The quarterly newsletter for JSBSim, an open source flight dynamics model in C++

VOLUME 2, ISSUE 4
15 JANUARY 2006

Back of the Envelope

JSBSim, XSL Transformations, and XML Schemas

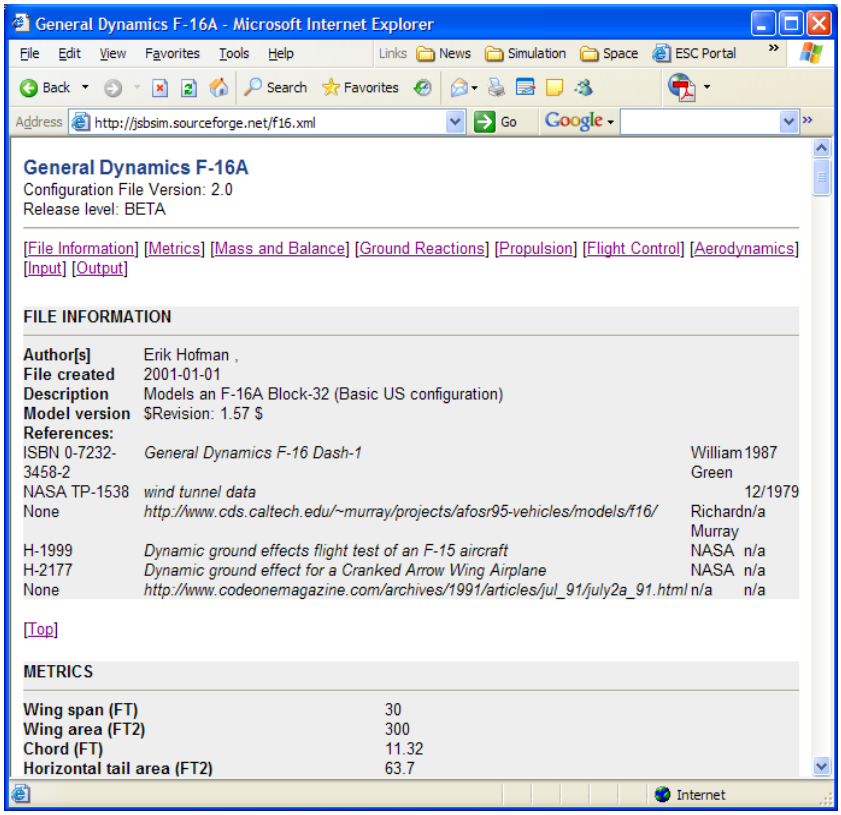
Big News. See Page 3.

Inside this issue:

The use of XML to specify configuration files for applications makes sense in many ways. It also opens up a wider variety of technologies and applications, ready made for use. This article will discuss some of those.

XSL Transformation

The eXtensible Stylesheet Language is an XML-based language used to write stylesheets used for transforming XML documents from one form into another. One can write a stylesheet that allows for processing each element in the target XML document and outputting HTML code that displays the element in a more friendly, useful, and appealing way.



This line causes the browser to read the stylesheet and process the target file (f16.xml), resulting in the display you (hopefully) see in your browser (see image at left). The resulting displayed file is much more readable than the original XML file – which is of course not meant to be easily human readable, but application parsable. Most browsers support XSL transformations.

Over the past couple of months a preliminary JSBSim XSL transformation has been written. It is called "JSBSim.xsl", and it is available at the JSBSim web site at:

http://jsbsim.sourceforge.net/JSBSim.xsl

The official, current, file will always be kept there (as well as in JSBSim cvs).

It may not be clear what can be offered by the use of an XSL transformation file. An example would be useful. In your web browser, try opening this file:

http://jsbsim.sourceforge.net/f16.xml

What you download is the straight, JSBSim F16 configuration file. At the top of that file is this stylesheet reference line:

```
<?xml-stylesheet type="text/xsl"
href="http://jsbsim.sourceforge.net/
JSBSim.xsl"?>
```

A future version of the transformation might apply to any data tables used in the configuration file. It is hoped that by employing SVG (Scalable Vector Graphics), the data tables might be displayed using

(Continued on page 2)

(Continued from page 1)
graphs instead of simply presenting the data tables by themselves.

XML Schema

Another capability that can be taken advantage of is in validation of XML files using some kind of formal file definition. The allowable format for an XML file (for example an aircraft or engine configuration file) can be defined in a Document Type Definition (DTD), or an XML Schema. A Schema defines what kinds of elements can be expected in a file, and the data type of the elements. A Schema can even set limits on the numeric values expected in an XML file. For example, an XML Schema definition of the Wing Area element in a JSBSim config file is presented in the listing, below.

```
<!-- Wing Area -->

<xs:element name="wingarea">
 <xs:annotation>
  <xs:documentation>
    Wing area (reference area) for the vehicle
    being modeled. The default unit is square feet (FT2).
  </xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:simpleContent>
   <xs:extension base="PositiveNumber">
    <xs:attribute name="unit" use="optional" default="FT2" type="AreaType"/>
   </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

This definition states that there must be an element *wingarea* in a file, that it can take the value of any positive number, and that the element definition may optionally include an attribute that defines the units that the value is expressed in. There is a default unit of FT2 (square feet).

Currently, the XML parser used by JSBSim (easyXML, a part of the SimGear library, based on eXpat) does not support on-the-fly validation of input files. However, there are several web sites that provide a tool to validate files. Two online validators are:

http://apps.gotdotnet.com/xmltools/xsdvalidator/
http://tools.decisionsoft.com/schemaValidate.html

It may be possible to adapt easyXML to do validation at the time the XML files is parsed. Whether this is desirable, or efficient, is questionable.

Another luxury afforded by the use of some XML editors (such as OxygenXML, the tool this author used to create the JSBSim XSL transformation stylesheet as well as the XML Schema) is the automatic generation of documentation for the XML file. As seen in the above example, there are "annotation" and "documentation" elements that can be embedded in the Schema. OxygenXML uses these two generate documentation for the JSBSim XML configuration file, as defined by the XML Schema file. The documentation will be online in the near future, and Schema and documentation files will also be added for the various other JSBSim files (engines, propellers, etc.).

Using a schema to check a config file prior to using it will provide a good level of error checking and reduce frustration at runtime.

For more information on XML technologies, visit the World Wide Web Consortium (W3C) web site at:

http://www.w3.org/

There, you will find the current standard and draft standards for many technologies including SVG, XML Schema, and XSL Transformations.

Another wonderful web site for being instructed in XML technologies is the W3 Schools web site here:

http://www.w3schools.com/

Finally, there are several high quality XML editors. Two of them are:

http://www.oxygenxml.com
http://www.stylusstudio.com/

About this newsletter ...

Edited by Jon Berndt

"*Back of the Envelope*" is a communication tool written generally for a wider audience than core JSBSim developers, including instructors, students, and other users. The articles featured will likely tend to address questions and comments raised in the mailing lists and via email. If you would like to suggest (or even author) an article for a future issue, please email the editor at:

Jon@jsbsim.org

News Items

Landing Gear Model Changes Eliminate Jitter

The long-suffered problem of unstable ground reactions while at low or zero velocity appears to have been solved. The problem had been particularly apparent when using JSBSim as the selected FDM (flight dynamics model) in FlightGear.

Several steps were taken to fix the problem:

- The code was reviewed and reorganized to be more concise and easily readable.
- The use of winds and turbulence while at rest on the runway has been eliminated. Winds and turbulence are now slowly ramped in as aircraft velocity increases. This is fine, as it is assumed that the landing gear ground forces will match wind and turbulence forces, anyhow, resulting in no movement.
- Centrifugal acceleration due to the rotation of Earth is not incorporated into the total aircraft forces and moments calculations until the aircraft leaves the runway. Again, the end effect is that the landing gear seems to counter that small acceleration.
- At low and zero velocity, the slip angle of the tire is assumed to be the steering angle of the wheel.
- A precise filtering of steering angle is applied in order to reduce noise due to small motions. This filter is tuned for the operation of the flight model at 120 Hz – the rate that FlightGear calls JSBSim. Any other FDM frame rate may result in noise and jitter. This is being investigated in the hope of making the filter operate for all FDM frame rates.
- Side and rolling forces produced by the landing gear are ramped out at very low velocities and as zero is approached.
- Side and rolling forces are filtered to reduce noise.

This combination of carefully selected techniques has been tested in operation within FlightGear and has shown that several aircraft ranging in size from the C-172 to the B747 now show rock-solid operation on the runway at any velocity.

"New" JSBSim moving to FlightGear

As I write this the new JSBSim code – over a year in the making – is on its way into FlightGear CVS. This is a critical move prior to the release of FlightGear v1.0, as well as for the subsequent release of JSBSim v1.0. Despite a year's worth of testing in the standalone mode, the inclusion of "new" v0.9.10 JSBSim into FlightGear will surely not be flawless. Many things have changed, and there will be growing pains. Documentation outlining the new v2.0 JSBSim config file format is not quite ready.

But, it will be worth the struggle. JSBSim will be incorporating many new technologies in this release, afforded by the improved configuration file format. Among the new features (some have been mentioned in previous newsletters):

- A telnet control interface has been added,
- Functions can be specified in the flight control and aerodynamics sections,
- A sensor class has been added, complete with malfunctions,
- Propeller operation has been improved

Off-Standard Atmosphere Temperature Support Added

JSBSim can use either an internal atmosphere model or an externally supplied one, whichever the user decides. The internal model is based on the U.S. Standard Atmosphere of 1959. As with any standard atmosphere model the atmosphere depicted does not reflect the wide variations in temperature, pressure and density that one would encounter in the real world. Of particular concern is the lack of off-standard temperature, since aircraft and aircraft engines can be greatly effected by temperature. A good simulation environment will allow you the flexibility to set up a run at Mexico City at 100 degrees Fahrenheit, for example, since many aircraft would be unable to take off under that kind of real-world condition.

JSBSim provides users with the ability to set off-standard temperatures in two ways. The simplest way is to set a delta-T value in the property "atmosphere/delta-T", or by calling the method FGAtmosphere::SetDeltaT(). Remember that the Rankine scale is used by JSBSim internals, so delta_T should be in Rankine or Fahrenheit. When set this way the delta_T will be applied at all times during the simulation run. This is a good way to

(Continued on page 4)

(News, Continued from page 3)
lock in a desired delta_T for use in a constant altitude run, or for a run with a moderate range of altitudes. The fixed delta_T would not provide a realistic model for the whole atmosphere.

To get a more realistic model of off-standard temperature deviations there is a second method of setting up the off-standard temperature. This can be done by setting the property "atmosphere/T-sl-dev-F", or by a call of FGAtmosphere::SetSLTempDev(). This will apply the deviation to an altitude of zero (sea level) and then decrease it linearly until the standard tropopause altitude (about 36089 feet in the 1959 Std. Atmosphere). This method is best used for a simulation run which needs a more realistic off-standard temperature profile for the whole atmosphere.

David Culp's new F-80C model in-flight.

If you wanted to set up a simulator run to test, say, takeoff performance at Mexico City at 100 degrees Fahrenheit, then the first method shown above would be the best to use. If you wanted to look at takeoff and climb to cruise altitude, then the second method would be best, but you'll have to do some prior calculation first. Since the second method applies the supplied temperature deviation to sea level, you'll have to figure out what sea level temperature would result in a temperature of 100 degrees Fahrenheit at the altitude of the Mexico City airport. The standard day temperature at MMMX (elevation 7341 feet) is 492.5 degrees Rankine. You want the temperature to be 560 degrees Rankine (i.e. 100F) which is a local deviation of +67.5 . You need to set the sea level deviation to:

(67.5 / (36089 - 7341)) * 36089 = 84.7

Thus setting T-sl-dev-F to 84.7 will result in a temperature of 100F at 7341 feet, and provide you with a linearly decreasing temperature deviation until reaching standard tropopause height, above which standard temperatures will apply.

If you are using turbine engines, please ensure that the thrust look-up table in the engine configuration file is indexed using the property "atmosphere/density-altitude", rather than "position/h-sl-ft". Also ensure that you have a column for negative altitudes if you wish to model thrust gains due to lower-than-standard temperatures.
Turbine configuration files generated by AeroMatic, JSBSim's web based configuration tool, include these items.

Lockheed F-80C "Shooting Star" Aircraft Model Fielded

A recent addition to the JSBSim stable of aircraft is the Lockheed F80-C Shooting Star, the first production jet fighter in the US. It first flew in 1944 and was built around the turbine engine designed by Sir Frank Whittle. There were many obstacles to overcome in getting the airplane into operation with the Army Air Corps, including needed fixes for: engine ingestion of the forward fuselage boundary layer, unreliable engine components, unreliable fuel supply chain, and a lack of adequate pilot and maintenance training for jet operations.

By the time these fixes were in the Shooting Star was already obsolete, as German advances in high-speed aerodynamics became known outside of Germany after the end of the war. The F-80's replacement, the North American F-86, incorporated the new aerodynamics and first flew in 1947. Unbeknownst to the West, the Soviet Union had also acquired the new aerodynamics technology, and had used it in the design of the MiG-15, which also made its first flight in 1947. The problem with the F-80's design was high transonic drag, which limited it's speed to about 500 knots. This prevented it from being used as an air combat fighter, however it served well as a fighter-bomber, and was spectacularly successful as a jet trainer in its two-seat form, the T-33.

Flying the simulated F-80C will demonstrate some of the quirks of the real aircraft. It was underpowered, by modern standards, so acceleration and climb performance were merely adequate, however it was a very clean airplane and hard to slow down once it got going. The engine gulped fuel, limiting the F-80's range. The engine was also slow to accelerate because of the large inertia in its centrifugal compressor.

Turboprop Engine Model

Jiri Javurek

Concepts and Design

A turboprop (abbreviation of turbine and propeller) engine is a turbine engine that powers a propeller. It is a "subclass" of the turboshaft engine – a turbine engine that is powered by means of shaft, for example a helicopter rotor or an electric generator. A turboshaft or turboprop engine has at least one of the turbine spools connected to an output shaft. The turbine stage that powers the output shaft is called a free or power turbine. Because the turbine speed is much greater than the propeller speed, there is a speed reducer (gear box) included between the turbine and the propeller.

One of the basic characteristics of a turboprop engine is the number of shafts and the number of compressor and turbine stages. The compressor is powered by one or more turbine stages. This consumes a large portion of the turbine power. The remaining energy is used for the free turbine and powers the propeller. The compressor and the connected turbine are called a gas generator. The rest of the kinetic energy of the gas usually remains on the exhaust outlet. This energy can be used for additional thrust, but the power is usually not significant in comparison with the propeller thrust.

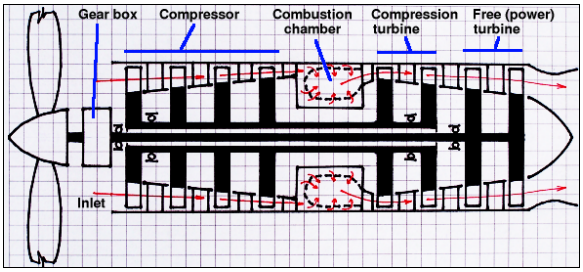


Fig. 1 Two shaft turboprop engine.

Figure 1 shows a "classic" turboprop concept with two concentric shafts. The outer shaft connects the generator (compression) turbine with an axial compressor; the inner shaft transmits the motion from the two stages free turbine to the speed reducer (gear box). Between the compressor and the turbine is – like in an ordinary turbine engine, a combustion chamber. Because this concept has some disadvantages, many other configurations are used. Another concept with two shafts is introduced in figure 2. The compressor has two axial stages and one centrifugal stage and is powered by one turbine stage. A one stage free turbine powers the gearbox.

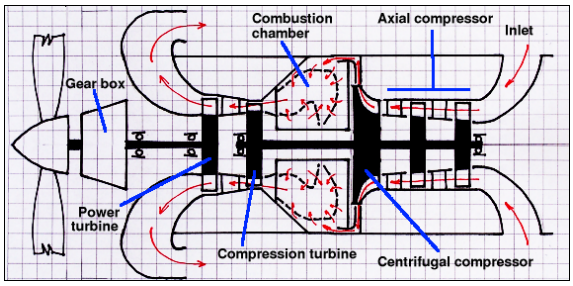


Fig. 2 Two shaft turboprop engine.

Interesting in this concept is the fact that the airflow in the engine is opposite to the flight direction. The air inlet is at the back and the exhaust outlet is in the front. This concept is used for the Walter M601 engine and also (with a three stage axial compressor) for the Pratt & Whitney PT6. In figure 3 we see an example of a three-shaft engine. There are three concentric shafts used. The outer shaft is powered by the first turbine stage and is connected to the second centrifugal compressor. The middle shaft connects the second turbine stage with the first centrifugal compressor. This concept is used for the PW100 series of engines from Pratt & Whitney.

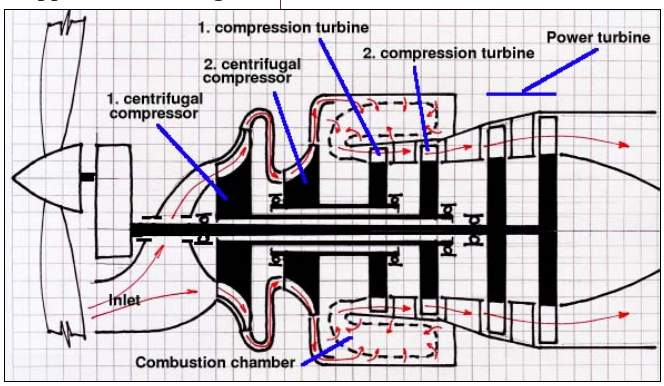


Fig. 3 Three shaft turboprop engine.

Turboprop engines are usually used with constant speed propellers. These propellers have variable pitch controlled by a regulator so, that the speed of the propeller (and thereby the speed of the free turbine) is constant for ordinary modes. The variable pitch also enables reversing for braking after landing and feathering. Feathering means setting the propeller pitch so that the drag in the direction of flight is minimal. It is used when the engine doesn't operate during the flight (in the case of malfunction, etc.). It is

(Continued on page 6)

(Continued from page 5)
also possible to use small pitch angles for slow taxiing. In diagram 5 is the table for a variable pitch propeller from the propeller configuration file in graphical form.

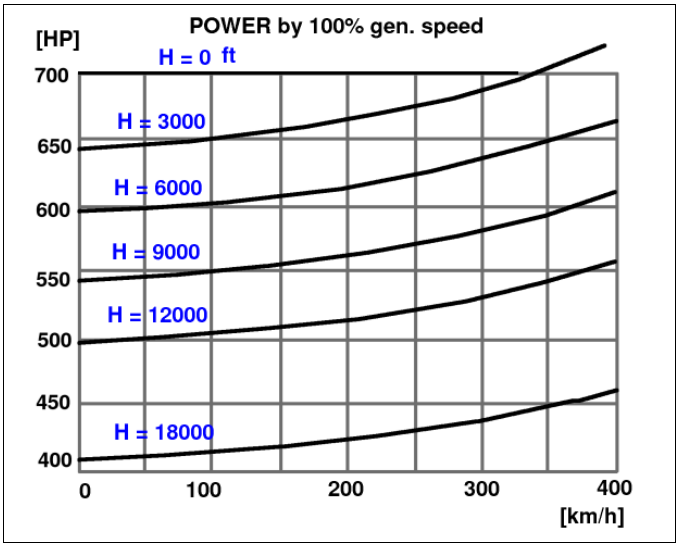


Diagram 2 Power dependency.

Controlling the Turboprop Engine

In the cockpit are two levers. A throttle lever and a propeller control lever. A Turboprop engine uses two regulators. The first is called the fuel regulator. It sets the needed fuel quantity, so the generator speed corresponds to the throttle lever position. The second regulator sets the propeller pitch, so the propeller RPM reaches the value set by the propeller lever. The throttle lever is also used for propeller reversing and for small angle setting (called *beta range*). In this range the fuel regulator is joined to the propeller regulator, and this lever sets the propeller pitch. Moving the lever in the opposite direction from full throttle sets Beta range and reverse.

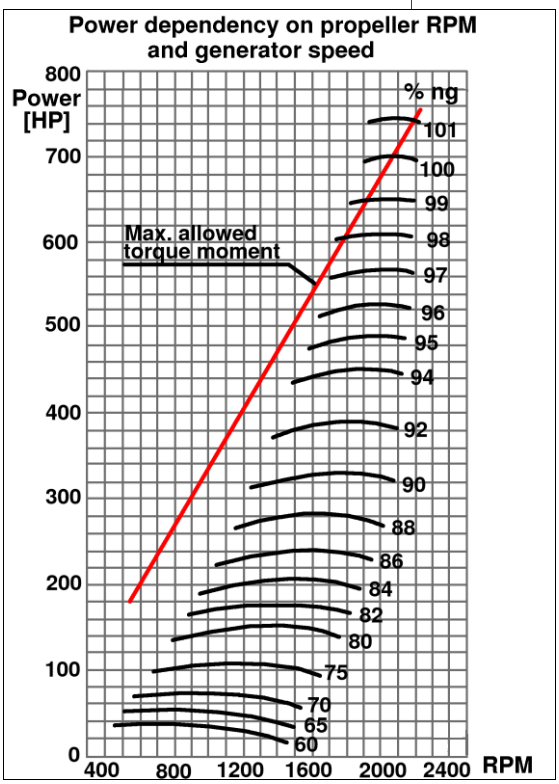


Diagram 1 Power dependency.

The propeller lever is used for feathering and also for decreasing RPM.

When we decided to make the L410 Turbolet for FlightGear our problem was that although JSBSim offered very good aerodynamic modeling, there were only piston and turbine engine models available. The connection of a propeller to the turbine engine didn't work properly (and was not designed for that purpose). Therefore, we decided to rewrite the turbine engine for use with a propeller and include the simulation for small two-shaft engines. It should accurately model engines with power up to 1500 HP.

At first we collected information about the engine regulation and controlling. We have got some parameters and diagrams that describe the engine behavior of the Walter M601 engine. The first diagram (diagram 1) shows how the power depends (for the constant air pressure or flight level) on the generator speed (i.e. throttle lever position), and on the shaft RPM. These dependencies are shown in diagram 1. The next diagram (diagram 2) shows the dependency of the power on flight level and flight speed for constant generator speed. This diagram exists for every important regime (take-off, maximum cruise, and so on). The values from diagram 1 and diagram 2 were used for tables ENG_POWER_COEF_AIRPRESSURE_AIRSPEED and ENG_POWER_RPM_N1. The values from these tables are shown in diagrams 3 and 4.

The binding between the throttle lever and the generator turbine speed is at this moment done only by a simple exponential function. This is exact enough for slow changes (as specified in the pilots guide), but does not simulate the effects of fast throttle changes.

The input parameters are:

- Position of throttle lever - number (The interpretation depends on the Reversed flag. For reversed propeller this is the maximum power reverse and although the engine runs in the same direction, the throttle lever is set to the opposite side),
- Reversed - boolean

- Cutoff - boolean
- Start - boolean
- Shaft RPM (i.e. propeller RPM)

The outputs are:

- Power [HP]
- Oil temperature
- ITT (Inter-turbine temperature)
- Fuel flow
- N1 (generator speed) [%]
- Starting - boolean
- IELU intervent (Integrated Electronic Limiters Unit) - boolean

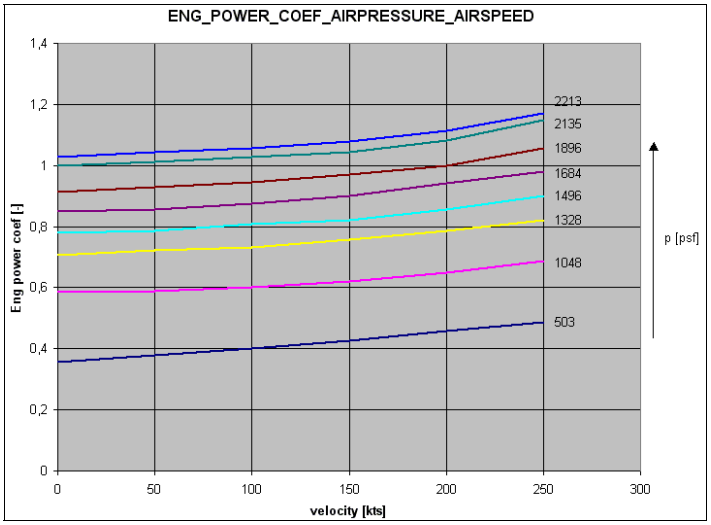


Diagram 3

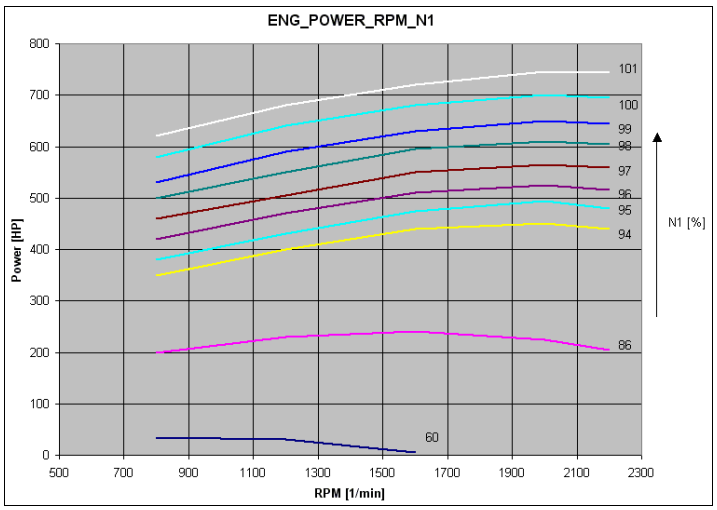


Diagram 4

Note:

The IELU unit evaluates important parameters (generator speed, propeller RPM, ITT and the torque on the propeller shaft. If one or more parameters run out of the limits, the fuel flow is reduced to avoid engine damage (only torque limiting is modeled now.)

The following are the parameters for specifying a turboprop engine in the configuration file:

maxpower — maximal power [HP]
idlefuelflow — idle fuel flow
psfc — power specific fuel consumption
n1idle_max_delay — time constant for throttle lever and generator speed dependency,
maxstartingtime — time for automatic start
startern1 — [%] generator speed for starting
ielumaxtorque — max. allowed torque on the propeller shaft
itt_delay — time constant for ITT temperature calculation
betarangeend — deg
reversemaxpower — [% of maxpower]

Tables:

EnginePowerVC (see diagram 3)
EnginePowerRPM_N1 (see diagram 4)
ITT_N1 — dependency between generator speed and ITT temperature

New parameters in the propeller configuration file:

reversepitch — pitch for reversing

We have written configuration files for the Walter M601 engine and used it for the FlightGear flight simulator.

Future improvements:

Better modeling of fast changes.

Highlighted References

Books:

XML Schema, Eric van der Vlist, O'Reilly Press

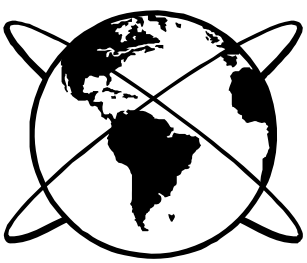
XSLT Cookbook, Sal Mangano, O'Reilly Press

SVG Essentials, J. David Eisenberg, O'Reilly Press

Online:

http://xml.oreilly.com
http://www.w3.org
http://www.w3schools.com

Visit us on the web at:
www.jsbsim.org



JSBSim in Use . . .

Here is a sampling of articles that I am aware of in which JSBSim is mentioned:

1. **Simulated Flight Testing of an Autonomous Unmanned Aerial Vehicle Using FlightGear**, *Eric F. Sorton and Sonny Hammaker*, Institute for Scientific Research, Inc.

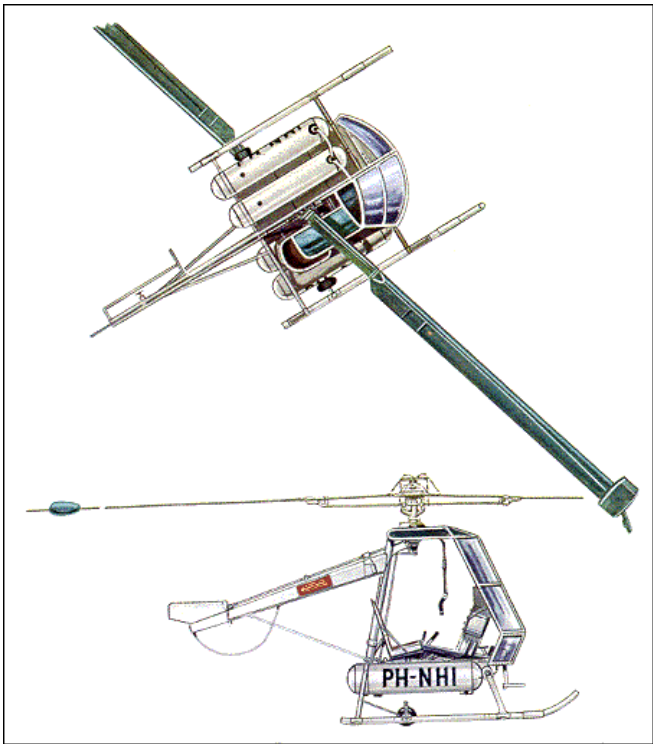
2. **Development of a Low-Cost Simulator for Demonstration and Engineer Training**, *R. Burns, M. Duquette, J. Howerton and R. Simko*, Air Force Research Laboratory, Wright-Patterson AFB AIAA-2003-5758

3. **Autonomous Dynamic Soaring Platform for Distributed Mobile Sensor Arrays**, *M. Boslough*, Sandia National Laboratories, 2002

4. **Robust Non-linear Control through Neuroevolution**, *Faustino John Gomez*, Report AI-TR-03-303, August 2003

5. **A Design Approach for Low Cost, "Expendable" UAV Systems**, *C. Munro and P. Krus*, Linkoping University, Linkoping, Sweden, AIAA-2002-3451

Simulate This! NHI H-3 "Kolibri"



The SOBEH Foundation (Stichting voor de Ontwikkeling en Bouw van een Experimenteel Hefschroefvliegting) was established in the early 1950s to research and build small ramjet-powered helicopters designed by J. Meyer Drees. The **SOBEH-1** helicopter, which flew in 1954, was an open-frame single-seat machine with a skid undercarriage. Two small ramjets were fitted at the tips of the two-blade rotor, which embodied an automatic pitch adjustment system, and the pilot controlled the machine through a suspended overhead stick The SOBEH-1 was written off through ground resonance, but was succeeded by the **SOBEH H-2** (PH-NFT) which was flown in May 1955. The **H-2**, was an improved version with a large windshield and a tiny strutted tail unit with a small anti-torque propeller. It was taken over by Nederlandse Helicopter Industrie N.V. which was formed by Aviolanda and Kromhout and based at Rotterdam, and they refined the design into the two-seat **NHI-3 Kolibri**.

The first **NHI-3** (PH-NHI) flew in May 1956, and featured a more substantial cabin enclosure and a stronger tail unit. The power units were two 55hp NHI TJ-5 ramjets but these were upgraded to the more powerful 100hp TJ-5A in production aircraft. The main role for the Kolibri was in crop spraying, for which role it had spray tanks mounted beneath the fuselage floorpan with standard spray bars. An initial production batch of ten aircraft was initiated in 1958 for customers in Dutch New Guinea, the United Kingdom, Germany and Israel. Further development of the **Kolibri** ceased after the company was acquired by Aviolanda in May 1959.

R.Simpson "Airlife's Helicopter and Rotorcraft", 1998