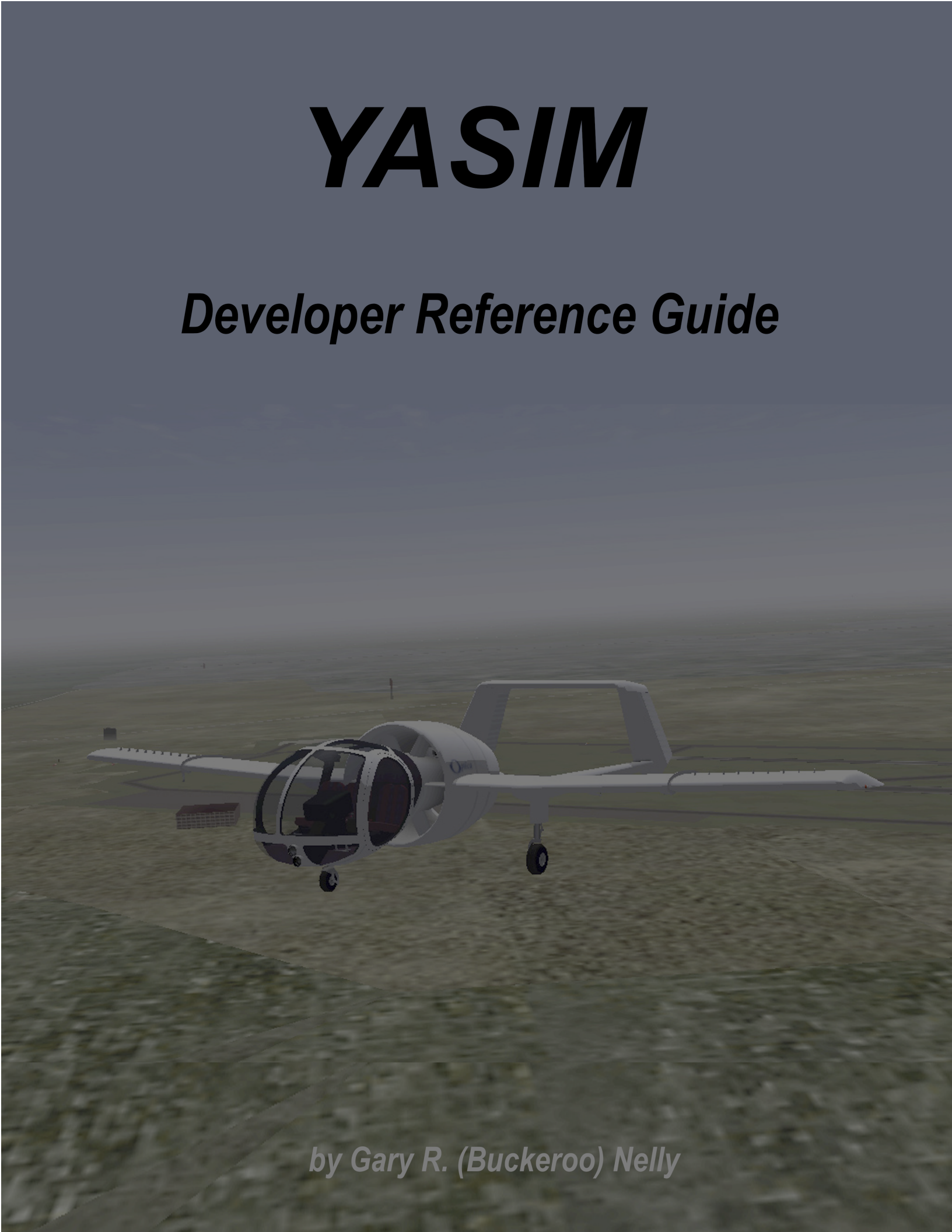
YASIM

Developer Reference Guide



by Gary R. (Buckeroo) Nelly

Guide to YASim

by Gary R Neely, -Buckaroo

I've been slowly collecting notes from my experiences with YASim, one of Flightgear's two primary flight dynamics models (FDMs). YASim is poorly documented, and this is an effort to make the path easier for others than it was for me. There's still work to be done, but I'm making sections available as I assemble them in the hopes that they prove useful. Note that my guides will likely be helpful only for fixed-wings as I have no experience with YASim helicopters.

Buckaroo, April 30 2015

grneely@gmail.com

Guide to YASim

Contents

About this YASim Guide

This guide is the result of years of working intensely with YASim. I started knowing not much, and anyone flying some of my early efforts will need more than a wing and a prayer. But gradually I learned more, from studying the FDMs of others, my own trials and errors, the many comments on the Flightgear forums, digging deep into developer list archives, and long hours working through the code, often working numbers through the algorithms by hand to see what they do. I ended up learning a lot more than I expected. My understanding of what keeps a plane in the air is vastly greater than it was when I began working with Flightgear FDMs.

I can't say I understand all there is to know about YASim, far from it, so I can't guarantee that this guide has no errors or wrong interpretations. It probably has many. But I try to write about things for which I have direct experience. An observant reader will find gaps, and those gaps are likely areas where I have no experience or where I don't yet feel confident in my understanding.

I regret that the guide will not be very useful to rotary-wing enthusiasts. I began using Flightgear by flying helicopters-- it was a novelty because at the time no other simulator had a good helicopter flight model. But my focus shifted to fixed-wings, and I currently have no experience with rotary-wing FDMs. Maybe someday...
I hope those working with Flightgear get something from this guide. It took a long time to assemble, and there's work still to be done. I welcome thoughts, suggestions for additions, and corrections.

I believe a French translation of this guide is in progress, but probably won't be completed for some time.

Acknowledgements
This guide is dedicated to the Flightgear community, and especially to those who have helped me in the past, flight tested my modeling efforts, encouraged me, or flown with me on those long, laid-back flights from here to there in the virtual Flightgear world. These folks include but are not limited to Peter "Farmboy" Brown, Lanny "Someguy" Chambers, Wolfram "Yakko" Gottfried, Syd Adams, Jacob "Tuxklok" Burbach, Emmanuel "Helijah" Baranger, Jörg "Jomo" Emmerich and quite a few others who it's been my honor to know and work with.
- Gary "Buckaroo" Neely

Contact me as "Buckaroo" on the Flightgear forums or as grneely on Google's gmail service

Chapter 1
What is YASim?

By Gary "Buckaroo" Neely

YASim is a flight dynamics model (FDM), a mathematical construct of algorithms and data that determines the physics of flight within the simulator. The 3D aircraft model has nothing to do with flight dynamics-- the 3D model is essentially just a picture to look at. It's the flight model, the FDM, that dictates how the model flies. YASim's author, Andy Ross, amusingly titled his work "Yet Another Simulator"YASim uses the geometry of the aircraft to generate basic flight characteristics. For example, you feed the FDM informaton about the a wing, its shape, its chord, sweep, taper, incidence, span, stall characteristics, etc., and the FDM calculates the wing's contribution to the physics of flight. Provide the FDM information about the fuselage, its length, max width, taper, etc., and the FDM calculates the effect of this body on flight. Add in details about the power plant, the position of landing gear, fuel tanks, the aircraft weight, weight of extra payload. Describe the the aircraft's low speed, high angle of attack flight and the low angle of attack cruise flight, and the FDM uses these to model the extremes of flight. When you're done, you've created a mathematical model of the aircraft that can be used in a real-time simulation.

Out of the box, a YASim FDM is a crude approximation at best. YASim is not a magic solution where you plug in the aircraft dimensions and out pops a realistic flight model. Any YASim FDM will require much adjustement before you get a result that approaches authentic behavior. With experience, YASim can quickly give you a flyable solution, but obtaining realistic results requires research and effort. Be prepared to spend a lot of time refining your solution. In the end, your YASim result is only as good as your knowledge of the aircraft and your time spent tweaking the FDM.

YASim is capable of generating simulations for both fixed-wing and rotary-wing aircraft. This guide is aimed at fixed-wing developers. I have no experience developing helicopters, so at present this guide does not include sections dedicated to rotary-wing elements. Even so, helicopter enthusiasts may find other portions of this guide useful.

YASim and JSBsim

YASim is one of the two most commonly used FDMs in Flightgear. JSBsim is the other popular FDM used by the Flightgear community. I began working with FDMs using JBSim, but found the learning curve rather steep and moved over to working with YASim. YASim, despite very poor documentation, seemed more approachable and intuitive. My problem was not with JSBsim itself, rather it was my insufficient understanding of aerodynamics. I don't regret my YASim experience, far from it, yet if I knew four or five years ago what I know now, I might have stayed with JSBsim given its more versatile approach.

JSBsim and YASim have strengths and weaknesses. JSBsim is a richer, more flexible development environment, but the learning curve is steep and you will need more than a casual knowledge of aerodynamics. It also helps if you are comfortable with programming or scripting. If you have aerodynamic data for your aircraft and desire programatic control over all elements of the simulation, then JSBsim is the better choice. If you lack such data but know the geometry of the aircraft and have access to the same information available to a real pilot, then YASim can provide a simulation that is more than adequate for general aviation aircraft.

Make no mistake-- both YASim and JSBsim require a working knowledge of the fundamentals of flight.

YASim Issues

Any appraisal of YASim must acknowledge its limitations. The most consequential have little to do with flight simulation:

Documentation: YASim has almost no documentation except a short guide that lists and briefly defines parameters used to create a YASim FDM. Likewise, documentation of the underlying code is very thin. This situation is hard to justify given that YASim has been around for more than ten years and is heavily used by the Flightgear community. This guide is meant to help address that issue for FDM developers of fixed-wing aircraft. Unfortunately I know of no effort addressing the code documentation problem.

The code base: YASim has not seen dedicated development for a number of years and the code base has no active

maintainer. The original developer has moved on to other projects. The code, while brilliant in places, has its share of ugly sections and is very poorly documented. Some portions of the simulation are crude hacks meant for later replacement. There are ongoing questions regarding possible bugs in the simulation. (I've discovered at least one significant bug myself.) Since there is no code maintainer, getting a change made to the code base is like waiting for an ice cube to melt in Antarctica. Some excellent modifications by experienced developers have gotten lost waiting for review and never made it into YASim. In other cases, hacks that never should have been implemented have somehow entered the code base.

YASim has some limitations as a flight simulator. Here are some I've observed or suspect:

- - YASim was originally built to simulate subsonic general aviation aircraft. If the developer is not demanding precise realism, YASim can simulate aircraft up to the transonic realm, but it lacks provisions that make it suitable for accurate simulation of supersonic aircraft. Simulation of the transonic realm is mitigated by the fact that many aircraft like modern airliners have features that minimize transonic effects to the point where a pilot does not even notice them. This allows the YASim developer to ignore the problem for many aircraft in the transonic realm. In other cases, transonic effects can be simuated with some clever Nasal scripting. But the YASim developer should be aware that the FDM will not yield realistic characteristics at near-supersonic speeds or faster.

- - YASim cannot simulate a true flying wing. YASim requires that the aircraft have a horizontal stabilizer to offset pitching moments generated in flight. In a flying wing, these moments are offset by reverse or "reflex" camber in the trailing edge of the wing's airfoil and other methods. If exact fidelity is not required and you just want your model to fly reasonably, you can cheat in YASim by placing a horizontal stabilizer close to the wing's trailing surface. But it won't be a true flying wing simulation.

- - Currently there is no way to change the incidence of a flight surface in flight. YASim cannot simulate an all-moving flight surface such as stabilators or alter trim by changing incidence.

- - The nature of YASim does not lend itself to easily creating multiple models with different engine specifications. You cannot simply "bolt on" a different engine by changing engine parameters and get the expected performance change. If you simply change engine parameters, YASim will try to balance the changes elsewhere to give the same approach and cruise characteristics. This means you won't get the results you expect without alterations elsewhere in the FDM. Similarly, adding floats to an aircraft requires significant FDM refinement. To be fair, this would be true of a real-life aircraft designer.

- - Many engine elements are not well simulated. The YASim piston engine simulation is adequate for engine performance, but gives poor results for fuel consumption, manifold pressure values, and engine temperature values. For a good simulation, the developer should replace or modify YASim results with custom values using Nasal scripting. Jet engine simulation is rather simple and works best for older low-bypass engines. Turboprop engine code is rather rudimentary. Propeller feathering is not yet implemented.

- - YASim is not an advanced simulator. The effects of flight surfaces are cummulative on the model, but all surfaces operate in their own space and don't affect the results of other surfaces. You won't see deep-stall issues where a wing or fuselage blocks airflow over the tail. You won't see induced drag benefits from adding little winglet surfaces at wingtips. Aircraft like the Cessna 172 won't pitch up when flaps are deployed due to increased airflow over the stabilizer. This isn't to say you can't simulate such things while using a YASim FDM, only that YASim isn't going to automagically do it for you.

- - There are some basic simulation issues. Ground interaction has always been a bit twitchy. Induced drag has a bug (but there is an easy work-around for that). There may be a bug that causes aircraft to unduly turn into the wind upon landing. Airfoil lift and moment calculations are not all they could be. It's difficult to get exact pitch behavior upon flap deployment. Flap lift and drag cannot be interpolated to different values over the range of flap deployment. Etc. Most of these can be mitigated with tweaking and patience, or overcome by some clever thinking.

YASim And This Guide

YASim is well-suited to simulation of general aviation aircraft and most sport aviation. If you are modeling an aircraft in those categories, YASim can do a good job. I've created aircraft that fly close to flight manual parameters and reported behavior. But YASim won't do this for free. You will have to research your aircraft and understand what YASim is doing with the values you give it. Helping you understand how YASim uses these values is the purpose of this guide.

Chapter 2
Basic Layout of a YASim Flight Model

By Gary "Buckaroo" Neely

A YASim FDM is an XML file made up of elements that define the structure and characteristics of the aircraft. It consists of elements that bracket the flight characteristics, describe the flight surfaces, the fuselage or other non-flight surface bodies, propulsion systems, landing gear, fuel tanks, and a few other features and options.

The following is a suggested structure for a conventional sport aircraft having a single piston engine, a constant-speed propeller, tricycle gear, flaps, and a fuel tank in each wing.

```
<?xml version="1.0"?>

<airplane mass="..." >

<approach ... >
  <control-setting axis="/controls/engines/engine[0]/throttle" value="..."/>
  <control-setting axis="/controls/engines/engine[0]/mixture" value="1"/>
  <control-setting axis="/controls/engines/engine[0]/propeller-pitch" value="1"/>
  <control-setting axis="/controls/gear/gear-down" value="1"/>
  <control-setting axis="/controls/flight/flaps" value="1"/>
  ...
</approach>

<cruise ... >
  <control-setting axis="/controls/engines/engine[0]/throttle" value="1"/>
  <control-setting axis="/controls/engines/engine[0]/mixture" value="0.65"/>
  <control-setting axis="/controls/engines/engine[0]/propeller-pitch" value="1"/>
  <control-setting axis="/controls/flight/elevator-trim" value="..."/>
  <control-setting axis="/controls/gear/gear-down" value="0"/>
  <control-setting axis="/controls/flight/flaps" value="0"/>
  ...
</cruise>


<cockpit x="..." y="..." z="..." />


<fuselage ax="..." ay="..." az="..." bx="..." by="..." bz="..." width="..."
taper="..." midpoint="..." />


<wing ... >
  <stall ... />
  <flap0 ... />
  <flap1 ... />
  <control-input axis="/controls/flight/flaps-norm" control="FLAP0"/>
  <control-input axis="/controls/flight/aileron" control="FLAP1" split="true"/>
  <control-output control="FLAP0" prop="/surface-positions/flap-pos-norm"/>
  <control-output control="FLAP1" side="left" prop="/surface-positions/left-
aileron-pos-norm"/>
  <control-output control="FLAP1" side="right" prop="/surface-positions/right-
aileron-pos-norm"/>
  <control-speed control="FLAP0" transition-time="..."/>
</wing>
```

```
<hstab ... >
  <stall ... />
  <flap0 ... />
  <control-input control="FLAP0" axis="/controls/flight/elevator" invert="true"/>
  <control-input control="FLAP0" axis="/controls/flight/elevator-trim"
invert="true"/>
  <control-output control="FLAP0" prop="/surface-positions/elevator-pos-norm"/>
</hstab>

<vstab ... >
  <stall ... />
  <flap0 ... />
  <control-input axis="/controls/flight/rudder" control="FLAP0" invert="true"/>
  <control-input axis="/controls/flight/rudder-trim" control="FLAP0"
invert="true"/>
  <control-output control="FLAP0" prop="/surface-positions/rudder-pos-norm"/>
</vstab>

<propeller ... >
  <actionpt x="..." y="0" z="..."/>
  <control-input axis="/controls/engines/engine[0]/propeller-pitch"
control="ADVANCE"/>
  <piston-engine ... >
    <control-input axis="/controls/engines/engine[0]/throttle" control="THROTTLE"/>
    <control-input axis="/controls/engines/engine[0]/starter" control="STARTER"/>
    <control-input axis="/controls/engines/engine[0]/magnetos" control="MAGNETOS"/>
    <control-input axis="/controls/engines/engine[0]/mixture" control="MIXTURE"/>
  </piston-engine>
</propeller>

<gear x="..." y="0" z="..." ... >
  <control-input axis="/controls/gear/gear-down" control="EXTEND"/>
  <control-output control="EXTEND" prop="/gear/gear[0]/position-norm"/>
  <control-speed control="EXTEND" transition-time="..."/>
</gear>

<gear x="..." y="..." z="..." ... >
  <control-input axis="/controls/gear/brake-left" control="BRAKE"/>
  <control-input axis="/controls/gear/brake-parking" control="BRAKE" split="true"/>
  <control-input axis="/controls/gear/gear-down" control="EXTEND"/>
  <control-output control="EXTEND" prop="/gear/gear[1]/position-norm"/>
  <control-speed control="EXTEND" transition-time="..."/>
</gear>

<gear x="..." y="-..." z="..." ... >
  <control-input axis="/controls/gear/brake-left" control="BRAKE"/>
  <control-input axis="/controls/gear/brake-parking" control="BRAKE" split="true"/>
  <control-input axis="/controls/gear/gear-down" control="EXTEND"/>
  <control-output control="EXTEND" prop="/gear/gear[1]/position-norm"/>
  <control-speed control="EXTEND" transition-time="..."/>
</gear>


<tank x="..." y="..." z="..." capacity="..." />
<tank x="..." y="-..." z="..." capacity="..." />


<ballast x="..." y="..." z="..." mass="..." />
```

```
</airplane>
```

It's not as scary as it might appear. With a little experience, many things are cut-and-paste jobs, changing a few parameters.

The YASim Coordinate System

Before continuing with descriptions of the elements, a word on the YASim coordinate system. Many YASim elements specify an x, y, z position for a component. Within the YASim XML file, the positive x axis is forward, the positive y axis points left, and the positive z axis is up. When positioning elements like fuel tanks, landing gear, or engines along the right wing, their y axis coordinates will be negative. In YASim, all coordinates are in meters. YASim curiously mixes both English and metric units.

The origin of your coordinates can be anywhere. The simulation will rotate the model about its center of gravity in flight, and the CG will always be placed correctly with respect to the origin you define. Some developers place their origin at a guess for the CG location. I use the nose of the aircraft as my origin, because this is the standard historical origin often used by engineers and dating back to early naval drafts. This is helpful when working with engineering plans, as positions are often designated along the station-line or fuselage station line, which is a measurement taken from a point in front of the aircraft. The equivalent for the z axis is the water-line, and for the y axis, the butt-line. Usually engineers will place the origin of the station-line and water-line at a location such that all measurements result in positive numbers. The butt-line is the exception, as it is measured from the axis of symmetry. I follow this standard when laying out my YASim models.

It helps reduce errors if your 3D model uses the same origin. This way you can take the numbers straight from your model and use them in your FDM, with no more trouble than perhaps reversing the sign of the y axis.

In flight, the aircraft's orientation will pivot about its center of gravity, but when rotating an external view about the model, the model will appear to rotate about the model's origin. In most views this will feel unnatural. This is easily remedied by setting an offset to position the view closer to the model's approximate CG. The Flightgear wiki has an article on this: Model View Offsets

The YASim Elements

Let's look at each of the fundamental YASim elements and see what they do. Not all elments are required, and many will be different from the above example, but this will give you an idea how to begin.
Elements take the form "<element-name [list of attributes]>", where an attribute looks like "attribute-name="n", where n is some value. Attributes should be separated by spaces, not commas. Many elements will contain sub-elements. Those that do have a different form: "<element-name ...>[list of sub-elements]</element-name>. You'll quickly get the idea from the examples.

The Airplane Element

The primary element for the FDM is the element.

```
<airplane mass="..." >
...
</airplane>
```

This has a single required attribute, mass, which should be the dry, empty weight of the aircraft in pounds.
All other YASim elements are contained within the <airplane> element. The order of elements does not matter. I use the above order in my FDMs, which is a common convention with many other YASim FDMs. Make sure that your FDM ends with the closing </airplane> tag.

Approach and Cruise Elements

Next are the approach and cruise elements.

```
<approach ... >
...
</approach>

<cruise ... >
...
```

```
</cruise>
```

These two elements bracket the flight envelope of the aircraft. Both are required and are critical to the flight behavior. For details on how to set up and tune these elements, see Approach and Cruise Settings.

The Cockpit Element

Next up is the <cockpit> element.

```
<cockpit x="..." y="..." z="..." />
```

This element specifies the pilot's viewpoint for the model. This may be a legacy element. It is parsed but not used within YASim and appears to be optional. Possibly it defines a default eyepoint if no other viewpoint is specified in the aircraft's -set.xml file. I've always defined my viewpoints in a separate views.xml file, so I've never tested to see if this element is used. I typically include it as a quick reference for a base viewpoint.

Fuselage Elements

Fuselage elements define structures that have mass and aerodynamic influence but otherwise don't determine primary flight characteristics. They serve as ground contact points, act as simple surfaces that contribute drag, and affect mass distribution. The structure of a fuselage element will usually look like this:

```
<fuselage ax="..." ay="..." az="..." bx="..." by="..." bz="..." width="..."
midpoint="..." taper="..." />
```

Most aircraft will have at least one fuselage element representing the actual aircraft fuselage. Additional Fuselage elements can be used to represent engine nacelles, or wheel pants, external fuel tanks, static weapon structures, etc. You can have as many fuselage structures as you need, or none at all. For a detailed description, visit Fuselage Elements.

Flight Surface Elements

Flight surfaces like the wing, horizontal stabilizer, and vertical stabilizer are the main lifting and control surfaces of an aircraft. A flight surface element looks like this:

```
<wing ... >
  <stall ... />
  <flap0 ... />
  <flap1 ... />
  ...
  <control-input .../>
  ...
  <control-output .../>
  ...
</wing>
```

The element name is followed by a number of attributes that describe the geometry of the surface. The stall sub-element describes stall and lift behavior. Optional flap0 and flap 1 sub-elements describe moveable control surfaces such as a flap, rudder, elevator, etc. Control inputs bind a Flightgear property to a flap for manipulation of the moveable surface, and control outputs create properties tht show the current position of a moveable surface.

A "wing" is just one of the available flight surface types. An "hstab" (horizontal stabilizer) is another. YASim requres that the flight model have exactly one wing and one hstab, but other flight surface types allow you to describe vertical stabilizers and additional wings.

You can read more about flight surfaces and their many options here: Flight Surfaces

Propulsion Elements

YASim has a number of ways to propel your aircraft through the virtual skies. The most common are jet engines and piston-driven propellers, followed by turboprop-driven propellers. There is also a simple thruster. A basic propeller-driven unit has this form:

```
<propeller ...>
  <actionpt ... />
  <piston-engine ...>
    <control-input .../>
    ...
  </piston-engine>
</propeller>
```

A jet engine definition is a little simpler:

```
<jet ...>
  <actionpt ... />
  <control-input ... />
  ...
</jet>
```

Each consists of a primary element with a set of descriptive attributes, a sub-element that describes where thrust is applied (the action point), and a set of sub-elements describing engine controls. In the case of the propeller, the actual powerplant is a sub-element.
You can have any number of propulsion elements, 1 to 4 being common. The engines will be enumerated in Flightgear property lists by the order in which they are defined in the YASim file. So the first engine defined will be 0, the second will be 1, etc.
Propulsion elements are a broad topic. Piston engines and jet engines are described in much more detail here:
Piston-driven Propeller Engines
Jet Engines

Writing guides for turboprops and thrusters is on my to-do list.

Landing Gear
Each landing gear unit receives its own element in the flight model. A typical gear element looks like this:

```
<gear x="..." y="..." z="..." ... >
  <control-input axis="/controls/gear/brake-left" control="BRAKE"/>
  <control-input axis="/controls/gear/brake-parking" control="BRAKE" split="true"/>
  <control-input axis="/controls/gear/gear-down" control="EXTEND"/>
  <control-output control="EXTEND" prop="/gear/gear[1]/position-norm"/>
  <control-speed control="EXTEND" transition-time="..."/>
</gear>
```

The primary gear element describes the coordinates for the gear's ground contact points at full extension and a set of attributes for other gear parameters. Full extension in this case means the gear's fully extended length, with no compression on the shock absorber. Sub-elements describe the gear's braking controls and characteristics, optional controls for extending the gear in the case of retractable landing gear, and transition time for retraction.

You can have any number of gear elements, 3 being common. Landing gear are enumerated in the property tree by the order in which they are defined in the YASim file. So the first gear will be 0, the second will be 1, etc.
A guide for landing gear is on my to-do list.

Fuel Tanks
Fuel tank elements are simple:

```
<tank x="..." y="..." z="..." capacity="..." />
```

The tank has a set of coordinates locating its center of mass, and a capacity expressed in pounds. You can have as many as you require, and each is enumerated according the order in which it is defined. See Weight and Balance for more information on setting up and using fuel tanks.

Ballast
Ballast elements are also simple:

```
<ballast x="..." y="..." z="..." mass="..." />
```

Ballast is simply a mass and coordinates for its position. Ballast elements do not add weight, they re-distribute weight. This topic is covered in more detail in Weight and Balance and Inertia Tensors and YASim.

Other Elements
YASim features additional elements for aerotowing, catapult launching, anchoring (useful for flying boats and seaplanes), and helicopter simulation. Anchoring and aerotowing are on my to-do list. Unfortunately I have no experience with YASim rotary wings and so I have no plans to write a guide on the YASim rotor elements.

Documenting Your Flight Model
Documentation of your flight model is important, not only for others who may view your FDM, but for yourself. You may find yourself coming back to a flight model a year after developing it, and wondering "Why did I set this value to 1.5?"

Document your FDM settings in the FDM itself. A comment takes the form:

<!-- My really illuminating comment. -->
Beware of using "--" within your comments. The parser will interpret that as an end to the comment and you will likely get an error in your FDM.

Many settings are not intuitive especially to inexperienced designers. You may have spent a great deal of time and effort researching a behavior and tweaking an attribute for flight results, but if you don't document your rationale no one will know why those values were chosen, and it's very possible some enthusiastic but inexperienced follow-on developer may change your values without understanding them. Lack of proper documentation is probably the greatest flaw of YASim itself. Don't make the same mistake in your FDM.

Chapter 3
YASim Fuselage Elements

By Gary "Buckaroo" Neely

Fuselage elements define structures that have mass and aerodynamic influence but otherwise don't determine primary flight characteristics. They serve as ground contact points, act as simple surfaces that contribute drag, and affect mass distribution.

A typical fuselage element looks like this:

<fuselage ax="0" ay="0" az="0" bx="-5.569" by="0" bz="0.255" width="1.200" taper="0.3" midpoint="0.5" />

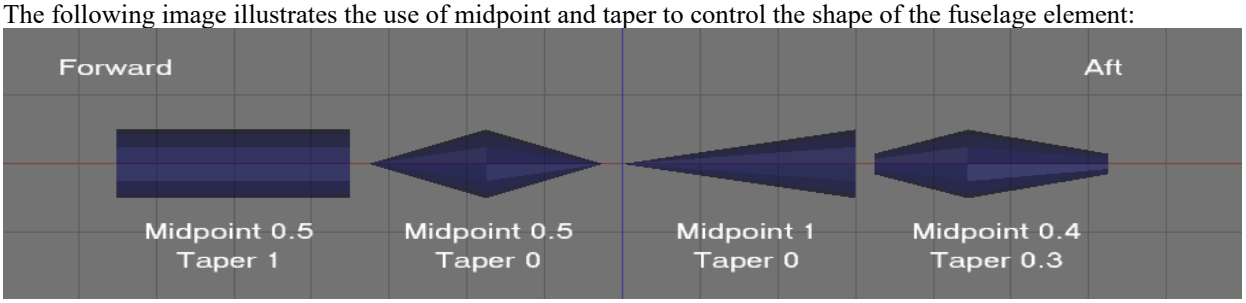
Most aircraft will have at least one fuselage element representing the actual aircraft fuselage. Additional Fuselage elements can be used to represent engine nacelles, or wheel pants, external fuel tanks, static weapon structures, etc. You can have as many fuselage structures as you need, or none at all.

Fuselage Attributes
The structure of a fuselage element should always contain the following attributes:
<fuselage ax="..." ay="..." az="..." bx="..." by="..." bz="..." width="..." />

A fuselage element also has these optional attributes:
midpoint (defaults to 0.5)
taper (defaults to 1)
cx, cy, cz (each defaults to 1)
idrag (defaults to 1)

In most cases you will provide values for midpoint and taper rather than use the defaults. The attributes ax, ay, az are the coordinates for the center of the forward end of the structure, and bx, by, bz determine the center of the aft end. Width is the width of the structure at its midpoint.

midpoint and taper
The defaults create a cylindrical structure, but changing the defaults can create anything from a cone to a truncated biconic structure. Midpoint defines the fractional distance of the widest point of the structure from the front to the aft end. So 0.5 is a true midpoint, 0.25 has the widest part 1/4 the distance from the forward end. Taper determines how much the structure deviates from a cylinder. A taper of 1 creates a cylindrical structure, a taper of 0 creates what is in essence two cones glued together at their bases. Any taper value between 0 and 1 creates two truncated cones glued together at their bases, with the structure becoming more cylindrical as taper is set closer to 1.
The following image illustrates the use of midpoint and taper to control the shape of the fuselage element:

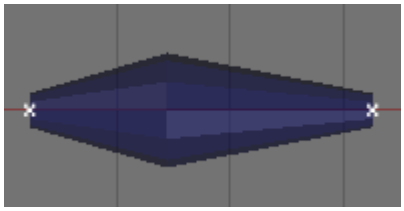


cx, cy, cz
The attributes cx, cy, cz are scalars that multiply the drag forces acting on the structure. Fuselage elements are symmetrical about the x axis, but you can use these attributes to make them appear less symmetrical for drag calculations. Note that these are scalars to forces, not to physical dimensions, so they have no effect on mass distribution.

Let's say you are modeling a fuselage like that of a Piper Cub that has a cross-section approximately twice as tall as it is

wide. In this case, you might set cy=2 and cx=2. To see why, consider a wind blowing sideways against the fuselage (from the y direction). Because the fuselage is twice as tall as it is wide, it has double the surface area when viewed from the side as compared to the top. Therefore the wind has double the force when acting on the side (cy=2). If wind is blowing from the front (the x direction) it would also see double the surface for the same reason, so the force is doubled (cx=2). If blowing from the top (z direction) it would see no change in force because we're using the standard dimensions given by the length and width of the fuselage, unmodified by cz (cz=1, the default).
As a second example, say your fuselage is short and wide like that of a Velocity, with a height about 2/3 the width, you could set cy=0.67 and cx=0.67, providing less sideways and frontal surface area than the fuselage attributes would normally give.

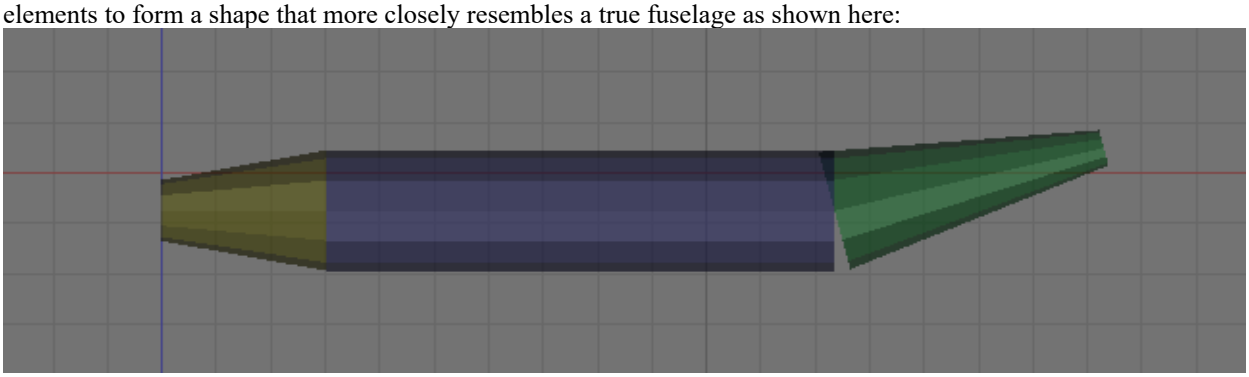
idrag
I haven't used the fuselage idrag attribute and don't trust it. Since fuselage elements are internally divided into segments with each segment treated as a simplified surface, then fuselage idrag likely suffers from the bug that reverses the effects. If so, then decreasing idrag from 1 actually increases induced drag. To decrease idrag, try setting idrag > 1. I haven't tested this for fuselage idrag attributes, so I if you use idrag for fuselage elements, I suggest you perform flight tests to be certain you are getting the results you expect. See Induced Drag and YASim.



Contact Points
The foreward and aft coordinates of each fuselage element are contact points for ground interaction. Ground contact with the object itself will do nothing, only the end points trigger a contact. The small white x's at right show the location of contact points for a typical fuselage element.

Mass and Drag Distribution
Internally, YASim sees each fuselage element as a number of sub-elements. Each fuselage object is divided into a number of segments equal to the ratio of object length to width. So a fuselage that is five times longer than it is wide would be distributed as 5 segments. Each segment is weighted and scaled acording its location in the object and the object's taper and midpoint. Each segment is added to the mass distribution of the aircraft. Drag effects are calculated separately for each segment. Segments are independent-- there is no slipstream or drafting benefit. Unless I've misinterpreted the code, since the relative wind passes through each segment in turn, this suggests YASim may actually be generating excessive drag along the x axis. By tapering the object this effect is considerably reduced because drag of any given segment is based on the segment's 'footprint', but the algorithm still seems less than optiimal to me.

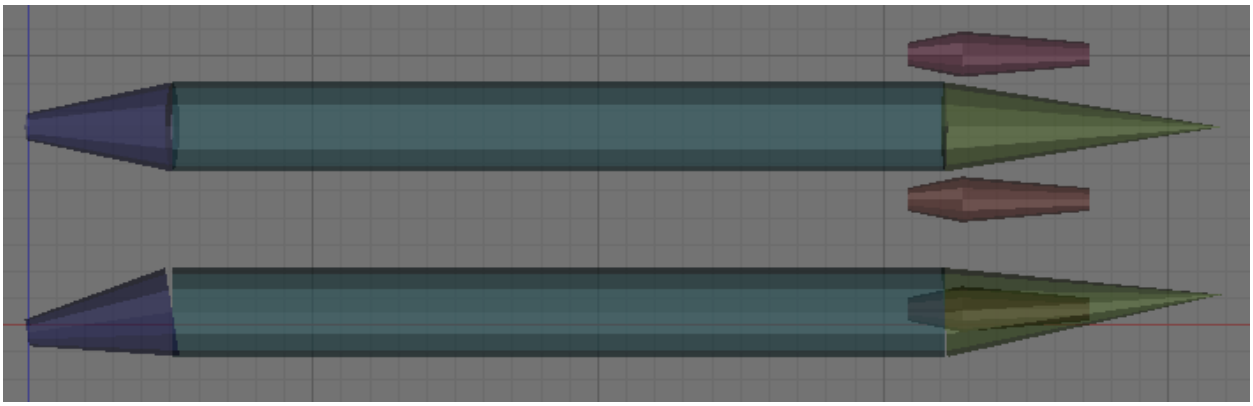
Combining Fuselage Objects
You can usually use a single fuselage element to simulate most objects. If I can, I try to make a fuselage or nacelle using a single fuselage element, using the philosophy of Keep-It-Simple-Stupid. But you can also combine three or more fuselage elements to form a shape that more closely resembles a true fuselage as shown here:



Looking at the YASim code, this seems to have benefits and penalties. Combining objects can give a better mass distribution, though you could do the same with ballast elements. Combinining objects yields approximately the same number of segments as a single object having the same length and width. As noted above, there is no drafting benefits from combinations since all segments are considered separately. Combining objects creates additional unnecessary contact points, since the only points that matter are the ones at either extreme of the combined object.
Here is an example fuselage layout based on my MD-81 model. This layout features three bodies making up the fuselage, and two engine nacelles, shown as a top and side view in the illustration.

<fuselage ax="0" ay="0" az="-0.26" bx="-5.09" by="0" bz="0.51" width="3.33" taper="0.3" midpoint="1" />
<fuselage ax="-5.09" ay="0" az="0.51" bx="-32.16" by="0" bz="0.51" width="3.33" taper="1" midpoint="0.5" />
<fuselage ax="-32.16" ay="0" az="0.51" bx="-41.89" by="0" bz="1.16" width="3.33" taper="0" midpoint="0" />

<fuselage ax="-30.85" ay="2.7" az="0.62" bx="-37.25" by="2.7" bz="0.62" width="1.7" taper="0.5" midpoint="0.3" cx="0.1" />
<fuselage ax="-30.85" ay="-2.7" az="0.62" bx="-37.25" by="-2.7" bz="0.62" width="1.7" taper="0.5" midpoint="0.3" cx="0.1" />



Other Thoughts
When creating fuselage bodies for structures like engine nacelles, keep in mind that these structures usually pass air through them, either as an enclosure for a jet engine, or to direct airflow past engine blocks in the case of piston engines. They usually have very little mass and their drag effects are much less than a similar body that does not pass air through it. Consider scaling down the properties of such fuselage objects to reduce their effects. One way to mitigate the airflow problem is to reduce the drag effects along the object's x axis using the cx attribute, for example, cx=0.2.

Chapter 4
YASim Flight Surfaces

By Gary "Buckaroo" Neely

Flight surfaces are the main lifting and control surfaces of an aircraft. The wing and stabilizers are the most common examples. Defining your aircraft's flight surfaces is likely to be your first step in creating a YASim aircraft FDM.

Here is a flight surface configuration for a wing featuring flaps, ailerons, slats and spoilers:

```
<wing x="-22.25" y="1.630" z="-0.648"
      length="15.632"
      chord="6.308"
      taper="0.17"
      sweep="19"
      dihedral="3"
      incidence="1.5"
      camber="0.1"
      twist="-2.0">
  <stall aoa="14" width="8" peak="1.5"/>
  <flap0 start="0" end="0.597" lift="1.5" drag="2.5"/>
  <flap1 start="0.597" end="0.824" lift="1.25" drag="1.2"/>
  <slat start="0" end="1" aoa="3" drag="1.1"/>
  <spoiler start="0.078" end="0.554" lift="0.7" drag="2.0"/>
  <control-input control="FLAP0" axis="/controls/flight/flaps"/>
  <control-input control="FLAP1" axis="/controls/flight/aileron" split="true"/>
  <control-input control="SLAT" axis="/controls/flight/slats"/>
  <control-input control="SPOILER" axis="/controls/flight/spoilers"/>
  <control-output control="FLAP0" prop="/surface-positions/flap-pos-norm"/>
  <control-output control="FLAP1" side="left" prop="/surface-positions/left-
aileron-pos-norm"/>
  <control-output control="FLAP1" side="right" prop="/surface-positions/right-
aileron-pos-norm"/>
  <control-output control="SLAT" prop="/surface-positions/slat-pos-norm"/>
  <control-output control="SPOILER" prop="/surface-positions/spoiler-pos-norm"/>
  <control-speed control="FLAP0" transition-time="10"/>
  <control-speed control="SLAT" transition-time="10"/>
  <control-speed control="SPOILER" transition-time="1"/>
</wing>
```

Fortunately most flight surfaces aren't quite so busy.

Flight Surface Basics

A surface is any wing-like component of the aircraft that is primarily concerned with aerodynamic properties. The wing, stabilizer, and vertical stabilizer are all flight surfaces. For many aircraft those will be the only flight surfaces you need be concerned with. But things are often more complex. Sometimes the geometry of a wing is more complicated than a simple Hershey-bar configuration, and can't be expressed easily as a single surface. YASim gives us the flexibility of defining a variety of flight surfaces that in combinantion will determine how the aircraft interacts with the air around it.

Types Of Surfaces

YASim flight surfaces come in four flavors:

wing (required)
hstab (required)
vstab
mstab

They share the same possible attributes and sub-elements and differ only in usage. Let's look at each one.

wing
A YASim FDM must have one and only one <wing> element. This is the primary flight surface. You're not limited to this surface, you can still simulate biplanes, but your FDM must have a single <wing> element. The wing is a mirrored surface. In other words, you define only half of it (the left half), and YASim automatically defines the right half. The wing must have a <stall> sub-element to determine stall behavior, and will likely have one or more control surface sub-elements to represent ailerons, elevator, flaps, slats, spoilers. Each control surface will have at least one control input and a control output. More on these later.

hstab
A horizontal stabilizer. You must have one and only one <hstab> element. This and the <wing> are the only required flight surfaces. The hstab differs from the wing only in that the YASim needs to know which surface it will use to trim the aircraft. It does this by setting the incidence of the hstab flight surface. YASim tries to find a setting that will offset the aircraft's pitching moment at cruise using the aircraft's given parameters and a neutral elevator setting. For this reason, you do not define the incidence of an hstab. Like a <wing>, the <hstab> is a mirrored surface.
There is no reason that the hstab must be positioned behind the wing-- the YASim solver can trim the plane using a canard configuration. Canards have some special considerations and some special behaviors, and I'll touch on those later in this guide.

mstab
Recall that you can have only one wing, yet a biplane is still possible. An <mstab> is just another wing-like surface, and is defined in exactly the same way. An mstab is also mirrored like a wing, so you define only the left surface. The difference is that you may have as many mstabs as you like, or none at all. Many planes will not have any mstabs.

The YASim documentation states that mstabs are not involved in the solver computations, but I have found this is not true. I've experimented by splitting a single wing into wing and mstab elements with a combined geometry being the same as the single wing, and found that the solver results are similar though not precisely the same. Examination of the code base shows that mstabs are in fact considered by the solver, so the documentation is likely out of date.
In addition to defining a second wing for a biplane, mstabs have other uses. Many wings have geometry that can't be defined by a single quadrilateral. Airliners for example often have an inboard wing section and an outboard wing section with different geometries. Often these can be simplified into two quadrilaterals. One should be defined using a <wing>, and the second can be defined as an <mstab>. When choosing which should be the <wing>, I pick the section with the greatest surface area and the most control surfaces.

vstab
These are optional flight surfaces like mstabs, but they differ in two ways: they are not mirrored and their dihedral defaults to 90 degrees. Since the vstab is not mirrored, if you use one as a left wing segment, you must define a second, separate vstab for a right wing segment. Dihedral for other flight surfaces defaults to 0 degrees, but for vstabs the default is 90. I'll talk more about dihedral in a moment. Like mstabs, vstabs also appear to be factored into the solver computations, despite the YASim documentation. If the geometry of the wing segment is the same, the solver will generate the same results using an mstab or two vstabs.

The most common use of a vstab is to define a vertical stabilizer with a rudder, which is why dihedral defaults to 90 degrees and why they are not mirrored. Most aircraft will have one vstab, but sometimes there are two (A-10, Beech Twin) or even three (Lockheed Constellation). But vstabs have other uses. They can be employed as winglets with their own control surfaces, handy for more complex wing profiles. The YASim documentation suggests using vstabs for the second wing of a biplane, but unless you need separate controls for the left and right flight surfaces, an mstab is simpler to configure.
Using vstabs for left and right wing segments have some special considerations. If the left wing segment's dihedral is 3 degrees, the right wing segment will be set to 177 degrees (180 - 3). Also, remember the orientation of the wing's lift vector; for the right wing segment, you'll likely want to reverse the sign of your incidence, twist, and camber.

Flight Surface Geometry
The geometry or physical attributes of a flight surface determines where the surface is positioned in relation to the aircraft, its appoximate shape, its orientation, and its aerodynamic properites. Physical properties are defined using these attributes:

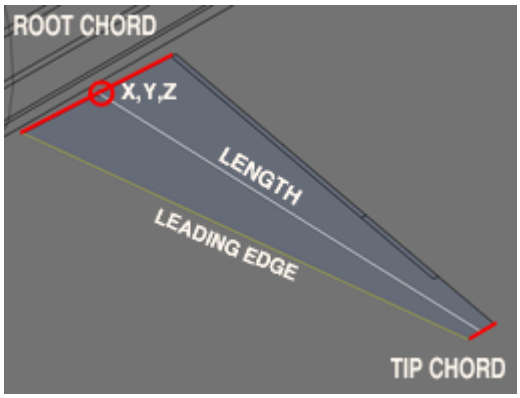
location

length
chord
taper
sweep
dihedral
incidence
twist
camber
idrag
effectiveness

For example, a basic vertical stabilizer might be defined as:

```
<vstab x="-38.578" y="0" z="1.952"
       length="5.185"
       chord="4.507"
       taper="0.928"
       sweep="44">
  ...
</vstab>
```

For any surface you will always define a location, a length, and a chord. The other attributes are optional, but you will typically also define taper, sweep, dihedral, incidence, twist, and camber, at least for wings and mstabs. Let's examine each attribute in detail.

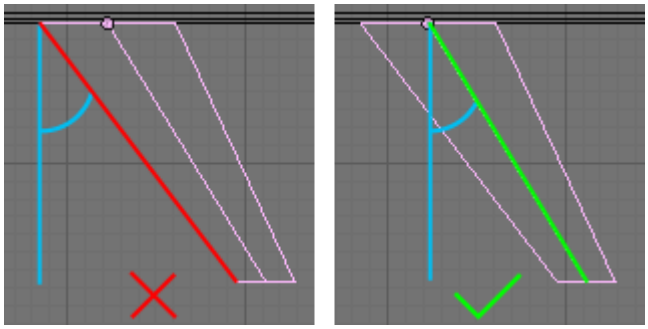


location
The mid-chord root of the wing serves as the base location of the flight surface. Provide x,y,z coordinates for the base as shown in the picture. This is always the left surface for wing, hstab and mstabs-- the right surface is automatically generated via mirroring. Vstabs are not mirrored, so if such flight surfaces are to be symmetrical, you must provide two separate vstabs where the right-side y coordinate is likely to be the negative of the left-side y.

length
Measure length from the middle of the root chord to the middle of the tip chord. Make certain you are not measuring the leading or trailing edge.

chord
This is the length of the root chord.

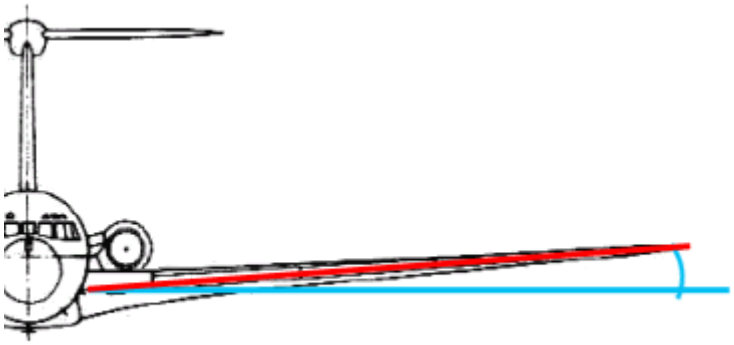
taper
Taper is the ratio of the tip chord length to the root chord length and is one of the two attributes (with sweep) that defines the shape of a flight surface. If a wing tip has a chord of 2 meters and the root a chord of 4 meters, then taper is 0.5 (2/4). A simple straight wing where all ribs have the same chord would have a taper of 1, the so-called "Hershey bar" wing. The default taper is 1.



sweep
This is the angle of the wing's mid-chord line with respect to the station line (usually the x-axis running front to back through the fuselage). Sweep should not be measured based on the wing's leading edge. A wing with a sweep of 0 has a mid-chord line that is 90 degrees to the station line-- it sticks straight out for the fuselage. A wing with a 45 degree sweep would have the classic delta appearance. A wing with a negative sweep would actually bring the wing tips forward, as is done in some sailplanes and experimental designs. Measure sweep from the wing's origin. Sweep defaults to 0.

The image at right shows the wrong way and then the right way to find sweep. Don't use the leading edge, use the mid-chord line.

dihedral



The angle of the wings with respect to the fuselage, as seen from the front or aft view of the aircraft. A positive dihedral is a rotation of the wing upward, lifting the wing tips. A negative value (also called anhedral), is a rotation of the wing downward, drooping the wing tips. Most aircraft, especially low-wing aircraft, have a positive dihedral. Some like the An-124 and the Harrier have an anhedral. Dihedral is measured from the wing's origin. The default is 0.

Dihedral affects stability along the roll axis by helping to keep the wings level in flight without pilot input. In flight, a roll to one side allows a component of the wing's lift vector to pull the aircraft sideways, causing a sideslip. With a positive dihedral, the sideslip slightly rotates the lowered wing forward, bringing it into a higher angle of attack than the opposing wing. The result tends to bring the aircraft level again. Too much dihedral can make the plane too resistant to roll or cause wallowing dutch roll behavior. An anhedral has the opposite effect and is often used as compensate for roll resistance in aircraft where the center of gravity is lower than the wing. (The B-25 has a curious gull-wing dihedral because the prototype with its continuous dihedral exhibited stability issues, dutch rolls, and couldn't do flat rudder-only turns, a requirement for bomb runs. The cranked-wing configuration solved these issues.)

A simple measurement of the dihedral angle may not give you the desired simulation results. Other design factors contribute to roll stability. Some aircraft have very little dihedral yet are very stable in rolls. Some with extreme anhedrals suggest an unstable aircraft, when in fact the aircraft is very docile. Apparent dihedral angle is only the first step in choosing a value for YASim. You must have a good feel for the design and the behavior of the aircraft to select a good value.

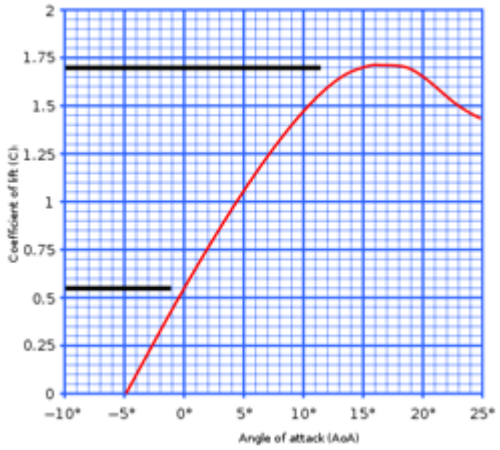
YASim's fixed stabilizer incidence is in some ways an unfortunate design choice. YASim determines stabilizer incidence such that the aircraft will fly

level with no elevator input at max cruise speed. But many planes were not optimized for cruise flight. The original Beech Twin models had a low stabilizer angle optimized for low-speed flight, making them a delight on approaches, but cruise performance suffered. After the war when models were being produced primarily for civilian markets, the demand was for better cruise performance, so Beech increased the stabilizer incidence. Pilots used to the old performance often disliked the change, though owners paying for fuel appreciated it. You can observe the difference by looking for the curiously high fuselage-stabilizer fairing on the later Beech 18 models.

incidence
This is the angle between the fuselage station-line or longitudinal axis, and the wing's chord line. Most wings have some positive incidence (the leading edge is higher than the trailing edge) to improve visibility on approach, shorten takeoff rolls, reduce drag by better aligning the fuselage with the relative wind, etc. If incidence is not known, 2 or 3 degrees is a good guess for general aviation aircraft. Incidence defaults to 0. Incidence affects stall behaivor. See YASim's stall element.

You cannot define the incidence of horizontal stabilizer (hstab) element. YASim sets this for you, it's part of the purpose of the solver. If you try to define an incidence for an hstab element, it will be ignored. You can offset this effect somewhat by applying a non-zero trim control setting in the cruise settings. More on this later.

twist
Most conventional planes have some degree of wing twist that makes the inboard wing stall before the outboard region where the ailerons live. Twist spreads out stall effects over time and helps maintain control during the onset of the stall. Wing twist can be geometric or aerodynamic. A geometric twist is a progressive decrease in incidence from inboard to outboard wing stations-- the wing is literally twisted. This is also called 'washout'. Aerodynamic twist is the difference between the zero-lift angles of the root and the tip airfoils-- the airfoil actually changes shape from root to tip. If not known (and it often isn't given in pilot handbooks or other sources), use -2 or -3 degrees for twist. Twist defaults to 0. For more on twist, see YASim's stall element.



camber

Camber usually refers to the asymmetry between the top and bottom surfaces of an airfoil, but in YASim it is defined as the ratio of the airfoil's lift at 0 degrees angle of attack to the maximum lift at critical alpha. In other words, the percentage of the maximum (stall) C_L that occurs at zero degrees angle of attack. More camber means more lift at low alpha values, but also more drag. The figure at right shows a lift curve for an airfoil. Here, the C_L value at 0 AoA is 0.55, and is maximum at 1.7 when the critical stall AoA is about 16 degrees. Camber for this airfoil can be estimated by 0.55/1.7 = 0.32. This is a thick, high-lift airfoil that no doubt has a ton of drag. Most will have cambers much less than this.

A camber of 0 represents a symmetrical airfoil, and this is the YASim default. Most general aviation aircraft will have some camber. If you cannot find YASim camber for your airfoil, try values like 0.05 up to 0.1. These work reasonably for old but common airfoils like NACA 23015 etc.

Airfoil C_L findings will be higher than those of an actual wing using that airfoil due to the effects of the spanwise pressure gradient. This is the same effect that causes [induced drag]. The spanwise pressure gradient reduces the effective C_L by 10-20%. So a good high-aspect wing can expect a true lift coefficient of about 90% of theoretical C_L. A low-aspect wing might get 80%. For something in the middle, try using 85% of the theoretical airfoil C_L.
Stabilizers are usually thin and symmetrical as they are not usually expected to produce much lift and should have minimum drag. Horizontal stabilizers sometimes have negative camber values, particularly with STOL or heavy-lift cargo aircraft where high-cambered wings with large flaps are designed for maximum lift and have strong negative pitch moments. Canard hstabs usually have high stabilizer loadings and tend to have significant positive camber. Aircraft with all-moving stabilizers will usually have symmetrical airfoils.

induced drag
This takes a little explaining and so I have a separate page dedicated to Induced Drag. Here's the short version: The YASim idrag value defaults to 1. Increasing the value above 1 reduces induced drag. Decreasing the value increases induced drag. This is opposite of what you might expect and what the documentation says. It's likely due to a long-standing bug in the YASim code. I recommend not messing with idrag until you have a good solution and know what you are doing.

effectiveness
This is a scalar applied to the drag derived from the form factor of flight surfaces. In effect it's a modifier for parasitic drag. The attribute is very badly named, and maybe should have been called "pdrag", similar to "idrag". The value defaults to 1. [Note: In my original YASim guide I mistook "effectiveness" with the similarly named "flap effectiveness" control, which is a different beast. The confusion resulted from the unfortunate use of the same name used by a control, the almost non-existent documentation, and my mistake in examining the code base.]
When applied to a wing, increasing this drag factor will generally bring solution drag coefficient and lift ratio numbers closer together while reducing elevator authority. Decreasing it will have the opposite effect. When applied to an hstab, increasing the drag factor increases the spread between drag coefficient and lift ratio numbers (though the effect is slight due to the much smaller surface area typical of stabilizers), and tends to increase elevator authority, radically in the case of canard aircraft, probably because the elevator is ahead of the CG and the increased drag is trying to pull it behind the CG. Decreasing it does the opposite.

When applied to hstabs, "effectiveness" can have a strong effect on elevator authority. This can lead a developer to use it to force solutions where reasonable values for elevator lift failed, when the real problem was poor CG positioning or unbalanced wing moment effects. In reality, a magnified tail surface windvane effect is obscuring more fundamental problems. In my opinion, use of this attribute should be avoided, especially for hstabs, at least until you have a good solution using proper weight and balance numbers.

The Flightgear wiki has a line in "Howto: Understand Console Output" that recommends adding an "effectiveness" attribute when dealing with solutions that fail to converge. This is a terrible idea, like slapping a patch on an already bad repair. Don't do this. If you have to resort to "effectiveness" to get a solution, you're doing something wrong.
Though poorly documented and badly named, "effectiveness" when used right may be interesting, allowing you to change the parasitic drag of flight surfaces. In YASim, this drag is based on the surface area of any given flight surface, which will not account for many factors such as rivet-studded wings, excessive pylons, disruptive engine nacelles, unusually clean surfaces, vortex generators, etc. The "effectiveness" attribute might be useful for increasing the drag of surfaces with lots of struts or wire supports. Using "effectiveness" to modify parasitic drag of a wing or mstab might be helpful for "dirty" aircraft" when you have trouble bringing drag coefficient and lift ratio values closer together. Avoid using it to get better elevator authority.

The Stall Sub-Element
The stall sub-element controls the stall behavior of the flight surface, in other words, what happens at the critical angle of attack. It also affects the lift of the surface. Each surface must have a stall sub-element. A typical stall sub-element looks something like this:

```
<stall aoa="14" width="8" peak="1.5"/>
```

Width will default to 2 and peak to 1.5. The peak default is fine-- don't mess with it until you understand what it does. A width of 2 is very narrow for general aviation wings-- I recommend using higher width settings for most general aviation applications. You should always provide an aoa attribute.

Determining values for stall AoA and width is a matter of airfoil study. Optimally, you want to identify the airfoil used and

locate a report that plots lift vs. angle of attack (AoA or alpha) for a given airfoil at Reynolds numbers appropriate to your aircraft. This is the same data you'll need to calculate YASim camber as discussed above. (Check the report carefully-- some graphs may be plotting values at low Reynolds numbers for small-scale aircraft. The data will be very different for full-scale aircraft.) Often airfoils used by modern aircraft will be either proprietary or a hybrid, in which case you can either make an educated guess or interpolate from similar known airfoils. Keep in mind that airfoil data is for the airfoil section-- wings using that airfoil won't perform as well due to spanwise pressure gradient effects.

Understanding how YASim's stall element works is important to the creation of a good FDM. I have a separate guide for a more complete discussion on YASim's stall element.

Generally speaking, you want the wing to stall before the horizontal stabilizer in order to maintain elevator response up to the aircraft's stall. For canards this is reversed-- the stabilizer should stall first. Wing loading for a canard stabilizer is higher than that of a conventional stabilizer. For starters, set stabilizer stall AoA a couple of degrees higher than than the wing stall AoA, or a degree or two lower in the case of a canard.

If you have nothing else to go on, try these values for a generic sport aviation configuration:

wing: <stall aoa="14" width="8" peak="1.5"/>
hstab: <stall aoa="16" width="4" peak="1.5"/>
vstab: <stall aoa="16" width="4" peak="1.5"/>

Some guidelines: Most general aviation airfoils will begin to stall in the 14-17 degree range, before considerations for twist or incidence which will lower stall angles. High-lift airfoils might begin to stall as low as 10 degrees or so, but it really depends on the particular airfoil. Delta wing configurations often achieve very high alphas before they stall. As a rule, widths should stay in the 2-12 range, with 2 being a very sharp post-stall lift falloff, and 12 being a fairly gentle falloff. Leave peak at 1.5, or consider reducing peak if you understand peak. Numbers outside of these ranges should be viewed with suspicion unless you are certain of what you are doing. When testing stall characteristics, keep in mind that stall is a function of angle of attack, not airspeed.

Lions and tigers and segments, oh my!
Control surfaces have an effect on the amount of work YASim must do. A surface with no control element (flaps, slats or spoilers) is a single segment, with all math operating on that lone segment. When you add control elements, the surface begins to be split into two or more segments, and the calculations are replicated for each segment. For example, if you have a wing with a single control element for ailerons starting at 0.5 and ending at 1, the surface would be split into two segments, one spanning 0 to 0.5, and a second spanning 0.5 to 1. If your aileron ended instead at 0.9, then the surface would be split into three segments: 0 to 0.5, 0.5 to 0.9, and 0.9 to 1. If you have flaps, slats, and spoilers, YASim will create many segments, each having the required combined characteristics required for that slice of the surface. So the more surfaces you have, and the more segments each flight surface has, the more work YASim has to do.

Control Surface Sub-Elements
Now that you know something about flight surfaces, you might be wanting to know how to set up things like ailerons and elevators. These are created by adding control sub-elements to the surface elements. There are three kinds of control sub-elements:

flap[0/1]
slat
spoiler

Control sub-elements have these possible attributes:

start

end
lift
drag
aoa

Let's examine each control element and see how the attributes are used.

Flaps
Each surface may define up to two "flap" elements. YASim uses the name "flap" to denote any movable trailing edge surface, so it could be an elevator, a rudder, an aileron or a true flap. A surface is not required to have any flap elements, but most will have at least one. Wings will usually have one for ailerons and often a second for true flaps. Stabilizers will have one for the elevator or rudder. You cannot define more than two flap elements for a surface. If you need more, you might have to split the surface using mstabs.

A flap element typically looks like this:

```
<flap0 start="0" end="0.597" lift="1.5" drag="2.5"/>
```

You must name your flap element either "flap0" or "flap1". You can have two sub-elements if you use both names. You cannot have a flap sub-element called "flap2"; you're allowed only two. Usually "flap0" is used for true flaps, and "flap1" is used for ailerons, but it doesn't matter. The actual assignment is controled by input/output sub-elements, which I'll talk about a little later in this guide.

> Flap implementation is one of my YASim annoyances. The only values you can tweak are "lift" and "drag". Real flaps allow for an increase of the coefficient of lift. They do so by increasing incidence and changing the camber of that wing segment. Many implementations also extend the wing's chord. One of YASim's strengths is its ability to calculate the effects of a unique wing segment and contribute those effects to the flight surface. YASim could easily have a flap attribute for incidence, camber, and even chord, but instead it provides only a simplified lift attribute with a rather arbitrary moment effect. A great strength of YASim is wasted.

For a flap sub-element you will have start and end positions indicating where the flap resides along the length of the surface, a lift attribute, and a drag attribute. For start and end positions, start should be nearest the root of the surface, and end should be nearest the tip. The are normalized values, i.e., 0-1. If the flap starts at the very root of the surface, then set start to 0. If it ends at the tip, set start to 1. If a flap starts at 25% of the surface length and ends at 90% of the surface length, set start to 0.25 and end to 0.9.

Lift and drag are more subjective. These are multipliers to lift and drag, so the default values are 1, which leaves lift or drag unchanged with flap deployment. Be careful in your flap lift settings. A high lift value for true flaps has a large effect on the pitching moment of the aircraft and can make it difficult to get good solution results. 1.3 to 1.4 are reasonable values for flap lift. Sometimes you can find references that suggest good starting values. For example, split flap characteristics are often cited as providing a maximum of 35% cL increase and 250-300% drag increase. While this doesn't translate nicely to YASim flap lift values, still you would not go far wrong if you began with lift="1.35" and drag="2.75" for split flaps. Drag for true flaps will often greatly exceed lift.

Aileron lift values will rarely exceed 1.2. Be conservative in your aileron lift settings. Most of mine tend be on the order of 1.15 or so. Elevator lift values are often fairly strong, in the 1.3 to 1.8 range. A discussion of elevator lift value belongs in a section relevant to YASim solutions. Rudder lift values tend to be 1.2 to 1.4. We can't enter moment coefficients for yaw, so all we can do is validate values against known performance in crosswind conditions or side-slipping. For aileron, elevator and rudder drag, I use values slightly less than the lift settings. Fly and test your results.

Slats

```
<slat start="0" end="1" aoa="3" drag="1.1"/>
```

In YASim, slats are used to increase the effective stall AoA for that surface. A given surface may have a single slat

definition, or none. A slat definition should have start, end, aoa, and drag attribute. Slats cannot have a lift attribute. Slats work by increasing the stall AoA defined in the wing's <stall> element for the surface segment defined by the slat start and end points. For example, if a wing has a stall AoA defined as 14 and a slat with an AoA of 2, then when slats are fully deployed, the surface segment will have an effective stall AoA of 16. If slats where 50% deployed, the stall AoA would be 15. I'll talk about how control surfaces are deployed later in this guide.

Spoilers

```
  <spoiler start="0.078" end="0.554" lift="0.7" drag="2.0"/>
```

Spoilers in YASim are a generic term for anything that kills lift or adds drag based on a control. So they might be true lift spoilers, or drag-inducing airbrakes, or both. They operate much like flaps, but lift should be a fractional value and is a multiplier to the surface's pre-stall lift. So a lift value of 1 does not effect lift. A value of 0 would mean that at full deflection the spoiler would eliminate all pre-stall lift. You'll likely want to start by setting lift somewhere in a middle range, 0.5 to 0.7 or so, and tune the results with test flights. Note that true spoilers will have both lift and drag effects, while airbrakes will likely have high drag values and leave the left setting at 1. A given surface may have a single spoiler definition or none.

Control-Input Elements

Control inputs are used to map an input property to a surface control. For example, if you want to use the pilot's elevator keys or joystick functions to control a YASim surface used as an elevator, you have to assign an elevator control property to the YASim surface control. Let's look at an example:

```
<hstab ...>
  ...
  <flap0 start="0" end="1" lift="1.3" drag="1.3"/>
  <control-input control="FLAP0" axis="/controls/flight/elevator"/>
  ...
</hstab>
```

Here we have flap0 being used as an elevator. The only thing that defines flap0 as an elevator is the mapping of an elevator control-input property to the surface control. The input is called the "axis", and is just a standard Flightgear property of your choice. The flap's lift and drag properties are interpolated based on the control property value. For example, if "/controls/flight/elevator" is 0.5, the lift attribute of this flap would be 1.3 * (1 + 0.5) = 1.15. This is straight-forward and simple example. Let's look at a wing with two control surfaces, flaps and ailerons:

```
<wing ...>
  ...
  <flap0 start="0" end="0.597" lift="1.5" drag="2.5"/>
  <flap1 start="0.597" end="0.824" lift="1.25" drag="1.2"/>
  <control-input control="FLAP0" axis="/controls/flight/flaps"/>
  <control-input control="FLAP1" axis="/controls/flight/aileron" split="true"/>
  ...
</wing>
```

Here we've selected flap0 to be the true flap control by creating an association with the property "/controls/flight/flaps". We've mapped flap1 as ailerons by assigning it the property "/controls/flight/aileron".
Any control assumes an input of 0 means no control deflection, an input of 1 is maximum control deflection in the positive direction (assuming normalized inputs, we'll get to that later) and -1 is the maximum negative control deflection. With flaps, as we increase the control input, flap deflection is increased proportionally, in the same direction. We don't use negative values for flaps. But ailerons work opposite each other. An input that deflects one downward generally sends the opposite one upwards. We don't want to mess with dual controls for ailerons, one positive, one negative, so we set the optional attribute "split" to be true. This negates the control input for the right wing on surfaces that are mirrored and simplifies our task.

What if you want to reverse an input before it is applied to the control surface? There are several ways, but the easiest is to use the optional "invert" attribute:

```
<vstab ...>
```

```
  ...
  <flap0 start="0" end="0.829" lift="1.4" drag="1.3"/>
  <control-input control="FLAP0" axis="/controls/flight/rudder" invert="true"/>
  ...
</vstab>
```

The most common application of "invert" is the rudder control.
I've mentioned the "split" and "invert" attributes. There is another atttribute set that can help you refine control input values without relying on external methods like scripting. This uses the attributes, "src0", "src1", "dst0" and "dst1". Here's an example:

```
  <control-input control="FLAP0" axis="/controls/flight/mycontrol" src0="-1.0"
src1="1" dst0="0" dst1="0.5"/>
```

In this case, we're taking the possible input range of -1 to 1 and interpolating it to an output range of 0 to 0.5. In other words, if the source property value is -1, the YASim control surface will see 0. If the property is 1, the control surface will see 0.5. These attributes are especially handy if you need to normalize input properties.

Control inputs can also be mixed. You can assign different properties or axes to the same input control by defining two or more controls on the same surface control element. The results are added before being applied to the input control. I've never done this, but I know it's possible. If your inputs are not already well clamped to the desired ranges, you'll probably want to use src and dst attributes to limit the mixed inputs.

There's one special case where you will nearly always use a mixed input: assigning a controls for trimming the elevator:

```
<hstab ...>
  ..
  <flap0 start="0" end="1" lift="1.3" drag="1.3"/>
  <control-input control="FLAP0" axis="/controls/flight/elevator"/>
  <control-input control="FLAP0" axis="/controls/flight/elevator-trim"/>
  ..
</hstab>
```

For an hstab, always provide a control-input sub-element for both primary elevator control and elevator trim control. Both inputs are critical for YASim to solve for approach elevator and tail incidence. You must provide both axes on hstabs or YASim will freak and not give you a solution. Usually these controls are mapped to the same elevator control surface, but they could be different control surfaces. Don't be too literal here-- a trim control may physically operate a trim tab, but its function is to trim the main control surface.

We now have an idea how flap control inputs work. YASim offers several other kinds of controls in addtion to flaps:

flap[0/1]
slat
spoiler
flap[0/1]effectiveness
incidence (inoperative)

Let's look at each one and see what it does.

flap[0/1]
As we've seen, the flap control is used for ailerons and true flaps, controlling the deflection of a flap surface. These inputs are clamped to the range of -1:1. In practice you'd use the full range for ailerons, but for true flaps you'd use a range of 0:1.

slat
This is the control input used to map a control property to a slat extension of a wing. Unlike flaps, you are allowed only one slat. It works much like the flap controls. Example:

```
  <control-input control="SLAT" axis="/surface-positions/slat-pos-norm"/>
```

spoiler
The spoiler extension for a wing. Example:

```
  <control-input control="SPOILER" axis="/controls/flight/spoilers"/>
```

Unfortunately you are allowed only one spoiler for a surface, which can be a problem for some aircraft that feature multiple spoiler or airbrake devices that are not necessarily deployed according to the same rules. This is common with airliners. There are two things you can do: split the surface into multiple surfaces using <mstabs>, or treat multiple spoilers as a single FDM unit, and use custom Nasal scripting to interpolate the spoiler's deployment based on the status of each "virtual" spoiler. For example, if two virtual spoilers are used and only one is deployed, a controller would deploy the single FDM spoiler to 50%.

flap[0/1]effectiveness
These controls allow a dynamic change to the 'lift' attribute of the specified flap.

```
  <control-input control="FLAP0EFFECTIVENESS"
axis="/controls/flight/blownflaps"/>
```

It is a simple multiplier for the "lift" attribute of a flap. If you had a flap with a lift attribute of 1.3 and a control property having a value of 0.5 and an effectiveness property of 1.25, the result would be 1.3 * (1 + 0.5) * 1.25 = 1.1875.

Don't confuse this with the surface "effectiveness" attribute, a different creature entirely. With the flap effectiveness control and some creative scripting you can simulate blown flaps, or you can use the control to eliminate lift over part of the flap deployment range (maybe; see my final note), which is a step toward simulating Fowler flaps which offer increased lift and minimal drag in the early part of the extension range, but the latter part is mostly drag. These controls are available for both possible surface flaps: "FLAP0EFFECTIVENESS" and "FLAP1EFFECTIVENESS". They do not affect flap drag, and surprisingly there is no flap drag equivalent. Note: the allowed values are supposed to be clamped to a range of O:10, but the code base indicates an actual range of 1:10. This suggests it may not be possible to reduce flap lift, which in my opinion severely devalues this control.

incidence
This control was evidently intended to allow dynamic changes to the incidence angle of a wing, but currently it is inoperative. The control is parsed but not mapped to anything in the YASim code. This is odd since the nature of YASim would make an incidence control very easy to implement. As things are now, you cannot dynamically alter wing incidence in YASim.

The Control-Speed Element
For general aviation, most surfaces respond nearly instantaneously to control inputs. Ailerons are directly linked to a stick or yoke and respond as you move your control. But surfaces like flaps and airbrakes are usually activated by means of electric motors or hydraulics and take a while to extend or retract. YASim gives you an easy means to do this using the control-speed sub element.

```
<wing ...>
  ...
  <flap0 start="0" end="0.597" lift="1.5" drag="2.5"/>
  ...
  <control-speed control="FLAP0" transition-time="10"/>
  ...
</wing>
```

Here we've assigned a 10 second interval for flaps to move from from the fully retracted position to the fully deployed position. Any transitions in between will have interpolated time requirements, so a move from retracted to half-deployed requires 5 seconds.

Transition times can be defined for any control surface but in practice you'll use them only for flaps, slats and spoilers.

The YASim document tells us that this sub-element is semi-deprecated, but this is nonsense. For most surfaces requiring transition times, this function is good enough and is much more efficient than Nasal solutions. It should be your first choice. But it is a simple, linear transistion. If you have a more complex or non-linear situation, don't hesitate to pre-process control

inputs using Nasal scripting rather than use the control-speed sub-element.

Control-Output Elements
Control outputs are the reverse of inputs. They allow you to export YASim's internal control position values as Flightgear properties.

```
<hstab ...>
  ...
  <control-output control="FLAP0" prop="/surface-positions/elevator-pos-norm"/>
  ...
</hstab>
```

Here we're exporting YASim's value for the current position of an elevator to the property "/surface-positions/elevator-pos-norm".

This has several uses, the primary one being animations. While inputs issue control signals to YASim, but they are not necessarily indicative of where a surface actually is. An aileron response might be virtually instantaneous, but flaps take a while to deploy-- it might be many seconds before flaps have reached the position indicated by the input control.

Since YASim can split control inputs into two inverted values for the left and right side surfaces, it's sometimes useful to extract those different side values and set them on separate properties. The most common use of this is for aileron animations:

```
<wing ...>
  ...
  <control-input control="FLAP1" axis="/controls/flight/aileron" split="true"/>
  ...
  <control-output control="FLAP1" side="left" prop="/surface-positions/left-
aileron-pos-norm"/>
  <control-output control="FLAP1" side="right" prop="/surface-positions/right-
aileron-pos-norm"/>
  ...
</wing>
```

You can constrain the range of these output values using the optional "min" and "max" attributes. If you wanted to constrain an output to a maximum of 0.5, you could set max="0.5" and the output property would be clamped to a maximum of 0.5. I've never had reason to do this, but the options are there.

Thoughts on creating flight surfaces
Keep it simple. Don't define unnecesary surfaces and keep the geometry basic. If a wing can be done reasonably with a single surface, then express it as a <wing>, not a <wing> and <mstab>. Don't add vstabs for every pylon or extra fin seen on the plane-- quite often the intent of these surfaces is not obvious or beyond the simulation capabilities of YASim. Many of the small surfaces commonly seen on T-tailed aircraft are meant to reduce deep-stall effects, and YASim doesn't simulate deep-stall conditions. Some designers add vstabs to represent winglets on the wingtips of airliners, but the point of such things is to minimize induced drag-- their other aerodynamic and weight effects are very small and ususally not worth creating a surface. Instead, tweak the idrag value for the main wing. The simpler your FDM, the more likely it is that you will get good results and the easier it will be to adjust the portions that matter most.
When defining flap positions, don't try to be ultra precise. If ailerons begin at mid wing and extend 98% of the way to the tip, then don't set start="0.5" end="0.98", set start="0.5" end="1". That tiny .02 sliver forces YASim to create an extra surface segment for dubious benefits. Chances are you are approximating positions anyway, so use values that create the least number of surface segments and allow YASim to operate efficiently.

YASim doesn't give us the capability of entering wind tunnel numbers to get our results, that's not its method, but this doesn't mean you can't get reasonably realistic flights. It does mean you have to research your aircraft and do very many flight tests to fine-tune individual attributes. The attribute values are not what you care about. The flight behavior is what you care about. Pilot reports and handbooks are your friend here. If you have no hands-on experience with the aircraft yourself (and even if you do!), study as many reports from others as you can. YASim is not an easy out-of-the-box solution.

If you want to approximate reasonably realistic behavior, you will have to work at it. There is no magic solution.

Document your FDM settings in the FDM itself. Many settings are not intuitive especially to inexperienced designers. You may have spent a great deal of time and effort researching a behavior and tweaking an attribute for flight results, but if you don't document your rationale no one will know why those values were chosen, and it's very possible some enthusiastic but inexperienced follow-on developer may change your values without understanding them. Lack of proper documentation is probably the greatest flaw of YASim itself. Don't make the same mistake in your FDM.

Chapter 5
Understanding YASim's Stall Element

By Gary "Buckaroo" Neely

Every YASim flying surface (wing, hstab, mstab, vstab) should have an element describing stall behavior. A Typical stall element looks much like this:

<stall aoa="16" width="8" peak="1.5" />

Here's what the YASim documentation has to say about the stall element:

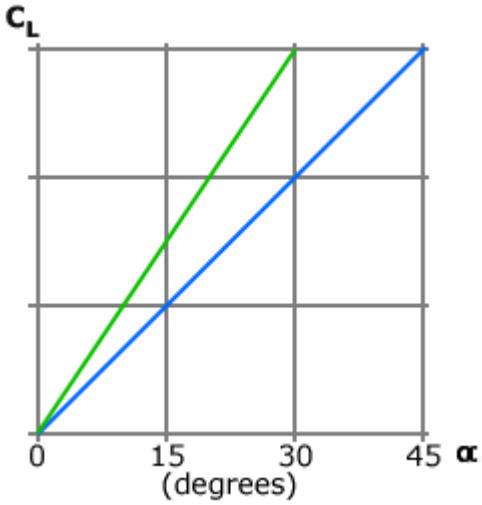
```
stall:    A subelement of a wing (or hstab/vstab/mstab) that specifies the
          stall behavior.
          aoa:   The stall angle (maximum lift) in degrees.  Note that
                 this is relative to the wing, not the fuselage (since
                 the wing may have a non-zero incidence angle).
          width: The "width" of the stall, in degrees.  A high value
                 indicates a gentle stall.  Low values are viscious
                 for a non-twisted wing, but are acceptable for a
                 twisted one (since the whole wing will not stall at
                 the same time).
          peak:  The height of the lift peak, relative to the
                 post-stall secondary lift peak at 45 degrees.
                 Defaults to 1.5.  This one is deep voodoo, and
                 probably doesn't need to change much.  Bug me for an
                 explanation if you're curious.
```

AoA isn't too difficult to understand, but width and peak are rather mysterious. It took me a long time to understand exactly what these two do. While "deep voodoo" is funny, it's also a bit of a disservice. It's hard to make good decisions without knowing what's going on, so let's take a closer look at these attributes.

AOA
I've interpreted this as critical angle of attack, in other words, the angle of attack (alpha) at which lift is greatest before the onset of stall. As the documentation says, Critical AoA is with respect to the wing, not the fuselage or station line. More on that at the end of this guide.

Typical critical stall AoA is 14-17 degrees for common sport aviation wings. Critical alpha for delta or semi-delta winged aircraft will tend to be a bit higher. If you lack any numbers at all, you might be able to guess at critical alpha by looking at photos of the aircraft on approach, estimating the approach AoA, and doubling that for critical AoA.

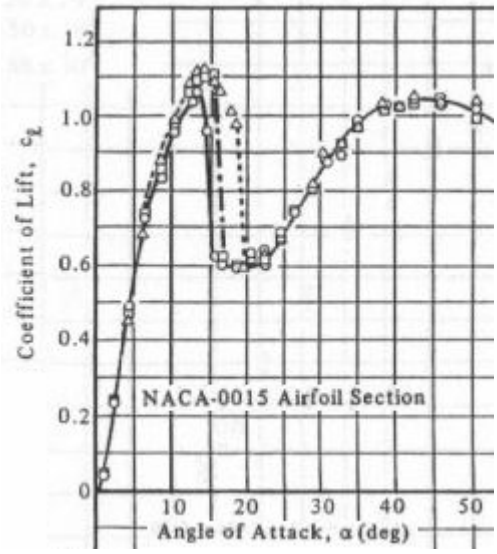
Important: All this is based on my original interpretation of YASim stall AoA, founded on the documentation's "maximum lift" reference and before I began exploring the actual YASim code base. After you read about 'peak' and 'width', see the AOA Revisisted section for another more likely interpretation and my recommendation for setting the stall AoA value.



Peak
YASim calculates lift by assuming lift rises linearly with alpha up to 45 degrees where maximum or peak lift occurs. If graphed as a line with lift on the y axis and alpha on the x axis, the line would have a slope of 1. This is the blue line on the graph at right. This peak lift is actually secondary, post-stall lift. For most airfoils, there are two lift peaks. The first occurs at the critical alpha, followed by a dip and then a rise again to peak lift at 45 degrees. After 45 degrees, lift falls off dramatically as the airfoil approaches 90 degrees to the airstream.

So YASim first determines a lift function for secondary lift up to 45 degrees. To find the pre-stall lift for alphas less than or equal to the critical-AoA, YASim applies a scalar to the secondary lift function, essentially creating a new function by changing the slope of the original. In my illustration, post-stall lift is indicated by the blue line and pre-stall lift is indicated by the green line.

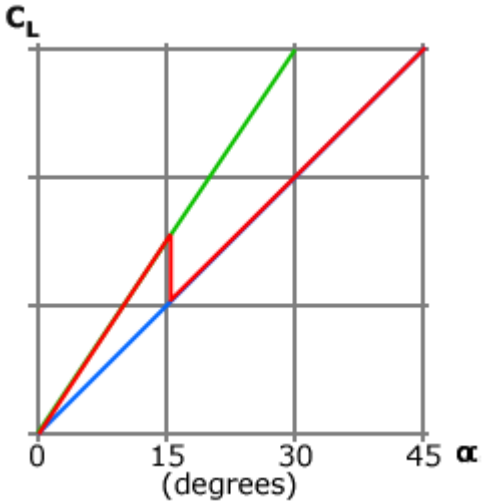
Andy Ross in his simulation notes describes the stall 'peak' value as the ratio of the heights of the initial stall lift peak to the post-stall 45 degree peak. Peak modifies the slope of the lift line in pre-stall conditions. The actual scalar formula is peak/(2 * stall AoA), so the slope scalar is a function of both the peak value and the critical alpha.
Peak defaults to 1.5. Increasing peak above 1.5 will increase the slope of pre-stall lift, yielding greater lift before stall and a greater loss of lift post-stall. Decreasing the 'peak' yields less pre-stall lift. A peak that results in a lift scalar of 1 would match pre-stall lift to post-stall lift, yielding no lift falloff at stall AoA and therefore no stall at all. For conventional horizontal stabilizers with a positive camber, increasing peak tends to require larger elevator flap "lift" values to achieve good approach elevator values, since the stabilizer serves to counter the wing's negative twisting moment.



The YASim default peak value of 1.5 may be high for many applications. The peak coefficient of lift of a flat-plate is 1.0 at 45 degrees, and most airfoils have 45 degree results slightly greater than this. At critical alpha, a C_L of 1.5 is near the top-end, discounting high-lift devices like flaps which can greatly increase lift at the cost of huge drag. Some airfoils like the Clark-Y and the NACA 2412 get max coefficients around 1.6, but many airfoils are lower, often significantly lower.

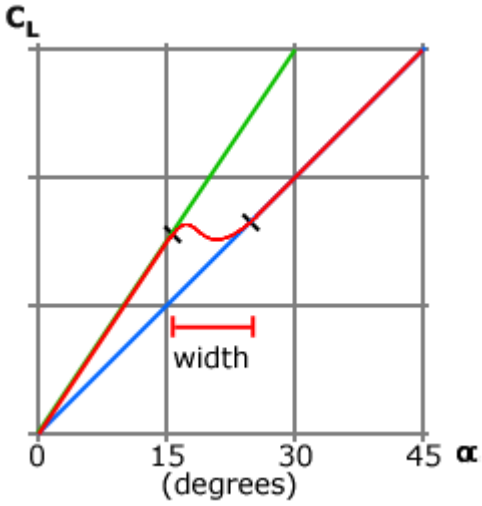
Symmetrical airfoils like those commonly used for stabilizers will often have lower lift coeffcients. For example, an NACA-0015 would probably have a YASim peak value of about 1.05.

Width



If you've followed me so far, you might be wondering what happens to lift after alpha reaches critical AoA. In my illlustration at right, I show an example where critical AoA is 16 degrees. Follow the red line to see what happens as alpha approaches and then exceeds critical AoA. Before stall, lift follows the green line. After the airfoil stalls, lift begins to follow the blue line. Note the sharp discontinuity at critical AoA. If nothing were done about this, the flight effects would make you regret having a big lunch before the flight.

From the illustration you can see that increasing the peak value above 1.5 not only increases pre-stall lift, but also creates a sharper drop in post-stall lift. Decreasing the peak value yields less pre-stall lift and decreases the post-stall drop in lift.



To counter the discontinuity problem, YASim does a cubic interpolation to join the point where the pre-stall lift line at critical alpha meets another point on the post-stall lift line. Since we don't want to join the secondary lift line at the same critical alpha (which would result in the nasty sharp drop we're trying to avoid), YASim offsets the point at which we join the post-stall lift line by some number of degrees. This offset value is the 'width' attribute. Think of this as the width of the stall valley. I've illustrated a case using a critical AoA of 16 and a width of about 8 degrees. The interpolation makes a curve that joins the two points, with a valley between them. (Don't get hung up on the values and shapes of the illustration-- the point is to give an idea of what's happening rather than illustrate a real airfoil.)

Setting a width of 0 would be the same thing as joining the secondary lift line where it is intercepted by critical alpha, essentially the same thing as shown in the second illustration, the nasty drop. Setting a broad width makes for an increasingly shallow drop in lift, or maybe no drop at all, possibly eliminating the effects of stall. The maximum value that has any mathematical sense would then be width=45-criticalAoA, which would essentially create a linear interpolation from critical alpha to maximum lift at 45 degrees. In other words, no valley or drop-off at all. This would be highly unrealistic, at

least for most airfoils and certainly for high-aspect wings.

> Why do we care about this drop in lift at all? If there's a secondary lift peak, why not keep pushing forward in flight until lift picks us up again? The answer is that most planes simply don't have the power to keep muscling forward into higher AoA regions. Also, typical flight surfaces will have stalled in that region, with the corresponding loss of control. Some high-performance aircraft can fly in these high-AoA regimes by means of extreme engine power, wing configuration, specialized control surfaces, and vectored thrust.

What should width be? The only way to know for sure is to have a good airfoil data showing lift plotted against angle of attack. If you don't have that or you have partial data up to critical AoA or maybe a little beyond, you can estimate width by examining the curve at critical AoA where lift begins to fall off. Estimate the radius of that curve and double it for a reasonable width value. For example, if the radius appears to be 4 degrees or so, you might set width to 8. If you lack this data, try using a value of 6 to 10 for wings, and about 4 for stabilizers. A narrow width for a high-aspect wing with little or no twist can give very nasty stalls because the entire wing starts falling into that post-stall valley at the same time. Wing twist mitigates this, since the inboard wing surfaces begin to feel stall effects before the outboard surfaces, spreading the effects of the stall out over time.

AOA Revisited

I've wondered about the effects of the interpolation associated with width. Due to the interpolation, lift could continue to rise at a decreasing rate until the interpolated curve begins to dip down into the valley. The effect would be minimal for narrow widths, but significant at larger widths. In the past I've interpreted YASim stall AoA as critical AoA because the documentation reads "The stall angle (maximum lift)", but I believe what YASim is calling stall AoA is actually the beginning of the stall region where lift begins to decline, not the point of maximum lift before the decline. There are other indications that support this. In a dev list post I once saw Andy state that for typical airfoils the YASim stall AoA should be in the 12-14 degree range. This puzzled me, because most general aviation airfoils have a critical AoA of 15-17 degrees. I've long had a gut feeling that YASim aircraft tend to stall at higher alphas than they should. Running some numbers through the relevant YASim code seems to support this argument.

If this reasoning is valid, and I think it is, then here's my suggestion for setting the YASim stall AoA value: critical AoA - width/4. In other words, research the airfoil's critical AoA, select a reasonable stall width (4-12 degrees), and deduct a quarter of the width from the critical AoA to get your YASim stall AoA value. So if critical AoA is 16 and you are using a width of 8, try a stall AoA of 14. Test fly your results and change from there. This is the method I've been using lately, and I get better results.

Putting It All Together

The stall element is the prime influence on stall behavior in a YASim aircraft, but two other factors from the <wing> element are important: **incidence** and **twist**.

Wing incidence is the angle between the wing's chord line and the fuselage longitudinal line. If the wing chord line was parallel to the longitudinal axis, then angle of attack for the wing is simply the relationship between the airstream and the longitudinal axis. But often the wing is set with a positive incidence-- the chord line is inclined a few degrees from the longitudinal axis. Aerodynamically speaking, the fuselage doesn't matter a whole lot-- you're flying the wing, not the fuselage, which is more like baggage for the wing. But YASim needs a reference axis, which typically runs along the longitudinal axis of the fuselage in most models.

What does this mean for stalls? If an aircraft with a 2 degree wing incidence is flying at an AoA of 4 degrees (as reported by the simulator with respect to the longitudinal axis), then the wing is actually seeing an AoA of 6 degrees. If this same wing begins to stall at 14 degrees, the pilot will begin to notice the effects at 12 degrees.

Twist complicates things a little further. Most wings have some twist, usually a 'washout' that reduces the effective angle of attack of the wing from the chord to the tip. The function is to make sure the inboard wing stalls before the outboard wing where the ailerons live, so the pilot can maintain control through the initial stall regime. If a wing has no twist, the entire span stalls at the same time. A wing with some twist spreads the effects of the stall out over time.

The effect of twist begins at 0 degrees at the root and increases to maximum at the tip. If a wing begins to stall at 14 degrees and has a washout twist of -3 degrees, the wing's root will begin to stall at 14 degrees. But the tip of the wing has its incidence reduced by 3 degrees due to the twist, and it will not begin to stall until the angle of attack reaches 17 degrees. So the effects of the stall are spread out not only by the width of the stall, but also by the twist.

So let's put everything together. Say you have a wing with an incidence of 2 degrees and a twist of -3 degrees. You also have a stall AoA set to 14 degrees. The root will begin to stall at 14 - 2 + 0 = 12 degrees. The tip will begin to stall at 14 - 2 + 3 = 15 degrees.

Twist can have a significant effect on wing-dropping at stall. A plane with little or no twist is likely to drop a wing soon after the onset of stall. Adding significant twist can reduce or eliminate this effect. Be aware that if the plane is pushed further into the stall regime, especially past critical alpha, it may drop a wing anyway. Many planes are notorious for wing-dropping at stall, but others are very gentle and may just drop the nose or begin to mush downwards. Adjusting the twist and width values are your best tools for affecting this behavior.

Be aware that load and balance can also affect stall behaivor. A nose-heavy load can result in insufficient elevator authority to pull the plane back into a stall. Having a CG too far back can make a plane too inclined to stall, and with some planes it can create deep-stall behaviors that are unrecoverable.

Many planes are designed such that in normal flight the aircraft can't be pulled back into a stall because there's not enough elevator authority or thrust or some other consideration. At high angles of attack with the stick full back, the aircraft simply begins to "mush" or wallow downwards rather than experience the sudden loss of lift of a stall. Study flight manuals and pilot reports to learn how your aircraft behaves at high angles of attack.

The YASim attributes give you the ability to change the characteristics of a stall, but they're just numbers. Don't be a slave to them. Set some reasonable values and fly your plane. Tweak your numbers until you get the behavior you want. Hopefully what I've provided above will give you some guidance on how to do that.

Chapter 6
Induced Drag and YASim

By Gary "Buckaroo" Neely

What is Induced Drag?
In flight, air doesn't simply flow over and under the wing, there is also a spanwise movement of air under the wing, around the wing tips, and spilling over to the top of the wing. The lower pressure above the wing causes air under the wing to flow around the wingtip to the top of the wing. The reduced pressure differential causes an almost total loss of lift at the wingtip. The effect gets smaller as measured toward the wing root and is known as the "spanwise pressure gradient". This disrupts the flow over the wing, causing nasty vortices which reduce lift. To compensate, you can increase the angle of attack to gain more lift. But this tilts the wing's lift vector backwards slightly. Though most of the lift force is still upwards, a portion is now generating a force that tugs rearward on the wing. This is induced drag.

Induced drag is proportional to the square of the lift, which also means it's inversely proportional to the square of airspeed. Essentially, the slower you go and the more you increase AoA, the more induced drag you get. If you had a wing of infinite span, you'd get no induced drag because there are no wingtips to allow air to escape around to the topside. We can't build infinitely long wings, but high-aspect wings still produce less induced drag because wingtip vortex effects are reduced on long wings. In effect, induced drag is inversely proportional to aspect ratio (though technically it's more a function of span loading). Washout or twist also plays a role by proportionally generating more lift toward the inboard wing segments. It's been found that elliptical wing configurations are best for reducing induced drag, which is why aircraft like the Spitfire and P-38 had such curvy wing shapes. But elliptical configurations are difficult and expensive to make, so designers compromise by building tapered wings and playing tricks with the wingtips to divert vortices away from the wing.

Note that induced drag is not the same as parasitic drag, which is generated by bodies moving through the airstream and rises geometrically with airspeed. Induced drag is of particular importance to the takeoff, climb, and landing regions of a flight profile, where it can be more than 20% of the total drag forces, and at take off might be as high as 70% for some planes.

Induced Drag and YASim
Back to Flightgear's YASim. The YASim document states: "In general, low aspect wings will generate less induced drag per-AoA than high aspect (glider) wings." Whoever wrote this was probably sleepy. High-aspect wings diminish the effects of induced drag as explained above. A major reason that sailplanes have long wings is because they reduce induced drag.

In the YASim code, induced drag is more guestimate than calculation. The idrag attribute is a multiplier on this internal guestimate, which makes idrag a guestimate factoring a guestimate. This puzzles me, since it doesn't seem unreasonable for YASim to use the standard approximation for induced drag: CDi = Coefficient of Lift^2/(Pi x Aspect Ratio x e). Internally everything is known or could be calculated except e, which estimates how close a wing comes to the theoretical best-case elliptical wing. Typical values for e are between 0.7 and 0.9. The internal YASim guestimate value is 0.7, but there's no indication that this is related to the standard CDi approximation. The internal code documentation reads, "70% is a magic number that sorta kinda seems to match known throttle settings to approach speed." The code seems meant to circumvent more complicated (and realistic) lift calculations.

Incidentally, because YASim does not actually model loss of lift due to tip vortices (at least I've seen nothing relevant in the code base), this implies that YASim's calculations based on wing geometry are on the liberal side, giving full benefits for the entire span at all airspeeds and angles of attack. Since vortices spoil some lift especially during takeoff and landing, this suggests some possible improvements in the code. In the meantime, for this reason alone I'd advise erring on the short side for wing spans.

Using the "idrag" Attribute
The default idrag value is 1, which essentially leaves the internal guestimate unmodified. For most general aviation aircraft with conventional wings, I suggest omitting the idrag attribute, if for no other reason than to simplify your initial FDM attempts.

As currently defined in YASim, decreasing the value of idrag actually /increases/ induced drag. This is the opposite of what you might expect and what the documentation says. Conversely, increasing idrag decreases induced drag. I have verified this by careful testing on many different Flightgear models. YASim induced drag test results
I've not validated the cause yet but I strongly suspect the problem is due to an inadvertant placement of a factor of -1 in the calculations. So until there is a fix:

to reduce induced drag: set idrag > 1
to increase induced drag: set idrag < 1

After you get a good solution from your FDM, if you want to experiment with idrag try these ideas:

If your aircraft has a particularly good method for reducing wingtip vortices, consider a 10-20% reduction of induced drag by setting idrag to 1.1 or 1.2. Most modern wings have some wingtip modification to reduce vortex effects, for without them some of the outboard wing lift is lost. (The original OV-10 Bronco design had no wingtip mods, not even Hoerner wingtips, and the designers later lamented that if they had employed a good wingtip mod, they could have shaved off a considerable portion of the outboard wingspan.)

If your aircraft has significantly tapered wings, and especially if it has elliptical wings (Spitfire, P-38, Lockheed Constellation, etc.), consider increasing the idrag value. (Remember, increasing idrag reduces induced drag.) For taper values greater than 0.5, try a 5% reduction (1.05). For those of 0.5 or less, try 10% (1.1). For elliptical wings, try 15% (1.15).

If your aircraft has long high-aspect wings like most sailplanes, consider reducing induced drag considerably, 20% to 40% or more, depending on your research and flight tests. If your aircraft has short delta wings or no wingtip anti-vortex modifications, consider raising the induced drag by reducing idrag to 0.9 or smaller.
Combine any of the above as seems reasonable. Fly the results. The effects may be subtle and hard to notice but should be more observable at low airspeeds with high angles of attack (typical of takeoff, climb, and approach profiles). When testing, make sure you always fly the same flight profile. The generic autopilot is your friend here, just make sure to give it time to let the numbers settle.

If your aircraft employs a stabilator rather than the conventional stabilizer-elevator, you might consider reducing the idrag of the stabilator hstab. As the wing AoA increases, the stabilizer incidence will decrease to compensate for the wing's tendency to pitch down (at least with a cambered airfoil), therefore the stabilator will be at a lesser AoA with respect to the wing and have less induced drag. YASim can't simulate this due to YASim's exclusive internal control of hstab incidence, but you can reduce the hstab's induced drag as a kludged step in that direction. This is speculation on my part; I'd be interested if anyone experiments with this.

Chapter 7
YASim's Piston-driven Propeller Engines

By Gary "Buckaroo" Neely

YASim's piston-driven propeller engines are easy to set up, but can be difficult to tune. It can give a pretty good result, but you're going to have to work for it through trial-and-error experimentation. Don't be discouraged though, because I'm going to show you what I've learned and how you might solve some problems.

A sample piston-driven propeller engine looks like this:

```
<propeller x="-2.429" y="1.905" z="1.549"
 radius="1.2"
 mass="731"
 moment="33"
 min-rpm="1200"
 max-rpm="2300"
 fine-stop="0.51"
 coarse-stop="4"
 cruise-rpm="2200"
 cruise-power="300"
 cruise-speed="143"
 cruise-alt="5000">
  <actionpt x="-2.260" y="1.905" z="1.563" />
  <dir x="0.9986" y="0.0" z="0.05234" />
  <control-input axis="/controls/engines/engine[0]/propeller-pitch"
control="ADVANCE"/>
  <piston-engine
        eng-rpm="2300"
        eng-power="440"
        displacement="985"
        compression="6"
        supercharger="1"
        turbo-mul="1.3"
        wastegate-mp="36.5">
    <control-input axis="/controls/engines/engine[0]/magnetos" control="MAGNETOS"/>
    <control-input axis="/controls/engines/engine[0]/mixture" control="MIXTURE"/>
    <control-input axis="/controls/engines/engine[0]/throttle" control="THROTTLE"/>
    <control-input axis="/controls/engines/engine[0]/starter" control="STARTER"/>
    <control-input axis="/controls/engines/engine[0]/boost" control="BOOST"/>
  </piston-engine>
</propeller>
```

There's a fair amount there but not all of it is required for all cases. I'll break everything down and talk about each component in detail.

Each propeller-engine unit should have its own propeller definition. You'll have one definition for a single-engine plane, two for a twin, etc.

Propeller Basics
Setting up the fundamentals is easy. Before you start, you'll need to collect some information:
propeller radius in meters
propeller assembly mass in kilograms
propeller assembly + engine weight in pounds
maximum engine RPM

maximum engine power
engine displacement in cubic inches
engine compression ratio
the approximate location of the propeller + engine mass CG
the approximate location of the thrust bearing
the direction of thrust (optional)

Many of these things you can get from commonly available references or the aircraft model. Certification sheets often provide much of this information. Some information can be hard to find, like the weight of the propeller assembly. If you can't find exactly what you need, try to find information for similar components for other aircraft. A good estimate is better than a wild guess or numbers taken from another unrelated YASim FDM.

Any propeller engine definition is likely to have these elements:

```
<propeller ...>
  <actionpt ... />
  <piston-engine ...>
    <control-input axis="/controls/engines/engine[0]/magnetos" control="MAGNETOS"/>
    <control-input axis="/controls/engines/engine[0]/mixture" control="MIXTURE"/>
    <control-input axis="/controls/engines/engine[0]/throttle" control="THROTTLE"/>
    <control-input axis="/controls/engines/engine[0]/starter" control="STARTER"/>
  </piston-engine>
</propeller>
```

There are other possible elements in addition to these, but you're sure to have these basic ones. Each element will also contain a number of possible attributes. Let's look at the elements and their possible attributes in detail.

Propeller Attributes
These are the settings that live inside the <propeller> tag. Some are optional, but most will have to be populated with values. Refer to the above complete example to see how each might be used.

The propeller element must contain the physical location (ie, center of mass) of the combined propeller and engine assembly expressed as x, y, and z coordinate attributes. Remember the YASim coordinate system: X axis is forward, Y is left, and Z is up. Probably you'll have to estimate this location, unless you have really good engineering data for the aircraft. Don't base your estimate on an engine nacelle as the engine assembly often takes up only a fraction of a nacelle. A cut-away diagram showing the actual engine position can be a useful guide.

For mass, enter the weight of the combined propeller and engine assembly in pounds. Again, mass represents the entire engine and propeller assembly, not just the engine. Engine dry weights are easy to find, but finding propeller weights can be take a lot of digging.

For radius, enter the radius of the propeller in meters. Yes, YASim is oddly inconsistent on units.
Once you know the propeller mass and its radius, you can calculate the propeller moment. Moment calculations are approximations, and there are numerous methods available.

I recommend you consider these methods:

plain stick method:	mass(kg) * radius(m)^2 / 3
rods method:	mass(kg) * radius(m)^2 / 3 * (number of blades)
disk method:	mass(kg) * radius(m)^2 / 2

Choose one, or pick several and average the results. For smaller props I use the average of the rods and disk methods. For larger props I simply go with the plain stick. The reason is that large moments can greatly affect how long it takes a propeller to spin up or spin down, and can have too much roll influence. Not being a pilot of any of the aircraft I've modeled, I have no way to know what is realistic. Moment is very fuzzy, since accurate propeller weights are hard to find and even if you find it, the mass distribution is known only to the engineers. I recommend finding a reasonable number (on

the smaller side for larger props) and adjusting the values after flight trials.

Positive moments indicate a clockwise rotation. Moment can be negative, which indicates a propeller moving in a counter-clockwise (when looking forward from the cockpit) rotation. Negative moments are particularly useful for pusher-type configurations. You can also set the contra attribute:

contra="1"
This indicates a paired contra-rotating usually coaxial propeller configuration for the engine, such as used by the Tu-95. Doing this eliminates the engine's contribution to the net gyroscopic moment, asymmetric torque on the aircraft body, and asymmetric slipstream effects.

Cruise Settings
Next you'll need some reference figures for typical cruise settings and power usage:

cruise-speed
cruise-rpm
cruise-power
cruise-alt

I believe these YASim settings are meant to determine where the engine delivers the most efficient use of power. For example, if your propeller has these settings:

cruise-power="235", cruise-rpm="2400", cruise-speed="175" and cruise-alt="8000"

you are saying that it requires 235 HP to make 2400 RPM at 8000' and 175 knots. Since this is establishing a power curve, cruise-rpm and cruise-power should not exceed engine-power and engine-rpm.

Think of these as power settings for long range flight. If you can find typical long-range cruise power settings in a pilot's handbook, try using those figures for your cruise values. Failing that, try power settings in the 60% range at 4000 or 5000 feet. Cruise settings have a great influence on the solver results, so don't get too fixated on the exact numbers because you'll likely tweak these values later based on flight testing.

When using a fixed-pitch propeller, cruise-speed has a lot of influence over the efficiency of that propeller. The higher you set it, the more the fixed-pitch propeller is optimized for high-end cruise speeds, requiring longer take-off runs and slower climb rates. The lower you set it, the propeller delivers great low-speed performance, but might have trouble reaching cruise speeds. I think.

Constant-Speed Propellers
YASim assumes a fixed-pitch propeller by default. If your engine is using a constant-speed propeller, you'll also need to provide these attributes:

max-rpm, the maximum RPM governor range (usually max RPM)
min-rpm, the minimum RPM governor range
fine-stop (optional, defaults to 0.25)
coarse-stop (optional, defaults to 4.0)

If max-rpm is present, YASim assumes you are defining a constant-speed prop. Make sure you also provide min-rpm.
The governor range specified by min and max RPM tells YASim what RPM values the propeller should try to seek. You can often find these in a pilot's handbook or related reference.

Fine-stop and coarse-stop are rather fuzzy YASim values that you can either omit or try to tune. Originally they were a hack to get around fixed internal YASim values. Leave these alone until you're happy with the rest of your engine settings. They won't affect the solver results and it's a waste of time messing with them until you've got good solver results.

Fine-stop defaults to 0.25. Increasing this value will give lower idle RPMs, but if you push it too far it will begin lowering the maximum power RPM, not a good thing. My suggestion is to set this value to 0.25 and note where the engine idles. If idle speed is higher than the desired idle range of the engine (which will likely be the case), begin increasing fine-stop. Observe the new idle results, and also the maximum-power RPM. Keep increasing fine-stop until one of two things occurs: idle speed enters the desired idle RPM range, or maximum power RPM begins to fall short of the maximum desired RPM.

Course-stop defaults to 4. I've yet to find that playing with course-stop alters much-- it doesn't seem able to compensate for high-RPM issues with fine-stop, so I recommend leaving course-stop at 4 or simply omitting it.

If using a fixed-pitch propeller, omit the above four attributes.
Cruise settings become somewhat more difficult to tune for constant-speed propellers, since it's hard to know what the propeller pitch will be at any given time. I suggest trying to tune your settings as a fixed-pitch prop first, and then add settings for constant-speed after you have good fixed-pitch results.

Variable Pitch Propellers
In addition to fixed-pitch and constant-speed, YASim also lets you define a variable-pitch propeller. In this case pitch is set manually by the pilot. To specify a variable-pitch propeller, include the attribute:

manual-pitch="1"

Nothing else is required except a propeller control element. I cover control elements at the end of this tutorial.
Some fixed-pitch propellers may have their pitch altered on the ground but not in flight. Assigning the prop as a YASim variable-pitch propeller with special pre-flight only options to set the pitch may be an interesting solution.

Geared engines
If your engine output is geared, add the following attribute to the propeller element:
gear-ratio
This specifies the output multiplier. For example, if the gear ratio is 16:9, use gear-ratio="0.5625". The default value is 1.0. If using a gear ratio other than the default, all propeller attributes referring to propeller RPM should be factored by the gear ratio. For example, if the governing range is 1200-2700 RPM and the engine uses a 16:9 gearing, set: min-rpm="675" and max-rpm="1519". Factor cruise-rpm by the gear-ratio values as well. Do not do this for the eng-rpm attribute of the <piston-engine> sub-element covered in the next section. That value should be the raw, ungeared maximum engine RPM.

Takeoff Power Settings
These are optional attributes:

takeoff-power
takeoff-rpm

These attributes provide a way to limit thrust at low propeller advance ratios. This is typical of takeoff conditions, where the propeller is spinning fast but the aircraft is moving slow. This could be used to represent a sharp drop in propeller efficiency at low advance ratios, where large portions of the propeller are essentially stalled. As the advance ratio increases, thrust will eventually match rated thrust, depending on conditions.

Though optional, I recommend setting these values, especially for fixed-pitch propellers. If the engine is capable of max-RPM, max-power when sitting at rest on the tarmac, then set these values to the same settings as the engine, or, if the propeller is geared, the same RPM as the maximum geared RPM value. If you have static thrust numbers, try using these for takeoff power and rpm. Lacking those, try values of about 85% engine power and rpm numbers.
If you provide an attribute for takeoff-power, be sure to provide takeoff-rpm. Providing takeoff-rpm alone will be ignored by YASim.

Action Point and Direction Subelements
The propeller element should contain an action point subelement with attributes of x, y, z. This is the point on the aircraft where the thrust is actually applied. I believe it defaults to the engine assembly location, but I am not certain. In any case, you should specify an action point as it will rarely if ever be the assembly's center of mass. What I do is find a diagram of the engine and locate the approximate position of the thrust bearings and use that location as my action point. In typical engines, this will be roughly at the end of the engine block immediately behind the propeller.
The direction subelement is optional. If used, this is a vector to offset the thrust from the usual forward, station-line vector. For example, if thrust were offset 3 degrees up from the station-line, you would include the following subelement in your propeller definition:

<dir x="0.9986" y="0.0" z="0.05234" />

Here, most of the thrust is forward in the x axis, but a small fraction is diverted upward along the z axis.

For most situations you will not need a direction element. It will default to x="1", y="0", z="0".

The Piston-Engine Subelement
The piston-engine subelement is placed within the propeller element. Most of this subelement is easy to populate. Dig out your reference information and fill in the blanks for:

eng-rpm
eng-power
displacement
compression

RPM and power are maximum-power settings. All of these are easy to find even from Wikipedia pages. (Which I normally would not recommend as a reference.) If the engine is normally-aspirated, you're done.

Supercharged and Turbocharged Engines
Many engines have a means for ingesting more air than normally-aspirated engines. This allows higher manifold pressures, enabling the engine to burn more fuel and deliver more power, particularly useful at higher altitudes where low air pressures would result in low fuel-air ratios and corresponding weak power. These engines require a few more attributes and a little tweaking. You'll need to set two more values:

wastegate-mp
turbo-mul

Wastegate-mp is easy. Simply set this to the maximum manifold pressure setting allowed by the engine. The MP result will be scaled up to this value and will never exceed it.

Turbo-mul (turbo-multiplier) is not so easy. The documentation says, "Static pressure will be multiplied by this value to get the manifold pressure." Somewhat informative, but not very helpful. To find this value, I look for a reference in a pilot handbook that indicates the altitude at which maximum MP can be sustained. Through flight trials I then try to find a turbo-mul value that yields that MP up to that altitude and no more. MP should start falling off above that altitude. For example, if a manual says the aircraft should be able to get 48" inHg up to 4000', then I try flights up to 4000' using different turbo-mul settings until I get MP values that maintain 48" and only begin to fall-off after 4000' is exceeded. Make sure you do this with the appropriate boost control settings. See the control section below.
Superchargers and turbochargers are mechanically different means to cram in more air. Superchargers rely on direct power from the engine via gears to drive a compressor. Turbochargers use exhaust gasses to drive a compressor. In YASim, turbocharging is the default method. If the engine is supercharged, supply the following attribute:

supercharger="1"

In YASim, a turbocharged engine experiences some lag in boost pressure but a supercharger responds instantly. A turbocharger effect also works only when the engine is running, possibly affecting engine re-starts. You can change the turbocharger lag with this attribute:

turbo-lag

The documentation reads, "Time lag, in seconds, for 90% of a power change to be reflected in the turbocharger boost pressure." The default is 2 seconds, so you can simply omit this attribute unless you're fooling with the value.

Tweaking Manifold Pressure
YASim's calculates a manifold pressure value that is usually too low for accurate engine MP reporting. You can alter this behavior by setting an optional piston-engine attribute:

min-throttle (defaults to 0.1)

This attribute is a YASim hack that prevents manifold pressure from indicating a total vacuum when the engine is idling. If you know where your manifold pressure typically sits when the engine is idling, then you can play with min-throttle until you get a result close to that value. A value of 0.3 works reasonably for common general aviation engines, typically giving a reading a little over 9" inHg. The min-throttle attribute should not be confused with throttle settings or effects.

Unfortunately, using min-throttle is problematic. If you use it to get an MP value that is reasonable at idle, you will likely find your engine has too much power at idle. The problem is that YASim power is linear with manifold pressure. A real engine has all sorts of friction and pressure effects that cause a non-linear power response. I've had better luck leaving min-throttle at the default (or even lowering it slightly to get better idle RPM numbers) and using my own scripted functions to correct the manifold pressure values seen by instruments.

Control Input Subelements
If you've gotten this far, you're almost done. The last, and perhaps easiest thing to do is map Flightgear properties to serve as control inputs for the engine and propeller. This consists of creating mappings between your selected properties and the desired YASim control axis.

Propeller Controls
The propeller itself needs no control subelements if it is a fixed-pitch propeller since you cannot change the propeller's pitch. If it is a constant-speed propeller, then provide a control input within the propeller element:

```
  <control-input axis="/controls/engines/engine[0]/propeller-rpm"
control="ADVANCE"/>
```

This creates a mapping between the property "/controls/engines/engine[0]/propeller-rpm" and the the YASim ADVANCE control axis, which regulates the propeller governor. The property can be anything you want. This property uses the index 0 for engine 0, identifying a single engine configuration, or the left-most engine for a multi-engine airplane if using the standard engine number convention. If a multi-engined airplane, the next engine to the right would be specified as "/controls/engines/engine[1]/propeller-rpm", etc.

Though "/controls/engines/engine[0]/propeller-pitch" is a commonly used convention for this control in Flightgear, in my opinion "/controls/engines/engine[0]/propeller-rpm" is be a better choice since ADVANCE actually controls the propeller governor. The choice is up to you. The value of this property should range from 0 to 1, with 0 being lowest RPM and 1 indicating highest RPM.

Note that PROP is mentioned in the YASim documentation in place of ADVANCE, but PROP does not appear to be recognized by the YASim parser. I believe this to be a legacy from the original YASim documentation post from 2001.

If the propeller is a variable-pitch (manual) propeller, then use the PROPPITCH control axis instead of ADVANCE:

```
  <control-input axis="/controls/engines/engine[0]/propeller-pitch"
control="PROPPITCH"/>
```

Like ADVANCE, this control expects a value of 0 to 1. Again, the name of the property is up to you to decide. Make sure your property names are correctly bound to the controls or animations that can change their settings.

There is another YASim propeller control axis called PROPFEATHER. Though it is recognized by the YASim parser and the value sets an internal propeller property, as far as I can tell from experience and looking at the YASim code, the control seems to be a stub with no current operational effect.

Engine Controls
Now let's look at the possible controls for the engine itself:

```
  <control-input axis="/controls/engines/engine[0]/magnetos" control="MAGNETOS"/>
  <control-input axis="/controls/engines/engine[0]/mixture" control="MIXTURE"/>
  <control-input axis="/controls/engines/engine[0]/throttle" control="THROTTLE"/>
  <control-input axis="/controls/engines/engine[0]/starter" control="STARTER"/>
  <control-input axis="/controls/engines/engine[0]/boost" control="BOOST"/>
```

These controls should be placed within the piston-engine subelement. Make sure the engine index number matches the correct engine being defined.

MAGNETOS, MIXTURE, THROTTLE and STARTER should all be fairly obvious in function and will most likely always

be present in any propeller piston-engine definition.

A BOOST control is required only if the engine is supercharged or turbocharged. Boost is essentially a multiplier for the manifold pressure benefits of supercharging or turbocharging. For a single-stage, always-engaged supercharger or turbocharger, boost should be set to 1 otherwise you will not get the full effects. For a two-stage supercharger or turbocharger, boost might be initially set to 0.5 or some other fractional value, and to 1 if the second stage is engaged. Remember to define and initialize a boost property for each engine.

A Word on Control Element XML Positioning

The YASim parser does not enforce the position of control elements. You could have an engine control element positioned outside of the piston-engine tags, or even outside of the propeller tags, and YASim will still map the axis to the control correctly. But this is not a good idea. YASim may eventually be revised to use a more strict XML parser, so it would be wise to position controls in their logical container elements. ADVANCE is a propeller control and should be placed within the propeller tags and not within the piston-engine tags. MAGNETOS, MIXTURE, THROTTLE, STARTER and BOOST are engine controls and should live inside the piston-engine tags.

Chapter 8
YASim Turboprop Engines

By Gary "Buckaroo" Neely

A turboprop engine is similar in configuration to a piston powered propeller engine. A propeller can be driven by either a piston engine or a turbine engine. The engine is a sub-element of the propeller element, so many configuration elements are the same. The differences lie in the turbine sub-element. For other propeller configuration details see YASim's Piston-driven Propeller Engines.

A sample turbine sub-element looks like this:

```
<propeller x="-2.429" y="1.905" z="1.549" ... >
  ...
  <control-input axis="/controls/engines/engine[0]/propeller-pitch"
control="ADVANCE"/>
  <turbine-engine
    eng-power="620"
    alt="10000"
    flat-rating="620"
    eng-rpm="2200"
    n2-low-idle="45"
    n2-high-idle="65"
    n2-max="101.5"
    bsfc="0.6"/>
    <control-input axis="/controls/engines/engine[0]/throttle" control="THROTTLE"/>
    <control-input axis="/controls/engines/engine[0]/condition"
control="CONDLEVER"/>
    <control-input axis="/controls/engines/engine[1]/starter" control="STARTER"/>
  </turbine-engine>
</propeller>
```

A propeller configuration may have only one turbine-engine sub-element.

Currently it's OK to have engine control elements live inside the propeller element tags but not inside the turbin-engine tags, because the YASim parser plays loose with XML components. But this is not a good idea. ADVANCE is a propeller control, while the other three are engine controls. At some point someone may revise the parser to act more strictly, so it would be wise to position controls in their logical container elements.

Turboprop Basics

A turboprop uses a relatively small turbine engine to turn a shaft which is used to turn a propeller. The main advantage of a turboprop is its ability to move a lot of air efficiently at low velocities. This is the same effect as a helicopter rotor, turned sideways for thrust rather than lift. A turboprop engine produces an exhaust just like a turbofan engine, but in modern turboprops very little of this exhaust contributes to thrust. The majority of thrust comes from the propeller.

The turbine's easiest work conditions are at sea level on a cool day. There's lots of air to play with and the compressor doesn't have to work very hard to feed air into the combustion chambers. Air pressure decreases as altitude or ambient temperature increases, so in those conditions the compressor has to work harder to cram the same mass of air into the engine. The compressor spins faster, parts get hotter, and air exiting the compressor gets hotter. At some point the turbine's material and RPM (thermodynamic) limits will be reached and if exceeded, the turbine will melt or the blades will fly apart. The engine's safe thermodynamic operating limit is called the thermal rating.
There's another limit on power-- the strength of the propeller gear train and the engine mount that must handle the engine's torque. This is expressed in shaft horsepower (SHP), the maximum power that can be delivered to the propeller. The output of many turbine engines is "flat-rated" to this limit, which is generally much lower than the turbine's thermodynamic limit.

At lower altitudes (or lower OAT) the flat-rated engine is capable of delivering more power than the gearbox can handle and

so power must be limited. But at higher altitudes this extra power means better high-altitude or hot weather takeoff ability and better cruise performance. Engines lacking a flat-rating will see their performance degrade with altitude and temperature, starting at sea level. There is no spare margin for engine performance. The effect of flat-rating is a little like turbocharging on a piston engine, where the wastegate limits engine power up to a certain altitude to prevent exceeding the limits of the engine at lower altitudes.

Armed with a little information about turboprops, let's look at YASim's implementation.

Turbine Engine Attributes
These are the settings that live inside the <turbine-engine> tag.

eng-power
eng-rpm
alt
flat-rating
n2-low-idle (default 50)
n2-high-idle (default 70)
n2-max (default 100)
bsfc

The N2 settings are optional, but other settings will have to be populated with values.

The YASim README document does not list n2-low-idle, n2-high-idle, or n2-max. Instead it lists only min-n2 and max-n2. Unfortunately (and unsurprisingly) the README document does not agree with the YASim codebase. Use the attributes shown above. YASim will ignore settings for min-n2 or max-n2.

Note that there is no spool-time attribute. Unlike YASim jet engines, turbine spool-time change appears to be fixed.

Engine Power Attributes
The power attribues are all related:

eng-power
alt
flat-rating

There are no defaults, so be sure to provide reasonable values. Internally the calculations are really about torque but the external references are SHP (shaft horsepower).

In YASim, maximum turbine engine power output is calculated using engine power (eng-power) and altitude (alt). This is essentially setting the engine's thermal rating. So if you give YASim the values eng-power="800" and alt="7200", you are saying that the engine will produce 800 SHP at 7200' at standard temperatures at a given RPM. What happens at other altitudes and temperatures? At altitudes higher than 7200', engine power at the same RPM will decrease. At altitudes below 7200', engine power will increase. At sea level, the output would be a whopping 1700 SHP. This is where flat-rating comes in.

The flat-rating attribute is used to place a cap on power. A YASim turbine engine designed to produce 800 SHP at 7200' will deliver almost 1700 SHP at sea level. That kind of power would destroy the gearbox. If the engine is flat-rated at 620 SHP, then power will always be restricted to no more than 620 SHP. Above 7200', calculated power will begin to fall from 800 SHP. But for several thousand feet we're still getting the maximum allowed ouput before there's any power falloff. At altitudes a bit greater than 9000' the power will have fallen to 620 SHP, at which time true power output will begin to decline as the aircraft gains more altitude.

Flat-ratings are not hard to find for common turboprop engines. The thermal ratings are more difficult to find or estimate. One method is to simply set engine power to the flat rating and experiment with altitude settings via flight tests until you get the desired cruise and ceiling performance.

Propeller RPM and Turboprop Engines
If you have experience with YASim piston engines, you may be expecting to set a gear ratio for turboprops. YASim turboprop configuration doesn't work the same way. When using a turboprop in YASim, set the turbine RPM setting to the

maximum RPM of the propeller:

eng-rpm

Don't set a gear ratio for the propeller. This isn't intuitive, especially if you're used to YASim piston engines, so be careful. There is no default, so always set the eng-rpm value. There is no RPM setting for the engine's internal N2 spool.

A real turbine engine's internal RPM is geared down to produce an RPM suitable for a propeller. For example, a turbine might be turning at 4500 RPM, but the geared propeller might be turning at 2200 RPM. Governed turboprop propeller speeds are typically 1500 to 2500 RPM so set your YASim propeller governor speeds appropriately.

N2 Idle Settings
YASim allows you to specify the settings for the N2:

n2-low-idle (default 50)
n2-high-idle (default 70)
n2-max (default 100)

These values set the upper and lower bounds of the output shaft speed. The use of these values is explained below in more detail in the section related to the condition lever.

Be sure to note that the YASim README document incorrectly lists the attributes min-n2 and max-n2 rather than the above attributes. At some point someone likely updated YASim and failed to update the README. Do not use min-n2 or max-n2.

Brake Specific Fuel Consumption
BSFC is the rate of fuel consumption divided by the power produced, a measure of fuel efficiency expressed in lbs/hr per horsepower.

bsfc (defaults to 0.5)

This value is essentially a multiplier to fuel flow (after conversion to kg/s/watt). The calculation is simple:
fuel flow = torque * RPM * bsfc

You can use BSFC to adjust the fuel consumption, or you can modify the Nasal script that handles fuel consumption for YASim aircraft.

The default value is rather low. A better default for common turboprop engines would be 0.55 to 0.6. The PT6A models commonly used in aircraft like the King Air and Twin Otter typically have a bsfc of about 0.6.

Turbine Engine Control Input Subelements
Turboprop engine controls can differ considerably from piston engines. The "throttle" of a turboprop is often called "power" and is commonly combined with other functions like reverse thrust. Some turboprops have a propeller RPM control, while others combine this function with another lever. Turbine engines will often have an additional lever called the "idle condition lever". The purpose of this lever requires a little background information.

Turbines have internal compressors or "spools" which spin at 30,000 RPM or higher. They require time to spool-up or spool-down in response to power setting changes. An aircraft on approach might require a sudden increase in power in the event of a go-around. A turbine that is idling will take precious seconds to spool-up to full power. To address this, many turbines have a "ground idle" condition setting and a "flight idle" setting. Ground idle is simply a lower N2 RPM than flight idle. If flight idle were used on the ground, the aircraft would have too much power when at rest or maneuvering at taxi speeds, accelerating too rapidly. So the pilot selects ground idle when at rest or taxiing, and flight idle for other conditions.

The propeller "ADVANCE" control is the same for both piston and turbine engines, being a function of the propeller element, not the turbine-engine sub-element.

YASim turbine engines use the "THROTTLE" control similar to piston-driven configurations. A real turbine engine has no carburetor or throttle, but the result is effectively the same, especially with a little scripted aid. The developer may have to customize portions of the throttle/power control range to simulate other features such as reverse thrust.

The Condition Control
Turbine engines have an additional control, "CONDLEVER".

```
<control-input axis="/controls/engines/engine[0]/condition" control="CONDLEVER"/>
```

The condition control has two functions in YASim. When set to less than 0.001, the control operates as a fuel cut-off switch. Fuel is available at any setting equal to or greater than 0.001. Note that this operates in addition to the usual fuel state property. Both must be valid for the engine to be considered "running".

The second function serves to regulate the minimum idle setting of the N2 spool. Idle speeds are determined by n2-low-idle, n2-high-idle and the value of the CONDITION control. The minimum idle speed is calculated this way:

min idle = n2-low-idle + (n2-high-idle - n2-low-idle) * CONDITION
where condition can range from 0 to 1. A value of 0.001 or less is idle cut-off, so in in practice CONDITION will range from some >0.001 to 1.
What does this mean? If CONDITION is near 0, minimum idle will be very close to n2-low-idle. If CONDITION is 1, minimum idle will be n2-high-idle. If CONDITION is 0.5, minimum idle will be half way between n2-low-idle and n2-high-idle.

The value for n2-max determines the maximum speed the spool will attain, so speeds will always vary between the minimum idle and n2-max.

Note that it isn't required that a turbine engine have a condition control. If the control is not defined, condition defaults to 1. If the control is used, condition should always be set to 1 for approach and cruise settings.

YASim's somewhat limited turbine control options won't cover all the possible real turbine control options, but with some clever scripting many turbine controls can be adequately simulated.

Reverse Thrust
YASim currently has no ability to reverse the thrust of a propeller. Until that functionality is added, it's still possible to simulate reverse thrust by using YASim thrusters. Syd Adam's has a good example of this in his DHC-6 Twin Otter model.

Residual Thrust
The turboprop engine is built around a gas turbine which can produce some thrust from its exhaust output depending on the design. In older turboprops this "residual" thrust could be considerable. For example, one model of the Bristol Proteus engine of the early 1950's produced 2500 SHP at sea level plus 820 lbs of jet thrust. For many modern turboprop configurations, residual thrust velocity is low and contributes little or no useable thrust. Many modern engines are very efficient and have already reclaimed most of the power from the turbine output.

For most modern situations you can simply ignore the problem of residual thrust. Some turboprop engines will list output in terms of ESHP (Equivalent Shaft Horsepower) which sums the propeller shaft horsepower and the thrust horsepower produced by the turbine exhaust. Using ESHP for engine output isn't totally realistic since the portion of thrust from exhaust has no torque component, but if ESHP is not terribly greater than SHP then it's not unreasonable to use it for a power rating.

If residual thrust is a significant component and the thrust factor is known, it's possible to use a simple YASim "thruster" element in tandem with a turboprop propeller element. This will require some additional scripting to tie the two control systems together.

Engine Startup/Shutdown
A YASim turboprop engine is "running" if both of these conditions are satisfied:

a) fuel is available (engines/engine[n]/out-of-fuel = false)
b) condition is >= 0.001 (controls/engines/engine[n]/condition)

As long as the engine has fuel and condition is not in the cutoff position, the YASim turbine engine is running. Toggle the fuel supply off, and the engine stops running. Toggle the fuel supply on, and the engine is suddenly running again. The condition control has a similar effect. If the engine isn't using a condition control, then the condition property will be ignored (internally it defaults to 1) and only the out-of-fuel property will matter, much like a YASim jet engine.

YASim gives no provisions for anything fancier than this, but it's enough to provide the tools for scripted controls to provide a realistic startup. A simplified startup procedure for a PT-6 engine goes something like this:

engage the electric starter (which should also engage the igniters)
when N1 reaches 15%, move condition lever from fuel cut-off to low idle
inter-turbine temperature (ITT) should start to rise, RPM should start to rise
watch for N1 to rise, at 50%, disengage the starter

This can all be handled by Nasal scripts. YASim does not model the N1 spool, so your scripts will have to do that. Fortunately this is not difficult using the Nasal "interpolate" function. In the above procedure, the only interaction with YASim parameters is in step 2, where after N1 reaches 15% you would set both out-of-fuel and condition properties for the engine.

For an example of a scripted startup procedure, see my MD-81 model. Though it is a JT8D jet engine simulation, the Nasal methods used can be simplified and adapted for a turbine startup.

Chapter 9
YASim's Jet Engines

By Gary "Buckaroo" Neely

The good news is that Flightgear's YASim jet engine simulation is much easier to set up and somewhat easier to tune than piston-driven propeller engines. The internal code for a jet engine is simpler than propeller engines, so it's easier to follow and understand what each element is doing. Simplicity, like complexity, comes with a cost, and I'll address some of the issues and suggest some work-arounds at the end of this guide.

A typical YASim jet engine configuration looks like this:

```
<jet
 x="-33.757" y="2.7" z="0.687"
 mass="4567"
 thrust="20694"
        exhaust-speed="1555"
 egt="625"
 tsfc="0.518"
 n1-idle="30"
 n2-idle="60"
 n1-max="99.2"
 n2-max="102.5"
 epr="2.12">
  <actionpt x="-35.0" y="2.7" z="0.687"/>
  <control-input axis="/controls/engines/engine[0]/throttle-fdm"
control="THROTTLE"/>
  <control-input axis="/controls/engines/engine[0]/reverser"
control="REVERSE_THRUST"/>
  <control-output prop="/surface-positions/reverser-norm[0]"
control="REVERSE_THRUST"/>
  <control-speed control="REVERSE_THRUST" transition-time="2"/>
</jet>
```

There are a few other interesting options like afterburners which I'll cover in this guide.

Like propeller engines, each jet engine should have its own engine definition. If your aircraft has two jet engines, you'll have two <jet> definitions, with their engine ID numbers corrresponding to the order of definition.

Jet Basics
Before configuring your YASim jet engines, you'll need to gather some data about the engine:

maximum dry or military thrust
maximum wet thrust (thrust with full afterburner) (optional)
exhaust gas temperature at takeoff
engine pressure ratio at takeoff
rotor speed at idle (n1 idle)
maximum rotor speed (n1 max)
compressor speed at idle (n2 idle)
maximum compressor speed (n2 max)
engine weight in pounds
the approximate location of the engine mass
the approximate location where thrust is applied
the direction of thrust (optional)

Most of these details are not too hard to find from engine specifications. I've had good luck finding data from published service or overhaul records. If you can't find exactly what you need, try to find information for similar components for other aircraft. A good estimate is better than a wild guess or numbers taken from another unrelated YASim FDM.

Any jet engine definition is likely to have these elements:

```
<jet ...>
  <actionpt ... />
  <control-input ... control="THROTTLE"/>
</jet>
```

Many jet engines will have a thrust-reversing mechanism, and military jet engines will often have afterburner controls as well. Let's look at the jet engine elements and their possible attributes and controls in detail.

The Jet Element
Most attributes for jet engines are optional. All you're required to populate are the engine location, the mass, and thrust. In practice, however, most should be populated with values. Refer to the above example for a typical case.
The jet element must contain the physical location (ie, center of mass) of the engine expressed as x, y, and z coordinate attributes. Remember the YASim coordinate system: X axis is forward, Y is left, and Z is up. Probably you'll have to estimate this location, unless you have really good engineering data for the aircraft. Lacking hard data, try to find a cut-away diagram showing the actual engine position within the fuselage or a nacelle and make a guestimate for the center of mass.

For mass, enter the weight of the jet engine in pounds.

Let's look at the other engine attributes.

Thrust Ratings
Thrust may be declared using a single thrust attribute. If the engine has afterburner capability, a second attribute is also used. If thrust can be reversed, a third attribute can be used:

thrust
afterburner (optional, defaults to 0)
reverse (optional, defaults to 0.2)

For simple thrust, find the engine's maximum rated no-afterburner thrust and enter that for thrust attribute. If the engine uses an afterburner, find the total afterburner or wet thrust and enter that for the afterburner attribute. This number will of course be considerably greater than the no-afterburner dry or military thrust.

In YASim, afterburner thrust creates a power ratio that is added to normal thrust after being scaled by a reheat control: thrust = thrust * (1 + reheat * (afterburner/thrust - 1)), where the reheat control varies from 0 to 1. The reheat control is a throttle on the amount of fuel being dumped into the exhaust stream. I discuss engine controls at the end of this guide.

As you might expect, using afterburners radically affects fuel consumption, consuming additional fuel at the rate of 3.5 times the afterburner/thrust factor at full reheat. Unfortunately the fuel consumption rate is set in the YASim code and can't be tuned.

Reverse is a scalar applied to thrust when reversers are engaged. For example, using the default 0.2 setting, reverse thrust is 0.2 * forward thrust. If you know how much reverse thrust the engine can produce, you can find a good reverse value by calculating reverse_thrust/thrust.

Exhaust Speed
Exhaust speed is the air leaving the turbine. (Not the bypass air leaving the fan in turbofan engines.)

exhuast-speed (defaults to 1555)

Exhaust speed may be difficult to find. The default value is 1555 knots (800m/s) and represents a somewhat early turbojet. High-bypass turbofan engines will have significantly lower exhaust speeds. As a rough guess, try numbers around 1000

knots (500m/s) for turbofans. For modern afterburning supersonic military engines, exhaust will likely be higher than the default. A high-performance pure turbojet might be very much higher. It's likely that you will have to experiment to get the behavior you want.

In YASim, maximum thrust is scaled by (1 - V0/Ve), where V0 is the velocity of the airstream and Ve is the exhaust velocity. So if the aircraft is moving at 250 kts and Ve is 800 kts, then maximum thrust would be factored by 0.6875. Essentially, thrust diminishes with airspeed. If you wanted to do Mach 2.2 at FL300, you'd be doing about 1300 kts, so Ve would have to greatly exceed 1300 or you'd never be able to accelerate to that speed in a reasonable time let alone maintain that speed.

Force equals mass times acceleration, so in a real turbofan, the lowering of exhaust speed is offset by the larger mass of air moved by the fan. Potential airspeeds are lower for a turbofan (assuming no afterburning) but efficiency at those speeds is significantly improved, yielding lower tsfc numbers and higher bonuses for airline CEO's. In my opinion, YASim doesn't represent this well. If you lower the exhaust speed from the default value, the difference between the solver's lift and drag results will increase. This is YASim compensating for what it believes is a weaker engine. This is not necessarily reasonable if the engine has a greater bypass ratio to offset the lower turbine exhaust speed. Thrust is thrust, but I think YASim could benefit from a bypass ratio attribute for turbofans, compensating for lesser exhaust speeds with the effect peaking at some specified optimum airspeed.

N1 and N2
Most jet engines have a low-pressure rotor (often called a spool) and a high-pressure rotor. Part of the N1 spool is the big fan you can often see at the front of a jet engine. The components of the N2 spool are internal. Rather than use RPM, their speeds are represented by a value compared against a nominal 100% maximum-power rating. These values are N1 and N2 respectively. Depending on the engine manufacturer, N1 is often used as the rating for thrust (others use EPR). N2 is often used on engine start-up to indicate when the engine is sufficiently spooled-up to being introducing fuel. In YASim these are represented by:

n1-idle (defaults to 55)
n2-idle (defaults to 73)
n1-max (defaults to 102)
n2-max (defaults to 103)

Information about N1 and N2 can be somewhat hard to find, particularly minimums, unless you have a flight manual. If an attribute is not provided, YASim will default to the values shown. Real values vary considerably from the defaults, especially for more modern jet engines and particularly with regard to minimums. If you are planning to do a comprehensive engine simulation that includes start-up procedures, you really need to find the true values.

YASim thrust is tied to the N1 value. Some simpler engines use a single rotor and a few use three rotors. YASim doesn't directly allow you to simulate these cases, though you could fake it with some Nasal scripting, as long as it's understood that thrust is tied to N1's value.

Engine Pressure Ratio
Engine pressure ratio (EPR, pronounced ee-per, I think) is the total pressure ratio across the engine-- the pressure of air leaving the engine is greater than the pressure of air ingested.

epr (defaults to 3)

In YASim, a glance at the code suggests EPR is doing something important and interesting, but closer inspection suggests it's used only for calculating an instrument response for EPR; it appears that it could be factored out of other uses, which makes me wonder if the code could be simplified. In any case, it has no significant effect on solver results.

Try to find your engine's maximum takeoff EPR and use that value. If you lack a manual, you might be able to find this by examining instrument readouts, particularly if the aircraft uses analog gauges. Whatever you do, don't set EPR to 1, as this gets into the division by 0 realm and makes the solver freak-out. Besides, a value of 1 in real life would mean your engine isn't doing anything but passing ingested air. There's a bad joke lurking there, but let's move on to EGT.

Exhaust Gas Temperature
This is the exhaust gas temperature in degrees Kelvin, measured at takeoff.

egt (defaults to 1050)

It's unfortunate that the documentation doesn't make the units clear, since YASim outputs the resulting EGT in degrees Fahrenheit. The default value is 1050 K (777 C).

EGT has no effect on the solver. It's simply a maximum temperature that is linearly interpreted from the atmosphere temperature as EPR rises. It is used only for instrumentation output: egt-degf.

Thrust-Specific Fuel Consumption
TSFC is the ratio of the engine fuel mass flow rate to the amount of thrust produced, in other words, a measure of efficiency.

tsfc (defaults to 0.8)

In YASim, TSFC is used to compute fuel flow. Usually you can pull this number from a reference. I believe the units are (Kg/Hr)/N, as internally YASim converts thrust to Newtons and fuel flow in kg/s. As long as thrust and fuel are measured in the same units, it shouldn't matter since TSFC is a ratio. Modern turbofans will have TSFC values much lower than the default.

This number is not terribly important to YASim. Use it to make a rough estimate at fuel consumption. Try to get real fuel consumption data from flight profiles and conduct test flights to compare the simulated numbers against real data. You can use TSFC to adjust the fuel consumption, or you can modify the Nasal script that handles fuel consumption for YASim aircraft. For more information about the latter, see the section of my guide that concerns YASim fuel tanks.

Spool Time
Spool-time controls the response of the engine to throttle changes. Essentially, it's an inertia delay while the rotors accellerate or decellerate.

spool-time (defaults to 4)

The documentation reads, "Time, in seconds, for the engine to respond to 90% of a commanded power setting." So this is not the maximum time required to go from idle to maximum thrust, it is the time time to respond to /any/ power setting change. In my opinion this is not an optimal attribute. It should be the time required to spool-up from idle to maximum thrust. Older jet engines responded fairly quickly to power changes at cruise settings, but going to or from idle could take a while. Modern engines apparently have much better response times, but there may still be differences, and YASim spool-time won't let you represent that. Do not set spool-time to 0.

Action Point and Direction Subelements
The jet element should contain an action point subelement with attributes of x, y, z:

```
<actionpt x="-35.0" y="2.7" z="0.687"/>
```

This is the point on the aircraft where the thrust is actually applied. I believe it defaults to the engine location, but I am not certain. In any case, you should specify an action point as it will likely not be at the exact center of mass. What I do is find a diagram of the engine and locate the approximate position of the thrust bearings and use that location as my action point.

The direction subelement is optional. If used, this is a vector to offset the thrust from the usual forward, station-line vector. For example, if thrust were offset 3 degrees up from the station-line, you would include the following subelement in your propeller definition:

<dir x="0.9986" y="0.0" z="0.05234" />

Here, most of the thrust is forward in the x axis, but a small fraction is diverted upward along the z axis.
Both action point and direction subelements should be located between the <jet></jet> tags. For most situations you will not need a direction element. It will default to x="1", y="0", z="0".

Control Subelements

Control-input elements allow you to map a Flightgear property to a YASim value, allowing you to channel user input to be used for that YASim value. In YASim, the channel is called an "axis". Control-ouput elements allow you to create a Flightgear property that contains the current value of a YASim control channel or axis. This is particularly helpful for animations. You are also able to control the rate-of-change of a control channel through use of the control-speed element. Control subelements should be located between the <jet></jet> tags.

Now let's look at the possible controls for a YASim jet engine:

```
  <control-input axis="/controls/engines/engine[0]/throttle" control="THROTTLE"/>
  <control-input axis="/controls/engines/engine[0]/reverser"
control="REVERSE_THRUST"/>
  <control-input axis="/controls/engines/engine[0]/reheat" control="REHEAT"/>
  <control-input axis="/controls/engines/engine[0]/vector-angle" control="VECTOR"/>
```

You will always have a throttle control, but the others are optional. All controls should be placed between the <jet></jet> tags. Make sure the engine index number matches the correct engine being defined. Axis property names are yours to define. Make sure they are the same property names used in your animations and keyboard controls, etc.
The THROTTLE and REHEAT channels allow an input range of 0 to 1. REVERSE_THRUST input should be 0 or 1 (boolean false/true). VECTOR allows an input of +/-degrees.

I said earlier that control-outputs are useful for animations. Say you want to display the position of thrust reversers on the aft end of an engine nacelle based on the status of reverse thrust. First, you would map the control channel to a Flightgear property

```
  <control-output prop="/surface-positions/reverser-norm[0]"
control="REVERSE_THRUST"/>
```

In this case, the current value of the YASim REVERSE_THRUST channel is copied to the output property, which YASim creates for you. Your animations can then reference this property, controlling the location of the model object based on the property value.

To make this animation more fluid rather than an on-off position, you could add a control-speed element to the control:

```
  <control-speed control="REVERSE_THRUST" transition-time="2"/>
```

In this case, it would take 2 seconds for the REVERSE_THRUST channel to transition from 0 to 1. Note that it's likely the reverser will be functionally engaged given any input value other than 0.

YASim Jet Issues
Thrust and Standard Temperatures
Engine thrust values are rated at certain test environment conditions, usually sea level and some standard temperature. The rating temperature may not be the same as YASim's standard atmosphere temperature of 25C. If the standards are different, you could ignore it, especially if the difference is small, but you could also calculate the difference in thrust. Various online tools and sites are helpful in doing this. I did this for my MD-81 model, but it was a while back and I can't recall which sites and tools I used to calculate the adjusted thrust value. Yeah, OK, I'm useless. I'll revise this with notes if I ever remember how I did it.

Engine Idling
In my experience, YASim jet engines idle much too low, often sitting at near 0 thrust on idle. This also means fuel flow at idle drops to almost nothing which is terribly unrealistic given that airlines go to great lengths to reduce engine use on the ground to save on fuel expenses-- starting engines only when necessary, sometimes operating on one engine until ready for takeoff, etc.

There are at least two methods to deal with this. One method is to set an input range on the throttle control. For example:

```
  <control-input axis="/controls/engines/engine[0]/throttle" control="THROTTLE"
src0="0" src1="1" dst0="0.07" dst1="1"/>
```

This maps the input range of 0 to 1 to a THROTTLE input range of 0.07 to 1 on the YASim side. So when the throttle appears to be sitting on 0 or full retarded, it is actually sending a value of 0.07, delivering a small but significant amount of thrust, and perhaps more importantly, a realistic idle fuel consumption. The value of 0.07 is used in my MD-81 model and was calculated using figures from the flight manual.

A more flexible but complex method to handle this is to maintain two properties, one for the YASim throttle control property, and another for user control and animation feedback. A Nasal listener function then watches for changes to the user/animation property. Upon a change, the listener performs an interpolation on the property similar to the control-input interpolation above, and copies the result to the property used to control the YASim THROTTLE channel. This is the method used in the MD-81 model. It has the advantage of being subject to other factors if desired, and the interpolation range can be changed during the simulation.

Engine Startup/Shutdown
YASim jet engines have no real startup/shutdown methods. As long as they have fuel, they are running. The controlling parameter is the engine "out-of-fuel" boolean value:

/engines/engine[n]/out-of-fuel

The engine will always run as long as this value is false. Toggle the fuel supply off, and the engine stops running. Toggle the fuel supply on, and the engine is suddenly running again. YASim has no provisions for anything fancier than that.

Obviously this is not optimal, but while jet engine startup is not usually complex it does have a few procedures that would be difficult for YASim to simulate sufficiently broadly for most applications. A typical (simplified) startup procedure might go something like this:

provide a startup power source (electric or pneumatic)
engage necessary fuel pumps
set the igniter switch
enable the starter
watch for N2 to rise to some required % range
open fuel valve
watch for N2 to rise to another required % range
hope nothing nasty happens
disengage the starter

In addition to these things, it's necessary to provide instrumenation results for engine properties that YASim isn't providing when the engine isn't running. For example, until the engine is running, YASim will not report N1 and N2 at any values other than 0%, but in reality, once a starter is engaged N1 and especially N2 should be reporting non-0% values. This means the developer has to intercede and provide meaningful values during the pre-ignition startup stages.

For an example of a complete and by-the-book startup procedure and implementation, feel free to have a look at my MD-81 model.

Jet Engines as Black Boxes
Since YASim's jet engine simulation is somewhat simplistic, it can lead to arguments over whether YASim can realistically simulate this or that. For example, the SR-71 Blackbird at cruise speeds actually gets most of its thrust from inlet air that bypasses the turbine and is burned, effectively a ramjet. The turbine itself serves as a kind of pump to keep the process going. YASim has no provisions to simulate this. Nor does it balance fan bypass airmass vs. exhaust speeds, or the effects of afterburners on exhaust speeds and temperatures. These issues can get ugly fast.
The solution, at least for speeds that are transsonic or less, is to treat the engine as a black box that makes thrust at a certain point and vector on the airframe and not worry about how it works. Tune the performance and behavior as best you can, and don't worry about the mechanics.

Supersonic Issues
A YASim FDM can be made to be realistic or at least reasonable at speeds up to and including the transsonic region. YASim does not appear to have been designed to account for supersonic regions. Nothing says you can't design an FDM that flies in those realms, but the simulation becomes less ideal. I address YASim and supersonic issues elsewhere in my guide.

Chapter 10
YASim Landing Gear and Surface Interaction

By Gary "Buckaroo" Neely

The <gear> element is the primary contact point for ground or water surfaces and the only contact points which can be disabled (retracted). Most aircraft will define three landing gear elements similar to the following example:

```
<gear x="-0.796" y="0" z="-1.070"
 dfric="0.5"
 sfric="0.6"
 spring="0.9"
 damp="0.8"
 compression="0.08"
        castering="1">
  <control-input axis="/controls/gear/nosewheel-lock" control="CASTERING" src0="0"
src1="1" dst0="1" dst1="0" />
  <control-input axis="/controls/gear/gear-down" control="EXTEND"/>
  <control-output control="EXTEND" prop="/gear/gear[0]/position-norm"/>
  <control-speed control="EXTEND" transition-time="12"/>
</gear>

<gear x="-3.750" y="0.967" z="-1.015"
 dfric="0.5"
 sfric="0.6"
 spring="0.8"
 damp="0.8"
 compression="0.22">
  <control-input axis="/controls/gear/brake-left" control="BRAKE"/>
  <control-input axis="/controls/gear/brake-parking" control="BRAKE" split="true"/>
  <control-input axis="/controls/gear/gear-down" control="EXTEND"/>
  <control-output control="EXTEND" prop="/gear/gear[1]/position-norm"/>
  <control-speed control="EXTEND" transition-time="12"/>
</gear>

<gear x="-3.750" y="-0.967" z="-1.015"
 dfric="0.5"
 sfric="0.6"
 spring="0.8"
 damp="0.8"
 compression="0.22">
  <control-input axis="/controls/gear/brake-right" control="BRAKE"/>
  <control-input axis="/controls/gear/brake-parking" control="BRAKE" split="true"/>
  <control-input axis="/controls/gear/gear-down" control="EXTEND"/>
  <control-output control="EXTEND" prop="/gear/gear[2]/position-norm"/>
  <control-speed control="EXTEND" transition-time="12"/>
</gear>
```

This is a tricycle arrangement with gear elements for the nose wheel, left main, and right main. Let's look at how to use the gear element in a YASim aircraft.

Gear Element Basics

Each gear element consists of a contact point, a number of optional attributes, and a number of optional controls and outputs. A retractable gear element will generally be defined as follows:

```
<gear x="-0.796" y="0" z="-1.070"
```

```
        ...optional attributes...
        >
  <control-input ... />
  <control-output ... />
  <control-speed ... />
</gear>
```

A non-retractable element is even simpler:

```
<gear x="-0.796" y="0" z="-1.070"
        ...optional attributes...
        >
  <control-input ... />
</gear>
```

A gear element should always have an x, y and z attribute which defines the contact point. Note that there are two other potential contact types other than gear which I'll discuss shortly. Many gear elements will also have control subelements for assigning functions like braking. Those with retractable gear will have a control for extending the gear, a control output for showing the position of the retractable gear, and a control-speed to regulate the time required to extend or retract the gear.

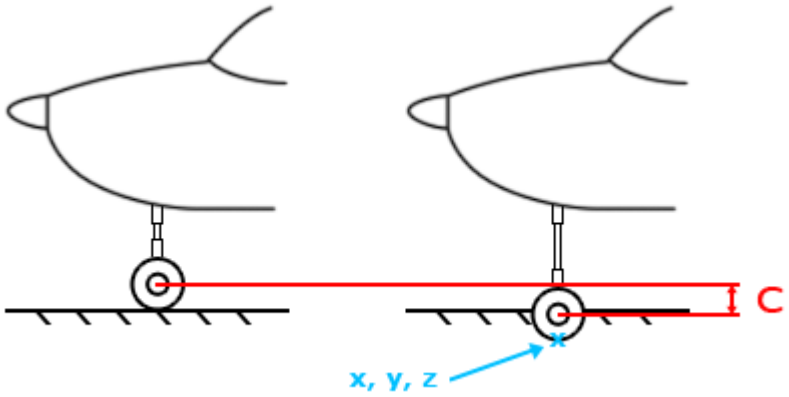
You can declare your gear in any order, but order determines the sequence of properties in the property tree. For example, in the above tricycle example, the nose gear is gear "0", the left main is "1", and the right "2". You may have as many gear elements as you need. Some tricycle arrangements also feature a tail skid, while some gliders have small wheels attached to the wing tips. These can all be configured using <gear> elements.

Gear Element Attributes
Landing gear have quite a few interesting attributes:

x, y, z
compression
upx, upy, upz
spring
damp
initial-load
sfric
dfric
castering
skid
on-solid
on-water
ignored-by-solver
spring-factor-not-planing
speed-planing
reduce-friction-by-extension

If this looks complicated, then fear not, because most are optional and many are for special applications. We'll look at each one in detail.

The Surface Contact: x, y, z
The x, y, and z attributes define the gear's surface contact location relative to the aircraft's reference datum. YASim calls this "the location of the fully-extended gear tip". Translation: this is the contact point of the gear with the hypothetical ground when the gear has no compression, in other words, as if the gear is supporting no weight. This is not the location of the contact point when the aircraft is sitting on the ground, as that location assumes weight on the gear and some degree of compression. Since YASim calculates compression at all times that the landing gear is deployed, it needs to know the fully-extended contact point as a base reference. You should always provide values for these three attributes.



The illustration shows a ground contact point for a typical tricycle nose gear. Note that the gear is fully extended. "C" refers to the compression length, which I'll discuss next.

When considering tires, I often assume some tire deformation when contacting the ground, something like 5-10% of the tire radius. In other words, my contact point won't be the outermost circumference of the tire, but a point slightly inside the tire. This will look more natural when the aircraft is sitting on the tarmac, particularly for heavier aircraft.
Gear aren't the only elements that can interact with a ground or water surface. There are two other possible types of contact points:

fuselage elements
flight surface tips

Ground contact with a fuselage contact point will result in a crashed condition. This is what triggers a "crash" if you land gear-up in a retractable gear aircraft. The midpoint of the tip chord of a flight surface also serves as a contact point but will not trigger a crashed condition.

Compression
The compression attribute sets the maximum distance in meters that the gear will compress under load.
compression (default 1)

Be sure to define this value since the default of 1 meter is much too large for general aviation aircraft. The above image shows how the compression length (shown as "C") is determined. Different gear mechanics will have different means of absorbing shock. All YASim cares about is how the load translates the point of contact with the ground (or water in the case of floats).

The compression attribute has a potentially significant effect on drag. See Gear Drag Effects below.
When gear is under load, YASim defaults to a compression orientation that is straight up along the z axis. You can define three optional attributes to change this behavior:

upx, upy, upz (default 0, 0, 1)

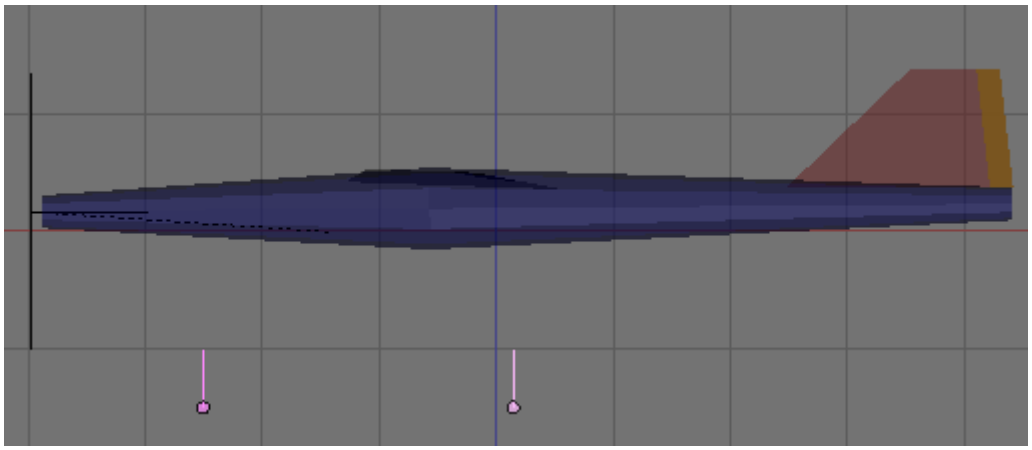
These values change the strut compression orientation. The default values of 0, 0, 1 indicate a compression that is vertical. For most configurations the default will work fine, especially if the angle of compression varies only slightly from the vertical. Where this is not the case, the direction of compression can be expressed as a vector.
Let's look at an example of compression orientation. A tricycle landing gear configuration is shown below in order of nose, left main, and right main. In the first case, they all use the default values of upx="0" upy="0" upz="1".

```
<gear x="2.5" y="0" z="-1.5"
  compression="0.5"
  ...
</gear>

<gear x="-0.15" y="1.2" z="-1.5"
  compression="0.5"
```

```
  ...
</gear>

<gear x="-0.15" y="-1.2" z="-1.5"
  compression="0.5"
  ...
</gear>
```



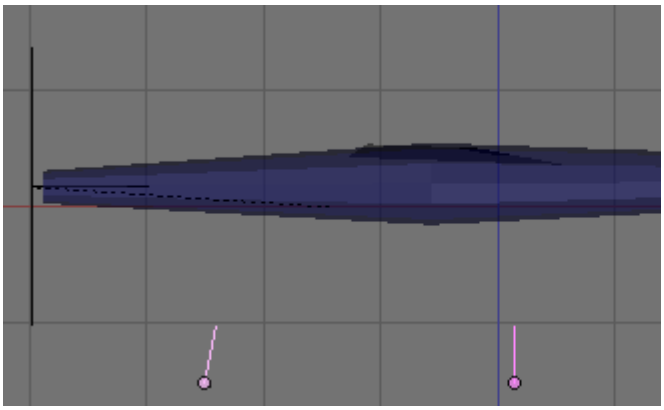
The illustration shows the gear compression orientation as vertical. All compressions are straight upwards.

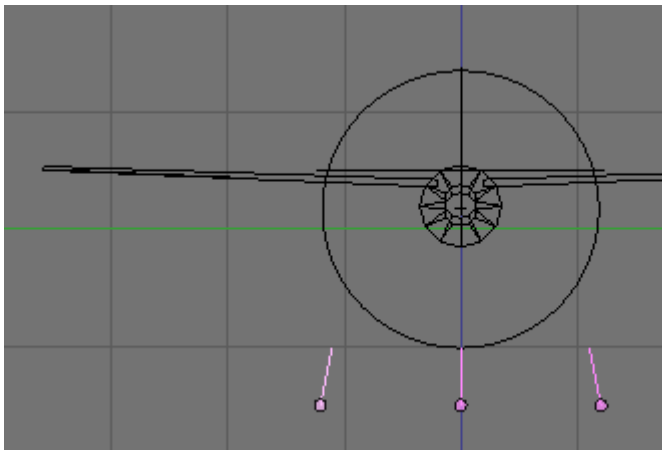
The same set of gear again, with changes to strut compression orientation:

```
<gear x="2.5" y="0" z="-1.5"
  compression="0.5"
  upx="-0.2" upy="0" upz="1"
  ...
</gear>

<gear x="-0.15" y="1.2" z="-1.5"
  compression="0.5"
  upx="0" upy="-0.2" upz="1"
  ...
</gear>

<gear x="-0.15" y="-1.2" z="-1.5"
  compression="0.5"
  upx="0" upy="0.2" upz="1"
  ...
</gear>
```





The gear compression orientation has changed. The nose gear now compresses with a slightly aft angle, and the main gear

now have a negative camber (the compression is aimed inward toward the fuselage).

If you use a configuration other than the default, be careful to define all three components of the vector, even if the value is 0. Otherwise YASim will either use the defaults or might generate an error.

Spring Behavior
Gear compression is governed by a physical spring simulation. YASim initially configures this for you, giving you two multipliers to alter the default behavior:

spring (default 1)
damp (default 1)

Internally YASim generates a spring constant such that at full compression the gear will support 10 times the mass of the aircraft. The "spring" attribute is an optional multiplier for this constant. Setting the spring value to greater than 1 will allow the gear to support a greater mass at full compression, making the gear more stiff, in other words, less compression for the same weight. Setting the value less than 1 will decrease the stiffness of the gear, allowing more compression at the same weight.

Below left is an Edgley Optica with main gear spring attributes set to 1. Below right is the Optica with the spring attributes set to 0.6.

Notice that with the smaller spring values, the Optica sits lower on the main gear. The struts are considerably more compressed.

Compression length appears to be a target value rather than a hard limit. It is possible to exceed the compression length if enough force is applied to the gear, such as a hard landing or even the simple weight of the aircraft at rest. The spring attribute has a significant effect this. With the Optica, a spring setting of 0.5 will result in the aircraft sagging to the ground-- the non-gear contact points will then intersect the ground, resulting in a "crashed" condition. Some real-world compression simulations will be hard to achieve. For example, the MD-80 series has a very short nose gear compression. Getting a good compression reaction in YASim for such a large aircraft is difficult without exceeding the compression length, even with fairly mild nose gear touch-downs. Unfortunately there is currently no way to set a hard stop on gear compression, though actually hitting the stop would probably damage most real aircraft.

The spring damping coefficient is another automatically generated internal value. The damp attribute is an optional scalar for that value. A perfect, frictionless spring system will continually oscillate back and forth with the same amplitude if nothing is acting to resist the movement. The damping effect provides this resistance.
The damping effect is easily observed by selecting a general aviation Flightgear model with a tricyle gear arrangement. Allow the aircraft to begin moving on the tarmac and then brake quickly. Observe the effects using an external view. With damping set to the default of 1, the nose of the aircraft plunges down and then recovers, with little or no oscillation. Change the nose gear damping to a very small value like 0.1 and repeat the test. This time the nose plunges down and recovers but then repeats the behavior with decreasing amplitudes until the oscillation stops entirely.

For most aircraft the default damp value will usually work fine, particularly with heavier models. Lighter aircraft might have more "bounce" in some gear effects, so reducing the damp value from 1 may be appropriate. If you have personal experience or can observe a video of the aircraft, you might have some insight and can adjust damping accordingly. Rarely will you have to increase damp much beyond 1, and the YASim document warns that high values may make the algorithm unstable. I've never experienced problems with too much "bounce" in a gear except with jitter issues, which I believe to be a different problem.

A third seldom-used spring-related attribute is available to alter spring behavior:

initial-load (default 0)

This value can give a spring an initial load, allowing the use of a lower spring multiplier. It may help to think of tires as absorbing some of the shock that would otherwise be absorbed by the gear. The value is supposed to be expressed in multiples of the compression. Presumably this means that if your compression length is 0.2, then initial-load could be set to 0.2, 0.4, 0.6, etc.

I don't fully understand this attribute and have not used it. I believe some developers have used it to help reduce gear jitter issues. Small values can sometimes introduce jitter, and large values can make the gear too "mushy", such that the aircraft may sink through the ground if too much force is applied, even simple braking. I've not given the attribute much study.

The initial-load attribute was added by Maik Justus.

Static and Dynamic Friction
Static and dynamic friction coefficients influence the gear's resistance to movement when on a surface. YASim generates static and dynamic value based on the current simgear terrain material friction values, but you have two gear attributes which allow you to modify the generated values:

sfric (default 0.8)
dfric (default 0.7)

The static coefficient influences the force necessary to overcome the initial resistance to movement. The dynamic (or kinetic) coefficient influences the force necessary to keep the gear moving once in motion. A larger friction value decreases the stopping distance of the vehicle. The static value should be larger than the dynamic value. The static friction resistance force will rise with the applied force until it reaches a threshold, where the friction force falls off quickly to the dynamic range and remains relatively constant regardless of the applied force.

The defaults work reasonably well for configurations using wheels on concrete or asphalt, though I often play with values ranging from 0.6 to 0.8. Dynamic friction will be only slightly less than static for wheels. For skids, the values will likely have a greater separation-- try changing the default dfric to 0.45.

The Castering Wheel
Most wheel gears are fixed, free-rolling forward or backwards but resisting sideways skidding. A castering gear allows relatively free skidding movement left or right once static friction has been overcome. Many tail-draggers feature free-castering tail wheels. Internally, the ground velocity of a gear consists of a rolling and a skidding component. Setting castering to 1 greatly magnifies the influence of the skidding component.
castering (default 0)

Castering should normally be off (the default) for main gear and for steerable nose or tail wheels. Learn more about controlling castering in the Control Input section.

As a side note, I think YASim would benefit from allowing the developer to place an optional scalar on the skidding component of castering so that the developer has some control over the effect in addition to the friction values.

Skids
Not all landing gear are wheels. Many helicopters employ simple skids for gear. You can declare a gear to be a skid by setting the following attribute to 1:

skid (default 0)

Internally "skid" is assigned to a variable called "brake". This appears to be a multiplier to the rolling and skidding friction values of the base ground material and applied to the steered (vs. skidding) component of gear interaction with a surface. The default value of 0 grants the base ground friction factor no influence for steering, while ground rolling friction gets maximum influence. A value of 1 grants maximum influence to ground friction factor, and no influence for ground rolling friction. The base factors are derived from the simgear scene material.

Limiting Applicable Surfaces for a Gear
Gear can interact with ground surfaces, water, or both. The following two attributes determine the surface where the gear is applicable:

on-solid (default 1)
on-water (default 0)

For example, if a gear is set for use only on a solid surface then it will have no effect on water. The gear will be ignored and the aircraft will likely contact the water with some other contact point, either another gear element, a fuselage segment, or a wingtip.

If using conventional ground landing gear, there is no need to include these attributes as the defaults will work well. When using a gear element as a float or flying boat contact point, you should provide these attributes using the following values:

on-solid="0"
on-water="1"

Other applications are skid-floats for helicopters, where you might have a contact point that is valid on water and on land.

The "Ignored-By-Solver" Attribute
YASim normally uses all defined gear elements for calculating the spring constants which determine the base physical modeling of spring behavior for each gear. This is fine for aircraft having only the usual set of three ground-based landing gear. But some gear definitions are best left out of the calculations for spring constants. YASim has a special attribute for doing just that:

ignored-by-solver (default 0)

This attribute tells YASim to ignore a gear when calculating spring constant weightings. If an aircraft does not have water-based gear, then you can usually use the default or simply not include this attribute. If an aircraft has both terrestrial landing gear and water-based gear, then include ignored-by-solver="1" for each water-based gear element. If you do not, YASim will include the water-based gear elements when determing the spring constants for the terrestrial gear.

Consider using ignored-by-solver for any gear that is not used for significant distribution of the aircraft's weight when at rest under normal circumstances. This might include situations like small wheels at the wingtips of some sail planes.

Important: do not set all gear elements to be ignored-by-solver. The command-line YASim solver may not gripe at you, but Flightgear will spew nasty errors that may not be easily traced to this setting.

Flying Boats and Seaplanes
YASim features a serviceable simulation of interaction with a water surface that can be used for flying boats and seaplanes. To use a gear element for this purpose, be sure to set these attributes:

on-solid="0"
on-water="1"
ignored-by-solver="1"

Using gear as contact points to simulate flying boat hulls or seaplane floats requires different thinking than land-based gear. It helps to know something about takeoffs using boat hulls or floats. There are numerous good online pilot references that discuss the procedures, but here is a very brief summary of the basics.

For floats on water, the problem of friction is more complex than ground-based operations. Floating drag is governed by frictional resistance, wave-making resistance, the shape of the hull, and other factors. This resistance or drag varies with the square of the speed of the object through the water. Resistance also depends on whether the object is ploughing though the water or hydroplaning on the surface, also known as being "on the step". For a flying boat or a seaplane it's important to start planing as soon as possible, otherwise the drag of plowing through the water will skyrocket with the increase in speed.

For many real-world seaplanes, water drag peaks around 27 knots. At that point the aircraft begins planing and water drag begins to fall off. This depends greatly on the aicraft design, the loading of the aircraft, the condition of the hull, and the

surface conditions of the water. Depending on these conditions and altitude, the aircraft might not be able to get airborne.

Back to YASim. There are three additional attributes applicable to water-based gear. When used wisely and with lots of testing and tweaking, these attributes can give a reasonable simulation of flying boats or seaplane takeoffs.

spring-factor-not-planing (default 1)
speed-planing (default 0)
reduce-friction-by-extension (default 0)

The first two attributes, spring-factor-not-planing and speed-planing, influence the spring constant of a float in water. The spring-factor-not-planing attribute is a multiplier to the spring coefficient when below planing speed, defined by speed-planing. The aircraft will sit lower in the water when not planing. When planing, the water begins acting "harder" and the vehicle will sit higher on the surface. Anyone who has tried water skiing has likely noticed this effect, which is particularly annoying when one loses control and smacks into the water. Trust me on this one.

Experiment for best results with your flying boat or seaplane. Keep in mind that planing usually begins somewhere around 27-30 knots, and the spring effects below planing are likely to be softer than when hydroplaning, so your spring-factor-not-planing attribute will likely be some fractional value rather than a number greater than 1.

The third attribute, reduce-friction-by-extension, can reduce friction based on gear compression. Using the default of 0, friction does not vary with compression. Using a value of 1, friction is linearly reduced as compression varies from fully compressed to no compression. With no compression on the gear, friction is then 0. Using a value of 0.5, friction varies with compression from full at max compression to half at no compression.

This is a key value for simulating the transition from ploughing through the water to hydroplaning. As the aircraft moves faster, more lift is generated, taking some load off the gear and reducing compression. When using reduce-friction-by-extension, this begins to reduce the friction imposed by the water surface. This is important when considering the squared velocity effects of water drag. The actual value used must be found by experimentation while knowing the water takeoff characteristics of your aircraft.

The reduce-friction-by-extension attribute is especially useful when used with the speed-planing/spring-factor-not-planing attributes, since a stiffer spring means less compression and therefore less friction when using reduce-friction-by-extension.

If the reduce-friction-by-extension value is greater than 1, the point of zero friction is achieved before the gear is fully extended. The algorithm doesn't clamp the friction to 0 at extensions beyond this point-- in other words, the friction multiplier can become a negative value. Since this is meant to be used with floats, this is likely intentional. In practice, I've found it to be sometimes necessary to use a value greater than 1, otherwise the aircraft may not get over the water-drag hump and start planing.

Gear used as floats have some special concerns regarding drag and YASim solutions. Be sure to read the sections below concerning landing gear and drag.

Control Input Subelements
Control inputs are used to map an input property to a gear control. For example, if you want to use the rudder to control the steering of a nose or tail wheel, you can assign a rudder control property to the gear control responsible for steering.

```
  <control-input axis="/controls/flight/rudder" control="STEER"/>
  <control-input axis="/controls/flight/rudder" control="STEER"
square="true"/>
```

Control inputs have some interesting optional attributes:

invert
src0/src1/dst0/dst1
square

For aircraft having a steerable tail wheel, the "invert" attribute is useful to correct for the rudder control input, which is itself normally inverted due to how vertical flight surfaces are configured.

```
  <control-input axis="/controls/flight/rudder" control="STEER"
invert="true"/>
```

You will not need to do this for a nose wheel.

The optional attributes "src0", "src1", "dst0" and "dst1" allow you to alter the range of control inputs. Here's an example:

```
  <control-input axis="/controls/flight/rudder" control="STEER" src0="-1"
src1="1" dst0="-0.5" dst1="0.5"/>
```

In this case, we're taking the possible input range of -1 to 1 and interpolating it to export an output range of -0.5 to 0.5. In other words, if the rudder control value is 1, the gear steering control will receive 0.5. These attributes can be used as an alternate method of inverting a control:

```
  <control-input axis="/controls/flight/rudder" control="STEER" src0="-1"
src1="1" dst0="1" dst1="-1"/>
```

The "square" attribute provides a method of giving more precise control in the small ranges of steering by squaring the value of the control input.

```
  <control-input axis="/controls/flight/rudder" control="STEER"
square="true"/>
```

In this example, a rudder control input of 1 creates an geer steering control input of 1, but a rudder input of 0.1 gives a steering input of 0.01. This can be useful for aircraft with longer or faster takeoff runs where too much authority on the nose wheel can cause the plane to track wild.

Applying Brakes
The previous examples have mapped the rudder control to a gear, but of course it's also useful to map braking controls to gears. This is used for differential braking and implementing the parking brake.

```
  <control-input axis="/controls/gear/brake-left" control="BRAKE"/>
  <control-input axis="/controls/gear/brake-parking" control="BRAKE"/>
```

This example shows the left-brake and parking break controls being mapped to a gear's braking function. So if either control is applied, the gear's brake function will be activated. Normally you would want to do this for the left main and right main gear (substituting "brake-right" for the right main gear).

Gear Extension

```
  <control-input axis="/controls/gear/gear-down" control="EXTEND"/>
```

This subelement maps a controlling property to the extended condition of the gear. A value of 0 indicates the gear is retracted, 1 indicates fully deployed.

A gear element's contact point is active only when it is in the fully extended position. In other words, the associated property must have a value of 1. Any value less than 1 will disable the contact points. If the aircraft is sitting on the ground and you initiate a gear retraction, the gear contact points are lost and the aircraft will fall to the ground. The fuselage contact points will intersect with the terrain and a "crashed" state will be set.

Locking the Tail Wheel
Many tail-draggers have a tail wheel that is either free-castering or can be locked into a straight, in-line position for takeoff and landing. YASim provies a control subelement to toggle the castering feature of gear elements.

```
  <control-input axis="/controls/gear/tailwheel-lock" control="CASTERING"/>
```

Castering is enabled if this property is true or 1. Since a property like "tailwheel-lock" is usually understood to be locked

(non-castering) if set to 1, you might wish to invert the control using either of the following methods:

```
  <control-input axis="/controls/gear/tailwheel-lock" control="CASTERING"
src0="0" src1="1" dst0="1" dst1="0"/>
  <control-input axis="/controls/gear/tailwheel-lock" control="CASTERING"
invert="1"/>
```

Control Output Subelements
Control outputs are the reverse of inputs. They allow you to export YASim's internal control position values as Flightgear properties.

```
<control-output control="EXTEND" prop="/gear/gear[0]/position-norm"/>
```

Here we're exporting YASim's value for the current position of a gear extension to the property "/gear/gear[0]/position-norm", where "norm" refers to normalized to the common range of 0 to 1.

Assigning an output property to a control allows you to export what would otherwise be an internal FDM value. In this case, we can export the value of a gear position for use with gear animations. The gear's control input would not work well for this, since it will likely be set to 0 or 1 (up or down), and won't account for the interpolation of the gear from one position to the other. Exporting the actual interpolated position allows the developer to not only animate the gear realistically, but also set up proper cockpit displays to indicate whether the gear is up, in transition, or down and locked.

The Control-Speed Element
Retractable landing gear require time to deploy or retract. You can control this time by using the control-speed sub element.

```
<control-speed control="EXTEND" transition-time="7"/>
```

Here we've assigned a 7 second interval for the landing gear to move from from the fully retracted position to the fully extended position. The value of the property associated with the EXTEND control will be interpolated between 0 and 1 over the transistion time in seconds. The default for transition-time is 0, so it's best to provide a value for this subelement.

The YASim document tells us that this sub-element is semi-deprecated by script-based solutions, but for most applications this function is good enough. It uses less resources and is much easier to implement than script-based solutions. It should be your first choice. But it is a simple, linear transistion. If you have a more complex situation, don't hesitate to pre-process control inputs using Nasal scripting rather than use the control-speed sub-element.

Issues Related to Landing Gear and YASim

Gear Elements and Drag
YASim's implementation of landing gear drag is, for lack of a better word, lousy. It's also bizarre, in at least one respect and without knowing the design logic.

For each gear element, YASim creates a simple "surface" to generate drag effects, using much the same constructs as flight surfaces. The drag factor of the "surface" is based on compression:

gear drag factor = (compression * 3)^2

Compression lengths of a small fraction of a meter yield little or no drag. These situations are typical of an airborne aircraft with gear deployed. But if the compression length gets relatively large, drag can skyrocket and have a major effect on the YASim solution. Be wary of this if your gear elements have long compression lengths or you are using floats.

When implementing floats or boat hulls, some developers use fairly long compression lengths to help simulate a large body floating in water. When the gear is fully extended, as it will be when the gear is unloaded (i.e., when flying), drag is at

maximum. With a long compression length this can greatly impact solutions and flight results, far beyond what may be intended.

The consequences of this are not serious for small seaplanes or flying boats where compression values are likely to be a fraction of a meter. But for large aircraft the developer might create large compression lengths approaching a meter or more to allow the hull or float to sit deeper in the water when not planing. Unfortunately this creates huge drag values for the solver and can severely alter an otherwise good solution. Currently the only way I know to deal with this is to keep the compression lengths relatively short and play with other gear attributes.

It's unclear why compression was chosen as a basis for a drag factor. Probably this was a hack solution to supply gear drag. It would have been wiser to allow the FDM developer to set a multiplier for gear drag and a location, since the center of gear drag effects will not be the ground contact point. Addressing the gear drag problem is high on my list if I ever get around to learning how to compile Flightgear.

There are several ways to add gear drag effects, and none of them are particularly great. Some developers create artificial flight surfaces to represent the landing gear struts, pants and wheels. This can work, especially for fixed gear. For retractable gear it generates additional flight surfaces which will always be in existance requiring extra calculations even when not deployed, a generally undesirable thing. Another method is to employ the "weight" element to represent landing gear drag. This element can simulate droppable external stores and features a "size" attribute that generates drag. To be honest, I've never used either method. I understand the cries of how significant gear drag can be, but most YASim aircraft have so many other FDM issues and implausibilities that gear drag is fairly far down the list of concerns.

"Jitters" and Sliding

YASim is notorious for situations where an aircraft sitting on the ground slowly moves or slides over the tarmac, usually with the prevaling wind. If the aircraft has wheels, the wheels can often be observed to be "jittering"-- the wheels spinning backward and forward in quick, slight motions. This effect is sometimes more pronounced in tricycle gear configurations or helicopters using skids. The motion can be numerically observed by opening a property window and inspecting the values for the gear, especially the compression and rollspeed properties.

I don't know what causes this and haven't researched it, though it is a significant YASim annoyance with some aircraft. The effect is rooted in the relationship between the ground surface and the YASim ground contact algorithms and seems to be some sort of insufficiently damped feedback between ground and gear contact, possibly causing a small, undamped micro-bouncing. The effect is often more pronounced on tricycle gear aircraft where the CG is fairly close to the main gear. The problem is influenced by aircraft weight, as it is more commonly seen with lighter aircraft than large planes like airliners.

I don't have a cure-all solution. Sometimes moving the CG or adjusting mass balance can mitigate ground contact jittering. Sometimes changing the gear friction values can reduce jittering. Sometimes adjusting spring values can help the issue. The problem with these solutions is such changes may move these attributes away from desired ranges, compromising the flight model.

Chapter 11
Inertia Tensors and YASim

By Gary "Buckaroo" Neely

The YASim command-line solver gives you some additional information:

```
Inertia tensor : 2863.719, -0.000, -4.189
      [kg*m^2]   -0.000, 1006.259, 0.000
   Origo at CG   -4.189, 0.000, 4071.845
```

This may be confusing as it looks like three separate 3-element vectors: Inertia tensor, [kg*m^2], and Origo at CG, none of which make any sense. It should be displayed this way:

```
Inertia tensor about the CG expressed in kg*m^2:
  2863.719  -0.000   -4.189
  -0.000  1006.259  0.000
  -4.189  0.000   4071.845
```

This makes things more clear to those not familiar with inertia tensors.

What is an inertia tensor? Most people interested in flight simulation are familiar with this basic physics equation:

F = m * a

Force equals mass times acceleration. If you want to accelerate a mass in some direction, you need to give it a push. A similar idea applies to rotational acceleration-- spinning an object about its axis. To rotate an object about an axis, we need to know the rate we wish to accelerate it about each of the three axes x, y, and z, and we need to know something about the distribution of the object's mass. Then we can calculate the required force to do it, known in this situation as torque.

An inertia tensor matrix looks like this:

```
Ixx Ixy Ixz
Iyx Iyy Iyz
Izx Izy Izz
```

The inertia tensor tells us something about mass distribution. So:

T = I * dv

where T is torque, I is our tensor matrix, and dv is the change in angular velocity. Look familiar? Force equals mass times acceleration.

Let's examine the simple, user-friendly case:

```
Ixx 0 0
0 Iyy 0
0 0 Izz
```

These diagonal elements are called the "moments of inertia". Here we consider only the diagonal elements and assume the others are 0. This means the object is mass-symmetrical about each axis, i.e., weights are distributed uniformly along each axis, like a see-saw with one child at each end, each child having the same weight and being the same distance from the fulcrum. If the diagonal elements all had the same value, the object's mass could be modeled as a simple sphere. If we apply a torque about the x axis, the object will acquire an acceleration about the x axis, and only the x axis. Seems reasonable, right? This works the same way with the y and the z axes. In an aircraft, mass distribution along the y axis (wing tip to wing tip) will be more-or-less symmetrical if we assume wing fuel tanks are filled to the same level.

Moments of inertia can be thought of as resistances to angular acceleration. They are always positive values. The greater the value, the further the mass is distributed from the system's center or mathematical origin (aka "origo"), which is the aircraft CG in our case, yielding greater resistance to angular acceleration. Small values represent a collective mass that is distributed closer to the system's center. The values (in units of Kg*m^2) are relative to the overall mass of the aircraft, so a plane with a large mass like an airliner will need respectively larger torques to give the same angular accelleration as a smaller plane.

Think of an ice skater spinning in place as having angular momentum about the z axis, her feet to her head. In aircraft terms, the skater is yawing like crazy. As the skater pulls her arms inward, she decreases the size of her Izz element and spins faster. If she moves her arms outwards, Izz increases and she spins slower. This is known as conservation of angular momentum.

So for an aircraft in our coordinate system, Ixx becomes resitance to changes in roll, Iyy is resistance to changes in pitch, and Izz is resistance to changes in yaw.

The non-diagonal elements are called the "products of inertia". They are always symmetrical about the diagonal. These guys are weird and not easily explained. The basic idea is that a matrix with values for the non-diagonal elements implies the object has an assymmetrical mass, a kind of dynamic imbalance. In an aircraft, this commonly occurs when the mass distribution forward of the CG is very different from the mass distribution aft of the CG. Things start to get messy.

The effects of products of inertia are more difficult to understand. Ixy is the inertia against rotation around the y axis when a rotation about the x axis is applied. In this case a torque about the x axis (a roll maneuver) might introduce an acceleration about not only the x axis, but also the y or the z or both y and z. Viewed another way, if an object is rotating about the x axis and Ixx is the inertia against rotation about the x axis, then a non-zero value for Ixy is the additional contribution to inertia against rotation about the y axis. If we want to minimize these effects, we want the products of inertia to be small, since values of zero eliminate their effect.

YASim initially distributes mass according to the volumes or area of fuselage bodies and surface elements, along with a few other elements like engine weights. You can see how the mass was distributed in the solver's output of the inertia tensor. For example:

```
Inertia tensor : 2863.719, 0.000, -4.189
      [kg*m^2]   0.000, 1006.259, 0.000
   Origo at CG   -4.189, 0.000, 4071.845
```

This represents a case of a light aircraft with a total mass of about a thousand pounds. Here, 2863 is the resistance to rolling, 1006 is the resistance to pitching, and 4071 is the resistance to yawing. The products of inertia are all zero or small.

YASim makes an attempt at mass distribution based on the fuselage and surface elements, but it's a simplistic guess and often needs work beyond basic mass balancing for CG placement. How do you adjust this distribution? This is where you get fancy with YASim "ballast" elements. Recall that ballast does not add weight, it forces YASim to re-distribute existing weight. We use ballast to move the CG to where we want it. But now that we know something about inertia tensors we can do some more interesting stuff.

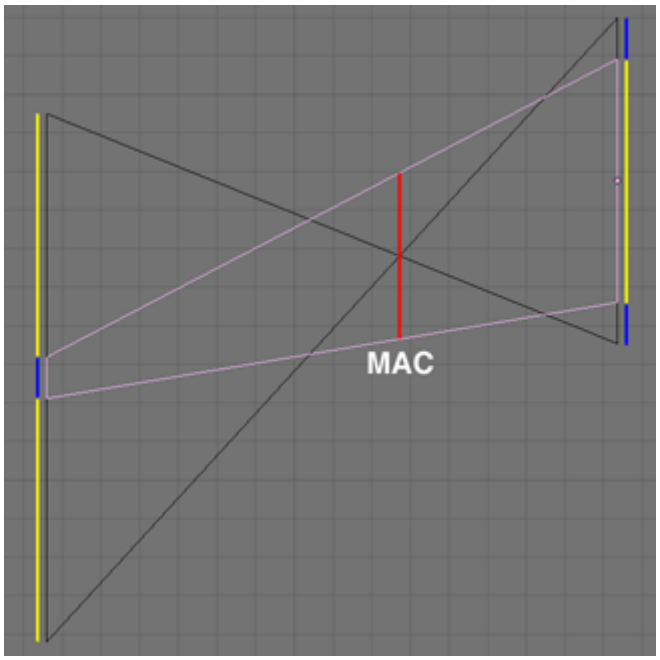
Most aircraft will fly perfectly fine with the default distribution after CG is moved to a good location. All you need is to place a ballast in the nose, or a negative ballast in the tail. But maybe you want to lower the center of gravity, perhaps in the case of a high-wing aircraft where the default CG is too high, making a plane with a narrow wheel-base tippy on landings. You could modify your CG ballast to include a negative value for the z axis. Or you could create a separate ballast element that specifies only the z axis and its mass redistribution.

When you do this, watch the results on the inertia tensor. Remember that the larger the inertia tensor element, the greater the resistance to rotation, at least for the moments of inertia (the products of inertial are weird). Especially watch for big changes to the products of inertia, the non-diagonals. I've observed that products of inertia can affect what happens when landing a tail-dragger. When rotating the tail down to the ground, the plane may exhibit an undersirably strong yaw effect. Altering the products of inertia can improve this behavior. Experiement and discover what works best.

Chapter 12
YASim Weight and Balance

By Gary "Buckaroo" Neely

Good weight distribution and aircraft balance is necessary for good flight characteristics. In a real plane, it's also critical for safety. You can get in all sorts of bad situations with a CG that is too far forward or too far aft. After geometry, center of gravity (CG) placement is the most important aspect of creating your FDM. Don't even think about tuning other YASim numbers until you have the CG positioned correctly. If the aircraft CG is positioned right, it's possible to get a solution that flies well on your very first try with the solver.
The first step is to identify where the real CG should be. For most general aviation aircraft, the CG needs to be located forward of the aerodynamic center for the entire plane. If it isn't, the plane could become unstable in normal flight attitudes, a Bad Thing. The CG limits can be found on FAA certification sheets expressed as a percentage of MAC (mean aerodynamic chord, aka SMC, standard mean chord) or as offsets from some datum, usually the aircraft's nose or the wing's leading edge at the fuselage. The datum will given on the certification sheet. If you cannot find a CG range, you can make a guestimate based on ranges of 20-30% of MAC. 25% MAC is the typical figure. I suggest choosing a location in the 25-30% range, favoring the aft end. It's easier to find YASim solutions for aft-placed CGs for reasons I'll explain later.



If you lack a datum reference for placing the CG range, you'll have to calculate MAC. The goal is to learn where the CG range exists along the station-line or x-axis. This is easy with a wing shaped like a Hershey bar, where the airfoil shape and location is constant for the entire span. In these cases, 25% MAC is just 25% of the way back from any point on the leading edge. It's trickier with wings that have a taper and a sweep, and worse with airliner wings that often have two fundamental geometries, one for the long-chord inboard section, and another for the slender outboard section. In these situations you'll need to do a little work to find where the MAC lies along the wing.

Geometry comes to the rescue. The image at right illustrates the use of geometry to find the MAC for a swept and tapered wing. Add the length of the root chord to the leading and trailing edges of the tip chord, and do the opposite at the root chord. Draw lines between the extremes at either end, and their intersection indicates the location of the MAC, the mean aerodynamic chord. Easy-peasy.

For aircraft with multiple wing segments featuring different geometries like many airliners, you can use the same strategy repeated. Treat the inboard and outboard sections as two different wings. Perform the above exercise for each wing section to find the MAC for each. Now, treat these two chords as the root and tip chord of a new "wing", and repeat the procedure to give a single MAC for both surfaces.

Once you know the MAC, you can locate your CG range as a percentage of the chord from its leading edge. Find the location of this point along the x-axis and you have your target x-axis CG position.

If you don't know the CG range and your model uses tricycle landing gear, you can make a good guess at the CG position by assuming about 8% of the aircraft's weight rests on the nose gear. This is a typical estimate used for many airliner designs to be certain enough weight rests on the nose gear to provide good traction for steering. It may be helpful in placing the CG.

After choosing a CG position within the range and identifying its position along the x-axis, the next step is to get YASim to place the CG at this desired location. Run a test against the command line solver. Don't worry about getting a solution, at this point all you care about is the "CG: x:" value.

The Ballast Element

Let's assume we have an small aircraft with an empty weight of about 2700 lbs, a nose at x=4 and a mid-chord wing located at x=1.446. A handbook calls for a CG range of 19% to 33% MAC (mean aerodynamic chord). YASim prefers solutions with the CG in the aft range, so let's shoot for something in the back half of that range, between 25% and 33% MAC. Running some quick calculations, we find the allowable range on the x-axis is:

25% MAC: CG at x=1.770
33% MAC: CG at x=1.590

So we'd like to see the CG lying on the x-axis somewhere between 1.590 and 1.770. Here's a trial run with the solver using no ballast:

```
E:\FlightGear 2.10\data\Aircraft\Yak-18T>yasim yak18t-yasim.xml
Solution results:       Iterations: 491
 Drag Coefficient: 20.699659
       Lift Ratio: 59.436172
       Cruise AoA: -1.541610
   Tail Incidence: 4.504022
Approach Elevator: -0.158732
               CG: x:1.159, y:-0.000, z:-0.135

  Inertia tensor : 5889.702, -0.000, 372.538
        [kg*m^2]   -0.000, 5150.064, 0.000
     Origo at CG   372.538, 0.000, 10834.974
```

The CG is lying much too far aft at x=1.159. We need to bring it forward to at least x=1.590. Let's redistribute some mass, placing a ballast just behind the aircraft nose and trying a mass of 500 for starters. That's about 20% of the aircraft's empty weight. Ballast adds no weight to the aircraft, all it does is shift's the aircraft's calculated mass according to the location defined in the ballast. In this case we're shifting the mass forward. We'll add a ballast element to the FDM like this:

<ballast x="3" y="0" z="0" mass="500"/>

and then run the solver:

```
E:\FlightGear 2.10\data\Aircraft\Yak-18T>yasim yak18t-yasim.xml
Solution results:       Iterations: 638
 Drag Coefficient: 20.207283
       Lift Ratio: 66.561714
       Cruise AoA: -1.550457
   Tail Incidence: 2.535978
Approach Elevator: -0.535204
               CG: x:1.566, y:0.000, z:-0.109

  Inertia tensor : 4562.464, -0.000, 236.895
        [kg*m^2]   -0.000, 4804.217, 0.000
```

```
        Origo at CG   236.895, 0.000, 9198.858
```

Better, but not quite in range yet. Let's try the same thing with the mass set to 700:

```
<ballast x="3" y="0" z="0" mass="700"/>
```

The solver now reports:

```
E:\FlightGear 2.10\data\Aircraft\Yak-18T>yasim yak18t-yasim.xml
Solution results:       Iterations: 665
 Drag Coefficient: 20.165134
       Lift Ratio: 68.989929
       Cruise AoA: -1.553766
   Tail Incidence: 1.839068
Approach Elevator: -0.666178
               CG: x:1.729, y:-0.000, z:-0.098

  Inertia tensor : 4031.066, -0.000, 190.487
        [kg*m^2]   -0.000, 4542.457, 0.000
     Origo at CG   190.487, 0.000, 8421.492
```

Not bad. That gives a CG at roughly 27% MAC. We could stop here and fly, or continue to refine the CG placement, perhaps changing the z-axis placement for different handling characteristics.

For purposes of establishing CG, add only a single ballast entry in the FDM. Don't get fancy with more than one entry at this time. I tend to weight the nose, as it is similar to how I would weight my R/C aircraft. But you can add ballast anywhere. The mass can be a negative number, indicating you are removing mass from that location. Remember that ballast is just mass-redistribution, so you're not adding or subtracting real weight. The y-axis should nearly always report 0 for CG. The z-axis will vary but is not critical to a solution so long as it is reasonable.
For a more refined CG placement, try learning a little about inertia tensors. The placement of mass has many effects on the behavior of an aircraft, and the YASim inertia tensor gives some information on how mass is distributed. I provide an introduction to the topic in Inertia Tensors and YASim.

Fuel Load
The CG is just a base target. One of the things that can affect CG is fuel load. The fuel load used by the solver is controlled by three things: the configuration of fuel tanks, the "fuel" attribute of the approach element, and the "fuel" attribute of the cruise element. Configuration of those elements is covered elsewhere. Here it's enough to know how they affect the aircraft weight when running the solver.

Fuel tank elements determine where fuel is located, and the total amount possible. For example:

```
<tank x="1.4" y="2" z="-0.500" capacity="150">
<tank x="1.4" y="-2" z="-0.500" capacity="150">
```

Here the aircraft has two tanks probably located in each wing root, each having a capacity of 150 pounds of fuel. The location is important, because 150 pounds in a light aircraft is a considerable amount of weight. It could have a significant affect on CG.
We need to tell the solver what fuel load will be in the tanks for the approach calculation and the cruise calculation. So we could use approach/cruise attributes like this:

```
<approach ... fuel="0.2">
<cruise ... fuel="0.7">
```

The solver will use 70% maximum fuel load for the cruise calculations, and 20% for the approach calculations.

Weights
Another thing that can affect CG is the non-fuel load of the aircraft: pilot, crew, passengers, baggage, cargo, real weights for

ballast, etc. For a small single-person aircraft we could dispense with this and simply total the weight with the dry weight of the aircraft and set that value as the airplane mass:

```
<airplane mass="2863">
```

But it's more common that the load of the aircraft will vary. You allow virtual pilots to determine load at run time using the Flightgear Equipment menu if you configure the FDM and the -set file appropriately. Let's look at how to do that.

Let's say we have a general aviation aircraft having 2 front seats, 2 rear seats, and a small compartment for baggage behind the rear seats. In the YASim FDM, we first add 3 weight elements:

```
<weight x="1.6" y="0" z="0" mass-prop="/sim/weight[0]/weight-lb"/> <!-- Crew -->
<weight x="0.7" y="0" z="0" mass-prop="/sim/weight[1]/weight-lb"/> <!-- Passengers -->
<weight x="0.3" y="0" z="-0.2" mass-prop="/sim/weight[2]/weight-lb"/> <!-- Baggage -->
```

These elements give the locations for three weights that can be modified at run-time by changing the associated properties.

While still working with the FDM we want to add matching sub-elements to the approach and cruise elements:

```
<approach ... fuel="0.2">
  ...
  <solve-weight idx="0" weight="180"/>
  <solve-weight idx="1" weight="0"/>
  <solve-weight idx="2" weight="0"/>
</approach>

<cruise ... fuel="0.8">
  ...
  <solve-weight idx="0" weight="180"/>
  <solve-weight idx="1" weight="0"/>
  <solve-weight idx="2" weight="0"/>
</cruise>
```

What these do is specify the weights used for these positions by the solver. In this case, I'm telling the solver to consider only the weight of a typical pilot. Different weight configurations will give you different flight results, so experiment.

Now let's leave the YASim FDM and move to the <my-aircraft>-set.xml file. In the -set file you should have an element called "sim". We need to add a few configuration lines to the sim element:

```
<sim ...>
  ...
  <weight n="0">
    <name>Crew</name>
    <weight-lb>180</weight-lb>
    <max-lb>408</max-lb>
    <min-lb>100</min-lb>
  </weight>
  <weight n="1">
    <name>Pax</name>
    <weight-lb>0</weight-lb>
    <max-lb>529</max-lb>
    <min-lb>0</min-lb>
  </weight>
  <weight n="2">
    <name>Baggage</name>
    <weight-lb>0</weight-lb>
    <max-lb>100</max-lb>
    <min-lb>0</min-lb>
  </weight>
```

```
</sim>
```

This tells the Flightgear Equipment menu what limits and defaults to assign to the weight positions. Here I've assigned the plane to have a default load of the pilot only. This matches my YASim solution configuration, so the base flight experience will be close to the solver results. I've also assigned the front seats position to have a maximum weight limit of 408 pounds, and a minimum of 100 pounds. The rear seat position and baggage position work the same way.

You can use any number of weights in your aircraft. Weights can be used to simulate many things, such as the dropping of crop dusting chemicals, or a load of water for fire-fighting.

Further Thoughts

I've seen many YASim FDMs with inappropriately placed CG, and a surprising number where the developer placed the CG behind 50% MAC on a general aviation aircraft with a Hershey bar wing. Flying these models feels odd, even to someone like me with very little experience behind the stick of a real aircraft. Some describe the results as "flying on rails", of flying "like a game". The aircraft simply isn't pivoting where it should. If the aircraft's CG isn't at least ahead of 36% MAC, it's likely the aircraft isn't going to feel right to someone with real stick-time.

There are exceptions. Some early vintage aircraft had CG's well behind 50% MAC, relaxing pitch stability at the expense of being demanding to fly. YASim won't force you to set good weight and balance numbers. Just like a real plane you can likely fly with numbers out of the recommended range. Fortunately you don't risk death in a flight simulator. For my guide and recommendations, I am assuming relatively modern, conventional general aviation aircraft.

CG doesn't have to be exact. In flight it will move due to fuel consumption and movement of passengers, etc. But it does have to be reasonable. If you don't have a good feel for CG placement, then study weight and balance worksheets for general aviation aircraft. There are lots of them out there. If you don't quite understand this stuff, consider reading-up on some fundamentals of flight before working with YASim. It's fascinating stuff.

Chapter 13
YASIM Approach and Cruise Settings

By Gary "Buckaroo" Neely

These two elements define the limits of the aircraft behavior. Approach settings determine the necessary low-speed, high angle of attack behavior. Cruise settings determine the high-speed, low angle of attack behavior. The solver must find a common elevator incidence solution that allows both behaviors. The primary function of approach and cruise settings is to determine the spectrum used to test the horizontal stabilizer's ability to balance pitching moments. I discuss this in more detail in YASim Solutions. Here I focus on the configuration of approach and cruise element values.

Approach Settings
Let's look at the approach settings first. Here is an example approach configuration:

```
<approach speed="80" aoa="5.8" fuel="0.2">
  <control-setting axis="/controls/engines/engine[0]/throttle" value="0.3"/>
  <control-setting axis="/controls/engines/engine[0]/mixture" value="1.0"/>
  <control-setting axis="/controls/engines/engine[0]/propeller-pitch"
value="1.0"/>
  <control-setting axis="/controls/flight/flaps" value="1"/>
  <control-setting axis="/controls/gear/gear-down" value="1"/>
  <solve-weight idx="0" weight="350"/>
  <solve-weight idx="1" weight="350"/>
</approach>
```

The primary approach attributes are speed, aoa (angle of attack) and fuel. You should always provide values for these attributes. The approach element can also have sub-elements consisting of control settings and solve weights. These are optional, but all useful approach elements will have control settings.

Approach AoA sets the bottom end of the aircraft's performance. It's really about configuring where the aircraft begins to stall and as such it will have a great effect on stall performance. For example, an aircraft with an approach angle of attack that is much lower than the wing's stall angle will need a lot of elevator authority to pull the plane into a stall attitude or beyond. More about this later under Solutions. Let's look at the approach attributes first.

speed
A good guess at approach speed for general aviation aircraft is find the stall speed of the aircraft when configured for landing (flaps and slats deployed, gear down, aka Vs0) and multiply it by 1.3. For example, if you know that the plane will stall at 55 knots, try an approach speed of 72 knots. Another method to set approach speed is to use Vs0 as the initial trial approach speed value, fly the results, and alter the approach speed in the FDM until the aircraft achieves a stall at roughly the plane's correct stall speed. Be sure that you do this with the aircraft configured for the proper load. An airliner with a full load of passengers might stall at 110 knots, but its stall speed will be much lower when unladen.

aoa
Setting a good approach angle of attack requires knowing something about the aircraft's intended design characteristics. Aircraft designed for STOL or high-lift cargo operations will often have very low approach angles because they use high-camber wings that generate a lot of lift even at an AoA of 0. This is particularly true when coupled with powerful flaps that change the wing's camber. Some high-lift aircraft like the DHC-6 Twin Otter will even approach at a negative angle of attack. At the other end of the spectrum are the delta-winged aircraft. With low-aspect delta wings, no flaps, and often a lifting-body effect, they tend to approach at high angles of attack, 15 degrees or more.

For general aviation aircraft using flaps, a good guess at an approach AoA is about 1/2 the stall AoA. For aircraft without flaps, try using the actual stall angle for approach. When judging the stall AoA, consider the entire aircraft, not the wing itself. For example, the wing might stall at 15 degrees, but a two degree washout will reduce this to an average of 14 degrees, and a wing incidence of 3 degrees will reduce it again to 11 degrees. In such a case, with an aircraft equipped with flaps you might try using an approach aoa of 4 or 6 degrees. If the aircraft has no flaps, try using an approach AoA of 10 or

11.

Sometimes you can find a good approach AoA by studying photos or videos of the aircraft on approach. Make sure that you take the aircraft's glidepath into account. A typical glideslope is 3 degrees, so if the aircraft appears to be approaching at an AoA of 3 degrees with respect to the ground, it's true approach AoA would be 6 degrees.
If you know the glide ratio of the aircraft and the nose angle with respect to the horizon when gliding, you could get a reasonable angle of attack by calculating: arctan(glide ratio) + nose angle.

Approach angle of attack is a key value for tuning a YASim FDM. Often a configuration using a certain aoa value will not solve, but changing the value slightly will give a solution. I have more to say on this here: YASim Solution Troubleshooting: Singularities

fuel
This is the fraction of fuel remaining in the tanks, 0-1 and is used to contribute fuel weight to total aicraft weight. A value of 1 indicates the solver calculations should be performed assuming full tanks, and 0.1 indicates the calculations should assume the tanks contain 10% of their maximum fuel capacity. For most situations, this value should be small, leaving enough fuel in the tanks for a comfortable reserve. The default value is 0.2. The exact value is not critical, but aircraft weight will have a significant effect on solver results.

glide-angle
There is a fourth and rarely-used attribute, glide-angle. This is a relatively late addition to YASim and is typically used for unpowered aircraft. I don't have experience with glide-angle. When I do, I'll expand this guide.

Control Settings for Approach
In addition to the speed, aoa, and fuel attributes, you will want to tell YASim what control settings should be used for "approach". Commonly used settings and recommended values are:v

full flaps (1)
full slats (1)
gear down (1)
throttle to some reasonable low-mid value (approximately 0.3)
props set to full-fine (max RPM) (1)
mixture full rich (1)
elevator trim if known

The controls and settings will vary with each aircraft. Obviously some aircraft lack flaps or slats, some will have fixed gear that are always down, some will have blower controls, etc. If using fixed landing gear, tell YASim that the gear should be deployed just as if the gear were retractable.

If you know that an aircraft will approach with a certain amount of elevator trim, you can and should specify that control in your approach setting:

```
<control-setting axis="/controls/flight/elevator-trim" value="-0.2"/>
```

You cannot set a control for elevator for approach and cruise, only elevator-trim. Elevator control is the province of the YASim solver. See YASim Solutions for more information.

Control settings are critical for getting good results. Double-check your control settings, since a mistake here can have significant effects on the solution.

Solve-Weights for Approach
These settings determine run-time distribution of weight. Typically they are used to give weights for the pilot and crew, the cargo, and sometimes ballast. Simple FDMs having a single pilot and no cargo might not have solve-weights. This also means there would be no provisions for customizing weight for a given flight. When working with the solver, you need to tell YASim what weights will be used to generate the solution. My recommendation is to use a typical common-case load.

For more information on setting up and configuring weights in your YASim model, see: YASim Weight and Balance.

Cruise Settings
Before getting into the specifics of cruise settings, we need to consider what "cruise" means in this context. Nowhere in documentation or code does YASim provide a definition for cruise, so I've had to come up with my own interpretation. "Cruise" is a somewhat nebulous term that often refers to an aircraft's flight between the climb and the descent phases. A cruise configuration might maximize enduranace, or maximize range, or minimize flight-times between two points. Cruise might also refer to a state where the aircraft is in equilibrium, lift equals weight, thrust equals drag, and the aircraft is moving at a constant speed and a constant altitude. Conditions and requirements may vary, and requirements might even change after the aircraft is designed. Many airliners were designed during a time when fuel was relatively cheap and emphasis was on endurance or flight-time, but now the emphasis is often on fuel economy so cruise requirements are not the same.

I have two schools of thought for configuring the YASim "cruise" element:

1. "cruise" as equilibrium at maximum power/performance
2. "cruise" as equilibrium at best economy/range/endurance

Both strategies involve differences in throttle, propeller pitch, and speed settings. Both should assume a constant speed in level flight at a fixed altitude.

Using method 1, throttle is always wide-open, propellers full-fine (max RPM), and speed is set to the maximum speed the aircraft is known to be able to maintain in level flight at that altitude. Method 1 allows the aircraft to meet top-end performance numbers, but other solution results may be forced into more extreme values. Drag Coefficients and Lift Ratios will be more separated, and stabilizer/elevator requirements are greater, meaning less authority without greatly increasing elevator lift.

In method 2, settings are based on a typical economy cruise configuration as suggested in a pilot manual. Throttle might be something like 70%, propellers set to a reduced RPM, and speed will be some economy cruise speed. It's generally easier to get good YASim solutions using method 2, probably because the configuration is less extreme. Drag Coefficients and Lift Ratios tend to be closer together, and stabilizer/elevator requirements are less demanding. The disadvantage of method 2 is the aircraft will not likely reach the aircraft's maximum power speed.

Method 1 gives me better general results, though it is more difficult to tune. When combined with the right approach settings, using maximum power settings brackets the aircraft's two flight performance extremes: max cruise and near-stall. Using method 1, the FDM developer determines the two extremes of flight and the pilot determines the economy settings in flight.

I can't tell you which method to use or which method is more correct. I prefer method 1, but only one thing matters: the end result. If a particular strategy gives a model that comes close to matching real aircraft behavior and the plane "feels" right, then don't be concerned with what is "correct" for YASim. Use what works.

Now let's look at the cruise settings. Here is an example cruise configuration:

```
<cruise speed="205" alt="8000" fuel="0.7">
  <control-setting axis="/controls/engines/engine[0]/throttle" value="1"/>
  <control-setting axis="/controls/engines/engine[0]/mixture" value="0.65"/>
  <control-setting axis="/controls/engines/engine[0]/propeller-pitch"
value="1"/>
  <control-setting axis="/controls/flight/flaps" value="0"/>
  <control-setting axis="/controls/flight/elevator-trim" value="0.15"/>
  <control-setting axis="/controls/gear/gear-down" value="0"/>
  <solve-weight idx="0" weight="350"/>
  <solve-weight idx="1" weight="350"/>
</cruise>
```

Cruise settings are similar to approach, with a few important differences. The primary cruise attributes are speed, alt (altitude) and fuel. You should always provide values for these attributes. The cruise element can also have sub-elements consisting of control settings and solve weights. These are optional, but all useful cruise elements will have control settings.

speed
If using method 1, set speed to the maximum possible at full power in level flight at a known cruise altitude. If using method 2, you'll need to do deeper research to determine a typical airspeed for your choice of cruise criteria. Sometimes pilot handbooks contain usable reference material. Make certain speed is expressed in KTAS, not KIAS.

altitude
The maximum reasonable cruise altitude, or one for which you have good numbers
.
fuel
Similar in function to the value used for approach. The default is 0.5. Choose a value that allows for a reasonable burn of fuel to reach a typical cruise altitude.

If you don't have a realistic value available, try this: Find the total weight of the aircraft after accounting for solve-weights and deduct this from the maximum take-off weight. Use the difference to calculate a maximum fuel load. After getting a flyable solution, fly the aircraft from the ground to cruise altitude and note the fuel remaining. Use this to get your fuel fraction (remaining/initial) and rework your FDM with this value.

glide-angle
See my notes under approach settings.

Control Settings for Cruise
In addition to the speed, alt, and fuel attributes, you will want to tell YASim what control settings should be used for "cruise". Commonly used settings and recommended values are:

flaps retracted (0)
slats retracted (0)
gear up (0)
throttle maximum (if using method 1) (1)
props set to full-fine (max RPM) (if using method 1) (1)
mixture depends on altitude
blowers engaged to the proper level, probably maximum (1)
afterburner engaged if present
elevator trim if known

If you have fixed-gear, then you will want to specify the down or deployed position (1). If using method 1, maximum power, set throttle to 1 and props to full-fine (1). If using method 2, throttle and propeller might be nearer 0.7.
YASim mixture setting doesn't work quite like a real mixture control, so a value is hard to define. For GA aircraft, try something like 0.6 at cruise. Mixture doesn't have much effect on the FDM solution.

If your aircraft uses a certain amount of elevator trim at your selected cruise setting, you can and should specify an elevator-trim control in your cruise setting. Conventional elevator aircraft designed for efficient cruise will tend to have little or no trim at cruise. Remember that you cannot set a control setting for the elevator, only elevator-trim.

Solve-Weights for Cruise
Set these to the same values as your approach solve-weights.

Chapter 14
Putting It All Together: YASim Solutions

By Gary "Buckaroo" Neely

The YASim Solver
The solver is a subset of the YASim program which takes in the FDM data you've provided and constructs the relationships that generate a flyable model. It calculates total drag, thrust, lift, and weight. It determines if your aircraft can maintain sustained, level flight with the pitching moments acting on it during approach and cruise flight attitudes. The solver pre-calculates certain values and some flight surface settings and, if the results are successful, it uses these for the duration of the flight. If not, it reports that it failed to solve for the specified FDM.

The solver runs automatically when you startup Flightgear with an aircraft configured to use a YASim FDM. The solver pre-calculates settings like horizontal stabilizer incidence that are used for the entire flight, then YASim proceeds with normal interations during the flight.

Even if you aren't expecting a good solution from the solver, it can be useful in finding flaws in your FDM or positioning the center of gravity. It will flag some of the more blatant errors, and tell you when you have problems with your XML. If you are working with YASim, you will be using the solver a lot.

Running the Solver from the Command Line
The primary function of the solver is to determine if the aircraft can fly with the pitching moments that will act on it, so it's a useful tool for a developer. It would be tedious to have to start up Flightgear every time a developer wants to test the FDM. Fortunately the solver can be run independently of Flightgear as a stand-alone program. This is a great time saver.

This independent solver application is normally found in /bin directory of the Flightgear installation. On my 32-bit Windows system it is found as:

<flightgear root>/bin/Win32/yasim.exe

To run the program from the command line, you should be able to enter something like this:

"C:\Program Files\FlightGear\bin\Win32\yasim.exe" "C:\Program Files\FlightGear\data\Aircraft\Optica\Optica-yasim.xml"
The exact command will vary with your operating system and installed files.

In practice, I make a copy of the yasim.exe program and place it in the same directory as the aircraft FDM file. Then all I need do is open a command prompt window and enter:

yasim.exe Optica-yasim.xml

where "Optica-yasim.xml" is the aircraft FDM I'm working with. For this to work, you may need to add the Flightgear /bin directory to your operationg system's environment path.

Working with the YASim Solver
The solver is all about pitch moments. It has two primary jobs: a) to determine if the aircraft's horizontal stabilizer can balance the pitching moments acting on it at the extremes of its flight envelope and b) to find a horizontal stabilizer incidence that allows the aircraft to cruise with neutral elevator input. The flight envelope extremes are set using the approach and cruise elements of the YASim FDM.

For the cruise calculation, the YASim solver sets the horizontal stabilizer incidence to adjust the lift produced by the tail surface in order to counter-balance the summed moments acting on the aircraft's pitch axis at cruise. Recall that you cannot set the stabilizer incidence yourself like you can the wing; the solver determines the stabilizer incidence. The solver does this using a neutral elevator, in other words, an elevator control value of 0, no elevator control input. You cannot specify an elevator control input in your cruise configuration, but you can assign a control input value for elevator trim. The solver

then tries values for stabilizer incidence until it finds an incidence that enables stable level flight or gives up. The solver reports the stabilizer incidence used, along with the cruise angle of attack when using that incidence.

The solver does a similar exercise with approach settings. This time, rather than assume a neutral elevator, we're telling the solver what angle of attack to hold, and the solver tells us how much elevator control input is necessary to hold that flight attitude. Again, the elevator must be able to balance the summed pitch moments acting on the aircraft in this attitude. If your configuration doesn't have enough elevator authority to hold the approach angle, YASim will report "Insufficient elevator to trim for approach".

If the solver can find settings for tail incidence that can trim for cruise and allow an approach with maximum elevator deflection or less, then its work is done. If not, it keeps trying different values until it finds workable numbers or exceeds a maximum number of attempts and reports "Solution failed to converge after 10000 iterations".
A typical solver report looks something like this:

```
Solution results:       Iterations: 813
 Drag Coefficient: 15.855506
       Lift Ratio: 110.167068
       Cruise AoA: 0.600153
   Tail Incidence: -0.246429
Approach Elevator: -0.683866
               CG: x:-23.852, y:-0.000, z:0.539

  Inertia tensor : 718058.188, -0.004, 497888.563
       [kg*m^2]   -0.004, 5542764.500, 0.001
     Origo at CG   497888.563, 0.001, 6015855.000
```

These results were taken from my MD-81 model. Solution results will vary widely for various aircraft configurations, so don't expect yours to have numbers like these. This example is only for illustration.

Let's look at the solution results in detail:

Iterations
This is how many attempts the solver made to find numbers that worked. The solver isn't smart-- it simply sets trial values, looks at the results, then tries a new guess, repeating the procedure until it gets a good result or exceeds how the number of guesses it's allowed to try. Iterations is not very useful to the developer, but if iterations gets much over 1200 or so, it may indicate that your configuration isn't optimal. I wouldn't worry about it much. Most decent YASim FDM's solve in under 1500 iterations. Most that go over that fail to solve.

Drag Coefficient and Lift Ratio
These are dimensionless numbers that roughly indicate total drag and lift. Don't try to make these conform to real L/D ratios, the numbers are not equivalent. Drag values are normalized to 10, so re-examine your settings if you are getting numbers wildly beyond this range. Don't be too alarmed at high drag values-- I've configured high-lift FDM's that resulted in Drag Coefficients well above 30 that gave me the flight characteristics I wanted. Lift values are normalized to 100. Lift ratio for GA aircraft should typically not exceed 100 by much, and will often be well under 100. Numbers well over 100 and especially above 200 should be re-evaluated, as this suggests performance requirements that are too great for the engine or the altitude. It can result in an aircraft that glides forever. If drag and lift results are very close together then your aircraft will glide like a brick, but this will often be correct for many planes.

Cruise AoA
The angle of attack of the aircraft at cruise speeds in degrees. High-speed performance aircraft will usually cruise with a very small AoA. High-lift configurations with a highly cambered airfoil are likely to cruise with a pronounced negative angle of attack.

Tail Incidence
This is the solver's setting for horizontal stabilizer incidence in degrees. Using this incidence value the aircraft will cruise with neutral elevator.

Approach Elevator
This is the solver's result for the elevator deflection necessary to hold an approach. For example, a result of -0.8 indicates

that an input of 80% "up" elevator control is necessary to hold the approach. A value of -1 would mean full "up" elevator is necessary to hold the approach, which would be give no margin of control.

In general cases, you will want the solver to give an approach elevator result in the range of -0.9 to -0.5, depending on the aircraft and the configured load. Values greater than -0.9 will not give much margin, but may be correct for your aircraft. Some notorious historic aircraft will approach with the stick held nearly all the way back. Beware of approach elevator values less than approximately -0.5, where small control inputs are necessary to hold an approach. This usually implies an elevator with too much authority. The aircraft may become pitch sensitive and difficult to fly, and autopilots may fail to find solutions to hold pitch. This results in porpoising or "spazzing" effects as some in the Flightgear forums have called it.

Approach configuration has a great effect on stall behavior. If the aircraft begins to stall at 12 degrees and the approach AoA is configured for 6 degrees and the solver reports an approach elevator value of -0.9, then it's telling you that you need 90% elevator to hold the approach angle of attack. But that is not stall. The aircraft won't be able to achieve a true stall (though it might achieve a kind of mushy pre-stall.) To bring the aircraft to a stall attitude, you'd need much more elevator authority, in this case something like -0.5. But be mindful that many aircraft, especially those with high-lift wings, aren't meant to fly at near stall attitudes-- that's why they're high-lift, so you can fly very slow or with very high payloads without falling out of the sky. So elevator authority for many such planes simply won't give you a stall in normal usage. The behavior depends on the specific aircraft, so be sure to study your aircraft's flight manual and pilot reports.

CG
This is where the solver has placed the aircraft's center of gravity. See YASim Weight and Balance for more information on the importance of CG placement. A solution is not necessary for the solver to place the CG. When working on weight and balance, I often use the command-line solver to show the CG result without caring about the other solver values.

Inertia tensor [kg*m^2] Origo at CG
This gives some idea of how the aircraft's mass is distributed about the center of gravity. Those new to YASim can safely ignore the inertia tensor and still obtain good results from YASim. For more information, see Inertia Tensors and YASim.

Closing Thoughts
If your YASim configuration is well thought-out, the solver may give you a flyable result on your first try. It's more likely that you will have to change and tune many settings before you get something flyable, and do much more before you get something that approaches the characteristics you want.

Here I've outlined the basics of YASim solutions and the solver. But there's a lot more to getting good solutions. For additional help, I've outlined some of the common problems and possible aids here: YASim Solution Troubleshooting
Getting a solution from the solver is only one step toward a good flight model. A successful solution will likely reward you with something that will fly, but that does not mean the flight model is realistic. That will take a lot of study, a lot of flying, a lot of FDM refinement, and a lot of patience. You will be using the command-line solver over and over as you refine your YASim FDM into something that approaches the characteristics of the real aircraft.

Chapter 15
Troubleshooting YASim Solutions

By Gary "Buckaroo" Neely

Getting a good YASim solution can be difficult. Some configurations solve with no trouble, particularly if the CG is correct, the model is simple, and the developer is meticulous. Others seem to willfully fight you. The following are situations I've often encountered when dealing with problematic FDMs, and my thoughts on how to fix the problem.

CG Location
Before diagnosing any YASim solver issue, make sure that your CG is reasonable. For a YASim aircraft, a CG in the 25-36% MAC range works best. YASim tends to be happier with solutions using CG's in the aft range. See Weight and Balance for more information.

Large Lift Ratios
If the solver gives you a lift ratio approaching 150, it's a good idea to re-examine the FDM. If lift ratio exceeds 200, the FDM is unlikely to have reasonable flight characteristics. When you see these high lift ratios, the solver is telling you that the aircraft's thrust, drag and lift results are not well balanced, resulting in a solution that is forcing the aircraft to be very "slick" to compensate for being under-powered or other configuration issues. In my experience, the problem is usually caused by choices for approach, cruise, wing incidence, and airfoil stall and camber settings. Sometimes it's caused by mistakes in engine settings.

Most conventional general aviation aircraft should have lift ratios under 100, in the 60-90+ range. A high-performance, fast or slick airframe might go over 100. It's been my experience that high-lift aircraft will have very low lift ratios, in the 50-80 range, with high drag coefficient values in the 20-40 range. These numbers will look extreme, but real-life high-lift aircraft often are extreme configurations.

"Insufficient elevator to trim for approach"
You'll see this a lot when building YASim FDMs. YASim is griping that it doesn't have enough elevator authority to meet the necessary approach profile. Should you increase the lift attribute on your hstab elevator flap? Maybe, but there are other things to check first.

An elevator usually needs a lift attribute ranging from 1.3 to 1.6. If you can get good solutions using such values, you're probably OK. But if the elevator lift needs to be well beyond this range, then there may be other factors causing the problem.
CG is the first thing to check. If you tweaked geometry positions or weights, CG may have changed. Check that your CG is where you think it should be. Is it in a good range? A forward CG requires more elevator authority. An aft CG requires less, but might also make the aircraft unstable or difficult to fly. One test is to research the real aircraft's stall characteristics and make certain your simulation stalls in a similar fashion.

Check your geometry. Melchior's YASim importer for Blender is useful for viewing this at a glance. Are the flight surfaces where they should be? A single wrong number or inverted sign can place a flight surface in a weird position and invalidate the FDM.

Approach angle of attack is another value that can cause this problem. A high approach AoA needs a lot of elevator authority to hold it there because it's fighting the wing's large negative pitching moment. This may be beyond the design configuration of the aircraft. Make sure your approach values are reasonable and not too great. Look at photos of the aircraft on approach and try to estimate their angle of attack with respect to the glide angle. High-lift aircraft and those with very effective flaps will often have surprisingly low approach angles, sometimes even negative.

Check other approach settings. Flaps should be set as fully deployed (1.0), fuel should be some smallish value like 0.2, propellers set to full-fine (1.0), and throttle should be be appropriate to hold the approach attitude, perhaps a low to mid power setting 0.2-0.4.

High-lift configurations can also give "insufficient elevator" headaches. I cover this situation in more detail further down in this guide.

If you get into "insufficient elevator" problems, try the above ideas before ramping up elevator flap lift. And whatever you do, don't start adding "effectiveness" values on your horizontal stabilizer to get a solution. All that does is add drag to your stabilizer. It might give you a solution, but it won't address the core problem.

High-Lift Configurations

High-lift or high-camber airfoils cause special problems for YASim. Airfoils with generous stall numbers or airfoils with high cambers (0.15 or more) can make it difficult to get good approach elevator results. This is particularly true in aircraft with airfoils having high camber, high stall configuration values, and high-lift flaps. Sometimes these configurations solve without much trouble, while others refuse to play nice. Just when you think you've got a configuration that works, you fly it and find you can't stall the plane no matter what. But re-configuring it to enable by-the-book stalls requires huge elevator authority which compromises other flight behaviors. It can be maddening.

When checking stall behavior, first make sure the real aircraft really can achieve a stall. Many won't stall by design. With the stick all the way back, the aircraft simply mushes downwards. Before fighting YASim's high-lift issues, make sure you really have to.

You can set extreme values on elevator lift to compensate, but this solution usually yields an elevator that has way too much authority for cruise. The plane will be sensitive in pitch, making it twitchy and fatiguing to fly. Even small elevator inputs will yield big pitch responses, so an auto-pilot will have trouble holding pitch. It's like using a sledgehammer to drive a small nail. The AP can't refine a solution and the aircraft will start porpoising. This has been called "spazzing" in the Flightgear forums. The problem is too much elevator authority.

Airfoils have very different responses for drag, lift, and moment as a function of angle of attack and flap deployment. These are major considerations for choosing a particular airfoil. Unfortunately YASim uses a kind of one-size-fits-all approach that doesn't allow a lot of tweaking of airfoil effects. The only factor we can manipulate to any degree is lift using the stall and camber settings, and even that is not very refined. If I could change a single thing about YASim, it would be to add the option to use tabular airfoil data for flight surfaces. Flap configuration is particularly simplistic.

If you have the data and are willing to learn another FDM scheme, consider using JSBsim when these problems become insurmountable in YASim. But if you don't have the time or inclination, there are still some options:

reduce camber
reduce stall alpha
reduce flap lift
move the wing slightly

"Camber" as YASim defines it is easy to find if you have airfoil data, but that doesn't mean it's realistically used at all possible values. Camber is a primary factor in the high-lift problem, so a reduction in camber is likely to make the task easier. Keep in mind that real wing C_L values are 10-20% less than airfoil section values due to spanwise pressure gradient effects. For most GA aircraft, consider using Camber values 15-20% less than those derived from airfoil C_L data. Just don't reduce camber so much that you lose the high-lift characteristics you are trying to simulate.

Try reducing stall characteristics such that the airfoil stalls sooner, or has a different stall width, or even reducing the value for peak below the default 1.5. Play with reducing stall AoA first, as this tends to give better results. Frankly, an airfoil with a high-lift camber, a high critical stall AoA, and a gentle fall-off at peak is probably too good to be true. The negative aspects of such an airfoil (monstrous drag, huge moments, etc.) may not be adequately simulated in YASim. You're dealing with a method that tries to simulate airfoils with a somewhat limited algorithmic approach, so you should feel free to tweak everything you've got to get something that matches real flight profiles.

Reducing flap lift may help. Flaps alter lift and drag, but they should also change the effective incidence and camber of the wing. In addition, flap lift and drag is often not linear with flap deployment. These factors are not represented in YASim. (Flap simulation is #2 on my list of things I would change in YASim, just behind using tabular data for surface sections.) In many flap cases, especially with split flaps and high-lift configurations, flaps are mostly drag, and lots of it. In YASim, flap lift has a great effect on the moment produced by a surface, but flap drag has no effect on YASim moment calculations, so try reducing flap lift and adding flap drag.

Moving the wing slightly along the x-axis (waterline) may relieve some of the problems. YASim places lift forces rather far back on the chord (33%), so moving the wing forward by up to 8% of the mean chord may help such issues. Don't be reluctant to change geometry if such a move can make a better FDM. When moving a flight surface, remember that you are shifting mass distribution, so check your CG and re-balance.

In the end, all you can do is play with the numbers and fly the simulation. Don't be a slave to the numbers or someone's notion of the "right way" to do something. If you get something close to the behavior of the real thing, you win.

"My Plane Won't Fly Fast Enough!"
Developers sometimes have difficulty getting their YASim aircraft to reach the desired maximum flight speeds. Usually this is a configuration problem, but sometimes it's a YASim limitation or a misunderstanding of exactly what maximum speed means.

The first thing to check is the aircraft's cruise settings. In my opinion, YASim "Cruise" speed is best used to represent the aircraft in sustained level flight at maximum continuous power at an appropriate cruise altitude with all flight surfaces clean and gear retracted. Throttles should be wide-open, flaps fully retracted, gear up, propellers full-fine. Yes, props at full-fine, maximum RPM. This is not an economy cruise or noise-reduction configuration, it's a flat-out, max power, best-the-plane-can-do configuration.

Next, check that the cruise speed setting is in KTAS appropriate for that altitude. Not KIAS. This is very important and often misunderstood. When you do your test flights at altitude with full power and read off the KIAS numbers, remember to adjust for altitude and temperature to get KTAS. You might find a "slow" plane is actually doing just fine.

Supersonic speeds: YASim was originally built to simulate conventional general aviation aircraft. It wasn't designed for supersonic (or even transsonic) aircraft. YASim will allow you to fly at supersonic speeds, but here the simulation falls to game levels. I cover some of these issues elsewhere. For this discussion, if you need to go supersonic and high-fidelity flight isn't the main concern, check that your jet engines are producing the necessary thrust, and in particular look at jet exhaust speed. See my discussion on YASim jet engines for more information.

"Solution failed to converge after 10000 iterations."
Don't panic. You'll likely see this a lot until you've gained experience with YASim. This result tells you that you need to go back to the basics and check all the fundamental stuff. With a reasonable configuration, good geometry, good engine definitions, reasonable attributes, and a reasonable CG, YASim can usually give /some/ solution, even if it's whacky or tells you it doesn't have enough elevator for an approach.

There are a couple of things to check if YASim stubornly refuses to cooperate. The most basic is to make sure you've got a properly configured XML file. Make sure you have no weird characters that the XML parser might hate. (Hyphens in comments can be a problem.) Make sure all the primary required components are present. YASim will usually gripe if they're not, but some errors aren't trapped. Also make sure you've got a control input sub-element for elevator trim in your hstab configuration. That one is often missed by people new to YASim and will give mysterious errors.

YASim Singularities
You may have a perfectly good flight model with all characteristics tweaked nicely, and then discover that your infomation had wing incidence set too low. You add a degree to incidence, and suddenly YASim goes crazy and can't find a solution. The solver spews numbers that aren't even close to what you had before. Welcome to what I call YASim "singularities".

I often encounter situations where the change of one value, say a degree of wing incidence, creates or magnifies gaps where certain mid-range approach aoa's won't solve. I can make it solve with a very low approach aoa, say 2 degrees, or a much higher aoa of, say, 7, but the middle ranges won't solve, even though I might have had a great solution at 5 degrees aoa with an wing incidence 1 degree less.

YASim is a complex mathematical model with lots of algorithms, lots of educated guesswork, and yes, lots of fudges. There are many ways in which it can get "lost" and return junk, even if your inputs are absolutely good. The math falls into spaces where the numbers go out of range. The worst part is that you don't know where this is happening.

Sometimes these situations can be solved with by applying a little aerodynamic logic. Think about your configuration, what you're trying to tell the plane to do. Sometimes the situation seems to make no sense, especially given that most of us working with YASim are not experts in either aerodynamics or YASim's rather complex code base. If the situation makes no sense, try making very small changes to various related values. Sometimes a 0.1 change to a value will free YASim from a singularity and you'll get a good solution. Try it with approach aoa, or wing incidence, or stall values, etc. On one occasion I discovered that moving the end position of a flap segment very slightly dislodged the singularity. Sometimes a small change in CG will enable a solution. I've found no single cure.

Chapter 16
Common YASim Problems

By Gary "Buckaroo" Neely

These are some of the more common problems I've noticed in YASim FDMs since I began a serious study of YASim. Be careful when examining an FDM made by another developer-- what looks like a mistake may actually be intentional and based on careful study of the aircraft's actual or reported behavior by the developer. As always in my guides, these suggestions are based on what's typical for relatively modern general aviation aircraft. In general, the following items are flags that signal possible problems in a YASim FDM. If you don't have any of these problems, chances are you'll get a good flight model.

Geometry mistakes
Easily identified by importing the FDM into Blender using Melchior's importer script and comparing against the real aircraft configuration. With most FDMs it's not a problem, but it's worth the test as it can reveal a glaring mistake.

CG placement
After geometry, this is the most important thing to get right and one of the most common problem areas. It's importance is stressed on type certifications and stressed in flight manuals. It needs to be stressed in YASim. The CG needs to be in a proper range or the FDM will not feel real in flight. For most general aviation aircraft, the CG should be ahead of 36% MAC. For YASim purposes, a CG in the range of 25-30% usually gives good results.

Extreme stall values
Conventional GA aircraft airfoils tend to stall in the 15 to 18 degree range. High-lift airfoils often stall much lower. These numbers do not reflect the effects of incidence and twist, and YASim needs to allow for a degree or two of interpolation before hitting critical AoA, so the aircraft itself will likely begin to stall 3-6 degress less than this. So when I see stall AoA values set to 19 or higher for high-aspectc general aviation wings, it's worth a second look. Stall widths wider than 12 degrees are also worth a second glance, as are those less than 4. And very few people understand what stall peak does, so values other than 1.5 are usually suspicious.

Unusual Approach settings
A quick look at approach AoA can say a lot. A high-lift aircraft having a high approach AoA value signals a problem, as these aircraft typically have very nose-low approaches. Heavily cambered airfoils designed for high-lift applications can have stall alphas as low as 10 degrees, so approach AoA will be much lower, possibly even negative. Delta-wings or a very fast aircraft lacking flaps will usually have high approach AoAs, so an FDM with a low one suggests a mis-configuration. Approach speed values should be somewhere between stall and stall * 1.4, anything outside of that should be re-examined. Check that control settings are reasonable: flaps set, a throttle setting somewhere around 1/4 to 1/3, and gear down.

Unusual Cruise settings
It should be obvious, but approach and cruise speeds should not be the same. Altitude should be a typical cruise altitude. Fuel should be something reasonable after a long climb-out. Check control settings. I recommend settings optimized for maximum continuous power in level flight at the optimal altitude. I think this gives better flight results than solutions using economy cruise settings since YASim cruise determines the top end behavior of the aircraft. In general, throttle should be wide-open, props full-fine, flaps and other surfaces clean, gear up. If the aircraft has a blower setting, make sure it is set appropriately for the cruise altitude.

Wing incidence values of 0
Most general aviation aircraft will have several degrees of incidence, so it is unlikely that you will have a wing incidence of 0. Be careful though, because some GA aircraft actually do have an incidence of 0. For example, the DHC-2 Beaver despite being a high-lift configuration actually has a wing incidence near zero, which some consider to be one of its very few

design flaws. Wing incidence naturally has a major effect on YASim, so try to find the right value before you do much work on your FDM. Also, remember that you cannot set horizontal stabilizer incidence-- the YASim solver sets this value.

Camber values of 0 on high-lift airfoils

It can be tough to find airfoil data, but if wing camber is 0 on an aircraft that clearly has a high-lift airfoil, it's obviously not right. Similarly, a large camber on a fast or aerobatic aircraft can also signal a problem as many of these have symmetrical or near symmetrical airfoils. Camber has a great effect on the aircraft's intended performance, so it's worth careful study.

Use of the "Effectiveness" attribute

This is commonly used on horizontal stabilizers as a cheat to get a solution when nothing else seems to work. Effectiveness alters the drag of a flight surface, so what it's doing is forcing the stabilizer to drag behind the rest of the aircraft. In most of these cases, the proper solution is to first get CG right, then examine other values for problems. Effectiveness should be used only to increase drag, not as a crutch to get solutions.

Index