.