AIAA Modeling and Simulation Technologies Conference and Exhibit
16 - 19 August 2004, Providence, Rhode Island

AIAA 2004-4923

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

Jon S. Berndt*
JSBSim Project
League City, TX

Abstract: This paper gives an overview of JSBSim, an open source, multi-platform, flight dynamics model (FDM) framework written in the C++ programming language. JSBSim is designed to support simulation modeling of arbitrary aerospace craft without the need for specific compiled and linked program code. Instead, it relies on a relatively simple model specification written in an extensible markup language (XML) format. Also presented are some key (perhaps unique) features employed in the framework. Aspects of developing an open source project are identified. Notable uses of JSBSim are listed.

I. Introduction

JSBSim[1] was conceived in 1996 as a batch simulation application aimed at modeling flight dynamics and control for aircraft.† It was accepted that such a tool could be useful in an academic setting as a freely available aid in aircraft design and controls courses. In 1998, the author began working with the FlightGear project.[2] FlightGear is a sophisticated, full-featured, desktop flight simulator framework for use in research or academic environments, for the development and pursuit of interesting flight simulation ideas, and as an end-user application. At that time, FlightGear was using the LaRCsim[3] flight dynamics model (FDM). LaRCsim requires new aircraft to be modeled in program code. Discussions with developers in the FlightGear community suggested that in order to make flight simulation more accessible, creating a generic, completely data-driven FDM framework would be helpful. That is, specific aircraft would be defined in data files, and no new program code would be required to model any arbitrary aircraft. Additional characteristics of such a framework include:

- Employs object-oriented design principles
- Compiles across common platforms and compilers
- Readily available as an open source application
- Is self-documenting

JSBSim was integrated with FlightGear in 1999, and is today the default flight model. JSBSim retains the capability to run in a batch mode. The volunteer development team has grown over the years, and vigorous development continues. Since JSBSim is provided and developed under the GNU General Public License, it is available for use in other simulation projects with few restrictions.

II. Architecture

A. Benefits in the Use of C++

The C++ programming language is ideal for use in flight simulation software in part because of its support of the primary object-oriented concepts: *polymorphism*, *inheritance*, *encapsulation*, and *abstraction*.

Polymorphism (i.e. "having many forms") is a characteristic of objects that operate differently internally, but have the same interface. For example, the JSBSim implementation of a flight control system (FCS) consists of various classes of filters, switches, and gains that each have a unique internal implementation. The public interface to the components consists of the Run() and GetOutput() methods, which are both virtual member functions of the component base class. These two methods are overridden in derived classes that represent specific components with

*JSBSim Project Architect and Development Coordinator, Senior Member AIAA.

†Many types of "craft" can be modeled in JSBSim (e.g. a ball, aircraft, rocket, etc.) but within this paper the term "aircraft" will be used for all types.

unique behavior. Each instance of a component is referred to by the base class pointer, but the behavior is determined by the logic defined within the Run() method of the specific derived component class.

Inheritance is the derivation of a class from another in order to provide base class characteristics to the derived class. Using the example of the JSBSim FCS component classes again, each of the specific component classes (switch, summer, filter, etc.) is derived from a more generic component class that features characteristics that all components share. Inheritance is a common way to implement polymorphism.

Encapsulation is the concept of hiding and protecting data in a class, preventing corruption by outside processes. For example, the output of the FCS components may only be retrieved, and not modified. In addition, some of the JSBSim FCS components store past values. These should never be accessible by any other part of the program.

Abstraction is "the elimination of the irrelevant and the amplification of the essential".[4] The FCS component base class, for instance, provides the two simple functions Run() and GetOutput() that are used to operate a component. The details of how each specific component operates do not need to be exposed.

The Standard Template Library (STL) is a library of containers and algorithms. The STL provides containers such as the vector class that operate like arrays that shrink or grow as needed. This capability is particularly useful in JSBSim, because the number of engines, coefficients, FCS components, etc. are unknown until the program is executed.

Other incidental advantages in the use of C++ include the ready availability of engineers, mathematicians, etc. who are also experienced C++ programmers, and the availability of mature – and also free – C++ compilers, utilities, tools, and integrated development environments (IDE) across the set of target platforms.

B. Class Hierarchy

The collection of classes that make up the JSBSim framework resembles a "family tree" with a common parent at the head of the hierarchy (see Fig. 1). This graphically illustrates the concept of inheritance and – indirectly – code reuse. An entire class framework does not necessarily have to conform to a completely hierarchical arrangement. In JSBSim, several classes represent generic objects or concepts that are each fully realized in derived classes. These base classes can be viewed as providing the abstract interface that is used to access each of the derived objects. There are also operations and *attributes* (a fancy name for class member variables) that are common to almost all classes, so it makes sense to implement those characteristics in a common base class. For instance, the framework base class (FGJSBBase*) implements a messaging capability, providing message handling methods and storage. Any descendant class can put a message on the queue to be retrieved and processed by the parent application, or by any class instance that is also derived from the base class. Unit conversion routines and constants also reside in the base class, available through inheritance to any of the many descendant classes that need them.

One of the key challenges faced in creating a generic FDM is in designing it to permit the modeling of completely arbitrary vehicle configurations. The framework needs to be able to transparently handle modeling craft ranging from a simple ball (useful for testing purposes, see Listing 1), to a missile, an aircraft, rocket, hybrid vehicle, a rotorcraft, and so

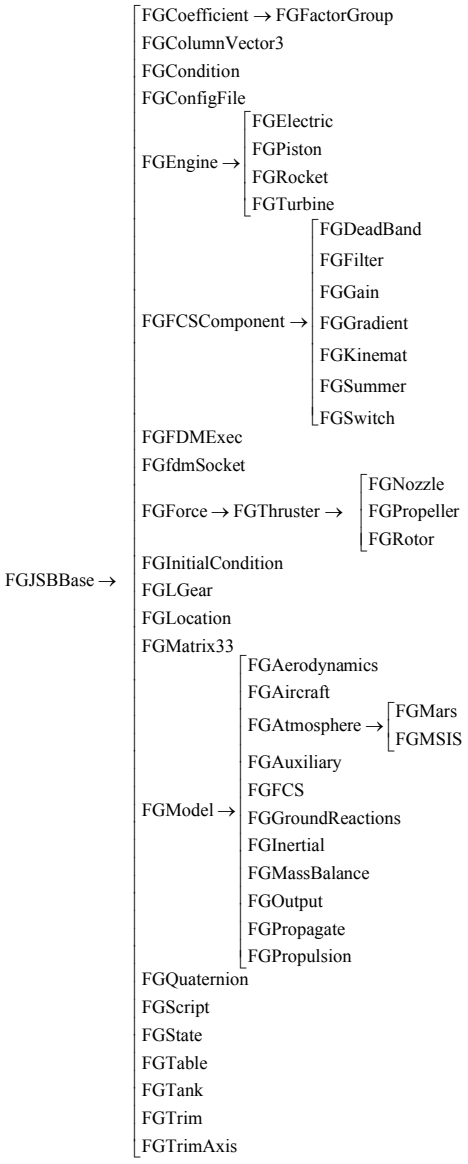

Figure 1. JSBSim Class Hierarchy

*The "FG" prefix refers to FlightGear.

on. These craft could feature different propulsion systems, ground reaction mechanisms, aerodynamic characteristics, and control systems (if any). All four of the previously mentioned object oriented design characteristics played a part in addressing this challenge.

For the propulsion system, an engine base class was created that provides a basic, common, engine interface. The class includes access methods for starter, throttle, and mixture settings, and methods for reading fuel flow rates, temperatures, etc. Specific engine types are derived from the engine base class, providing the actual implementations for turbine, rocket, piston, and electric engines. The derived classes each provide a Calculate() function (overriding the empty, virtual Calculate() function from the base class) that is called to run an iteration of the specific engine instance, determining the force and moment values that the particular engine transmits to the aircraft. In addition to the engine force and moment calculations performed inside the Calculate() function, other quantities are determined as well, including various temperatures, rotational rates, and pressures.

A manager class handles all aspects of the propulsion system operation, including loading the definition of each engine, tank, and thruster (the device used to turn engine power into a force), initialization, cyclic execution while running, and destruction at completion. By providing an abstract interface to the engine object, each specific engine instance (no matter what derived type) can be executed the same way, from the point of view of the propulsion system manager class:

```
for (i=0; i<number_of_engines; i++)
{
  Engines[i]->Calculate();
  vForces  += Engines[i]->GetBodyForces();  // sum body frame forces for all engines
  vMoments += Engines[i]->GetMoments();     // sum body frame moments for all engines
}
```

The same arrangement is used for the FCS. Several different types of components may be used in assembling the control laws for an aircraft, all of which are derived from the FGFCSComponent. The set of components are executed sequentially to process pilot inputs and arrive at effector positions:

```
for (i=0; i<FCSComponents.size(); i++) FCSComponents[i]->Run();
```

The FCS manager class is insulated from details of the operation of each distinct kind of component, needing only to know the interface provided in the component base class.

C. Model Definition in Specification Files

Early on in the development of JSBSim, the goal was set to provide the ability to model any arbitrary aircraft without the need for unique program code. By some definitions[5,6], this feature gives a statistic for *code* reuse for JSBSim at 100% – all of the program code needed to model any aircraft is present. The responsibility for modeling a unique aircraft has thus been offloaded from program code into a specification file read at runtime.

Specification files, script files, etc. for JSBSim are written in extensible markup language (XML) format. A cursory look at an XML file will show that it bears a close resemblance to a hypertext markup language (HTML) document. This format is well suited for aircraft specification and indeed a standard is being drafted for simulation model exchange using XML.[7]

Several files can be involved in running JSBSim:

- Aircraft specification file
- Engine specification file[s]
- Thruster specification file[s] (propeller, nozzle, etc.)
- Initialization file (for batch mode only)
- Script file (for batch mode only)

Over a development period of several years, the minimum set of parameters needed to specify a unique aircraft for JSBSim has been refined. The core structure of the specification file for aircraft is shown below. (In the XML definitions that follow, braces denote user-specified items, brackets surround optional items, the vertical bar separates choices, and upper case items refer to literal identifiers.)

```
<FDM_CONFIG NAME="{name}" VERSION="{number}" [RELEASE="{ALPHA | BETA}"]>

  <METRICS>
```

```
  …
</METRICS>

<UNDERCARRIAGE>
  …
</UNDERCARRIAGE>

<PROPULSION>
  …
</PROPULSION>

<AUTOPILOT {NAME="{name}" | FILE="{filename}"}>
  …
</AUTOPILOT>

<FLIGHT_CONTROL {NAME="{name}" | FILE="{filename}"}>
  …
</FLIGHT_CONTROL>

<AERODYNAMICS>
  <AXIS NAME="{DRAG | SIDE | LIFT | ROLL | PITCH | YAW}">
    …
  </AXIS>
  …
</AERODYNAMICS>

<OUTPUT NAME="{name | IP | COUT}" TYPE="{NONE | SOCKET | CSV | TABULAR | TERMINAL}">
  …
</OUTPUT>
</FDM_CONFIG>
```

The individual sections (e.g. AERODYNAMICS, FLIGHT_CONTROL) roughly correspond to the simulation models present in the JSBSim class framework. The models are described later.

D. Properties

Simulation programs need to manage a large amount of state information. With especially large programs, the data management task can cause problems:

- Contributors find it harder and harder to master the number of interfaces necessary to make any useful additions to the program, so contributions slow down.
- Runtime configurability becomes increasingly difficult, with different modules using different mechanisms (environment variables, custom specification files, command-line options, etc.).
- The order of initialization of modules is complicated and brittle, since one module's initialization routines might need to set or retrieve state information from an uninitialized module.
- Extensibility through add-on scripts, specification files, etc. is limited to the state information that the program provides, and non-code-writing developers often have to wait too long for the developers to get around to adding a new variable.

The *Property Manager*[*] system provides a single interface for chosen program state information, and allows the creation of new properties dynamically at run-time. The latter capability is especially important for the JSBSim FCS model because the components that make up the control law definition for an aircraft exist only in a specification file. At runtime, after parsing the component definitions, the components are *instantiated*, and the property manager creates a property to store the output value of each component.

Properties themselves are like global variables with selectively limited visibility (read or read/write) that are categorized into a hierarchical, tree-like structure that is similar to the structure of a Unix file system. The structure of the property tree includes a root node, sub nodes, (like subdirectories) and end-nodes (properties). Similar to a Unix file system, properties can be referenced relative to the current node, or to the root node. Nodes can be *grafted* onto other nodes similar to symbolically linking files or directories to other files or directories in a file system.

Properties are used throughout JSBSim and FlightGear to refer to specific parameters in program code. Properties can be assigned from the command line, from specification files and scripts, and – in the case of

[*]The property manager is a component of the SimGear library, an open source collection of simulation modules that is an integral part of FlightGear. See www.simgear.org.

FlightGear – even via a socket interface.* References to parameters as properties look like `position/h-sl-ft`, and `aero/qbar-psf`.

In JSBSim, selected property management functionality is wrapped in a class by itself. At program initialization the JSBSim executive creates a top-level property manager node. All other classes that expose variables refer to this node. Variables are exposed using the property manager `Tie()` function call. The `Tie()` function associates (or *binds*) a property string (such as "`fcs/left-aileron-pos-rad`") either with the relevant variable itself or with a function. This way of exposing variables might seem to break the concept of data hiding. However, variables can be (and usually are) bound to properties in a read-only state. Therefore, the programmer has control over both *which* variables in a class are exposed via properties, and *how* they can – or cannot – be used.

To illustrate the power of using properties and configuration files, consider the case of a high-performance jet aircraft model. Assume for a moment that a new switch has been added to the control panel for the example aircraft that allows the pilot to override pitch limits in the FCS. For FlightGear, the instrument panel is defined in a configuration file, and the switch is defined there for visual display. A property name is also assigned to the switch definition. Within the flight control portion of the JSBSim aircraft specification file, that same property name assigned to the pitch override switch in the instrument panel definition file can be used to channel the control laws through the desired path as a function of the switch position. No code needs to be touched.

III. Simulation Models

A. Overview

The set of models that comprise the JSBSim framework includes:

- Aerodynamics
- Propulsion
- Flight Control

The aerodynamic, flight control, propulsion, and ground reaction models all feature a manager class that loads model information from a file, creates and maintains a list of objects, and cyclically executes each object in the list. These models are described in more detail in the following sections.

B. Aerodynamics

JSBSim uses a coefficient build-up method for modeling the aerodynamic characteristics of aircraft. Any number of coefficients (or none at all) can be defined for each of the three axes (rotational and translational): lift, drag, side, pitch, roll, and yaw. Each coefficient specification includes a definition comment, and a reference to the parameters needed to turn the coefficient into a force or moment. The coefficient can be a simple value, or determined by lookup into a one, two, or three-dimensional table.† The general format of a coefficient specification is as follows:

```
<COEFFICIENT NAME="{name}" TYPE="{VALUE | VECTOR | TABLE | TABLE3D}">
  {description}
  [{number of rows} [{number of columns}]]
  [{table lookup property}]
  [{row lookup property} [{column lookup property}]]
  {property factor1} [{| property factor2} [| …]]
  {value or lookup table data}
</COEFFICIENT>
```

A specific example of a coefficient is shown below:

```
<COEFFICIENT NAME="Clda" TYPE="TABLE">
  Roll_moment_due_to_aileron
  8 2
  velocities/mach-norm position/h-sl-ft
  aero/qbar-psf | metrics/Sw-sqft | metrics/bw-ft | fcs/left-aileron-pos-rad
       0.0   80000
```

*FlightGear provides a telnet interface and an http page-serving capability. This provides the capability, for instance, to run Flightgear on one computer and have a user somewhere else control Flightgear by changing the value of properties through the telnet interface, then having data displayed via web pages served by FlightGear.

†Support for equations is under development.

```
  0.0   0.05   0.05
  0.4   0.07   0.05
  0.6   0.08   0.05
  1.0   0.11   0.05
  1.4   0.08   0.06
  1.6   0.07   0.06
  2.4   0.06   0.05
  6.0   0.03   0.02
</COEFFICIENT>
```

The example above defines a roll moment contribution due to aileron deflection. At run time, the coefficient value is determined from a table lookup using mach and altitude values as lookup indices into the table. The resulting coefficient is multiplied by dynamic pressure, wing area, wingspan, and aileron position to get the rolling moment contribution.

During simulation initialization, as each model section of the aircraft specification file is encountered, the file handle is passed to the Load() method of the relevant model class for parsing. For the aerodynamics model the FGAerodynamics class provides the Load() method. As each coefficient definition is encountered and parsed, a new FGCoefficient class object is instantiated (created) and stored in a Standard Template Library (STL) vector – which is effectively a self-sizing array. At runtime, each FGCoefficient object is queried for its value, and a total force or moment sum is subsequently calculated for each axis.

C. Propulsion

JSBSim models the operation of several types of engines. The models are not rigorously complete, high fidelity, engineering models – they exist for the purpose of providing a realistic perception of the propulsion system from the point of view of a pilot, as well as providing accurate forces and moments on the aircraft.

The FGPropulsion class manages all aspects of the propulsion system, which is comprised of zero, one, or several engines (perhaps of different types) each paired with a thruster, and one or more fuel tanks. The propulsion system definition contained in the aircraft specification file itself includes the filenames of both the engine and the thruster specification files to be used, as well as non-specific information such as the location and orientation of the engine and associated thruster. The engine and thruster specification files contain the unique characteristics for those objects. By storing the unique information separately, the engine and thruster specification files can be reused in other aircraft by simply including references to them in the aircraft specification files.

An engine/thruster combination is specified for an aircraft in the aircraft specification file as follows:

```
<AC_ENGINE FILE="{filename}">
  XLOC  {location along X-axis in structural frame coordinates, inches}
  YLOC  {location along Y-axis}
  ZLOC  {location along Z-axis}
  PITCH {pitch angle, degrees}
  YAW   {yaw angle, degrees}
  FEED  {tank to draw fuel from}  <!-- one or more -->
  <AC_THRUSTER FILE="{filename}">
    XLOC  {location along X-axis in structural frame coordinates, inches}
    YLOC  {location along Y-axis}
    ZLOC  {location along Z-axis}
    PITCH {pitch angle, degrees}
    YAW   {yaw angle, degrees}
    [SENSE {rotation sense: 1=CW, -1=CCW}]
  </AC_THRUSTER>
</AC_ENGINE>
```

Four engine types are currently supported: piston, turbine, rocket, and electric. Both propellers and nozzles (referred to generically as thrusters) are also modeled. The format used to define a piston engine is shown below:

```
<FG_PISTON NAME="{name}">
  MINMP          {minimum manifold pressure}
  MAXMP          {maximum manifold pressure}
  DISPLACEMENT   {displacement}
  MAXHP          {maximum horsepower}
  CYCLES         {number of cycles}
  IDLERPM        {RPM at idle}
  MAXTHROTTLE    {maximum throttle <=1}
```

```
  MINTHROTTLE    {minimum throttle >=0}
  NUMBOOSTSPEEDS {number of boost speeds}
  BOOSTOVERRIDE  {pilot controls boost: 0 or 1}
  RATEDBOOST#    {#=1,2,3; pressure boost, psi}
  RATEDPOWER#    {#=1,2,3; rated power at boost}
  RATEDRPM#      {#=1,2,3; RPM at rated power}
  RATEDALTITUDE# {#=1,2,3; max altitude for rated boost power}
  TAKEOFFBOOST   {Boost available at takeoff, psi}
</FG_PISTON>
```

The specification above reveals that turbocharging is supported, but turbocharging is defaulted to OFF and the statements dealing with turbocharging need not be included in the definition for a non-turbocharging engine.

As mentioned previously, each engine is paired with a thruster that ultimately converts engine power into a force that acts on the airframe. For the piston engine, the proper thruster to use is the propeller. A propeller is defined in a specification file that includes tables defining the thrust and power coefficient as a function of advance ratio and blade angle (for variable-pitch propellers):

```
<FG_PROPELLER NAME="prop">
  IXX         {Ixx}
  DIAMETER    {diameter}
  NUMBLADES   {#blades}
  GEARRATIO   {ratio of engine RPM / propeller RPM}
  MINPITCH    {minimum pitch angle}
  MAXPITCH    {maximum pitch angle}
  MINRPM      {minimum RPM} <!-- used for variable pitch propellors -->
  MAXRPM      {maximum RPM} <!-- used for variable pitch propellors -->
  C_THRUST    {number of rows} [{number of columns}]
    {thrust coefficient table}
  C_POWER    {number of rows} [{number of columns}]
    {power coefficient table}
</FG_PROPELLER>
```

The propeller thrust is determined by first calculating the power required to keep the propeller spinning in the current conditions. A comparison is made between required power and current engine power output, and the excess power is used to accelerate (or decelerate) the propeller. Propeller thrust can then be determined from the thrust coefficient table.

Additional effects modeled with the piston engine / propeller combination include gyroscopic effects due to the spinning drive shaft and propeller, and P-factor. P-factor is approximated by shifting the point of application of the thrust vector over the propeller disk as a function of alpha.

JSBSim features a turbine engine model. The turbine engine is paired with a thruster referred to as a direct thruster, which is effectively a pass-through for thrust calculated directly in the turbine engine model. A rocket engine is also modeled. The rocket engine is paired with a nozzle.

D. Flight Control System

JSBSim models a flight control system by providing a suite of components that can be connected to represent the control laws for an aircraft. Similar to the aerodynamic and the propulsion system models, there is a managing class (FGFCS) and a component base class (FGFCSComponent). Specific control components are derived from the base class. The components that are currently modeled include:

- Multi-purpose filter (lag, washout, lead-lag, second order, integrator, etc.)
- Switch
- Gain (has multiple functions)
- Scheduled gain (can be dependent on mach, dynamic pressure, etc.)
- Deadband
- Summing
- Kinematic (for actuators)

As is the case with the propulsion and aerodynamic model managers, the FCS manager parses a list of component definitions from the aircraft specification file and creates a list of components. At runtime, the list is iterated over and each component in turn is executed, with the result – the output – of one component in a string being the input of a subsequent component in the string until the last component in that string is reached. At that point, the result is assigned to the final destination. In many cases, the specified final destination is an aerosurface.

Table 1. Lag Filter Definition

	Filter	Filter definition

Generic form	$\frac{C_1}{s + C_1}$	`<COMPONENT NAME="{name}" TYPE="LAG_FILTER">`
		`INPUT {property}`
		`C1 {value}`
		`[OUTPUT {property}]`
		`</COMPONENT>`

Specific example	$\frac{600}{s + 600}$	`<COMPONENT NAME="LAG_1" TYPE="LAG_FILTER">`
		`INPUT fcs/aileron_cmd`
		`C1 600`
		`</COMPONENT>`

1. Specification and Mechanization of Selected Components

Filter Component

The filter component can model several types of filters. The Tustin substitution is used to take filter definitions from LaPlace space to the time domain. At the time of simulation initialization, the filter definitions are loaded, and filter coefficients are calculated. The general format for a filter specification is:

```
<COMPONENT NAME="{name}" TYPE="{type}">
   INPUT {property}
   C1 {value}
   [C2 {value}]
   [C3 {value}]
   [C4 {value}]
   [C5 {value}]
   [C6 {value}]
   [OUTPUT {property}]
</COMPONENT>
```

The specification for a lag filter is shown in Table 1. A specific example is also shown. The definitions for a second order filter and an integrator are shown in Table 2.

Table 2. Additional Filter Definitions

	Filter	Filter definition

Second order filter	$\frac{C_1 s^2 + C_2 s + C_3}{C_4 s^2 + C_5 s + C_6}$	`<COMPONENT NAME="{name}" TYPE="SECOND_ORDER_FILTER">`
		`INPUT {property}`
		`C1 {value}`
		`C2 {value}`
		`C3 {value}`
		`C4 {value}`
		`C5 {value}`
		`C6 {value}`
		`[OUTPUT {property}]`
		`</COMPONENT>`

Integrator	$\frac{C_1}{s}$	`<COMPONENT NAME="{name}" TYPE="INTEGRATOR">`
		`INPUT {property}`
		`C1 {value}`
		`[TRIGGER {property}]`
		`[OUTPUT {property}]`
		`</COMPONENT>`

The TRIGGER keyword that is part of the integrator definition specifies that while the property variable that is paired with it is non-zero then the inputs should be zeroed out. This protects against integrator wind-up. The trigger property might refer to a switch that goes high if an aerosurface reaches a hard limit.

The filter class models additional filters in a similar manner.

Switch Component

The switch component models a switch - either an on/off or a multi-choice rotary switch. The switch can represent a physical cockpit switch, or it can represent a logical switch, where several conditions need to be satisfied before a particular state is established. The value of the switch – which is the output value for the component instance – is chosen depending on the state of the switch. Each switch is comprised of two or more tests. Each test has a value associated with it. The first test that evaluates to true will set the output value of the switch according to the value parameter belonging to that test.

Each test contains one or more conditions, which each must be logically related (if there are more than one) given the value of the LOGIC parameter. The condition takes the form:

```
{property} {conditional} {property | value}
```

For example,

```
aero/qbar GE 21.0
```

or,

```
aero/roll_rate < aero/pitch_rate
```

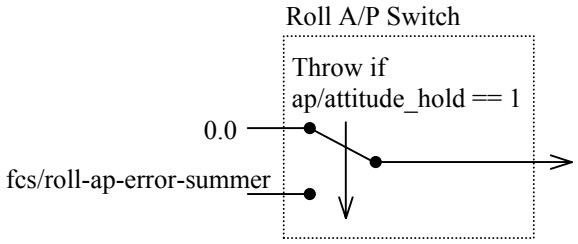

Figure 2. Example Switch

Within a test, a *condition group* can be specified. A condition group allows for complex groupings of logical comparisons. Each condition group contains additional conditions, as well as possibly additional condition groups. The format of the switch specification is as follows:

```
<COMPONENT NAME="{name}" TYPE="SWITCH">
  <TEST LOGIC="{AND|OR|DEFAULT}" OUTPUT="{property|value}">
    {property} {conditional} {property|value}
  [<TEST LOGIC="{AND|OR}" OUTPUT="{property|value}">
    {property} {conditional} {property|value}]
  [<CONDITION_GROUP LOGIC="{AND|OR}">
    {property} {conditional} {property|value}
    ...
  </CONDITION_GROUP>]
  ...
</COMPONENT>
```

Here is a simple example showing the definition of the switch shown in Figure 2:

```
<COMPONENT NAME="Roll A/P Switch" TYPE="SWITCH">
  <TEST LOGIC="DEFAULT" VALUE="0.0">
  </TEST>
  <TEST LOGIC="AND" VALUE="fcs/roll-ap-error-summer">
    ap/attitude_hold == 1
  </TEST>
</COMPONENT>
```

The above example specifies that the *default* value of the component (i.e. the output property of the component, addressed by the property, fcs/roll-ap-autoswitch) is 0.0. When the attitude hold switch is selected (property fcs/attitude_hold takes the value 1), the value of the switch component will be whatever value fcs/roll-ap-error-summer has. [Note that when an FCS component is created, an associated property is also created. The property is named by changing the name of the component to lower case, replacing special characters with hyphens, and applying the "fcs" property node prefix.]

Summer Component

The summer component sums two or more inputs. These can be pilot control inputs or other properties (such as environment or state variables), and a bias can be added in using the BIAS keyword. The form of the summer component specification is:

```
<COMPONENT NAME="{name}" TYPE="SUMMER">
       INPUT {[-]property}
       INPUT {[-]property}
       [BIAS {[-]value}]
       […]
       [CLIPTO {min} {max}]
       [OUTPUT {property}]
</COMPONENT>
```

Note that an input property name may be immediately preceded by a minus sign. Here is an example of a summer component specification:

```
<COMPONENT NAME="Roll A/P Error summer" TYPE="SUMMER">
       INPUT  velocities/p-rad_sec         <!-- Roll rate, rad/sec       -->
       INPUT -fcs/roll-ap-wing-leveler     <!-- From upstream components -->
       INPUT  fcs/roll-ap-error-integrator <!-- From upstream components -->
       CLIPTO -1 1                         <!-- Limit output             -->
</COMPONENT>
```

Note that there can be only one BIAS statement per component.

The flight control components can be used to define control laws that connect a stick and rudder, but they can also be used to construct an autopilot, as well. The FCS component manager class maintains two distinct lists of components: one for autopilot components and one for flight control components. Provision of autopilot controls is particularly useful for running test cases automatically. An example illustrating the use of the FCS components for an autopilot is presented later.

There are some limitations in the current implementation of the FCS:

- *Sensor* inputs to the FCS are perfect. No sensor modeling is done at this time.
- The FCS components all execute at the same rate.
- Filter component inputs are specified as LaPlace filter coefficients.
- Aerosurfaces are perfect, and are not affected by the environment.

These limitations can be overcome, but implementing them will depend on user demand, and time constraints.

E. Other Models and Features

Additional models and features are described briefly below.

It should be mentioned that some models require physical information about the aircraft, such as center of gravity (CG) location, landing gear location, etc. Locations are specified in inches, and in the structural frame. This frame is defined with the X-axis increasing aft, Y increasing towards the right, and Z completing the right hand system. The use of this reference frame was adopted based on the most commonly encountered frame used in technical reports, textbooks, and other reference material.

1. Atmosphere

The default atmosphere model for JSBSim was the 1959 Standard Atmosphere model. It has been updated to the 1976 Standard Atmosphere. More recently, a drive to allow environmental modeling of planets other than Earth has lead to some modifications to the FGAtmosphere class. Some member functions of the FGAtmosphere class have been reclassified as virtual methods, allowing them to be overridden in derived classes. Two new atmosphere classes have been created: FGMars and FGMSIS, representing a very simple Mars atmosphere and the NRLMSISE-00[8,9] atmosphere model, respectively. The Mars GRAM[10] source code is available, however it is not freely re-releasable, so it cannot be a formal part of an open source project such as JSBSim.

2. Mass Properties

The changing mass, center of gravity, and moments of inertia are continuously calculated as fuel burns off.

3. Equations of Motion

In the very early versions of JSBSim, the equations of motion (EOM) described motion in a flat earth reference frame. When JSBSim was being integrated with FlightGear, the EOM evolved to support a non-rotating spherical earth model. Further development has recently lead to support of aircraft motion over a rotating, spherical earth. Quaternions are used for angular state integration. The effects of rotating mechanisms (engine/propeller) are included. Steady winds can be specified. Support of turbulence is in place, and turbulence models are under development. A fixed time step is assumed.

4. Ground Reactions

The ground reaction model is loosely based on the approach presented in Reference 11. The model – like the FCS, propulsion, and aerodynamics models – consists of a manager class that handles a list of items. In this case, the items in the list represent contact points, skids, and tires. The landing gear model is a relatively simple one with gear struts that compress in the vertical direction only. The contact points and landing gear definition is located in the UNDERCARRIAGE section of the aircraft specification file. The definition for each gear or wingtip, etc. includes: static and dynamic friction coefficient, damping coefficient, rolling resistance, and steering and braking capabilities.

5. Output

JSBSim can output data to a file, directly to the console window, or through a socket interface. The console and file output is in comma-separated value (CSV) format. Related, predefined, sets of data can be logged by turning ON the associated set in the OUTPUT section of the aircraft specification file. For example:

```
<OUTPUT NAME="X15_data_log.csv" TYPE="CSV">
  RATE_IN_HZ       20
  SIMULATION       ON
  ATMOSPHERE       OFF
  MASSPROPS        OFF
  AEROSURFACES     ON
  RATES            ON
  VELOCITIES       ON
  FORCES           ON
  MOMENTS          ON
  POSITION         ON
  COEFFICIENTS     OFF
  GROUND_REACTIONS ON
  FCS              ON
  PROPULSION       ON
</OUTPUT>
```

Specific properties can also be logged if desired by including a PROPERTY definition in the OUTPUT section. For example:

```
  PROPERTY fcs/elevator-pos-deg
```

6. Trimming and Initialization

A trimming capability is present within JSBSim. A desired state can be specified using the initial conditions class (FGInitialCondition). The initial conditions class can either load the set of initial conditions from an initialization file, or the initial conditions can be set programmatically. Items that can be specified for initialization include velocities, map location, altitude, rate of climb, flight path angle, wind speed and direction, Euler angles, and angles of attack and sideslip.

Trimming is accomplished by finding the aircraft attitude and control settings needed to maintain the steady state described by the initial conditions passed to the FGTrim object. This is done by assigning a control to each state and iteratively adjusting that control until the state is within a specified tolerance of zero. States include the rectilinear body axis accelerations, the angular accelerations, and the difference between heading and ground track. Controls include the usual flight deck controls available to the pilot, with the addition of angle of attack, sideslip angle, flight path angle, pitch attitude, roll attitude, and altitude above ground. The latter three are used for on-ground trimming.

The state-control pairs used in a given trim are completely user-configurable and several pre-defined modes are provided as well. They are:

- Longitudinal: Trim body Z-axis velocity with alpha, X-axis velocity with thrust, and pitch rate with elevator.
- Full: Same as Longitudinal plus Y-axis velocity with roll angle, roll rate with aileron, yaw rate with rudder, and heading minus ground track with beta.
- Pullup: Same as Longitudinal, but adjust alpha to achieve specified load factor.
- Ground: Trim body Z-axis velocity with altitude, pitch rate with pitch angle, and roll rate with roll angle.

Sometimes the trimming routine cannot achieve the requested trim condition. In that case, a warning is issued.

IV. Using JSBSim

A. Getting and Building JSBSim

As an open source application, JSBSim source code is freely available from the project web site.[1] The web site is a resource for users, and serves as a configuration management tool for developers. Source code is managed using CVS (concurrent version system).

The source code can be (and has been) compiled into an executable across several platforms including Macintosh, PC/Windows, PC/Cygwin (a Unix-like environment that runs under Windows), Linux, and SGI IRIX. Under Windows, the Borland C++ and Microsoft Visual C++ compilers have also been used to build a JSBSim executable, or a Windows DLL for use in another program.

Additional open source tools are used in the development effort. Doxygen[12] is used to auto-generate application program interface (API) documentation for JSBSim. Doxygen parses the source code header files for formatted comments and creates developer documentation. This process is automatically performed periodically, and the resulting current documentation is available via the project web site.

B. Creating Flight Models

The existence of a data driven simulation application is of course meaningless without the data to drive it. The creation of a specification file comprising all necessary aspects of any particular aircraft is an adventure by itself, and deserves a more thorough discussion than it will get here. In this endeavor, one is reminded of the old axiom: "garbage in, garbage out", which is particularly relevant for freely available software due to the wide range of experience and expectations of the user base.

Fortunately, there are a fair number of useful resources available to aid in the creation of an aircraft flight model. Data for many aircraft have been harvested from:

- Aerospace engineering textbook examples and appendices
- The NACA, Dryden, and Langley technical report servers
- FAA aircraft Type Certificate Data Sheets (TCDS)[13]
- Aircraft flight manuals
- Aircraft manufacturers' web sites
- University aerospace engineering department web sites
- Longhand calculations

The NACA technical report server is especially useful for those interested in historical aircraft (X-1, etc.). The X-15 was the first aircraft that was modeled in JSBSim (see Listing 2) using the data presented in reports hosted on the NASA Dryden web site, as well as from a number of reports that were obtained using Freedom of Information Act (FOIA) requests. Flight test data is a preferred source of information for guidance in creating a flight model. However, it is inherently limiting in that only flight dynamics characteristics bounded by the flight envelope of the aircraft as tested and reported can be reproduced.

Additional tools are used in creating new models or refining existing models. Digital DATCOM is a tool that is publicly available that can calculate estimates for selected aerodynamic coefficients when supplied with information about aircraft geometry. The source code for Digital DATCOM is available, and one developer has modified Digital DATCOM to produce output directly in the format used by JSBSim.

Another solution considered for the tedious effort of creating a specification file for an aircraft would be the creation of a GUI application, perhaps with a "wizard", that guides the user through the process of creating a specification file. The resulting specification file could represent a close approximation to the chosen aircraft. The user would subsequently be expected to refine the generated specification as information that is more specific

became available. Towards this goal, a web application called Aeromatic has been created and is hosted at the main project web site. Aeromatic requires the user to select the closest similar aircraft to what the desired model will represent from a list of aircraft types ranging from a glider to a multi-engine transonic transport. Additional parameters such as wing area, span, and weight must be given. With these parameters, Aeromatic can then be commanded to generate a specification file. The resulting specification file can then be refined as needed.

C. Standalone and Integrated Use

Minimal effort is required to integrate JSBSim into a simulation application that provides a broader range of capabilities not earmarked as goals for JSBSim by itself (for instance, JSBSim provides no real-time out-the-window visuals). The capability to run in a standalone mode is also maintained for cases where visual modeling is not required. The standalone mode of operation is particularly useful for testing.

Integration of JSBSim within a simulator such as FlightGear is accomplished using an interface class. At the time of initialization, the interface class creates an instance of the JSBSim executive class, FGFDMExec. The executive is subsequently directed to load the chosen aircraft specification file:

```
fdmex = new FGFDMExec( … );
result = fdmex->LoadModel( … );
```

Subsequent to the creation of the executive and loading of the model, initialization is performed. Initialization involves copying control inputs into the appropriate JSBSim data storage locations, configuring it for the set of user-supplied initial conditions, and then copying state variables from JSBSim. The state variables are used to drive the instrument displays and to place the vehicle model in world space for visual rendering:

```
copy_to_JSBsim();   // copy control inputs to JSBSim
fdmex->RunIC();     // loop JSBSim once w/o integrating
copy_from_JSBsim(); // update the bus
```

Once initialization is complete, cyclic execution proceeds:

```
copy_to_JSBsim();   // copy control inputs to JSBSim
fdmex->Run();       // execute JSBSim
copy_from_JSBsim(); // update the bus
```

JSBSim can be used in a standalone mode by creating a compact stub program that effectively performs the same progression of steps as outlined above for the integrated version, but with two exceptions. First, the copy_to_JSBSim() and copy_from_JSBSim() functions are not used because the control inputs are handled directly by the scripting facilities and outputs are handled by the output (data logging) class. Second, the name of a script file can be supplied to the stub program. Scripting (described in the following section) provides a way to supply command inputs to the simulation:

```
FDMExec = new JSBSim::FGFDMExec();
Script = new JSBSim::FGScript( … );
Script->LoadScript( ScriptName ); // the script loads the aircraft and ICs
result = FDMExec->Run();

while (result) { // cyclic execution
  if (Scripted)
    if (!Script->RunScript()) break; // execute the script until it returns false
  result = FDMExec->Run(); // execute JSBSim
}
```

The standalone mode has been useful for verifying changes before committing updates to the source code repository. It is also useful for running sets of tests that reveal some aspects of simulated aircraft performance, such as range, time-to-climb, takeoff distance, etc.

D. Scripting

The value of scripting is realized any time a repeatable set of conditions needs to be performed for testing or analysis. The scripting feature of JSBSim is made up of language elements that describe conditional tests to be

performed and the actions to be taken when the conditional tests evaluate to true. A *runscript* also includes a reference to an aircraft specification file, to a file containing the set of initial conditions, to a start and stop time, and to the integration step size. The aircraft and initialization files are specified first:

```
<?xml version="1.0"?>
<runscript name="{description of script run}">

<use aircraft="{aircraft name}">
<use initialize="{initialization file name}">
```

The real work is done in the run section:

```
<run start="{start time}" end="{end time}" dt="{time step}">
  <when>
    {tests}
    {actions}
  </when>
  …
</run>
```

Within the run section are the statements that comprise one or more *directives*. A directive refers to the set of conditional tests to be evaluated and the resulting actions to be performed when the test or tests are all satisfied. The statements that comprise a directive are grouped within a `<when></when>` block.

A conditional test of a `parameter` is specified as follows:

```
<parameter name="{property}" comparison="{logical operator}" value="{value}"/>
```

The `name` element refers, once again, to a simulation property, such as "velocities/vc-kts" (calibrated airspeed in knots).

The `comparison` element refers to a logical operator that is one of the following:

- "le" or "<=" (less than or equal to)
- "lt" or "<" (less than)
- "ge" or ">=" (greater than or equal to)
- "gt" or ">" (greater than)
- "eq" or "==" (equal to)
- "ne" or "!=" (not equal to)

The `value` element refers to a number against which the comparison is made.

All `parameter` statements – each representing a conditional test – must be satisfied before the directive action or actions are carried out.

An action is specified on one line, but is broken up here for clarity:

```
<set
  name="{property}"
  value="{new value}"
  type="{type}"
  action="{change type}"
  persistent={true|false}
  tc="{tc}"
/>
```

The `type` element selects how the new value is to replace the old value for the chosen property:

`FG_DELTA` – the new value is added to the old one
`FG_VALUE` - the actual value is copied over the old one

The `action` specifies how the new value for the chosen property is arrived at:

`FG_EXP` – the new value is approached exponentially from the old value
`FG_RAMP` - the value ramps linearly from the old value to the new one
`FG_STEP` - the new value immediately takes effect

The `persistent` element is set true or false. This setting determines whether or not the directive would be reset once it becomes inactive – that is, when one or more of the conditional tests no longer evaluates to true. Set the `persistent` element to true if it is desired to reuse this directive indefinitely.

The `tc` element is a time constant used to specify at what rate the new setting should be faded in. The `tc` element is important for the ramp and exponential actions, but is meaningless for the step action.

An example of a `when` block looks like this:

```
<when>
  <parameter name="sim-time-sec" comparison="ge" value="0.25"/>
  <set name="fcs/aileron-cmd-norm" type="FG_VALUE"  value="0.20" action="FG_STEP"/>
</when>
```

In the above case, the aileron command will be set to a value of 0.20 when the simulation time passes 0.25 seconds.

E. Putting It All Together

It is useful to be able to run a series of quick tests in JSBSim to evaluate changes made to an aircraft model (that is, to changes made in the configuration files) or to source code. This task is made possible if the aircraft can be directed through a series of repeatable maneuvers. The addition of automatic control to an aircraft, coupled with the use of scripts that can direct the automatic control, provide the capability to perform repeatable tests. An example is presented here that shows how the FCS components can be used to model an autopilot capability and how a script can be used to direct a flight.*

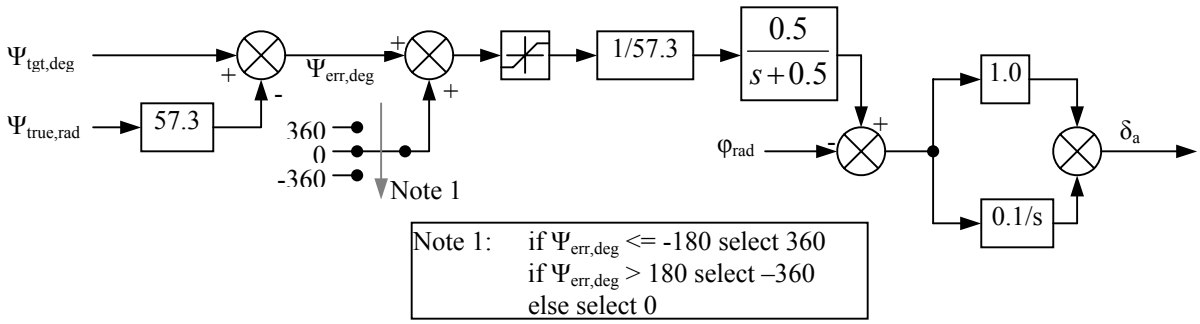

Figure 3. A Simple Heading Autopilot

A heading hold autopilot for a single engine general aviation aircraft has been constructed using the FCS components. The block diagram for this controller is shown in Fig. 3. The heading autopilot simply turns heading error into a bank command, and uses proportional plus integral control to arrive at an aileron command that achieves the desired effect. The specification for the heading autopilot in XML format is presented in Listing 5 in the Appendix. The script that directs the test flight is shown in Listing 6. The Ground Track plot is shown in Fig. 4. Note that an altitude-hold capability was used simultaneously with the heading-hold test.

F. Notable Uses of JSBSim

Besides being used by FlightGear, JSBSim has found uses in various other places. It has proven to be useful with students (one of the prime target audiences that JSBSim is tailored to), developers working on other simulators either for entertainment or research uses, and in industry for tasks such as a serving as a test driver for a Head Up Display (HUD) display. Since JSBSim is open source and freely available, it is difficult to ascertain the size of the user base.

* The example simply illustrates the use of FCS components and is not meant to represent an optimal autopilot.

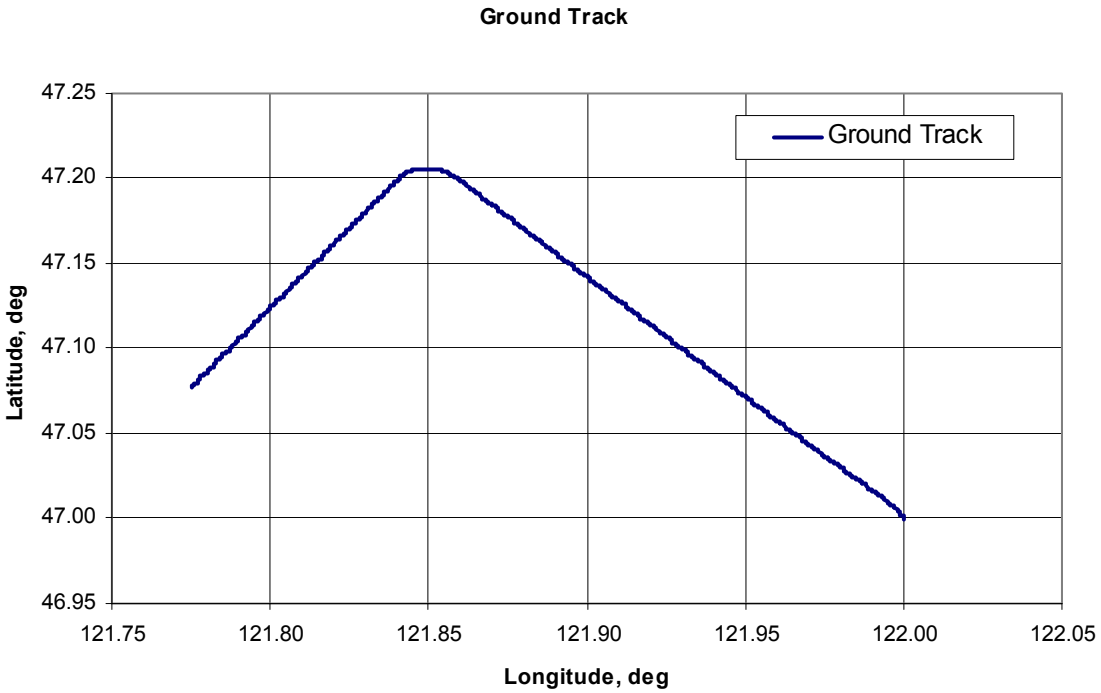

Figure 4. Ground Track resulting from the scripted test run

In one notable use, an augmented version of JSBSim was used to support the development of control laws for a sounding rocket. The changes made to JSBSim for that purpose were made available to the development team for incorporation back into the main source code line. The augmented version of JSBSim also served as a simulation tool supporting the development of an evolving neural network guidance system for a finless rocket.[14]

In another use, a detailed Space Shuttle approach and landing model was created as part of an educational/demonstration outreach effort. This project also shared some modifications made to support three-dimensional lookup tables. The actual vehicle definition as described in the aircraft specification file was not released, however, as it contained data that could not be disclosed.

V. Conclusion

JSBSim has – to this point – succeeded as an open source application in what could be called a *niche market*. Six years after being integrated with FlightGear (and over 30,000 lines of source code and comments later), development is continuing with a growing team of volunteer developers with relevant specialized skills. Users and developers both have created aircraft models and donated them to the project. Currently over 40 models of varying complexity and completion are managed within the JSBSim code repository, though some are still very much pre-release versions. Feature requests and bug reports are still being reported and fielded. Over a period of nearly four years, JSBSim project web site page-serving statistics indicate minimum interest in summer, and maximum interest in September and January (Fig. 5), which suggests that JSBSim is being found useful in an academic setting.

Yet, much remains to be done. Currently, refinements of the time-stepping algorithms are underway and actually exist in a development branch of the source code repository. This development will have an impact on the stability of the ground reactions model, which displays jitter that is particularly noticeable when at rest.

The landing gear model is also being reviewed for improvements. Techniques used in racing simulations are being analyzed for insight on improving the landing gear model in JSBSim.

Consideration has also been given to modeling multi-body configurations (e.g. multi-stage rockets, ordnance mounted on aircraft, etc.). However, this could result in a significant increase in complexity. One of the most difficult aspects of developing JSBSim is in balancing the benefits of new or expanded capabilities with the desire for simplicity and readability.

It is likely that a new XML parser will be written for JSBSim that is based on the EasyXML parser fielded in the SimGear library. Providing a more robust parser would allow for a better use of XML to be made, encourage better input checking, and ideally would allow JSBSim to migrate and adhere to the emerging DAVE-ML[7] standard. Improving the parsing capability of JSBSim would also support the capability to specify units for parameters specified in the aircraft, engine, and thruster specification files.

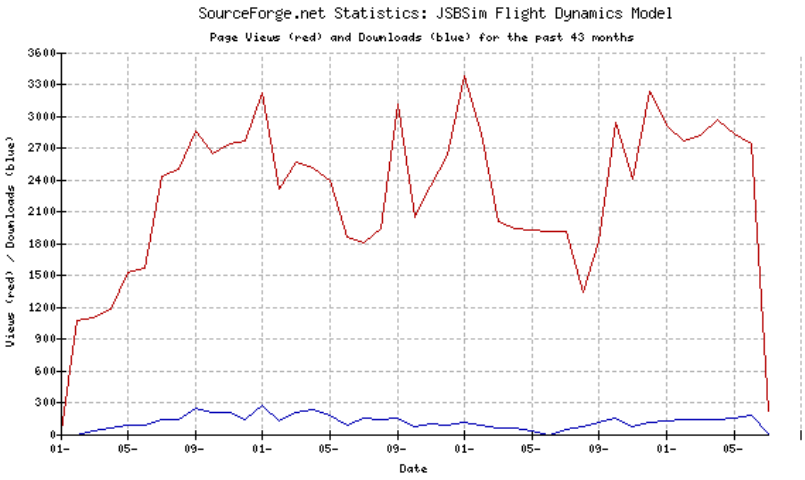

Figure 5. Page-Serving Statistics for JSBSim Project Web Site *(Statistics began in February 2001)*

In the best of all worlds, the ongoing use of JSBSim, with a close interaction and sharing between users and developers (in the spirit of open source development) will result in a continuing, synergistic, improvement.

Appendix

Listing 1. Specification file for a ball

```
<FDM_CONFIG NAME="BALL" VERSION="1.65" RELEASE="ALPHA">

  <METRICS>

    AC_WINGAREA 1.0
    AC_WINGSPAN 1.0
    AC_IXX      10.0
    AC_IYY      10.0
    AC_IZZ      10.0
    AC_IXZ      0.0
    AC_EMPTYWT  50.0
    AC_CGLOC    0.0 0.0 0.0

  </METRICS>

  <UNDERCARRIAGE>

   AC_GEAR  NOSE_CONTACT 0.0 0.0 0.0 10000 200000  0.0  0.0  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>
```

Listing 2. Specification file for the X-15

```
<FDM_CONFIG NAME="X-15" VERSION="1.65">
  <!--

  File:        X15.xml
  Author:      Jon S. Berndt
  CVS Version: $Id: X15.xml,v 1.44 2004/05/26 12:29:58 jberndt Exp $

  Function: Models an X-15
  -->

  <METRICS>

    AC_WINGAREA 200
    AC_WINGSPAN 22.36
    AC_CHORD    10.27
    AC_IXX      3650
    AC_IYY      80000
    AC_IZZ      82000
    AC_IXZ      590
    AC_EMPTYWT  15560
    AC_CGLOC    345.0  0.0    0.0
    AC_EYEPTLOC 115.9  0.0   26.0
    AC_AERORP   345.4  0.0    0.0
    AC_VRP      342.82 0.0    0.0

  </METRICS>

  <UNDERCARRIAGE>

    AC_GEAR     NOSE        63.0   0.0 -48.0 15000  1000  0.02 0.5 0.02 FIXED NONE 0 FIXED
    AC_GEAR     LEFT_SKID  540.6 -56.8 -76.6 15000  1000  0.2  0.9 0.20 FIXED NONE 0 FIXED
    AC_GEAR     RIGHT_SKID 540.6  56.8 -76.6 15000  1000  0.2  0.9 0.20 FIXED NONE 0 FIXED
```

```
</UNDERCARRIAGE>

<PROPULSION>

  <AC_TANK TYPE="OXIDIZER" NUMBER="0">
    XLOC 282.3
    YLOC 0.0
    ZLOC 0.0
    RADIUS 26.0
    CAPACITY 9470.0
    CONTENTS 9470.0
  </AC_TANK>

  <AC_TANK TYPE="FUEL" NUMBER="1">
    XLOC 408.3
    YLOC 0.0
    ZLOC 0.0
    RADIUS 26.0
    CAPACITY 8236.0
    CONTENTS 8236.0
  </AC_TANK>

  <AC_ENGINE FILE="XLR99">
    XLOC 600.0
    YLOC   0.0
    ZLOC   0.0
    PITCH  0.0
    YAW    0.0
    FEED     0
    FEED     1
    <AC_THRUSTER FILE="xlr99_nozzle">
      XLOC  600.0
      YLOC    0.0
      ZLOC    0.0
      PITCH   0.0
      YAW     0.0
    </AC_THRUSTER>
  </AC_ENGINE>
</PROPULSION>

<FLIGHT_CONTROL NAME="X-15 FCS with SAS">

  <COMPONENT NAME="Pitch Trim Sum" TYPE="SUMMER">
     INPUT        fcs/elevator-cmd-norm
     INPUT        fcs/pitch-trim-cmd-norm
     CLIPTO       -1 1
  </COMPONENT>

  <COMPONENT NAME="Pitch Command Scale" TYPE="AEROSURFACE_SCALE">
    INPUT        fcs/pitch-trim-sum
    MIN         -50
    MAX          50
  </COMPONENT>

  <COMPONENT NAME="Pitch Gain 1" TYPE="PURE_GAIN">
    INPUT        fcs/pitch-command-scale
    GAIN         -0.36
  </COMPONENT>

  <COMPONENT NAME="Pitch Scheduled Gain 1" TYPE="SCHEDULED_GAIN">
    INPUT        fcs/pitch-gain-1
    GAIN         0.017
    ROWS         22
    SCHEDULED_BY fcs/elevator-pos-rad
    -0.68  -26.548
    -0.595 -20.513
    -0.51  -15.328
    -0.425 -10.993
    -0.34   -7.508
    -0.255  -4.873
    -0.17   -3.088
    -0.085  -2.153
     0      -2.068
     0.085  -2.833
     0.102  -3.088
     0.119  -3.377
     0.136  -3.7
     0.153  -4.057
     0.17   -4.448
     0.187  -4.873
     0.272  -7.508
```

```
   0.357 -10.993
   0.442 -15.328
   0.527 -20.513
   0.612 -26.548
   0.697 -33.433
</COMPONENT>

<COMPONENT NAME="Roll Command Scale" TYPE="AEROSURFACE_SCALE">
  INPUT        fcs/aileron-cmd-norm
  MIN         -20
  MAX          20
</COMPONENT>

<COMPONENT NAME="Roll Gain 1" TYPE="PURE_GAIN">
  INPUT        fcs/roll-command-scale
  GAIN         0.42
</COMPONENT>

<COMPONENT NAME="Roll Gain 2" TYPE="PURE_GAIN">
  INPUT        fcs/roll-gain-1
  GAIN         0.027
</COMPONENT>

<COMPONENT NAME="Yaw Command Scale" TYPE="AEROSURFACE_SCALE">
  INPUT        fcs/rudder-cmd-norm
  MIN         -250
  MAX          250
</COMPONENT>

<COMPONENT NAME="Yaw Gain 1" TYPE="PURE_GAIN">
  INPUT        fcs/yaw-command-scale
  GAIN         0.082
</COMPONENT>

<COMPONENT NAME="Yaw Gain 2" TYPE="PURE_GAIN">
  INPUT        fcs/yaw-gain-1
  GAIN         0.040
</COMPONENT>

<COMPONENT NAME="Pitch SAS Feedback" TYPE="PURE_GAIN">
  INPUT        velocities/q-rad_sec
  GAIN         0.75
</COMPONENT>

<COMPONENT NAME="Yaw-Roll Crossover Gain" TYPE="PURE_GAIN">
  INPUT        velocities/r-rad_sec
  GAIN         -0.90
</COMPONENT>

<COMPONENT NAME="Yaw Coupled Aileron Feedback Sum" TYPE="SUMMER">
  INPUT        velocities/p-rad_sec
  INPUT        fcs/yaw-roll-crossover-gain
</COMPONENT>

<COMPONENT NAME="Roll SAS Gain" TYPE="PURE_GAIN">
  INPUT        fcs/yaw-coupled-aileron-feedback-sum
  GAIN         -0.50
</COMPONENT>

<COMPONENT NAME="Yaw SAS Gain" TYPE="PURE_GAIN">
  INPUT        velocities/r-rad_sec
  GAIN         0.30
</COMPONENT>

<COMPONENT NAME="Elevator Positioning" TYPE="SUMMER">
  INPUT         fcs/pitch-scheduled-gain-1
  INPUT        fcs/pitch-sas-feedback
  CLIPTO       -0.26 0.61
</COMPONENT>

<COMPONENT NAME="Elevator Filter" TYPE="LAG_FILTER">
  INPUT        fcs/elevator-positioning
  C1           600
  OUTPUT       fcs/elevator-pos-rad
</COMPONENT>

<COMPONENT NAME="Aileron Positioning" TYPE="SUMMER">
  INPUT         fcs/roll-gain-2
  INPUT        fcs/roll-sas-gain
  OUTPUT       fcs/left-aileron-pos-rad
  CLIPTO       -0.35 0.35
```

```
  </COMPONENT>

  <COMPONENT NAME="Rudder Positioning" TYPE="SUMMER">
    INPUT         fcs/yaw-gain-2
    INPUT        fcs/yaw-sas-gain
    OUTPUT       fcs/rudder-pos-rad
    CLIPTO       -0.52 0.52
  </COMPONENT>

</FLIGHT_CONTROL>

<AERODYNAMICS>

  <AXIS NAME="LIFT">

    <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>

    <COEFFICIENT NAME="CLDe" TYPE="VECTOR">
      Lift_due_to_Elevator_Deflection
      11
      velocities/mach-norm
      aero/qbar-psf | metrics/Sw-sqft | fcs/elevator-pos-rad
      0.0 1.00
      0.6 1.05
      1.0 1.15
      1.2 1.00
      1.6 0.66
      2.0 0.50
      2.4 0.40
      3.0 0.31
      5.6 0.21
      6.0 0.20
      9.0 0.20
    </COEFFICIENT>

    <COEFFICIENT NAME="CLM" TYPE="TABLE">
      Lift_due_to_Mach
      8 4
      velocities/mach-norm position/h-sl-ft
      aero/qbar-psf | metrics/Sw-sqft | velocities/mach-norm
           0.00  40000  60000  80000
      0.0  0.00   0.00   0.00   0.00
      0.6  0.00   0.00   0.00   0.00
      0.8  0.00   0.00   0.20   0.00
      1.0  0.01   0.04   0.60   0.00
      1.2  0.00   0.02   0.15   0.00
      1.4  0.00   0.00   0.00   0.00
      1.6  0.00   0.00   0.00   0.00
      9.0  0.00   0.00   0.00   0.00
    </COEFFICIENT>

  </AXIS>

  <AXIS NAME="DRAG">

    <COEFFICIENT NAME="CDmin" TYPE="VECTOR">
      Drag_minimum
      22
      velocities/mach-norm
      aero/qbar-psf | metrics/Sw-sqft
      0.000  0.061
      0.100  0.061
      0.500  0.061
      0.700  0.062
      0.800  0.065
      0.900  0.068
      0.990  0.090
      1.000  0.090
```

```
    1.010  0.090
    1.100  0.130
    1.200  0.120
    1.300  0.110
    1.400  0.100
    1.500  0.093
    2.000  0.080
    3.000  0.062
    4.000  0.048
    5.000  0.040
    6.000  0.038
    7.000  0.037
    8.000  0.037
    9.000  0.037
  </COEFFICIENT>

  <COEFFICIENT NAME="CDi" TYPE="VECTOR">
    Drag_induced
    10
    velocities/mach-norm
    aero/qbar-psf | metrics/Sw-sqft | aero/cl-squared-norm
    0.0 0.20
    0.5 0.20
    1.0 0.23
    1.5 0.40
    2.0 0.50
    3.0 0.80
    4.0 0.93
    5.0 1.05
    6.0 1.15
    9.0 1.33
  </COEFFICIENT>

</AXIS>

<AXIS NAME="SIDE">

  <COEFFICIENT NAME="CYb" TYPE="VALUE">
    Side_force_due_to_beta
    aero/qbar-psf | metrics/Sw-sqft | aero/beta-rad
    -1.4
  </COEFFICIENT>

  <COEFFICIENT NAME="CYda" TYPE="VALUE">
    Side_force_due_to_aileron
    aero/qbar-psf | metrics/Sw-sqft | fcs/left-aileron-pos-rad
    -0.05
  </COEFFICIENT>

  <COEFFICIENT NAME="CYdr" TYPE="VALUE">
    Side_force_due_to_rudder
    aero/qbar-psf | metrics/Sw-sqft | fcs/rudder-pos-rad
    0.45
  </COEFFICIENT>

</AXIS>

<AXIS NAME="ROLL">

  <COEFFICIENT NAME="Clb" TYPE="VALUE">
    Roll_moment_due_to_beta
    aero/qbar-psf | metrics/Sw-sqft | metrics/bw-ft | aero/beta-rad
    -0.01
  </COEFFICIENT>

  <COEFFICIENT NAME="Clp" TYPE="VALUE">
    Roll_moment_due_to_roll_rate_(roll_damping)
    aero/qbar-psf | metrics/Sw-sqft | metrics/bw-ft | aero/bi2vel | velocities/p-rad_sec
    -0.35
  </COEFFICIENT>

  <COEFFICIENT NAME="Clr" TYPE="VALUE">

    Roll_moment_due_to_yaw_rate
    aero/qbar-psf | metrics/Sw-sqft | metrics/bw-ft | aero/bi2vel | velocities/r-rad_sec
    0.04
  </COEFFICIENT>

  <COEFFICIENT NAME="Clda" TYPE="TABLE">
    Roll_moment_due_to_aileron
    8 2
```

```
    velocities/mach-norm position/h-sl-ft
    aero/qbar-psf | metrics/Sw-sqft | metrics/bw-ft | fcs/left-aileron-pos-rad
         0.0   80000
    0.0  0.05  0.05
    0.4  0.07  0.05
    0.6  0.08  0.05
    1.0  0.11  0.05
    1.4  0.08  0.06
    1.6  0.07  0.06
    2.4  0.06  0.05
    6.0  0.03  0.02
  </COEFFICIENT>

  <COEFFICIENT NAME="Cldr" TYPE="VALUE">
    Roll_moment_due_to_rudder
    aero/qbar-psf | metrics/Sw-sqft | metrics/bw-ft | fcs/rudder-pos-rad
    0.012
  </COEFFICIENT>

</AXIS>

<AXIS NAME="PITCH">

  <COEFFICIENT NAME="Cmalpha" TYPE="VALUE">
    Pitch_moment_due_to_alpha
    aero/qbar-psf | metrics/Sw-sqft | metrics/cbarw-ft | aero/alpha-rad
    -1.2
  </COEFFICIENT>

  <COEFFICIENT NAME="Cmq" TYPE="VALUE">
    Pitch_moment_due_to_pitch_rate
    aero/qbar-psf | metrics/Sw-sqft | metrics/cbarw-ft | aero/ci2vel | velocities/q-rad_sec
    -6.2
  </COEFFICIENT>

  <COEFFICIENT NAME="Cmadot" TYPE="VALUE">
    Pitch_moment_due_to_alpha_rate
    aero/qbar-psf | metrics/Sw-sqft | metrics/cbarw-ft | aero/ci2vel | aero/alphadot-rad_sec
    0.0
  </COEFFICIENT>

  <COEFFICIENT NAME="CmM" TYPE="TABLE">
    Pitch_moment_due_to_Mach
    10 4
    velocities/mach-norm position/h-sl-ft
    aero/qbar-psf | metrics/Sw-sqft | metrics/cbarw-ft | velocities/mach-norm
          0.00  40000  60000  80000
    0.00  0.00   0.00   0.00   0.00
    0.60  0.00  -0.01  -0.01   0.00
    0.80  0.00  -0.13  -0.13   0.00
    1.00  0.00  -0.11  -0.18   0.00
    1.10  0.00  -0.16  -0.18   0.00
    1.20  0.00  -0.25  -0.17   0.00
    1.40  0.00  -0.06  -0.15   0.00
    1.70  0.00  -0.03  -0.03   0.00
    2.00  0.00  -0.01  -0.01   0.00
    9.00  0.00   0.00   0.00   0.00
  </COEFFICIENT>

  <COEFFICIENT NAME="Cmde" TYPE="VECTOR">
    Pitch_moment_due_to_elevator_deflection
    7
    velocities/mach-norm
    aero/qbar-psf | metrics/Sw-sqft | metrics/cbarw-ft | fcs/elevator-pos-rad
    0.00 -1.50
    0.80 -1.60
    1.00 -1.75
    1.60 -1.30
    2.80 -0.60
    6.00 -0.25
    9.00 -0.20
  </COEFFICIENT>

</AXIS>

<AXIS NAME="YAW">

  <COEFFICIENT NAME="Cnb" TYPE="VALUE">
    Yaw_moment_due_to_beta
    aero/qbar-psf | metrics/Sw-sqft | metrics/bw-ft | aero/beta-rad
    0.5
```

```
      </COEFFICIENT>

      <COEFFICIENT NAME="Cnp" TYPE="VALUE">
        Yaw_moment_due_to_roll_rate
        aero/qbar-psf | metrics/Sw-sqft | metrics/bw-ft | aero/bi2vel | velocities/p-rad_sec
        0.0
      </COEFFICIENT>

      <COEFFICIENT NAME="Cnr" TYPE="VALUE">
        Yaw_moment_due_to_yaw_rate
        aero/qbar-psf | metrics/Sw-sqft | metrics/bw-ft | aero/bi2vel | velocities/r-rad_sec
        -1.5
      </COEFFICIENT>

       <COEFFICIENT NAME="Cnda" TYPE="VALUE">
         Yaw_moment_due_to_aileron
         aero/qbar-psf | metrics/Sw-sqft | metrics/bw-ft | fcs/left-aileron-pos-rad
         0.04
       </COEFFICIENT>

       <COEFFICIENT NAME="Cndr" TYPE="VALUE">
         Yaw_moment_due_to_rudder
         aero/qbar-psf | metrics/Sw-sqft | metrics/bw-ft | fcs/rudder-pos-rad
         -0.3
       </COEFFICIENT>

     </AXIS>

  </AERODYNAMICS>

  <OUTPUT NAME="X15_data_log.csv" TYPE="CSV">
    RATE_IN_HZ       20
    SIMULATION       ON
    ATMOSPHERE       OFF
    MASSPROPS        OFF
    AEROSURFACES     ON
    RATES            ON
    VELOCITIES       ON
    FORCES           ON
    MOMENTS          ON
    POSITION         ON
    COEFFICIENTS     OFF
    GROUND_REACTIONS ON
    FCS              ON
    PROPULSION       ON
  </OUTPUT>

</FDM_CONFIG>
```

Listing 3. Specification for the XLR-99 Rocket Engine

```
<?xml version="1.0"?>
<FG_ROCKET NAME="XLR-99">
  <!--
    SHR      = Specific Heat Ratio, for LOX/Ammonia this is 1.23
    MAX_PC   = Maximum chamber pressure in psf
    PROP_EFF = propulsive efficiency. This is an estimate.
  -->
  SHR             1.23
  MAX_PC       86556.0
  PROP_EFF         0.67
  MAXTHROTTLE      1.0
  MINTHROTTLE      0.4
  SLFUELFLOWMAX   91.5
  SLOXIFLOWMAX   105.2
</FG_ROCKET>
```

Listing 4. Specification for the XLR-99 Rocket Engine Nozzle

```
<?xml version="1.0"?>
<FG_NOZZLE NAME="X-15 XLR-99 Nozzle">
  <!--
    PE      = Nozzle exit pressure, psf.
    EXPR    = Nozzle expansion ratio, Ae/At, sqft.
```

```
    NZL_EFF = Nozzle efficiency, 0.0 - 1.0
    DIAM    = Nozzle diameter, ft.
  -->
  PE     40
  EXPR    9.8
  NZL_EFF 0.67
  DIAM    3.2
</FG_NOZZLE>
```

Listing 5. Specification for the C-172 Heading Hold Autopilot (see Fig. 3)

```
<COMPONENT NAME="Heading True Degrees" TYPE="PURE_GAIN">
  INPUT         attitude/heading-true-rad
  GAIN          57.3 <!-- convert to degrees -->
</COMPONENT>

<COMPONENT NAME="Heading Error" TYPE="SUMMER">
  INPUT         -fcs/heading-true-degrees
  INPUT          ap/heading_setpoint
</COMPONENT>

<COMPONENT NAME="Heading Error Bias Switch" TYPE="SWITCH">
  <TEST LOGIC="DEFAULT" VALUE="0.0">
  </TEST>
  <TEST LOGIC="AND" VALUE="360.0">
    fcs/heading-error < -180
  </TEST>
  <TEST LOGIC="AND" VALUE="-360.0">
    fcs/heading-error > 180
  </TEST>
</COMPONENT>

<COMPONENT NAME="Heading Corrected" TYPE="SUMMER">
  INPUT          fcs/heading-error-bias-switch
  INPUT          fcs/heading-error
  CLIPTO         -30 30
</COMPONENT>

<COMPONENT NAME="Heading Command" TYPE="PURE_GAIN">
  INPUT          fcs/heading-corrected
  GAIN           0.01745
</COMPONENT>

<COMPONENT NAME="Heading Roll Error Lag" TYPE="LAG_FILTER">
  INPUT          fcs/heading-command
  C1             0.50
</COMPONENT>

<COMPONENT NAME="Heading Roll Error" TYPE="SUMMER">
  INPUT          fcs/heading-roll-error-lag
  INPUT         -attitude/phi-rad
</COMPONENT>

<COMPONENT NAME="Heading Roll Error Switch" TYPE="SWITCH">
  <TEST LOGIC="DEFAULT" VALUE="0.0">
  </TEST>
  <TEST LOGIC="AND" VALUE="fcs/heading-roll-error">
    ap/heading_hold == 1
  </TEST>
</COMPONENT>

<COMPONENT NAME="Heading Proportional" TYPE="PURE_GAIN">
  INPUT          fcs/heading-roll-error-switch
  GAIN           1.0
</COMPONENT>

<COMPONENT NAME="Heading Integral" TYPE="INTEGRATOR">
  INPUT          fcs/heading-roll-error-switch
  C1             0.10
</COMPONENT>

<COMPONENT NAME="Heading Error Summer" TYPE="SUMMER">
```

```
  INPUT          fcs/heading-integral
  INPUT          fcs/heading-proportional
  CLIPTO         -1.0 1.0
</COMPONENT>
```

Listing 6. Specification for the Heading Hold Autopilot Test Script

```
<?xml version="1.0"?>
<runscript name="C172-01A test run">
  <!--
    This run is for testing the C172 heading and altitude hold autopilot
  -->

  <use aircraft="c172x">
  <use initialize="reset00">
  <run start="0.0" end="700" dt="0.0083333">
    <when>
      <parameter name="sim-time-sec" comparison="ge" value="0.25"/>
      <parameter name="sim-time-sec" comparison="le" value="1.00"/>
      <set name="fcs/throttle-cmd-norm" type="FG_VALUE" value="1.0" action="FG_RAMP"
                                                        persistent="false" tc ="0.05"/>
      <set name="fcs/mixture-cmd-norm" type="FG_VALUE" value="0.65" action="FG_RAMP"
                                                        persistent="false" tc ="0.05"/>
      <set name="propulsion/magneto_cmd" type="FG_VALUE" value="3" action="FG_STEP"
                                                        persistent="false" tc ="1.00"/>
      <set name="propulsion/starter_cmd" type="FG_VALUE" value="1" action="FG_STEP"
                                                        persistent="false" tc ="1.00"/>
    </when>

    <when> <!-- Set Aileron AP on -->
      <parameter name="position/h-agl-ft" comparison="ge" value="6.0"/>
      <set name="ap/attitude_hold" type="FG_VALUE" value="1" action="FG_STEP"
                                                        persistent="false" tc="1.0"/>
    </when>

    <when> <!-- Set Autopilot for 800 ft -->
      <parameter name="aero/qbar-psf" comparison="ge" value="15.0"/>
      <set name="ap/altitude_setpoint" type="FG_VALUE" value="800.0" action="FG_EXP"
                                                        persistent="false" tc ="3.0"/>
      <set name="ap/altitude_hold" type="FG_VALUE" value="1" action="FG_STEP"
                                                        persistent="false" tc="1.0"/>
    </when>

    <when> <!-- Set Autopilot for 850 ft -->
      <parameter name="sim-time-sec" comparison="ge" value="260.0"/>
      <set name="ap/altitude_setpoint" type="FG_VALUE" value="850.0" action="FG_EXP"
                                                        persistent="false" tc ="3.0"/>
      <set name="ap/altitude_hold" type="FG_VALUE" value="1" action="FG_STEP"
                                                        persistent="false" tc="1.0"/>
    </when>

    <when> <!-- Set Autopilot for 600 ft -->
      <parameter name="sim-time-sec" comparison="ge" value="320.0"/>
      <set name="ap/altitude_setpoint" type="FG_VALUE" value="600.0" action="FG_EXP"
                                                        persistent="false" tc ="3.0"/>
      <set name="ap/altitude_hold" type="FG_VALUE" value="1" action="FG_STEP"
                                                        persistent="false" tc="1.0"/>
    </when>

    <when> <!-- Set Autopilot for 2000 ft -->
      <parameter name="sim-time-sec" comparison="ge" value="430.0"/>
      <set name="ap/altitude_setpoint" type="FG_VALUE" value="2000.0" action="FG_EXP"
                                                        persistent="false" tc ="3.0"/>
      <set name="ap/altitude_hold" type="FG_VALUE" value="1" action="FG_STEP"
                                                        persistent="false" tc="1.0"/>
    </when>

  </run>

</runscript>
```

Acknowledgments

The author would like to acknowledge the efforts and contributions of the other JSBSim development team members: Tony Peden, David Culp, Mathias Fröhlich, David Megginson, and Erik Hofman. Thanks also to Bill Galbraith for DATCOM+ modifications, and Dave Luff for the development of the piston engine model.

References

[1]JSBSim Project web site: www.jsbsim.org.
[2]FlightGear Project web site: www.flightgear.org.
[3]E. Bruce Jackson, *Manual for a Workstation-Based Generic Flight Simulation Program (LaRCsim)*, Version 1.4, NASA TM-110164, May 1995.
[4]Martin, Robert C., *Designing Object-Oriented C++ Applications Using the Booch Method*, Prentice-Hall, Inc., Eaglewood Cliffs, New Jersey, 1995, p. 9.
[5]P. Sean Kenney, *Rapid Prototyping of an Aircraft Model in an Object-Oriented Simulation*, Modeling & Simulation Technologies Conference, Paper Number AIAA-2003-5816, August 2003.
[6]M. Madden, *Examining Reuse In LaSRS++ Based Projects*, Modeling & Simulation Technologies Conference, Paper Number AIAA-2001-4119, August, 2001.
[7]E. Bruce Jackson, Bruce L. Hildreth, *Flight Dynamic Model Exchange Using XML*, Modeling & Simulation Technologies Conference, Paper Number AIAA-2002-4482, August, 2002.
[8]J.M. Picone, A.E. Hedin, D.P. Drob, and A.C. Aikin, "NRL-MSISE-00 Empirical Model of the Atmosphere: Statistical Comparisons and Scientific Issues," J. Geophys. Res., doi:10.1029/2002JA009430, in press (2003).
[9]NRLMSISE-00 Empirical Atmosphere Model source code,
http://nssdc.gsfc.nasa.gov/space/model/atmos/nrlmsise00.html.
[10]C. G. Justus, *A Mars Global Reference Atmosphere Model (Mars-GRAM) for Mission Planning and Analysis*, 28th Aerospace Sciences Meeting, AIAA Paper Number 90-0004.
[11]W. A. Ragsdale, "A Generic Landing Gear Dynamics Model for LaSRS++", Paper Number AIAA-2000-4303, 2000.
[12]Doxygen, Open Source Software, Version v1.3.7, http://www.stack.nl/~dimitri/doxygen/, 7 July 2004
[13]Type Certificate Data Sheet Database, Federal Aviation Administration, http://www.airweb.faa.gov/tcds.
[14]Faustino J. Gomez, Risto Miikkulainen, "Active Guidance for a Finless Rocket Using Neuroevolution," *Proceedings of the Genetic and Evolutionary Computation Conference*, 2003, pp. 2084-2095 (Best Paper Award winner).