# JSBSim: An Open Source Flight Dynamics Model in C++

**Jon S. Berndt**
Project Architect and
Development Coordinator

# Open Source Approach

- Potentially rapid software evolution.
- Software is peer-reviewed.
- Users can modify the source code to meet specific needs.
- Modifications can be incorporated back into the main branch.

# Project Goals

- Make simulation more accessible
- Work towards a RAD environment
- Field a data-driven, generic FDM
- Multi-platform capable
- Versatile, configurable, extensible
- Use object oriented design techniques as appropriate
- Favor simplicity and readability

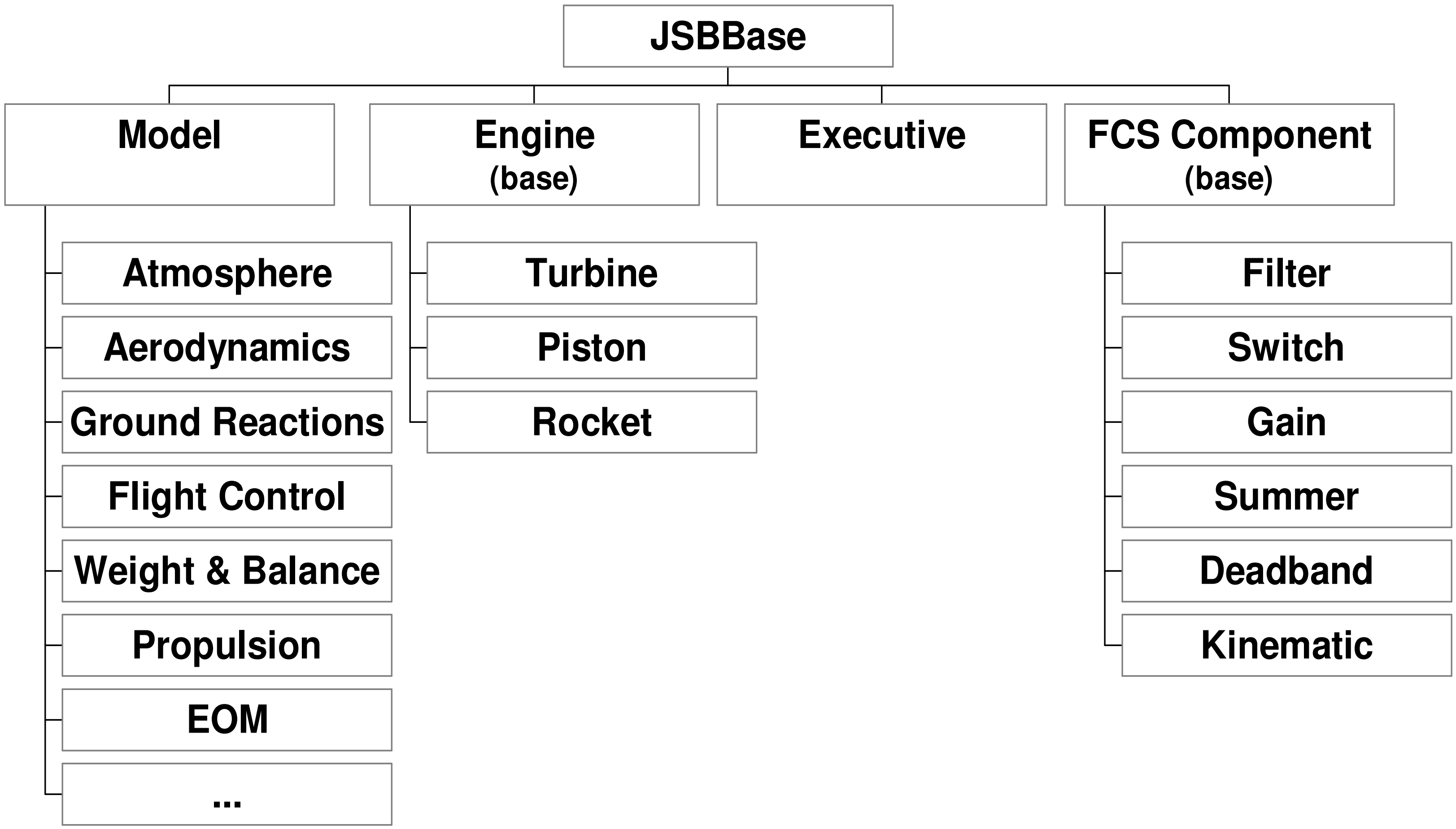
# Target Users

- Lightweight special projects
- Students
- Hobbyists (FlightGear)

# Architecture

- Hierarchical class framework
- Base class provides common features
- Object-oriented design techniques used (polymorphism, inheritance …)

# Class Hierarchy (partial)



# Architecture (II)

- An executive manages model loading, initialization, and cyclic execution.
- Execution rate is determined by the calling application.
- All models within JSBSim are executed at the same frequency.

# Example: Calling JSBSim

## The program interface to JSBSim is minimal:

```
FDMExec = new JSBSim::FGFDMExec();

Script = new JSBSim::FGScript(FDMExec);
result = Script->LoadScript(ScriptName);

while (FDMExec->Run()) {
    if (!Script->RunScript()) break;
}
```

# Architecture (III)

Each of the Models:

- Propulsion
- Flight Control
- Aerodynamics

Acts as a manager, maintaining a list of objects. The managed objects are viewed as specific instances of a generic type.

# Example: Flight Control

```
vector <FCSComponent*> Components;
…
Components->push_back(new Gain());
Components->push_back(new Filter());
…
for (int i=0; i<Components.size(); i++)
{
 Components[i]->Run();
}
```

# Architecture: Properties

Problem: How do we reference program parameters from a text file?

Solution: Properties bind a text string to a class parameter at initialization. Access is controlled by the programmer.

# Vehicle Specification: XML

- More formal than a simple text definition
- Reduces processing
- Several free parsers already exist
- Not merely a “good idea”: DAVE-ML

# Specification Files

- Aircraft
- Engine (piston, turbine ...)
- Thruster (propeller, nozzle ...)
- Batch run script
- Batch run initial conditions
- Plot command file

# Vehicle Specification

```
<FDM_CONFIG NAME="name" VERSION="1.65">
  <METRICS> … </METRICS>
  <UNDERCARRIAGE> … </UNDERCARRIAGE>
  <PROPULSION> … </PROPULSION>
  <AUTOPILOT> … </AUTOPILOT>
  <FLIGHT_CONTROL> … </FLIGHT_CONTROL>
  <AERODYNAMICS> … </AERODYNAMICS>
  <OUTPUT> … </OUTPUT>
</FDM_CONFIG>
```

# Vehicle Specification

- Metrics: Vehicle weight, CG location, Moments of Inertia, wingspan, etc.
- Undercarriage: landing gear, contact points, coefficients and constants.
- Propulsion: Engines, thrusters, tanks.
- Autopilot and Flight Control: Flight control components.

# Vehicle Specification

- Aerodynamics:
  - Axes (drag, side force, lift, roll, pitch, yaw) and coefficients.
- Output: Subsystems, properties, rate, format.

# Specification Example: Ball

```
<FDM_CONFIG NAME="BALL" VERSION="1.65" RELEASE= "ALPHA" >
  <METRICS>
    AC_WINGAREA  1
    AC_WINGSPAN  1
    AC_IXX      10
    AC_IYY      10
    AC_IZZ      10
    AC_EMPTYWT  50
    AC_CGLOC    0 0 0
  </METRICS>
  <UNDERCARRIAGE>
    AC_GEAR Ball 0 0 0 10000 200000 0 0 0 FIXED NONE 0 FIXED
  </UNDERCARRIAGE>
  <AERODYNAMICS>
    <AXIS NAME="DRAG">
      <COEFFICIENT NAME="CD" TYPE="VALUE">
         Drag
         aero/qbar-psf | metrics/Sw-sqft
         0.0001
      </COEFFICIENT>
    </AXIS>
  </AERODYNAMICS>
</FDM_CONFIG>
```

# Aerodynamics

- <u>Coefficient buildup method</u>: Total aerodynamic forces and moments are built up from contributions – as many or few as desired – for each axis.
- Coefficients are defined by value or lookup in a 1, 2, or 3 dimensional table.
- Coefficient definitions include information for turning the coefficient into a force.

# Coefficient Example

```
<COEFFICIENT NAME="CLalpha" TYPE="VECTOR">
    Lift_due_to_alpha
    8
    velocities/mach-norm
    aero/qbar-psf | metrics/Sw-sqft | aero/alpha-rad
    0.00 4.50
    0.40 3.80
    0.60 3.60
    1.05 4.50
    1.40 4.00
    2.80 2.50
    6.00 1.10
    9.00 1.00
</COEFFICIENT>
```

# Flight Control

- A flight control system is defined using a list of components (switch, gain, filters, etc.).
- Components are executed serially – “bucket-brigade” style.
- Currently, the components all execute at the same rate.
- Sensors are “perfect”.

# FCS Component Example

```
<COMPONENT NAME="HDot Error" TYPE="SUMMER">
  INPUT               fcs/hdot-command
  INPUT              -velocities/h-dot-fps
</COMPONENT>

<COMPONENT NAME="Alt Hold Switch" TYPE="SWITCH">
  <TEST LOGIC="DEFAULT" VALUE="0.0">
  </TEST>
  <TEST LOGIC="AND" VALUE="fcs/hdot-error">
    ap/altitude_hold == 1
  </TEST>
</COMPONENT>
```

# Creating Aircraft Models

- Aeromatic web application can be used for a first, rough cut.
- The resulting aircraft specification file is refined based on information from: TCDS (FAA web site), AIAA and NACA/NASA technical reports, textbook examples, university web sites, and longhand calculations.

# Creating Aircraft Models (II)

## DATCOM+

- Digital DATCOM has been extended to produce output directly in the format used by the JSBSim aerodynamic definition.

# Using JSBSim (batch mode)

- Build the aircraft model.
- Design a script or set of scripts.
- Select output parameters.
- Execute the scripts – possibly iterating over several test cases.
- Generate plot files.
- Compare performance with actual aircraft performance.

# Further Development

- Currently developing support for alternate integration schemes.
- Will implement a more robust and formal XML parser
- Will consider migrating to DAVE-ML.
- Multi-Body support possible.
- Companion tools and editors (DATCOM+, simplot, etc.)