Velcro Rockets Version 1.11

By Erik Anderson aka Sputnik erik_h_anderson@hotmail.com

Thanks to:
Hendo and Daver and erv, for the CVE-Lite code on which this is based.
Urwumpe, for showing me how to read the .cfg file.
TomPA, for releasing some excellent Saturn V meshes to the community.
Travmind, for the NovaSSTO (Martin S10E-2) meshes.
80mileshigh for the Nova Martin S10E-1 meshes.
BrianJ for the DeltaII meshes.
Erv for the Centaur-G mesh.
STS107 for numerous meshes, to include the Agena, STAR-37 and STAR-48, and more.
DaveS and STS-107 for the IUS meshes and textures.

And, above all, many thanks go to Martin Schweiger, for actually developing the simulator I used to daydream about in astrodynamics classes!

http://www.orbitersim.com

Unpacking:

Use Winzip to put each subfolder in its matching Orbiter folder.
Velcro Rockets employs meshes from several existing add-ons. Mesh and Config files will be in a separate location (in the /Velcro subfoder), but the Textures folder is shared, so changes to these add-ons' textures will affect both versions.
Some add-on sub-parts worthy of mention:

Agena upper stage, by Alex (STS-107). From Thor-Agena-B add-on, as used in CVEL Titans.
IUS upper stages, by DaveS and Alex (STS-107). From IUS add-on.
Star-48+Star-37 upper stages, by Alex (STS-107). Effectively replaces Star-48 / -37 add-on.
Transtage upper stage. Effectively replaces my previous stand-alone Transtage add-on.
Centaur-D upper stage. Effectively replaces my previous stand-alone Centaur-D add-on.
Centaur-G upper stage by erv. Effectively replaces his stand-alone Centaur-G add-on.
Energia-M launcher. Effectively replaces my previous Energia-M add-on.
Energia launcher by Nerull.
Delta-II add-ons by BrianJ. Velcro Rockets does not include BrianJ's payloads or scenarios..
Travmind's NovaSSTO. Effectively replaces my previous stand-alone NovaSSTO add-on.
Nova S10E-1 by 80mileshigh. New for Velcro Rockets.
Proton-K by Hendo (dh219).
Fregat by Hendo (dh219).
Zenit launcher by Urwumpe. New version with texturing for Velcro Rockets.
Conestoga 1620. Demonstrates easy assembly of launchers from existing parts (Castor-IV)
McDD Barbarian. Demonstrates easy assembly of launchers from existing parts.
Blue Scout. An X-15-launched rocket.

Add-ons used in some Velcro Rockets scenarios:

CVEL-Buran, launched by the Velcro Energia.
CVEL Titans, for some scenarios adding additional strap-ons.
Project X-15, to launch the Blue Scout upper stage.
World of 2001, as a payload for the Nova S10E-1.

Introduction:

Welcome to the Velcro Rockets add-on! Velcro Rockets is a meta-add-on, which means it exists to make development of OTHER add-ons easier. I've included a number of those with this release, such as Energias, Novas, a Delta II, and more. Those may properly be thought of as rocket add-ons using the Velcro Rockets code, that I've also included with the Velcro Rockets distribution.
Velcro Rockets began as a notion to create a generic CVEL (Common Vehicle Extensions-Lite) stage which used a simple code loop, but pulled its actual properties (mass, fuel mass, Isp, engine location) through its .cfg file. In that way, anyone could make a simple CVEL stage without programming a new one, or indeed needing to code at all.
An aside: thanks to Urwumpe for showing me how to do this, after I found my original approach simply wouldn't work.

It wasn't long before I realized I could do the same thing with my Flyback F-1 code. It would be possible to build a strap-on motor or lower stage that would attach itself as that add-on did, but again, get its properties from the .cfg file.

Eventually, I realized I didn't need to make these two separate add-ons. The same code would do both jobs. "Velcro Rockets" was born.

The name "Velcro Rockets" came about because it was thus possible to stick a simple .cfg-based stage anywhere – on top of another rocket, or to its side as a strap-on, or underneath as a lower stage, building multi-stage vehicles, etc.

After some additional work, it became possible to make a Velcro rocket operate in one of several modes:

-- As a CVEL stage. See the included CVEL documentation for how to use it. CVEL draws a mesh (you tell it where to find it), adds a mass you specify, and then creates a payload (you specify the .cfg file) at the time of jettison. Payload fairings, interstages, etc., can also be added through additional CVEL payload strings.

-- As an attached stage. The attached, or series-burn stage, uses an attach point to hold some other vehicle in place, generally on top. The parent stage will generally burn to exhaustion, then the attached stage lets fly! Disadvantages over the CVEL method is that the upper stages "exist" at all times, increasing CPU usage and complexity of the .scn file. Advantages are that the upper stages "exist" at all times, enabling the use of spacecraft.dll payloads, DG-IV, NASSP, AMSO, anything with a panel that you'd like to watch while flying to orbit on autopilot, and so on.

-- As a parallel stage. A parallel-burn stage is a strap-on motor. The parallel mode adds the stage's engine and fuel tank to the core stage, then deletes these resources again when the strap-on motor jettisons.

-- As a booster stage. As above, but the fuel source is the core stage itself. That is, we're just temporarily attaching an engine, not a fuel tank. Used for such structures as the booster stage of an Atlas; we would treat the Atlas sustainer stage as a single stage, and the Atlas booster module is a second, booster, stage in this mode. The Atlas booster module has two engines which feed from the core (sustainer) stage.

-- As a crossfeeding stage. As a parallel burn, but also feeds its propellant into the core stage. Not used in any current launcher, but it's been proposed now and then, and is expected to be used in the Falcon-Heavy. It is most useful, however, for making drop tanks! Just set the engine thrust to an extremely small (non-zero) value.

So build your stage, and attach it any way you like. Go nuts!

A word on Multistage.dll: it is not my intention to create an add-on that replicates or "competes with" Multistage.dll. Oribteers should have one, standard, application for creating multi-stage vessels, and Vinka's Multistage.dll is it. But a generic application that creates upper stages OR strap-on stages OR lower stages in any of many combinations, necessarily, can do many of the same things as Multistage.dll; I've even included some. Nevertheless, I recommend that the average Orbiteer use Multistage.dll as the standard for creating multi-stage vehicles, except where that doesn't accomplish everything you need. For that, Velcro Rockets offers a few more capabilities, at the cost of additional complexity.

Velcro Rockets Operation:

Keys:

O – Autopilot toggle on/off
J – Jettisons the next payload
U – Enter a new launch azimuth

During launch, a flight director cue will show the autopilot's idea of where you should steer. The

cue looks a bit like an airplane, with a vertical tail to show desired rotation. If the autopilot is engaged, the cue's round circle changes to a diamond shape.

The autopilot is designed to make a quick yaw about the zenith until it's on the right azimuth, then act as a simple one-axis pitch autopilot. Pitch attitude is based on current energy state, allowing lots of variability for different stages. Low-powered upper stages will suffer a bit from non-optimal high-flightpath-angle trajectories, but mostly the system is very flexible.

Velcro Rockets Care and Feeding (.CFG files):

The .CFG file for a Velcro Rocket looks a lot like any other, but there are some important differences. Since the Velcro Rockets code can load the .CFG file in directly, some features are added, and a few work differently.

Stock .cfg lines that are re-read or require discussion are labeled in blue.

NEW .cfg lines for Velcro Rockets are labeled in green.

Module = Velcro	Always include this!
Meshname = Velcro/mesh	Or whatever your mesh is named
Mass =	Set Mass, MaxFuel, and Isp normally
MaxFuel =	MaxFuel needs to be non-zero (1 will do) for use in
Isp =	any and all parallel-burn modes
IspSL =	The Isp at sea level, to model under-expansion
Touchdownpoints =	Make sure you include some, for the parallel modes
; === Fin specs ===	New toy! You can add wings and fins to your .cfg-based Velcro Rocket, in a very generic manner....
Wing = 0.0 1.0 0.1 2.0 5.0 1.0	This adds a wing, at location (0.0, 1.0 0.1), of span 2.0, area 5.0, and Aspect Ratio 1.0.
HStab = 0.0 0.0 -4.0 2.0 5.0 1.0	Adds a horizontal stabilizer, for stability. No lift.
VStab = 0.0 0.0 -4.0 2.0 5.0 1.0	Adds a vertical stabilizer.
	There are no provisions for control surfaces.
Mayfly =	In honor of Urwumpe's Mayfly.dll; does something similar but simpler. Sets the number of seconds after staging, after which the stage will delete itself. Used to clean up jettisoned boosters and the like.
Become =	Points to a config file. If this is set, then on jettison, the stage will delete itself, replacing itself with whatever is pointed to here. There is no error-checking; an invalid value will crash Orbiter. Used for flyback first stages, and the like; you can point to a custom dll, or to a spacecraft.dll craft, etc. Note that the new craft starts with propellant tanks full, so your flyback stage should be designed accordingly.
; === Attachment specs ===	These 4 points MUST be included, as shown below, for usage as anything other than a CVEL stage.
BEGIN_ATTACHMENT	
C 0.0 0.0 10.0 0 0 1 0 1 0 VELCRO1	This is the forward attach point for payloads
C 1.65 0.0 0.0 0 0 1 -1 0 0 VELCRO2	The side attach pt, for a strap-on motor

```
P  0.0 0.0 10.0     0 0 1  0 1 0  VELCRO3      Repeats VELCRO1 point, but as parent
P  1.65 0.0 0.0     0 0 1  -1 0 0  VELCRO4     Repeats VELCRO2 point, but as parent
END_ATTACHMENT
```

ExhaustType = SRM

Exhaust Type tells the code to place a specific type of exhaust texture/trail behind the following ref points. Allowable types are:

- SRM – Solid Rocket Motor (smoke, no shutdown)
- HYBRID – Hybrid Motor (smoke)
- LH2 – Liquid Hydrogen
- N2O4 – Storable (RealExhaust texture)
- KERO – Kerosene flame
- LF2 – Liquid Fluorine (greenish flame)

MEngineVec =

If you'd like the main engine thrust vector to point in some direction other than (0, 0, 1), set it here. Exhausts will all point in this same direction.

MEngineRef1 =

Set the location of the first exhaust point. Set MengineRef2 for the second point, etc. Can have up to 512, so splurge.
Thruster exhaust is defined by the exhaust type and by the main thrust value divided by the number of exhausts. There's no way currently to have exhausts of differing sizes, but you can make “fatter” exhausts by putting a few in close proximity.

Note that Velcro Rockets only notices or cares about values for the main engine, and attitude thrusters. Setting values for hover and retro engines will be ineffective, as these thrusters are not used by Velcro Rockets.

Velcro Rockets Care and Feeding (.SCN files):

As a general rule, list upper stages before lower ones in the .SCN file.

As a CVEL stage, including a payload is simple. Just add a CVEL string:

```
PAYLOAD probe probe probe 0 0 22.25 140000.0 0.0 1.0
```

This creates a payload, named probe, with a meshname of probe, and will create probe.cfg when jettisoned. Put it at (0,0,22.25), make it mass 140,000 kg, and load it up with fuel (1.0).

CVEL payloads are also used for fairings and the like. For example:

```
Fairing 2 0.5 -1.0 0.0 0.0
```

This line tells the code that there is a fairing, in 2 parts. The 0.5 means that jettison will happen at 0.5 Pa ambient pressure. The -1.0 0.0 0.0 string tells the code that the first part comes off with that added velocity vector; subsequent parts get this vector rotated about the Z-axis clockwise by 360 degrees divided by the number of fairing parts. So fairings with three or more parts will need to be in a specific order.

The fairing gets jettisoned when the static pressure decreases to the stated value, or when the stage jettisons. The fairing jettison strips off the requisite number of CVEL payloads; a 3-part fairing will need the 3 parts defined as CVEL payloads, and the actual payload will have to be defined later (higher up in the CVEL payload list).

SLA 4 -2.0 -2.0 0.0

As a fairing, but the ambient pressure value is not needed. This string tells us there's a 4-part Spacecraft Launch Adapter, like the panels that separated from Apollo. An SLA separates at the same time the payload is jettisoned. The first part comes off with a velocity of (-2, -2, 0) and the others are defined in clockwise order so that they come off properly.

LES 15.0

The LES command is followed by a time in seconds. In this instance, 15 seconds after this particular stage begins firing, it will strip off the next available CVEL payload, and ignite its engines. If the payload in question is defined as a Velcro Rocket with exhaust type SRM, you can get the smoke of the jettisoned launch tower, too.

Interstages and other hardware can be defined as a CVEL PAYLOAD; they should stay with the stage when it is expended.

The FAIRING, LES, and SLA commands can all be used on the same stage. Note, however, that each strips off the next available payloads; if for some reason these don't execute in the order that you expected, you will get unpredictable results. This is particularly likely if you have a pressure-dependent FAIRING, and a time-dependent LES, and the user manually flies an unusually low or high trajectory, triggering the FAIRING unexpectedly. This is bad; set the two so that this is unlikely, or split them on different stages.

Note that, when a CVEL payload is jettisoned, the CVEL stage mass is updated immediately. However, if an attached lower stage is burning using a SERIESBURN arrangement (see below), it doesn't know to update the mass of the stages above it, so it won't do so until the next staging event. In essence, you'll introduce an error by carrying the mass of the fairing further than you actually did. For this reason, make your FAIRING and LES on the lowest stage you can, so that ideally it's on the stage that's firing when the jettison event happens.

Attachment:

A series-attached stage, such as the Saturn V S-IC first stage attached to the S-II second stage, uses the SERIESBURN command, as follows:

SERIESBURN 0 S-II 0 0 -9.25 0 1 0

0	Chooses which attach point (on the first stage) will be used. Either 0 (front of the stage) or 1 (side of the stage).
S-II	The name of the specific stage which will be attached. WARNING: the name must be EXACTLY correct – no error-checking is performed here. An incorrect name WILL crash Orbiter.
0 0 -9.25	The location of the attach point ON THE ATTACHED STAGE (S-II, in our example). A bit confusing, since, to move the stage up, you need to move the point down (Z-number more negative).
0 1 0	Sets the rotation about the attach point (Y-axis is up for the S-II stage).

Note that this will create an attach point on the S-II stage:

Location (0, 0, -9.25)
Direction (0, 0, 1) Implicit; not normally spelled out. CAN be changed
Rotation (0, 1, 0)

The Long Specification: If you don't like the default direction of (0, 0, 1), enter something else here before the rotation value. The Velcro Rockets code will note the extra digits and use the new value for direction. Make sure you normalize the values, or odd graphics may ensue.

The act of attachment will also add the S-II mass to the S-IC stage. When the S-IC stage is depleted or jettisoned with the "J" key, focus will be transferred to the S-II, which will be detached and have its engines started.

Parallel-burn staging is done similarly:

PARALLELBURN 1 STS-101 -4.15 -2.0 -7.0 -1.0 0.0 0.0

This attaches an SRM's side point to the (-4.15, -2.0, -7.0) location on STS-101, rotated so that the -X axis is “up”. Specification, including the long specification if desired, works as above.

The code operates quite differently from the SERIESBURN command, however. PARALLELBURN creates its engines and fuel resources on the core vehicle, and engages them when the core engines start.

You will want the focus on the core stage, not the strap-on motor with the PARALLELBURN.

Because the core stage cannot jettison the strap-ons directly, they are programmed to jettison themselves whenever core stage thrust drops to zero. When flying the core stage and you'd like to jettison the strap-ons, just set thrust to zero momentarily, then proceed on your way.

If for some reason this behavior is undesirable, you can override it by settting JETT_ON_MECO to 0 in the .scn.

The PARALLELBURN code does something unusual: the first PARALLELBURN strap-on imparts its touchdown points to the core stage, too. This is needed for use with the stock Atlantis, for example, so that it will properly stand on its tail for launch. To do this, however, the code actually stacks the strap-on as if it were a SERIESBURN stage, then converts itself to a PARALLELBURN stage at the instant of lift-off. This sets the touchdown points properly, but prevents unwanted jumping as the touchdown points are changed.

For cases where this behavior is undesirable (you don't want the booster's touchdown points, as in for example a JATO motor attached to an airplane), add the following line in the .scn:

PRIMEBOOSTER 0

The “Prime Booster” is the only one to do this action; it will hoist the entire stack on its own touchdown points, holding the entire stack as the parent. At the moment of launch, it converts itself to a child attachment as though nothing had happened. Setting PRIMEBOOSTER 0 will disable the checking to see if it's the first PARALLELBURN booster; if all boosters are disabled, nobody will change the touchdown points of the core stage.

Booster-burn stage:

BOOSTERBURN 0 STS-101 0.1 0.0 -3.3 -18.20 0.0 -1.0 0.0

A “booster”, in this case, refers to the Atlas booster module; engines without a fuel tank. The fuel is supplied from the core stage. Specification is the same as PARALLELBURN, except an additional number is added into the middle (highlighted). This is the fuel state (in the core stage) at which point the booster module will be jettisoned. In this example, the fuel state will be 0.1; 10% fuel left.

Crossfeeding:

CROSSFEED 1 GL-01 0.5 0 5.5 -0.45 -0.5 0.0 -1.0 0.0

As with a PARALLELBURN specification, but the fuel will transfer to the core to keep it topped off. The stage will drop away when empty. A crossfeeding stage with no engine is a drop tank!

The extra numbers inserted in the middle are, first a feed rate; the rate of consumption that the tank will try to cover. In this case, for a symmetrical pair of tanks, the first one should get a 0.5 feedrate, and the second tank a 1.0, so that both will feed symmetrically until depleted. If there are N tanks, the first should be depleted at rate 1/N, the second at 1/(N-1), and so on.

The second number is an integer; it's what propellant tank we will crossfeed to. Almost

always 0, but we include the ability to change it in case you're trying to feed a complex vehicle with several tanks.

Secondary Payload:

All of the above options have one thing in common: you can only do one of them. For the case of the Saturn V, two attached (non-CVEL) payloads are a requirement; we need a secondary payload attach string. So, one is included. Example:

PAYATCH AS-506_LM_vessel 0.0 -0.0 0. 0.0 -1.0 0.0 1.0 0.0 0.0

This string works exactly the same as the PAYATCH string for CVEL Titans, except that it supports the "long" specification (as in the example here). It is essentially the same as the SERIESBURN command, except that SERIESBURN is followed by a 0 or 1, depending on the attach point you'd like to use. For PAYATCH, you don't get a choice; you're attached to a duplicate of point 0; hence, no attach point after PAYATCH.

The secondary payload gets jettisoned after the primary payload, and after the SLA and fairing, if any. To use it Apollo-style, you'll have to F3 back to the S-IVb stage and jettison the LM to free it up.

The secondary payload does not get its docking port copied the way the primary payload can; see GETDOCKINGPORTS.

Other .scn options:

CONFIGURATION	Normally 0; will be 1 after payload jettisoned.
DELAYSTART	Delays engine start by this many seconds. Actually, engine thrust is set to an extremely low value (1e-9), so that all the things that check for engines running are fooled into thinking we're thrusting.
COUNTINGDOWN	Used internally by Velcro Rockets to save the remaining DELAYSTART time if a scenario save occurs during the delay period.
NO_JETT	Normally 0; a 1 prevents jettisoning of payloads under any circumstances. I needed this for the fixed payload of Saturn SA-1 (Velcro Saturns).
JETT_ON_MECO	Normally 1, meaning that, if using PARALLELBURN, BOOSTERBURN, or CROSSFEED, the stage will jettison if the main engines on the core stage shut down for some reason. Set to 0 if you'd like them to stay attached, such as for aircraft drop tanks.
JETT_ON_EMPTY	Normally 1, meaning that, if using PARALLELBURN or CROSSFEED, the stage will jettison when its tank is empty. Set to 0 if you'd like to override this behavior. This would be for semi-permanently mounted tanks, such as those on a Shuttle-A. If JETT_ON_MECO and JETT_ON_EMPTY are set, the only way to jettison the tank or stage is to F3 to it and hit "J".
IGNITENEXT	Normally 1, meaning that if you're burning the engines when the payload or next stage is jettisoned, its engines will be set to max. Set 0 to suppress this behavior.
IGNITE_WITH_CORE	Normally 1, meaning that, if using PARALLELBURN, BOOSTERBURN, or CROSSFEED, the engines will ignite at the same time as the core. Set to 0 if this behavior is undesirable, for example a carried craft you're going to drop and launch later.

CENTERTHRUST	Normally 0. Velcro Rockets ensures the engine thrust from a stage acts on a point _V(0,0,Z), where Z is the Z-value of the first exhaust definition. In parallel staging, the parallel stage may have a significant offset in the X or Y direction; ordinarily, this is retained. Some other thruster will have to counteract this offset thrust. This behavior is not always desirable, however, as not all add-ons' center of mass is balanced to be acted on in this way. Set CENTERTHRUST to 1 to force our thrust to be on the centerline of the vehicle we're parallel-staging with.
PADBIAS	An offset to the Z-axis of our touchdown points, used to set the vehicle on a pad such as Pad 39A. Added directly to the touchdown point Z-axis, so to raise the vehicle up higher, you need a negative number.
THROTTLEBACK	If acceleration exceeds this value, the vehicle will throttle the engines back to maintain it.
SPINTABLE	A spin rate, in radians/sec, for the stage to assume when it's jettisoned from a lower stage.
TIPOFF	Normally 1. A multiple of the usual rotation rate upon jettison for side-attached parallel stages. Almost never needed, but I had to set it to 0 for the X-15-Blue Scout scenario to simulate the launching trapeze.
GUIDANCE	Just a flag to tell the scenario to re-initialize the autopilot if it's on when saved. You can force this to 1 in the .cfg file, meaning the vessel will default to autopilot-on operation.
ROLLATT	Normally 0; a flag which tells the autopilot to fly to orbit heads-down. If set to 1, autopilot will fly to orbit heads-up.
GETDOCKINGPORTS	Normally 0. If set to 1, your stage will cleverly steal the docking ports from its primary payload, and attach them to the root stage instead. When the stage is undocked, the process is reversed. Use caution, though; because when the docking port is replaced on the payload, its handle doesn't get replicated. This may cause bad effects, depending on how the payload is programmed. Use this if you need to use the payload's docking port while it's still attached to a lower stage; handy for dual-launch Apollo missions and the like. Leave this option off unless it's needed, and leave it off for lower stages, or the docking ports will propogate all the way down the stack.
CAMERA	A vector; location to set the camera for this stage. Unfortunately, the nature of Velcro Rockets means we probably have one stage piled on top of another, which means we'll need to bump up the camera location so that we're not staring at the inside of the payload or fairing while riding to orbit.
TGT_HEADING	Target heading for the autopilot to head for. Not relevant if the stage is not running the show (strap-ons). Upper stages assume the heading they have at staging, so this parameter doesn't matter for them either.
SCREENHEIGHT	Normally 125.0. This is the altitude (in km) at which the autopilot transitions to an altitude-holding mode. This is not the same as a target altitude, but the final orbit should be somewhat above this value. Raise it to obtain a higher orbit.

PITCHMULTIPLE	Normally 1.0. Lower values will cause the autopilot to pitch more slowly. Also used to set higher final altitudes, or to compensate for a known low-powered upper stage.
MINPITCH	Normally 0.0. Higher values set a minimum pitch angle relative to the horizon for the autopilot. Used for lower stages when the next stage is known to be low-powered; bends the trajectory up in anticipation.

Two passes:

Velcro Rockets can interpret lines in a .cfg file, to describe how the stage is built. It can also interpret lines in the .scn file, to describe how the stage will operate. The best part, though, is that, on startup, Velcro Rockets checks through the ENTIRE list of available commands, twice (once for the .cfg, and again for the .scn).

This means that you can set default desired behavior in the .cfg file. For example, an aircraft drop tank we would never want to jettison when the main engine shuts down; ordinarily, we'd set a .scn line to read:

JETT_ON_MECO 0

But instead, we will transplant this line into the .cfg file. Now the tank will always have that flag not set, which is the way we want it. We can always change it back in the .scn file, if we like.

It looks a bit odd there, the only value in the .cfg file without an equals ("=") sign. That's okay; .scn file commands don't have = signs. Even if they're in the .cfg file.

We can over-ride .cfg-file behavior in the .scn file, too. Change the mass, Isp, or thrust of a stage to tweak slightly different values without building a new .cfg. You'll need an = sign here, for these transplanted .cfg-file lines.

Some issues: a quicksaved file will save the .scn-based parameters, but not the .cfg-based ones (with the "=" sign). Plan accordingly!

Known issues:

Stages should be listed in the .scn file in order, top to bottom, so that stages created later can be attached to stages created earlier.

If a carried stage gets suddenly lighter (by staging, payload jettison, etc), a lower stage carrying it has no way of knowing to update its mass. This won't work correctly. If a true staging occurs, this should involve a ClearPropellantResources() call, which will also strip off the propellant of any parallel-burn stages. If JETT_ON_MECO is 1, the parallel-burn boosters will drop off immediately, and this is wasteful of propellant, but nothing too bad will happen. If it is zero...bad things will ensure, and Orbiter will likely crash. Don't do this.

As previously mentioned, the SERIESBURN and other attach strings do not perform any error-checking on the name of the vessel they'll attach to. Get it right, or Orbiter will crash!

Version history:

V1.11

Updated to Orbiter 2010.

Bugfix: Stages were firing engines after a short time at max acceleration.

V1.1

Modified autopilot routine to stay a bit lower longer; it's more energy-efficient.

Modified autopilot routine to add a pitch increase when stage power is low.

Added MINPITCH parameter to modify the minimum allowed pitch of the autopilot. Add to a

first stage to compensate for a low-powered second stage.
Bugfix: BOOSTERBURN stages were identifying multiple tanks to feed from when attached to a vessel with multiple same-size cores.
Bugfix: ASRM mesh was too big.
Bugfix: autopilot wasn't properly setting target heading equal to heading on staging.

V1.01
Bugfix: quicksave doesn't stitch expended stages back on.
Added Mayfly = and Become = parameters.

V1.0
First public release! Fixed a bug reading/writing CROSSFEED.

V0.91
Updated beta release. GETDOCKINGPORTS and payload carriage works correctly.

V0.9
Mostly-final beta release. Saturns broken out into separate project.

V0.3
Third beta release. Added BOOSTERBURN and CROSSFEED modes, drop tanks.

V0.2
Second beta release. Added first Velcro Saturns.

v0.1
First beta release.