Firespitter Module Documentation

Full source: https://github.com/snjo/firespitter

Index

Info: Node replacement string parsing

FSmeshSwitch

FStextureSwitch2

FSfuelSwitch

FSengine

FSengineSounds

FSanimateGeneric

FSplanePropellerSpinner

FSbuoyancy

FStextureSwitch [deprecated in favor of FStextureSwitch2]

FSanimateThrottle

FSswitchEngineThrustTransform

FSwheel

FSpartTurner

FSliftSurface

FSwing

FSmonitorInterface / FSmonitorScript

Some comments mark a field as Optional. Most fields are actually optional, in that if you don't define them, they will use their default value. But the Optional fields mark extra behaviour that you would not normally bother with.

Info: Node replacement string parsing

Modules like FSmeshSwitch, FStextureSwitch2 and FSfuelSwitch don't use configNodes (the values inside a module that are inside a further set of { }), they instead use a single line with a

list of values.
What used to look like this in FStextureSwitch:

```
MODULE
 {
       name = FStextureSwitch
       textures
       {
              name = MyMod/Parts/bp/biPlaneGreen
              name = MyMod/Parts/bp/biPlaneRed
       }
       objects
       {
              name = obj_main
              name = obj_ctrlSrf
       }
 }
```

Now looks like this in FStextureSwitch2:

```
MODULE
 {
       name = FStextureSwitch2
       textureNames = MyMod/Parts/bp/biPlaneGreen;MyMod/Parts/bp/biPlaneRed

       objectNames = obj_main;obj_ctrlSrf
 }
```

Options are separated with semicolons, and, if there's a set of values inside one option, they are separated by commas.
A floatCurve looks like this:

fuelConsumption = 0, 0.01; 0.7, 0.04; 1, 0.1

or with tangents:

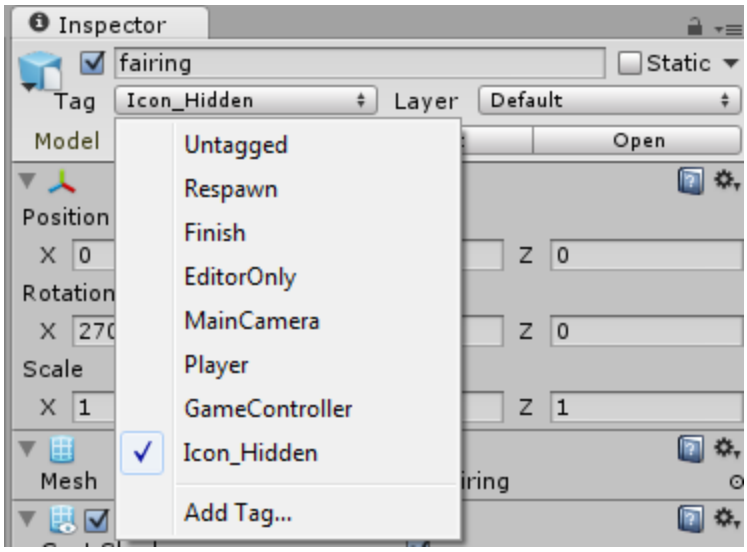
fuelConsumption = 0, 0.01, 0, 0; 0.7, 0.04, 0.1, 0.1; 1, 0.1, 0.3, 0

Note the difference between semicolon separators, commas separating values that belong together, and periods for decimal points. White space is allowed. (If you find a place where it is not, let me know. There have been rewrites to allow for whitespace)

This system isn't very pretty, but it avoids the use of configNodes, which is more prone to bugs, and can't easily be updated by reloading the database while the game is running.

FSmeshSwitch

Instead of switching textures, switch meshes instead, with clever use of UVs to reuse the same expensive bitmap. Meshes are cheap. textures are expensive. Also allows for parts with different shape variations.
To hide meshes from view in the parts list thumbnail, apply the tag Icon_Hidden on the gameObject in Unity. (To avoid z-fighting etc.)



MODULE
{
	// the name of the module. Don't change.
	name = FSmeshSwitch

	// If there are multiple instances of this module in a part, they must each have a unique module ID to avoid conflicts
	moduleID = 0

	// The text on the tweakable buttons in the hangar.
	buttonName = Next part variant
	previousButtonName = Prev part variant

	// The different variants can have a display name that explains what they are
	objectDisplayNames = Red;Blue;ClownCar

	// You can show or hide the Previous button to save GUI clutter
	showPreviousButton = true

	// Pair the switching of the meshes with switching fuel tanks in a FSfuelSwitch module

with no GUI
useFuelSwitchModule = false

// if useFuelSwitchModule is on, the list of fuel tank setups should be listed here. As you toggle between meshes, it selects tanks in the same order. You can have duplicate entries, like 0;1;2;0;0 to make the last two have the same tank setup as the first variant
fuelTankSetups = 0; 1; 2

// the objects that will be toggled through when clicking the buttons. One will be visible at a time. It also accepts sets of objects to be toggled on and off together, so in one one set you can turn on/off more than one object. Separate objects that belong together with a comma.
//**Important:** The code targets the object's renderer, which is always present on a gameObject with a mesh. If there is no renderer, it won't know what to do, so don't target empty gameObjects, at least give it a renderer. It can have further children which will also be toggled.
objects = redtank; bluetank, blueWidget; greentank, greenWidget

// When you switch this object's meshes, you also switch the objects on parts in the symmetry. This will always be on if useFuelSwitchModule is true, to avoid horrible GUI bugs in the editor.
updateSymmetry = true

// internal persistent value. Could be set in the cfg if it should default to something other than the first object in the list, but really, just set the default object to be the first in the list.
//selectedObject = 0

// If this is true, any colliders on the toggle objects also get turned on/off. It's still recommended that you use colliders on switched objects as little as possible, and instead use a single collider for the part.
affectColliders = true

// If true, it feeds info to the module info system that appears when you right click a part in the part catalog in the hangar. In sets of mesh switchers and fuel switchers, you will want to turn this off in one of them
showInfo = true

// Show messages about what the module is doing in order to find out how you done messed up, son.
debugMode = false
}

FStextureSwitch2

Replaces FStextureSwitch.

Switch between different textures on a part, or some objects in a part with this module. The names of the textures must match what's in the GameDatabase. There's a list of all loaded textures in the Alt+F12 menu.

To find the names of the objects to switch textures for, click on the part in the action editor and use the **List Objects** button. They will be logged to the Debug Log window (alt+f2 / alt+f12).

You can put multiple of these modules on a part as long as you give each one a unique moduleID.

You can use either \ or / in the folder names. \'s are replaced with / internally.

WARNING: If the part name has an underscore in it, KSP translates this to a period. This will trick this module into not finding the part and its textures. Drop the underscore in part name (in folder names it's OK).

```
MODULE
{
	name = FStextureSwitch2

	// If there are multiple instances of this module in a part, they must each have a unique module ID to avoid conflicts
	moduleID = 0

	//the names of the objects to swap textures on. Use semicolons to separate names. all objects with the same name that have a mesh will be switched, not just the first found.
	objectNames = fuselage;canopy

	// Optional: this text will be added to each the start of each entry in the textureNames and mapNames lists to make them neater. If you have textures in different folder, this value should be the lowest common folder. Each entry is treated like textureRootFolder + textureNames[i]
	textureRootFolder = MyMod/Parts/MyPart/

	// a list of textures, matching their names in the GameDatabase
	textureNames = green;red

	// An optional list of normal maps, or other maps such as specular to switch at the same time as the texture. If there are no textures in the textureNames list, you can switch just the normal map.
	mapNames = green_NRM;red_NRM

	// The different variants can have a display name that explains what they are
	textureDisplayNames = Green;Red
```

// The text on the tweakable buttons in the hangar.
nextButtonText = Next Texture
prevButtonText = Previous Texture

// There's a line in the gui showing the current display name, this is the title on that line. If you have more than one switcher, explain the difference here.
statusText = Current Texture

// allows switching by the player in flight mode
switchableInFlight = false

// If using additional map switching, replace the shader texture value with this name (using Material.setTexture). For example _BumpMap, _Emissive etc. It all depends on the shader in use.
additionalMapType = _BumpMap

// If using any other map type than normal maps, set this to false
mapIsNormal = true

// allows the player to repaint a part while walking around as a kerbal
repaintableEVA = true

// You can show or hide the Previous button to save GUI clutter
showPreviousButton = true

// Pair the switching of the meshes with switching fuel tanks in a FSfuelSwitch module with no GUI
useFuelSwitchModule = false

// if useFuelSwitchModule is on, the list of fuel tank setups should be listed here. As you toggle between meshes, it selects tanks in the same order. You can have duplicate entries, like 0;1; 2; 0; 0 to make the last two have the same tank setup as the first variant
fuelTankSetups = 0; 1; 2

// When you switch this object's meshes, you also switch the objects on parts in the symmetry. This will always be on if useFuelSwitchModule is true, to avoid horrible GUI bugs in the editor.
updateSymmetry = true

// If true, it feeds info to the module info system that appears when you right click a part in the part catalog in the hangar. In sets of tex switchers and fuel switchers, you will want to turn this off in one of them

showInfo = true

// Turns on more debug messages to troubleshoot the module/cfg setup
debugMode = false

// show or hide the List Objects button. Disabled by default. Should only be on while developing the part. Outputs a list of gameObjects in the part to the Alt+F2 log
showListButton = false
}

FSfuelSwitch

Switch between different resource setups in a single part. Can be controlled by FSmeshSwitch and FStextureSwitch2.

MODULE
{
name = FSfuelSwitch

// the resources in each *tank setup.* Separate setups with a semicolon. Separate tanks that go together in a single setup with a comma
resourceNames = ElectricCharge; LiquidFuel,Oxidizer; MonoPropellant

// the max amount of a resource in each tank in each setup. semicolons separate setups, commas separate tanks in a setup
resourceAmounts = 100; 75,25; 200

//Sets the amount of resource in each tank in each setup. semicolons separate setups, commas separate tanks in a setup.
initialResourceAmounts = 0;0,0;0

// the dry mass of the part with no tanks
basePartMass = 0.25

// each type of tank setup can weigh a different amount. Batteries should weigh more than an empty LF tank. Some resources should require sturdy tanks.
tankMass = 0.5; 0.1; 0.2

// Additional Funds cost for the different tanks. This is added to the part's base cost.
tankCost = 0; 0; 0

// If true, a line is added on the right click menu listing "Added Cost", because the price widget in the editor is not updated often enough to make it obvious

displayCurrentTankCost = false

// Whether the player can affect the tank setup selection directly, or must do it through a mesh or fuel switcher.
hasGUI = true

// If true, the user can switch resource types while playing. This will empty out the tank. Makes sense for converting a tank from LiquidFuel to Kethane after landing a mining base for instance. Default is off of course.
availableInFlight = false

// Is the player allowed to set up the tank in the hangar? In the above scenario, maybe you don't want that.
availableInEditor = true

// If true, it feeds info to the module info system that appears when you right click a part in the part catalog in the hangar. In sets of mesh switchers and fuel switchers, you will want to turn this off in one of them
showInfo = true
}

FSengine

This is a replacement for the stock engine more fit for atmospheric engines. Most importantly, it handles thrust efficiency in different atmospheric densities, which the stock engine can't do without doing some ugly hacks. You should never have to override the maxThrust in this engine in flight, there should always be a sensible coefficient to access instead.
The current throttle can be controlled by other modules by replacing the delegate function getThrottleDelegate.

FloatCurves use the format "key,value;key,value" or "key,value,inTangent,outTangent;key,value,inTangent,outTangent"

MODULE
{
name = FSengine

// The total force applied when at full RPM, and full throttle, scaled by the various curves
maxThrust = 100

// A separate RPM value allows for engine spin up/down to be slow, but throttle response

to be quick. Current Thrust is scaled by the (current RPM / maxRPM) value. RPM is built up by the powerProduction value, and taken away by powerDrain and engineBrake
maxRPM = 600

// FloatCurve: Sets how much thrust you get at different atmospheric densities. Fuel consumption remains constant.
atmosphericThrust = 0,0; 1,1

// FloatCurve: Basically the same as the velocityCurve in stock, tells you how much thrust to give at different speeds.
velocityLimit = 0,1;200,1;500,0

// FloatCurve: Sets how much fuel is spent at the current throttle setting. Allows for fuel use when idling, and fuel use ramping up as the throttle is higher.
// **IMPORTANT**: If the final requested fuel after taking maxThrust etc into account is very low (0.000001 or below), the game will return 0 fuel instead, so in those cases the engine uses a hardcoded minimum amount. With these values, this happens with a maxThrust below 10
fuelConsumption = 0,0.0001; 1,0.01

//Sets the thrust at various throttle settings. This means built in afterburner support. In tandem with fuelConsumption, you could have a steep thrust and fuel use ramp up at 70%-100%.
throttleThrust = 0,0; 1,1

// Fuel values in the format "resourceName,ratio;resourceName,ratio". I prefer normalized values over integers, but that's optional.
resources = LiquidFuel,1; IntakeAir,15

// The transform(s) in the model where the force is applied
thrustTransformName = thrustTransform

// The tweakable slider value for max throttle value seen in flight and in the hangar.
maxThrottle = 100

//If the engine receives less than this amount of fuel (normalized), it flames out. a low threshold makes sense for electric engines that could run on partial power.
// Should be less than 1f to account for math issues with the supplied fuel amount
flameoutThreshold = 0.7

// The smoothing value for the FX that looks at this engine. Makes engine sounds etc less jumpy. Low numbers are smoother

smoothFXSpeed = 0.1

// Engine RPM gain per fixedUpdate when ignited and fuelled
powerProduction = 20

// Engine RPM slowdown when not ignited
engineBrake = 15

// RPM loss when running and throttled up, to simulate propeller blade drag for instance. scales with throttle setting
powerDrain = 10

// Turn on more error debugging messages in the log
debugMode = false
}

FSengineSounds

This module provides sound to engine with some options that are better suited for propeller engines than the stock sound handling, like constant sound at 0 thrust while running, and more control over volume and pitch.
In KSP 0.20.0 this is the only way to get custom engine sounds to work as far as I know. In later KSP versions, the stock method will probably start working again.

Not all features are done yet for the engine sound module, but it will all work very soon.

MODULE
{
name = FSengineSounds

//the names of the sound in the game database. Open the alt+f12 debug menu/sounds to see list of loaded ones. Any unnamed sound will be disabled, so just comment out the lines for the sounds you don't use.

//the engine start sound
engage = Firespitter/Sounds/sound_fspropstart

//the constant thrust independent background hum of an engine
running =

// the thrust based pitched main engine sound
power = Firespitter/Sounds/sound_fspropidle

// the engine shutdown sound
disengage =

// loss of fuel flameout sound. If you don't have a separate sound, consider using the same as disengage.
flameout =

// engine overheat warning sound
warning = Firespitter/Sounds/sound_fsheatDing

// the default pitch of the engine power sound. 1 is the same pitch as the original sound file.
powerPitchBase = 1.0

// how much above the default pitch the power goes based on thrust. final pitch = powerPitchBase + (thrust * thrustAddedToPitch)
thrustAddedToPitch = 1.0

// how quickly the engine power sound volume ramps up from silent when started. set to 1 for immediate full volume. This value is added to the volume every 0.2 seconds until it reaches 1.
powerFadeInSpeed = 0.005

// how long the volume is off after starting the engine, to allow the engage sound to be heard
powerFadeInDelay = 0.5

// thrust values below this level will yield a silent engine sound. 0.1 means 10% throttle is the cutoff. A 0.0 always keeps the sound on while the engine is running.
powerLowerThreshold = 0.0

// volume adjustment for the different sounds listed above
runningVolume = 1.0
powerVolume = 1.0
engageVolume = 1.0
disengageVolume = 1.0
flameoutVolume = 1.0
warningVolume = 1.0

// normalized engine heat value at which the warning sound chimes. set above 1 to disable (or don't name the sound of course)
warningSoundThreshold = 0.8

```
	// how often the warning sound can play in seconds
	warningCooldownTime = 2.0

	// random start delay for the power sound, to make several engines not play in sync. makes for a better soundscape.
	randomStartDelay = 1.0 //for playing engines slightly out of sync
}
```

FSanimateGeneric

This is a replacement module for the stock ModuleAnimateGeneric. In addition to the functionality of that module, this one supports:
-Custom Action name.
-Can start deployed, or toggle the animation in the hangar, via popup menu in the Action editor.
-Custom animation speed
-EVA context menu actions
-Handles more than one animation per part
-The animation can be reversed mid deployment

Does not support:
-isOneShot
-Locking the animation while running

```
MODULE
{
	name = FSanimateGeneric

	// The name of the animation in Unity
	animationName =

	// if you want a faster or slower playback speed without having to redo the anim, adjust this. 1 is normal speed
	animSpeed = 1

	// The right click anim start Event name while in flight. Can be anything
	startEventGUIName = Deploy

	// The right click anim reverse Event name while in flight. Can be anything
	endEventGUIName = Retract

	// optional Action editor name. e.g. “Toggle Animation”. Defaults to the above Event names.
```

toggleActionName =

// If true, the part starts at the end of the animation in SPH and in flight, (unless altered in the action editor)
startDeployed = False

// GUI text for the hangar starting state toggle
startDeployedString = Start Deployed?

// animation speed multiplier. 1.0 is the default speed. Does not care about custom Unity animation speeds.
customAnimationSpeed = 1.0

// Is the right click menu Events available while on foot?
availableInEVA = False

// Is the right click menu Events available while controlling the craft?
availableInVessel = True

// The range the action is available in EVA
EVArange = 5

// Animation Layer. If you have more than one animation on a part, this must be unique for each one in order for them to not interrupt each other. If the stock engine heat animation module is used, that's probably on layer 0, so make this 1 or higher.
layer = 1

// If true, clicking on the part in the Action Group editor pops up a menu asking if the part should start deployed. Also useful for things like cargo bay doors where you want o to open them to place stuff inside, then close before launch.
useActionEditorPopup = True

// If you have more than one FSanimateGeneric on a part, give each a unique ID to. This makes the popup menu appear in a separate location for each module, and not steal each others windows. values: 0-9.
moduleID = 0

// To make it obvious that this part has animation abilities, the animation can play once, from deployed to retracted state when the part is spawned/loaded. This only happens in the editor, not in flight.
playAnimationOnEditorSpawn = True

// Set the default location of the hangar GUI window (Optional)

defaultWindowRect = 550, 300, 150, 100

// play a named part effect from an fx node when in different states of delpy/retract. Meant for sounds, but any fx should work.
fullyRetractedEffect =
fullyDeployedEffect =
startDeployEffect =
startRetractEffect =

}

FSplanePropellerSpinner

MODULE
{
name = FSplanePropellerSpinner

// the gameObject to Spin around its forwards axis. Can be a parent to other sub objects.
propellerName = propeller

// the propeller rotation speed in RPMs when the engine is ignited, independent of throttle. Positive or negative values can be used. Use 0 for a wholly throttle controlled rotation.
rotationSpeed = -120

// the propeller rotation speed in RPMS that are added to the rotationSpeed according to the final thrust of the engine. i.e., higher throttle makes spinny thing go fast.
thrustRPM = -30

// Normalized value to spin the propeller if the engine is shut down/flamed out, but moving fast. If the craft is moving beyond windMillAirSpeed, the engine will spin at the speed you would expect in a running engine * windMillRPM. This means a purely thrustRPM-controlled engine will not windmill.
windmillRPM = 0.1

// If the craft is moving beyond this speed, and atmospheric density is above 0.1, the windmilling will start. It currently does not scale with speed above the minimum threshold.
windmillMinAirspeed = 30.0

// divide engineResponseSpeed by this amount for dramatic effect. Slows the spin up time of the propeller

spinUpTime = 10

// ---
// The rotor disc swap, when enabled, will swap out the blades for a slowly spinning blurry disc graphic when the engine is spinning at top speed.

// whether the blade/disc swap at high RPMs is used
useRotorDiscSwap = 0

// a smooth disc gameObject that replaces the blades
rotorDiscName = rotorDisc

// the normalized RPM values when the blades are hidden and the disc is shown. Since it's lerping to this value from the curent value, it will take a long time to reach 0.99 or 1.0, so set it a bit lower to get there in a reasonable time.
rotorDiscFadeInStart = 0.95

// The speed at which the blurry disc spins. Should be much slower than the engine RPM. A static speed.
rotorDiscSpeed = -30

// You can name up to six propeller blades that will be hidden when the disc is shown. Empty or unlisted values will be ignored.
blade1 = blade1
blade2 = blade2
blade3 = blade3
//blade4 =
//blade5 =
//blade6 =

}

FSbuoyancy

This module kills the parts stock buoyancy, and replaces it with custom float code.

MODULE
{
name = FSbuoyancy

buoyancyForce = 12.0 // the force applied to lift the part, scaled by depth according to buoyancyRange

buoyancyRange = 1.0 // the max depth at which the buoyancy will be scaled up. at this depth, the force applied is equal to buyoancyForce. At 0 depth, the force is 0

buoyancyVerticalOffset = 0.1 // how high the part rides on the water in meters. Not a position offset inside the part. This will be applied in the global axis regardless of part rotation. Think iceberg/styrofoam.
// for subs you could play with that value to make it think you are at sea level buoyancywisse at an entirely different depth. You could even change this value at run time to create the whole up/down code (Using your own code I mean)

maxVerticalSpeed = 0.2 // the max speed vertical speed at which there will be a liftng force applied. Reduces bobbing.

dragInWater = 1.5 // when in water, apply drag to slow the craft down. Stock drag is 3.

debugMode = True // enables buoyancy control context menus

waterImpactTolerance = 125 // For sea planes, any part in the water during take off should have a resistance high enough to withstand take off speeds.

//**forcePointName** = // you can set an alternate transform name to set up multi point float. Optional

splashFXEnabled = True // on by default, creates splashes even deep under water currently...

}

FStextureSwitch [deprecated in favor of FStextureSwitch2]

Switch between different textures on a part, or some objects in a part with this module. The names of the textures must match what's in the GameDatabase. There's a list of all loaded textures in the Alt+F12 menu.
To find the names of the objects to switch textures for, click on the part in the action editor and use the **List Objects** button. They will be logged to the Debug Log window (alt+f2 / alt+f12).
You can put multiple of these modules on a part as long as you give each one a unique moduleID.
WARNING: If the part name has an underscore in it, KSP translates this to a period. This will trick this module into not finding the part and its textures. Drop the underscore in part name (in folder names it's OK). (I'll try to parse my way around it later)

MODULE

```
{
    // the name of the module. Don't change.
    name = FStextureSwitch

    // the heading on the popup box. Used to differentiate multiple modules on one part
    displayName = Nose Art Texture.

    // show or hide the List Objects button. Disabled by default. Should only be on while developing the part. Outputs a list of gameObjects in the part to the Alt+F2 log
    showListButton = false

    // each instance of the module on this part must have a unique moduleID to be able to load node values, and get unique popup windows
    moduleID = 0

    // allows switching in flight mode
    switchableInFlight = False

    //shows debug messages in the log
    debugMode = False

    // a list of textures, matching their names in the GameDatabase
    textures
    {
        name = Firespitter/Parts/Command/FS_bomberCockpit/model003
        name = Firespitter/Parts/Command/FS_bomberCockpit/noseArt/early_bird_model003
        name = Firespitter/Parts/Command/FS_bomberCockpit/noseArt/swift_demise_model003
    }

    // A list of objects (transforms) inside the part that will have their textures switched. Found by using the List Parts button if you don't know it from the Unity project.
    objects
    {
        name = noseartplate
    }

    // An optional list of normal maps, or other maps such as specular to switch at the same time as the texture

    additionalMap
    {
        name = Firespitter/Parts/Command/FS_bomberCockpit/normalPlain
        name = Firespitter/Parts/Command/FS_bomberCockpit/normalPlain
```

```
            name = Firespitter/Parts/Command/FS_bomberCockpit/normalBlahBlah
        }

        // If using additional map switching, replace the shader texture value with this name
(using Material.setTexture)
        additionalMapType = _BumpMap

        // If using any other map type than normal maps, set this to false
        mapIsNormal = true
}
```

FSanimateThrottle

This animation module, in addition to solving the issues with multiple animations on one part for the stock moduleAnimateThrottle, allows for multiple engine states, controlled by another module. The default mode 0 will use the animation span indicated by primaryModeRange. The thrust reverser module will add another mode to the list, using a different animation span, so that when the mode switches, you can have other throttle animation effects. for the thrust reverser, this can be used to give propeller blade pitches going from neutral to positive normally, and neutral to negative when the reverser module kicks in.

```
MODULE
{
        name = FSanimateThrottle
        animationName = throttle

        // If false, the animation reverts to the start position if the engine state is off/flameout
        dependOnEngineState = False

        // how quickly it goes from the last animation time to the target time. 1 is instant.
        responseSpeed = 0.5

        // to avoid conflict with other animations, choose a different layer for each animation on
the part if there's a problem
        animationLayer = 1

        // moves from anim times of different modes smoothly if true, if false it snaps to correct
the time when the mode changes
        smoothModeTransition = True

        // the start and end time of the default engine modes throttle animation. If you are using
a FSswitchEngineThrustTransform, this could be 0.5, 1
        primaryModeRange = 0, 1
```

}

FSswitchEngineThrustTransform

Allows for engines with reverse thrust, so you don't have to create one pulling and one pushing propeller.

```
Module
{
	name = FSswitchEngineThrustTransform

	// The thust transform used for normal/forwards flight
	defaultTTName = thrustTransform

	// Optional: The alternate thrustTransform used for reversed or alternate thrust direction.
	alternateTTName = alternateThrustTransform

	// Should a named gameObject transform be used for the alternate thrust direction? If not, the reverse direction of the default will be used.
	useNamedAlternate = 0

	// Optional: If an FSanimateThrottle module is present, you specify a range of the animation to use in the reversed state. If it should have the the regular animation, set it to 0, 1
	animateThrottleRange = 0.5, 0

	// Should the engine start reversed? (Persistent value for saved craft)
	isReversed = False

	// Used to avoid popup conflict if duplicate modules are on a single part (unlikely)
	moduleID = 0
}
```

FSwheel

A module for wheels with rotating meshes, suspension and retracting animation. If you use the module without one or more components needed by the stock module, this version will not freak out. You can opt out of retract anim, suspension setups, or even having wheels in the first place. If used with no wheelColliders in the model, it will simply be a landing leg.
Also supports wheel motors, friction overrides and an ingame debug test menu for altering spring settings.
Steering is handled by the module FSpartTurner.

Notes on friction from Nothke:
Unity doc on basics of friction curves:http://docs.unity3d.com/ScriptReference/WheelFrictionCurve.html
And a corrected curve explanationhttp://forum.unity3d.com/threads/wheel-friction-curve-revealed.153340/
Note: friction is added on high speed sideways slip (hence the Unity's tendency to flip)

MODULE
{
	name = FSwheel

	// the name of the Unity wheelCollider physics collider. There can be multiple, named wheelCollider, wheelCollider2, wheelCollider3 etc.
	wheelColliderName = wheelCollider

	//a Cube collider defining the bounding box of the deployed gear, used for Put To Ground to ensure the gears don't start below ground. Should extend a bit below the wheel. Will be removed afterwards.
	boundsCollider = Bounds

	//the visual part of the rotating wheel. Rotates around the X axis. (can be an empty parent of the mesh). If you have multiple wheels, they are named Wheel, Wheel2, Wheel3, etc.
	wheelMeshName = Wheel

	// the name of the visual suspension, a parent of Wheel. If multiple, named suspensionParent, suspensionParent2, suspensionParent3, etc.
	suspensionParentName = suspensionParent

	//If you have multiple wheels, they can have individual wheel meshes and colliders, they are named oddly, as described in Wheel and wheelCollider to maintain compatibility with stock wheel code. The minimum requirement for a wheel setup is the collider. suspensions and rotating meshes are optional.
	numberOfWheels = 1

	// The stock wheels rotate around Z, so that's the default. If they rotate around X, set it to 1,0,0 or -1,0,0 if it rotates bacwards
	wheelRotationAxis = 0, 0, 1

	// if true, the wheelColliders are disabled to avoid odd collision issues or unwanted animation. If false, the wheel can still roll when retracted, which can be very useful.
	disableColliderWhenRetracted = False

	// Disable the collider at a specific time during retraction, in case it would collide at a bad moment.

disableColliderWhenRetracting = False

// this option shouldn't be used. It will invert the time check for the disabling of the collider.
disableColliderTimeInverted = False

// If disableColliderWhenRetracting is true, when during the animation should the wheel be enabled/disabled? 0 is deployed, 1 is retracted.
disableColliderAtAnimTime = 0.5

// The name of the animation to retract the gear. You do not need an animation if it's a fixed gear.
animationName = Retract

// to avoid conflict with other animations, the animation layers should be unique on the part. (Stock anims are on layer 0)
animationLayer = 1

// Optional: alter the retract animation's playback speed
//customAnimationSpeed = 1.0

// How long to wait between gear action group key presses before you can reverse the animation.
deploymentCooldown = 0.5

// internal persistent value. do not use.
//*deploymentState* = Deployed // Deployed, Deploying, Retracted, Retracting, Stopped

// The stopping power of the wheelColliders when braking
brakeTorque = 15

// How quickly the brake power is applied. Makes it so that you can tap the brakes quickly without getting full brake power and nose diving.
brakeSpeed = 0.5

// internal persistent value. Do not use.
// *brakesEngaged* = false;

// Adjust the visual roll speed of the wheel if it looks off. You will probably need more than 1.0.
rotationAdjustment = 1

// Amount of part drag when gear is deployed

deployedDrag = 0

// Amount of part drag when gear is stowed
retractedDrag = 0

// If true, kerbals can right click the gear and extend/retract when EVA. useful for lowering the plane so you cen climb up.
guiActiveUnfocused = True
// The range for the EVA kerbal retract/extend event.
unfocusedRange = 5

// If on, the right click menu will have an option to enable a popup window to test spring settings, and options to increase or decrease friction.
debugMode = False

// --- Wheel motor settings ---

// if true, the wheels can move you when the motor is enabled.
hasMotor = True

// If true, the wheel motors start enabled. Toggled by action group or right click event.
motorEnabled = True

// The power of the wheels to move you.
motorTorque = 2.0

// A wheel at an angle will have more power in one direction than the other because of loss/gain of friction when suspension is under pressure or not. This is an attempt to get around that for a wheel that will typically be at an angle. 1.0 is no change, 0.3 will give 30% of the power in the reverse direction.
nerfNegativeTorque = 1.0

//Wheel motor torque will go from 100% at 0 speed to 0% at this speed value, meaning you will have a top speed quite a bit lower than this number.
maxSpeed = 30

// The resource consumed by running the wheel motors
resourceName = ElectricCharge

// How much of the reasource to consume per second at full power.
resourceConsumptionRate = 0.2

// Internal persistent value. Do not use. Use motorStartsReversed instead

//*reverseMotor* = False

//Internal persistent value. Do not use. Wheels on the left and right side will have their reverse value auto set opposite of each other.
//*reverseMotorSet* = False

// Reverse motor power makes sense if you want to use wheel motors on take off and still pitch up with the same key.
motorStartsReversed = False

// --- Brake emissive ---

// A mesh with an emissive texture to indicate brake status
brakeEmissiveObjectName

// The color values for the different states of gear deployment. Use 0-1 values, not 0-255/0-FF
onEmissiveColor =1,0, 0.3, 0.0
offEmissiveColor =0.0, 1.0, 0.0
deployingEmissiveColor =1.0, 0.0, 0.0
disabledEmissiveColor =0.0, 0.0, 0.0

// --- Wheel collider settings ---

// If false, the values on the model's wheelCollider will be used without change.
// These settings are for the Unity wheelCollider. They are explained (poorly) in the unity docs.

// The override bools make sure the settings below are used even if the values are negative. Otherwise, they will only be used if they are defined as >=0 (The internal default is -1 to mark them as inactive)
overrideModelFrictionValues = False
overrideModelSpringValues = False
overrideSuspensionDistance = False

//for tire friction. A value of 1 is not enough to get up hills. Try 10.
forwardsStiffness = 10.0
forwardsExtremumSlip = 1.0
forwardsExtremumValue = 20000.0
forwardsAsymptoteSlip = 2.0
forwardsAsymptoteValue = 10000.0

sidewaysStiffness = 1.0 //for tire friction

sidewaysExtremumSlip = 1.0
sidewaysExtremumValue = 20000.0
sidewaysAsymptoteSlip = 2.0
sidewaysAsymptoteValue = 10000.0

// Set up proper values if using overrideModelSpringValues. See http://docs.unity3d.com/Manual/class-WheelCollider.html for more info.
wheelColliderMass = -1
wheelColliderRadius = -1
wheelColliderSuspensionDistance = -1
suspensionSpring = -1
suspensionDamper = -1
suspensionTargetPosition = -1

// Part effects that will play when retract or deploy starts (Typically sound effects)
startDeployEffect =
startRetractEffect =

// used to separate popup windows in the hangar. If there are multiple of this module on one part (unlikely), use unique numbers.
//moduleID = 0
}

FSpartTurner

Rotates a gameObject in a part based on control inputs. Primarily used for wheel steering.
It's a bit old, so setup axis are defined with separate floats instead of Vector3s.
Includes speed sensitive steering and tweakable steering in the hangar or in flight.
For KSP 0.23 it switches to the tweakables system instead of the popup menu.

MODULE
{
name = FSpartTurner

// the name of the gameObject that will be rotated
targetPartObject = turnEmpty

// You can define up to five extra gameObjects that will be rotated in the same fashion
//targetPartObject2 = someOtherName
//targetPartObject3
//targetPartObject4
//targetPartObject5
//targetPartObject6

// The axis to rotate the game object on. You can define a negative number to invert the rotation, or a smaller or larger number to alter response, but you would normally do that through the later listed values. In this example the object rotates around the Z axis.

rotationDirectionX = 0
rotationDirectionY = 0
rotationDirectionZ = 1

// If your game object's neutral position has a non zero rotation, you can set it here, and any rotation will be added to this default.

defaultRotationX = 0
defaultRotationY = 0
defaultRotationZ = 0

// the range in degrees to rotate the object at full input.

steerMultiplier = 10

// if the steering is speed sensitive, it will be at its lowest impact at this speed, with full impact at 0 speed

steerMaxSpeed = 15

// Keeps you from flipping out at speed by alloing less steering while the surface speed is high

speedAdjustedSteering = True

// At top speed, use this much of the steering input (1 is full, 0.1 is 10%)

speedAdjustedSteeringMinimumMultiplier = 0.1

// Whether steering is on by default (Also stored as a persistent value on the craft/save game)

steeringEnabled = False

// if useWheelSteeringAxis is false, you can toggle between using the QE or AD keys to steer.

altInputModeEnabled = False

// Reversed input is used for tail steering wheels.

reversedInput = False

// yaw trim will affect the wheel if this value is false and useWheelSteeringAxis is false

ignoreTrim = True

// use the new separate axis for wheels instead of roll/yaw. will also disable ignore trim

and alternate input buttons. By default it still uses the same keys though.

useWheelSteeringAxis = True

}

FSliftSurface

This modules provides aerodynaimc lift on a part, and can be used by itself, or in combination with an FSwing controller, using multiple FSliftSurface for different parts of the wing, like main sectio, aileron, flap, etc.

MODULE

{

name = FSliftSurface

// the name of a transform in the part that dictates the wing direction, and it's the pint where the lift force is applied. Place it so Z is forwards into the wind, Y is up towards the sky, X is right.Also used by FSwing leading edge code (leadingEdgeLiftSurface = this)

liftTransformName = lift

// An over all multiplier to the efficiency of the wing, this should be the same on all wings. a value of 0.00024 gives a normal take off speed relative to stock KSP wings (80 knots in a Sopwith Camel, twice the real speed). 0.0008 gives an almost realisitc take off speed for a light air craft (50 knots instead of real 39 knots in a Sopwith Canel)

power = 0.0008

// the total area of the wing in square meters

wingArea = 12

// the length of the wing from base to wing tip. This has a large impact on drag, since the aspect ratio is a direct multiplier for the drag. Aspect ratio = $span^2$ / area

span = 6

// the name used in the module info in the editor part hint. If you name your ailerons "Aileron" and you main section "Fixed section" for instance, it's easy to see how big a wing is, and how much control it has in the info window.

displayName = Wing

// The efficiency affects the amount of drag a wing produces when off angle. a value of 1.0 is an ideal wing, with minimum drag. 0.3 would be good for an air brake. Most ral wings are around 0.75.

//Ideal: 1.0 / Wright's plane 0.7 / Boeing 247D: 0.75 / DC-3: 0.785 / P-51D: 0.69 / L1049G: 0.75 / Piper Cherokee: 0.76

efficiency = 1

// The amount of drag produced by the wing and plane even at 0 Angle of Attack, due to inefficiencies in shape, and things like struts, wires etc on the plane. This is actually an overall rating for the whole plane, but that would be impossible to calculate per craft, so here it is used per wing set. A mustang is very efficient with 0.0161. A Sopwith Camel has lots of junk on it, and has a bad rating of 0.0378

zeroLiftDrag = 0.0161

// This is a cheaty way to make the wing more efficient by reducing the overall drag. 1 is normal drag, 0.1 is 10%.

dragMultiplier = 1

}

FSwing

This module does not produce lift by itself, but control the aileron, flap and leading edge, which each have their own liftSurfaces or other logic.

MODULE
{

name = FSwing

// The name used in the part info popup module details in the part list

displayName = Wing

// The name of the aileron. This will be rotated when flight control input is receives, according to roll, pitch, yaw and flap settings. It should contain an FSliftSurface to provide aerodynamic force, unless you are using stock lift code.

controlSurfaceName = obj_ctrlSrf

// Optional. if present, a separate control surface is used as a flap/air brake. For now it simply rotates, and does not support animations.

flapName = string.Empty

// Leading edges provide more lift and drag when deployed. Because of symmetry contsraints, it uses two of the in the model, one on the top, and one on the bottom of the wing, and which ever one faces the sky in the SPH, that's the one that is actually moved when the leading edge deploys

// Optional: Change the name of the leading edge toggle action, in case you want to use the functionality on some other part of the wing, such as an extending flap that increases lift.

//**leadingEdgeToggleName** = Toggle leading edge

// the name of the leading edge model on the top of the wing
leadingEdgeTopName = string.Empty

// the name of the leading edge model on the bottom of the wing
leadingEdgeBottomName = string.Empty

// The leading edge currently does not support animation, but instead translates between two positions, and optionally rotates on the way.
leadingEdgeTopRetractedName = leadingEdgeRetracted
leadingEdgeTopExtendedName = leadingEdgeExtended
leadingEdgeBottomRetractedName = leadingEdgeRetracted
leadingEdgeBottomExtendedName = leadingEdgeExtended

// if True, the edge will not only be moved between transforms, but also rotated.
leadingEdgeCanRotate = True

// How fast the leading edge deploys
leadingEdgeSpeed = 0.01

// If using stock lift code, this amount of drag is added to the rigidbody (not to the part drag)
leadingEdgeDrag = 3

// For stock wings, this increases lift coef, for FSliftSurface, it measures the increase in wingArea in meters
leadingEdgeLift = 1

// If using FSliftSurface, extra wing area will be added to the module with this liftTransformName
leadingEdgeLiftSurface = lift

// If true, the leading edge is not player controlled, but extended if the speed is below autoLeadingEdgeSpeed or the AoA is above autoLeadingEdgeAoA.
autoDeployLeadingEdge = False
autoLeadingEdgeSpeed = 90
autoLeadingEdgeAoA = 15

// the at rest rotation of the separate flap.
flapTarget = 0

// how much a press of the flap action groups will rotate the flap

flapIncrements = 10

// The min and max rotation of the flap. You can go negative for a flap that rotates up as well as, or instead of down.
flapMax = 40
flapMin = 0

// How fast the flap rotates
flapSpeed = 0.3

// The axis around which the control surface rotates. Default is X (Vector3.Right)
controlSurfaceAxis = 1,0,0

// The axis or combination of axis the control surface responds wo. An aileron would be 1 toll, 0 on the rest. This can be tweaked in the editor, or in flight. There is no auto detection like in stock wings.
pitchResponse = 0
rollResponse = 0
yawResponse = 0
flapResponse = 0

// How fast the control surface rotates
controlRotationSpeed = 0.2

// Wings on the left and right side have opposite rotation directions. If you are making separate left and right wings, you might want to set this to false to completely control rotation.
allowInvertOnLeft = True

// If true, this module can be used on a wing with stock wing/ControlSurface code, to add leading edge control.
affectStockWingModule = True

// If using stock lift code, set this to the same value as the stock deflectionLiftCoefficient
deflectionLiftCoeff = 0

// how much the control surface can rotate
ctrlSurfaceRange = 20

// to reduce agressive control response at high speeds, and take into account wind resistance on the surfaces, this limits how much the surface can rotate. at max speed, it will only rotate ctrlSurfaceRange * controlLimiterMultiplier
controlLimiterMaxSpeed = 400
controlLimiterMultiplier = 0.3

```
		// Gives more error messages
		debugMode = False
}
```

FSmonitorInterface / FSmonitorScript

A note from Mihara on editing the FSmonitorScript values without referencing firespitter.dll

```
using System;
using System.Collections.Generic;
using System.Reflection;
using System.Linq;
using System.Text;
using UnityEngine;

namespace ASET
{
	public class MFSMonitorOverride : InternalModule
	{
		[KSPField]
		public int charPerLine = 23;
		[KSPField]
		public int linesPerPage = 17;
		private InternalModule targetScript;
		private string[] textArray;
		FieldInfo remoteArray;

		public void Start ()
		{

			foreach (InternalModule intModule in base.internalProp.internalModules) {
				if (intModule.ClassName == "FSmonitorScript") {
					targetScript = intModule;
					remoteArray = intModule.GetType ().GetField ("textArray");
					// Mihara: I'm not up to figuring how to correctly set an enum using reflection right now.
					// Thankfully I don't have to because this is the default.
					//targetScript.textMode = FSmonitorScript.TextMode.singleString;
					break;
				}
			}

			textArray = new string[linesPerPage];
			for (int i = 0; i < textArray.Length; i++) {
				textArray [i] = "";
			}

		}

		public override void OnUpdate ()
		{
			if (CameraManager.Instance.currentCameraMode == CameraManager.CameraMode.IVA && vessel == FlightGlobals.ActiveVessel) {

				for (int i = 0; i < textArray.Length; i++) textArray [i] = "";
```

```
                                    // And here we actually populate the array with data by an undisclosed method.

                                    textArray [0] = "ECTb KOHTAKT!";
                                    textArray [5] = "    This here          ";
                                    textArray [6] = "      is supposed      ";
                                    textArray [7] = "        to work        ";
                                    textArray [linesPerPage - 1] = "root/root";

                                    remoteArray.SetValue (targetScript, textArray);
                                }
                            }
                        }
}
```