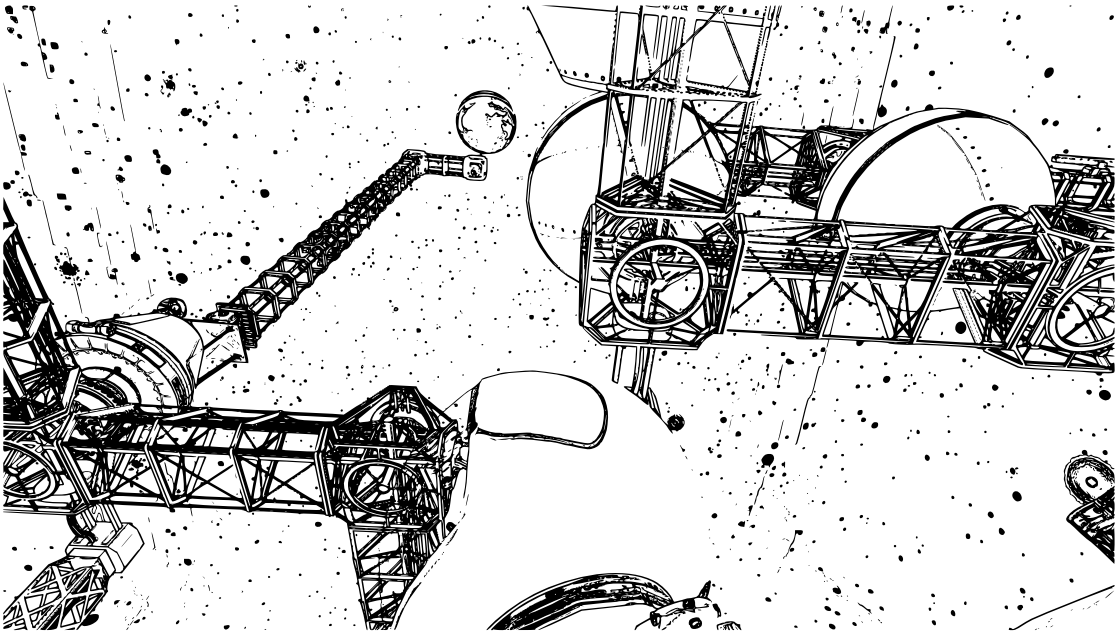
Extraplanetary Launchpads

User's and Modder's Guide

Bill Currie (taniwha)



Introduction

Building bases and space stations in Kerbal Space Program can be fun and rewarding on its own: they look good and allow for the production or storage of fuel and science, but they amount to little more than outposts. One main problem with such outposts is that when things go wrong and repairs are needed, they are highly dependent on resupply runs from the KSC[1]. A bigger problem is they serve only as way stations for grand missions: it is very difficult to use them as the origin of such missions as all the vessels comprising the grand mission must be launched from the KSC.

Extraplanetary Launchpads (or EL for short[2]) gives additional meaning to planetary bases and orbiting space stations by allowing for the construction of all manner of vessels away from the KSC. The construction can be carried out both on the surface of any body and in orbit. However, EL does not do anything for life support supplies (other mods have that covered), or the expansion of the kerbal population at the base or station.

EL defines the resources[3] used in the construction of vessels, and provides the parts required to obtain and process those resources into the final product: a vessel that can be an independent ship (or other vehicle), base, station, a module for a larger vessel, or even individual parts[4].

[1] Kerbal Space Center? Kerbin Space Center? Kerman Space Center? Kraken Snack Constructor?

[2] Just like "extraterrestrial", "extraplanetary" is (or would be) one word, and "launchpad" is also one word, thus in this case TLA is Two Letter Acronym.

[3] Other mods can define other resource that replace those defined by EL.

[4] Though currently a little awkwardly.

Contents

List of Figures

List of Tables

Part I.
Using Extraplanetary Launchpads

1. Getting Started

1.1. Installation

1.1.1. Minimal Installation

Download the zip file for the latest version via the EL forum thread[**EL**] and extract its contents to KSP's `GameData` directory[5]. Ensure that the latest version of Module Manager[**MM**] is installed correctly.

Note that the minimal installation will not support survey builds.

1.1.2. Survey Build Support

In order to support survey builds, Kerbal Inventory System[**KIS**] is required.

1.1.3. Recommended Mods

There are several mods that improve EL:

Kerbal Alarm Clock[KAC] Keep track of when a build will be finished. Really, the most important mod for anybody wishing to run more than one mission at a time.

KerbalStats[KS] Provides a means to extend kerbal attributes, but in the context of EL it provides time and activity based experience for kerbals.

Kethane[Keth] The original ISRU solution for KSP. Provides hot-spot mining of MetalOre.

Modular Fuel Tanks[MFT] Edit tank resources in the editor.

Talisar Parts[TP] High capacity spherical tanks (up to over $200m^3$) and various structural parts.

Kerbal Attachment System[KAS] Pipes and winches.

TAC Fuel Balancer[TACFB] Easier transferal of resources.

1.1.4. Mods that support or use EL

Kerbal Planetary Base Systems[kpbs] Several new parts that are designed to be used in a planetary base for the kerbals.

[5] Ok, folder. Now get off my lawn ;)

Pathfinder[Path] Space Camping & Geoscience.

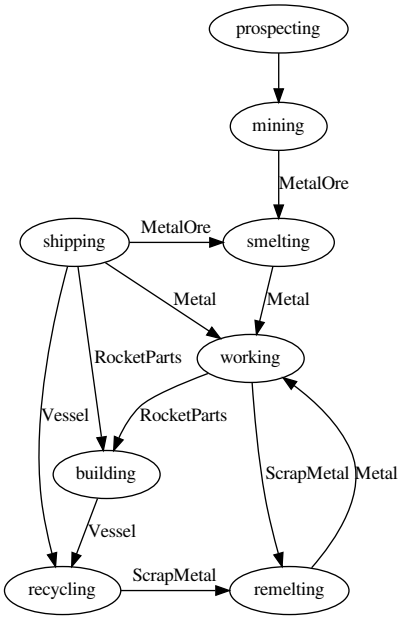
SimpleConstuction[SC] Uses EL but stock parts.

1.2. Setup

The first time the space center scene is entered, an options window will be presented allowing for the selection of various settings:

- Use of Blizzy's toolbar when available.
- Creation of KAC alarms and the default action for those alarms.
- Visibility of the build resources window in the editors.

2. Construction Basics



2.1. Resources

Currently, EL uses four resources for its production chain (though the recipe system (see section 5.2 on page 27) allows for much more complicated systems).

MetalOre Assumed to be hematite[6] (but not explicitly stated as such) and thus has a density of $5.2t/m^3$.

[6] A common form of iron ore.

Metal Assumed to be iron (but not explicitly stated as such) and thus has a density of $7.8t/m^3$.

RocketParts Assumed to be sub-parts ready for assembly into actual parts, and thus has a very low density of $0.5t/m^3$.

ScrapMetal The true product of any machine shop: all machine shops produce scrap metal in various forms and efficiencies. The lumps of metal handed over to the customer are really the left-overs from producing scrap metal. Scrap metal generally does not pack well, though better than parts, so a density of $0.8t/m^3$ was chosen as an average.

2.1.1. Prospecting and Mining: dirt? to MetalOre

In order to obtain MetalOre when away from the KSC, one of the augers is used to mine MetalOre from the surface of the planet or moon. EL uses the stock resource distribution system configured to distribute MetalOre, so prospecting is done as for stock's Ore resource, but with a focus on MetalOre instead.

Kethane and Karbonite Prior to KSP 1.0, EL relied solely on Kethane[**Keth**] for its prospecting and mining, and there was an adaptation to make EL use Karbonite[**karb**] instead.

As of KSP 1.0 (EL 5.1.90) Kethane is completely optional, but if present, will be used on top of the stock resource system. Scanning is quite separate, but mining is done using the exact same augers. Mining outside a MetalOre deposit created by Kethane will extract MetalOre at the rate dictated by the concentration given by the stock system (1% to 15%), but deposits created by Kethane effectively provide hot-spots of 100% concentration.

The Karbonite adaptation seems to have been mothballed, but it was mostly a parts mod with configs for EL, so it may still be usable.

It may be worthwhile thinking of the stock system providing a means to extract MetalOre from a larger mix of "dirt", while the Kethane system provides access to rich veins of MetalOre.

2.1.2. Smelting: MetalOre to Metal

MetalOre is converted to Metal via smelting. Smelting is the process of reducing[7] metal oxides. EL assumes MetalOre is Fe_2O_3 (the most common iron ore on Earth). Reducing Fe_2O_3 is a three-step process (from Wikipedia):

Stage One $3Fe_2O_3 + CO \rightarrow 2Fe_3O_4 + CO_2$

Stage Two $Fe_3O_4 + CO \rightarrow 3FeO + CO_2$

Stage Three $FeO + CO \rightarrow Fe + CO_2$

[7] Chemistry term, the opposite of oxidizing (or reduction vs oxidization).

However, this really happens all at once in a smelter so the effective process is:

$3Fe_2O_3 + 9CO \rightarrow 6Fe + 9CO_2$.

Fe has a molar mass of $55.845g/mol$, O has a molar mass of $15.9994g/mol$, so $479.0646g$ of Fe_2O_3 will produce $335.070g$ of Fe. This leads to a MetalOre to Metal mass conversion rate of 0.6994255[8].

In order to model the CO consumption, EL assumes that LiquidFuel is RP-1 ($C_{12}H_{16}$) and that Oxidizer is liquid oxygen (O_2). The stoichiometric equation for burning RP-1 is:

$C_{12}H_{16} + 16O_2 \rightarrow 12CO_2 + 8H_2O$

However, rockets tend to burn rich (reduced oxidizer quantity) in order to avoid oxidizing the engine, keep the engine cooler, and increase the number of lighter molecules in the exhaust. KSP's standard LiquidFuel and Oxidizer mix of 9:11 by volume very closely approximates the following when LiquidFuel is assumed to be RP-1 and Oxidizer is assumed to be liquid oxygen[9] (used for 0% efficient smelting):

$2C_{12}H_{16} + 17O_2 \rightarrow 2C + 17CO + 4CO_2 + CH_2O + 7H_2 + 8H_2O$

The resulting equation for 100% efficient[10] smelting thus becomes:

$10Fe_2O_3 + 2C_{12}H_{16} + 17O_2 \rightarrow 20Fe + 24CO_2 + 16H_2O$

$C_{12}H_{16}$ has a molar mass of $160.25544g/mol$, CO_2 has a molar mass of $44.0095g/mol$, H_2O has a molar mass of $18.01528g/mol$, C $12.0107g/mol$, CO $28.0101g/mol$, CH_2O $30.02598g/mol$, H_2 $2.01588g/mol$. Thus the two equations become the following when viewed as masses:

$320.51088 + 543.9796 \rightarrow 24.0214 + 476.1717 + 176.038 + 30.02598 + 14.11116 + 144.12224$

$1596.882 + 320.51088 + 543.9796 \rightarrow 1116.9 + 1056.228 + 288.24448$.

The mass conversion rate of MetalOre to Metal is still 0.6994255.

Smelting is assumed to consume $8.62MJ/kg$ of produced Metal[**TMETPS**].

The combustion of LiquidFuel and Oxidizer is assumed to produce $6MJ/kg$. The justification for this comes from analyzing the properties of KSP's stock LFO engines and noting that the Poodle (the most efficient) gets $5.9MJ/kg$ of LiquidFuel and Oxidizer (see table 1 on the next page).

Thus, EL's smelters nominally[11] consume $864.49048g/s$ of LiquidFuel and Oxidizer to produce $5186.94288kJ/s$ of heating along with the products necessary for smelting. At 100% efficiency, they consume $1596.882g/s$ of MetalOre and $9627.678kJ/s$ of heat while producing $1116.9g/s$ of Metal. Thus 100% efficiency is unobtainable as there is insufficient heat supplied by the LFO mix. In theory, the attainable efficiency is a bit less than 53.9%, but due to transferred resources cooling the smelter[12], the maximum

[8] 0.46628367 volume (resource unit) conversion rate.

[9] Note that the exact ratios of the products were arbitrarily chosen but are chemically "correct" (balanced), and that the inclusion of formaldehyde is the result of "Hmm, wonder if there's a CH_2O . Oh, there is, and it does appear in partial combustion of hydrocarbons. Neat."

[10] Arbitrary assumption.

[11] The actual rates vary with the smelter: $800g/s$, $3149.8g/s$ and $5000g/s$ of LFO, with the other rates scaled accordingly.

[12] And an engineering mistake in the smelter design. Some kerbals will miss out on snacks.

Engine	Thrust (N)	Isp(s)	Ve (m/s)	flow (kg/s)	power (W)	Esp (J/kg)
24-77	16k	290	2843.9	5.6	22.8M	4.0M
48-7S	20k	320	3138.1	6.4	31.4M	4.9M
aerospike	180k	340	3334.2	54.0	300.1M	5.6M
LV-1	2k	315	3089.1	0.6	3.1M	4.8M
LV-1R	2k	290	2843.9	0.7	2.8M	4.0M
LV-909	60k	345	3383.3	17.7	101.5M	5.7M
LV-T30	240k	310	3040.0	78.9	364.8M	4.6M
LV-T45	215k	320	3138.1	68.5	337.3M	4.9M
mainsail	1500k	310	3040.0	493.4	2.3G	4.6M
Mk55	120k	305	2991.0	40.1	179.5M	4.5M
poodle	250k	350	3432.3	72.8	429.0M	5.9M
SSME	1000k	315	3089.1	323.7	1.5G	4.8M
skipper	650k	320	3138.1	207.1	1.0G	4.9M
twin-boar	2000k	300	2942.0	679.8	2.9G	4.3M
rhino	2000k	340	3334.2	599.8	3.3G	5.6M
mammoth	4000k	315	3089.1	1294.9	6.2G	4.8M
rapier	180k	305	2991.0	60.2	269.2M	4.8M
vernor	12k	260	2549.7	4.7	15.3M	3.3M

Table 1: Comparison of stock KSP LFO engines.
$Thrust$ and I_{sp} are from the config files. V_e (exhaust velocity) is $I_{sp} * g_0$ (9.8066). Flow is $\frac{thrust}{V_e}$. Power is $\frac{1}{2} thrust * V_e$. E_{sp} (energy per kg) is $\frac{power}{flow}$ or $\frac{1}{2} V_e^2$.

efficiency appears to be about 40%.

2.1.3. Working: Metal to RocketParts

Metal is converted to RocketParts by working it. Currently, this is done using either the workshop (big blue part in Utilities), or the workbench (tower with little platforms in Pods). Unfortunately, the process is quite bogus: Metal is used for for everything, and the conversion speed is probably too fast. However, the efficiency (0.7 by mass) is reasonable: it is the estimated average of various means of production: cutting cast iron parts leads to high efficiency, but cutting lumps of steel can lead to fairly low efficiency depending on just how much metal needs to be cut away. At the same time, ScrapMetal is produced at a rate of 0.295 by mass (some scraps are lost).

2.1.4. Remelting: ScrapMetal to Metal

ScrapMetal can optionally be remelted to Metal using a smelter. The process is lossless (conversion rate of 1), the loss (very small) occurs when producing the ScrapMetal. Storing and reclaiming ScrapMetal is fully optional: running out of storage will not stop Metal to RocketParts conversion.

2.1.5. Building: RocketParts to Rockets

Building is done by the launchpads, orbital dock, or survey station (or just "pads" for short). The rate is governed by the overall vessel productivity (measured in kerbal-hours (Khr)) shared amongst active pads. Each ton of rocket (dry-mass) requires five kerbal-hours (i.e. $5Kh/t$).

There have been discussions that EL's build rate is too high compared to Kerbal Construction Time, but those arguing that side were unaware that RocketParts represent components to be assembled into the parts visible in KSP. The building process is really just the kerbals putting those components together. It is the smelting and working stages that are unrealistically fast.

2.1.6. Recycling: RocketParts to ScrapMetal

Recycling is done on a part-by-part basis. When a part is recycled, it is first drained of any resources it contains (e.g., LiquidFuel or Oxidizer), and those resources will evaporate, be broken down into other resources to be reclaimed, or transferred and thus reclaimed depending on the resource (most will be transferred). Once the part has been drained, it will be broken down from RocketParts to ScrapMetal and the ScrapMetal reclaimed.

Of course, any reclaimed resource needs storage space. Otherwise, it will be lost.

Some of you may have had spotty results recycling entire vessels: that is intentional, and there is a way around it (see if you can guess).

2.2. Productivity

All kerbals have a base productivity score determined by their stupidity, courage, and bad-ass characteristics. The more stupid a kerbal is, the less that kerbal will contribute

to the workshop's (and thus the overall vessel's) productivity, and more courageous kerbals will, in general, contribute less than less courageous kerbals, though bad-ass kerbals complicate the relationship. It is entirely possible for a kerbal to have negative productivity.

If the KerbalStats[**KS**] mod is installed, then the amount of time a kerbal has spent in certain workshops (currently only EL's blue workshop (afaik)) improves the kerbal's productivity.

A workshop's productivity is the sum of the productivities of all kerbals working in that shop. A vessel's productivity is the sum of the productivities of all workshops in that vessel. If the vessel's productivity is greater than zero, then construction will progress. Negative productivity does not cause production to become destruction, instead it causes a productivity deficit that must be overcome by better construction kerbals before construction will proceed.

2.3. Construction Skill

Kerbals with the construction skill (by default, engineers, but hereon referred to as construction kerbals) are the cornerstone of workshop productivity. However, their space-faring (stock) experience affects their productivity greatly.

0 stars The kerbal can work in a fully equipped workshop.

1 star The kerbal can work in any workshop.

2 stars The kerbal is always productive in a fully equipped workshop (base productivity still matters, but to get negative productivity, the kerbal would have to have infinitely negative base productivity).

3 stars The kerbal is always productive in any workshop.

4 stars The kerbal enables skilled workers in any workshop (a 4-star construction kerbal in an under-equipped workshop allows 0-star construction kerbals to contribute).

5 stars The kerbal enables unskilled workers in a fully equipped workshop (a 5-star construction kerbal in a fully equipped workshop allows any kerbal, even those without the construction skill, to contribute).

2.3.1. Unskilled kerbals

Unskilled kerbals cannot work unless a 5-star construction kerbal is present in the same workshop, and the workshop must be fully equipped, but if the kerbal's experience level is 2 or less, and the kerbal's base productivity is negative, the kerbal will detract from the workshop's productivity.

2.3.2. Non-career modes

In sandbox (and science?) mode, all kerbals are level 5, so there will be no negative contributions, and if there is at least one construction kerbal in the workshop, then all kerbals of sufficient ability will contribute.

2.4. Workshops

Workshops, too, affect productivity. All workshops have a productivity factor that is multiplied by the sum of the productivities of the kerbals working in that shop. The resulting productivity is then passed to the vessel.

2.4.1. Fully equipped

Fully equipped workshops are those with a productivity factor of 1.0 or more, or specially marked workshops with lower productivity factors. EL's blue workshop, and the rocket workbench are both fully equipped workshops.

For role-play purposes, "fully equipped" can be thought of as the workshop having all the necessary tools, and the productivity factor as being the quality of the tools or the level of automation available.

2.4.2. Other parts

All stock crewed parts act as under-equipped workshops. In addition, all crewed command pods, including those from other mods, act as under-equipped workshops. Many base-building mods (eg, KPBS[**kpbs**], Pathfinder[**Path**]) provide workshops (refer to those mods for details).

2.5. Pads

All construction is done at "pads", whether the pad is an orbital dock, a launchpad, a survey site (marked out using survey stakes and managed by a survey station), or a micro-pad.

Initiating construction is the same for everything: open the build window (via either the toolbar button (blizzy's toolbar[**TB**], or the stock app button), or the Show UI button in the PAW[13]), select the craft to build, and press the Build button. Between selecting the craft to build and pressing the Build button, the required and optional resources for the build will be displayed in a preview. There is no need to have all the required resources on-hand when beginning the build: if they run out during the build, the build will stop until the resources become available and then automatically resume. The resources can become available via supply runs or local processing.

2.5.1. Launchpads and Orbital Docks

Technically, there is no difference between a launchpad and an orbital dock: they operate exactly the same way. The difference comes in the physical form of the parts: launchpads

[13]Part Action Window (right-click menu)

are optimized for grounded operation, and the orbital dock is optimized for orbital operation.

Adjusting the optional resources in the preview will set the defaults for the amounts to be transferred upon release.

2.5.2. Survey Stations and Survey Stakes

Survey stations use local survey sites to specify the location and orientation of the built vessel. Survey sites are sets of one or more survey stakes with the same name and within range (200m) of each other.

Adjusting the optional resources in the preview will have no effect as no resources will be transferred.

A stake's name defaults to the name of the kerbal who planted it with "Base" appended. Thus if Valentina plants a stake, it will be named Valentina Kerman Base. Thus, when creating a site consisting of more than one stake, it is easiest to have only one kerbal do the stake planting. Also, if multiple local sites are desired, getting a different kerbal to plant the stakes for each site will make it easier.

2.5.3. Micro-Pad

The micro-pad is a single-use construction point. When the build is finalized, the pad self-destructs and the build is attached to the parent vessel as if the build had been placed there in the editor (VAB or SPH). Also, the micro-pad can be carried on a kerbal's back using KIS[**KIS**].

The orientation and position of the build is controlled by the combination of the orientation and position of the micro-pad, and the automatically selected attach node of the root part. The colored diamonds on the micro-pad indicate the orientation of the root part when its top node is selected (red = +X, blue = +Z, cyan = -X, yellow = -Z[14]. It is important to remember that when the micro-pad is facing up, the root part's selected node will be facing down, having been rotated around the part's Z axis.

The micro-pad automatically selects the attach node of the root part by searching for the first available (unattached) node. First the top node is checked, then the bottom node, then any remaining nodes in the order they are found. Thus a small amount of control over which node is selected can be obtained simply by attaching parts to the undesired nodes.

Adjusting the optional resources in the preview will have no effect as no resources will be transferred. Nor is there any need for them to be automatically transferred.

3. Survey System

When landed, orbital docks can be awkward for building as they tend to be on top of the building vessel (especially awkward for building rovers as getting the rover to the ground can be an issue), and launchpads are highly sensitive to ground conditions, and and have their own issues when building large vessels. Also, they provide no flexibility in placement or orientation of the build.

[14] Hopefully the colors are distinct enough that any suffering from colorblindness can distinguish at least one diamond. The pad's default orientation in the VAB is such that red (+X) points to the VAB door, blue (+Z) points to the north wall, magenta (-X) to the west and yellow (-Z) to the south.

EL's survey system greatly eases the seeding (or even complete build-out) of bases, and works equally well for building ships and other vessels. However, it does have one disadvantage: any optional resources (liquid fuel, oxidizer, electric charge, etc) will *not* be transfered: the build will be empty of such resources (freedom is not free), but as KIS[**KIS**] is required to place the stakes, and KAS[**KAS**] is almost always installed with it, this disadvantage should be only minor[15].

The survey system consists of two parts[16]: the survey station, and the survey site. The survey station (a re-purposed hitchhiker can) is used to keep track of the survey sites and do the actual building (it serves the same purpose as the orbital dock or a launchpad, but must be landed), and must be flown down to the surface in the vicinity of where the builds will occur. The survey site is ephemeral: it is marked out by one or more survey stakes and is used to specify the location and orientation of the build.

3.1. Survey Station

3.1.1. Survey Skill

Kerbals with the survey skill (by default, pilots, but hereon referred to as survey kerbals) affect the range of a survey station according to their stock experience level. The experience level of the most skilled survey kerbal in the survey station is used. However, even an unskilled kerbal can man a survey station, and an unmanned survey station is still operational.

unmanned 20 meters.

unskilled 50 meters.

0 stars 100 meters.

1 star 200 meters.

2 stars 400 meters.

3 stars 800 meters.

4 stars 1600 meters.

5 stars 2000 meters.

Note that the range is from the survey vessel's center of mass to the nearest stake of the survey site, but stakes may be separated by up to 200 meters[17].

[15] It can, however, lead to good entertainment: [**HMV1**]

[16] If you're thinking KSP parts, then it's three: survey station, survey stake, and mallet.

[17] This is a bit of a misfeature: the range should be from the survey station part and the maximum separation of the stakes should be dependent on the skill as well.

3.2. Survey Site

Stakes have two modes with seven settings in each mode (default is Direction:Origin):

Direction these are used to control the orientation of the build.

- **Origin** used to mark the location above which the build's root part will be placed, and also forms the reference point for other direction stakes that aren't in pairs.
- **-X and +X** used to specify the lateral (port (-X) and starboard (+X)) axis of the build (both VAB and SPH). If both -X and +X are used, then the origin is is ignored, otherwise the axis runs from -X to origin or origin to +X.
- **-Y and +Y** used to specify the "vertical" (nadir (-Y) and zenith (+Y)) axis of the build (relative to the floor in the VAB or SPH). If both -Y and +Y are used, then the origin is is ignored, otherwise the axis runs from -Y to origin or origin to +Y. NOTE: not recommended, very advanced usage.
- **-Z and +Z** used to specify the ventral(+Z)/dorsal(-Z) (VAB) or fore(+Z)/aft(-Z) (SPH) axis of the build. If both -Z and +Z are used, then the origin is is ignored, otherwise the axis runs from -Z to origin or origin to +Z.

The VAB orientation really is weird.

- ***** If none of the axis direction stakes are used, then the default orientation is such that the build's +Y axis is the local up, +X axis points east, and +Z points north (same as on the KSC launchpad).
- ***** If the axes marked out by the stakes are not perfectly orthogonal, then the build will be oriented such that the errors are balanced.

Bounds these are used to control the placement of the build based on its bounding box rather than its root part.

- **Origin** used to mark the location of the root part along any axis that has not been bound.
- **-X and +X** used to mark the lateral (port (-X) and starboard (+X)) edges of the build. If only one of -X or +X is used, then that edge of the build will be exactly on that stake, otherwise the the X-axis center of the build's bounding box will be centered on the midpoint between the two stakes.
- **-Y and +Y** used to mark the "vertical" (nadir (-Y) and zenith (+Y)) edges of the build. If only one of -Y or +Y is used, then that edge of the build will be exactly on that stake, otherwise the the Y-axis center of the build's bounding box will be centered on the midpoint between the two stakes. NOTE: use of the +Y bounds stake is not recommended unless you know what you are doing.
- **-Z and +Z** used to mark the ventral(+Z)/dorsal(-Z) (VAB) or fore(+Z)/aft(-Z) (SPH) edges of the build. If only one of -Z or +Z is used, then that edge of the build will be exactly on that stake, otherwise the the Z-axis center of

the build's bounding box will be centered on the midpoint between the two stakes.

* Bounds stakes and direction stakes work together: any unbound axis of the build slides along that axis of the reference frame created by the direction stakes (or the default frame if no direction stakes are used).

* There is actually only one origin stake: there is no difference between a bounds origin stake and a direction origin stake. The appearance of there being two origin stakes is due to the overly simple controls.

* If multiple stakes of the same type+setting have been placed, then they will be averaged together to form a virtual stake of the same type+setting. This can be very useful with multiple origin stakes to avoid the build clipping into the stake when the lowest part of the build is directly below the root part.

* If no origin stakes have been placed, then the average of all other stakes is used as the origin point.

* The actual location of the stakes is about 19cm above the ground.

* If no Y bounds stake has been placed, then the origin acts as an implicit -Y bounds stake (otherwise almost all builds would spawn in the ground).

4. Designing a construction capable vessel

4.1. Orbital Construction

Orbital construction is probably the easiest to get going as all that is required is a supply of RocketParts, an orbital dock, some qualified construction kerbals, and somewhere for them to work. This means that only two parts need to be added to the design of a station that has crewed command pods or any stock crewed part[18]: a reasonably sized RocketParts container, and either the orbital dock or the micro-pad. When extending an existing station, a suitable docking port would be required as well, and means to get the "construction unit" to the station. However, if the micro-pad is used, then the micro-pad needs to be attached to the station somewhere using KIS[**KIS**]. In fact, the micro-pad can be an excellent way of attaching an orbital dock to a station.

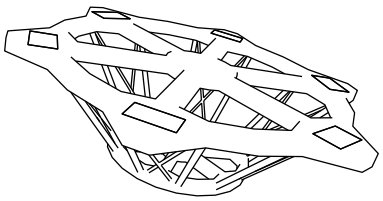


Figure 2: Orbital Dock

It can, however, be the most difficult to maintain due to the need for supplies to be flown to the station. That said, maintaining an orbital construction station can be quite interesting as there are a number of options for feeding RocketParts into the orbital dock's ravenous maw, each with additional design requirements for the station.

[18] there are mods that supply suitably configured crewed parts

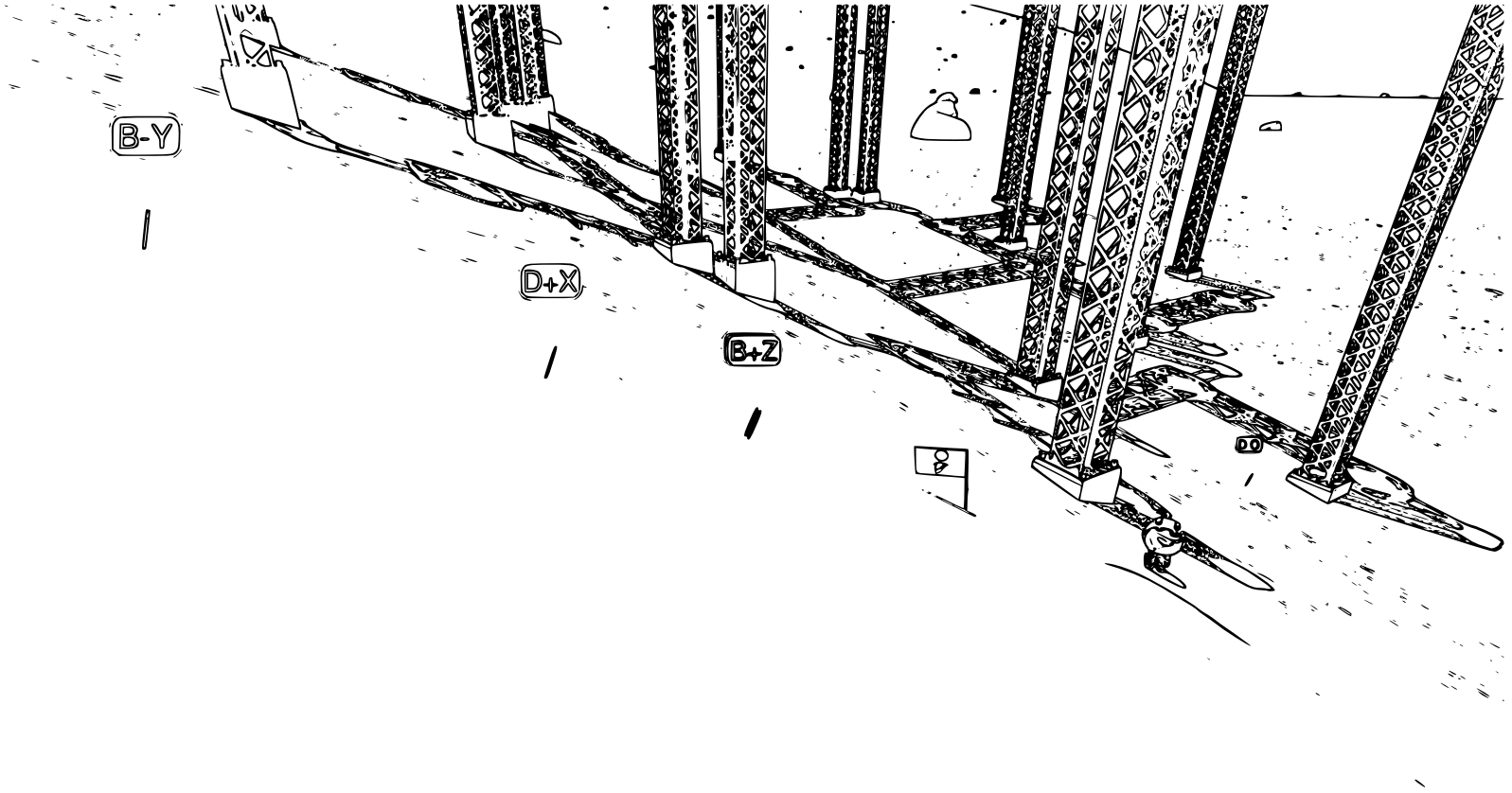


Figure 1: Hillside survey site, somewhere on Minmus.
This is one possibility for placing stakes for building a large structure on the side of a hill. From left to right, there's a -Y Bounds stake, a +X Direction stake, a +Z Bounds stake with a +X Bounds stake hiding behind it, a flag and kerbal (not relevant to the site), and near the far right launch clamp, an Origin stake (direction, but bounds would have the same effect). Although the top of the structure is not directly visible, it is horizontal.

- Firstly, and most simply, RocketParts can be flown up from a nearby, or even not so nearby, production base[19]. The only additional requirement for the station is an available docking port, but a station with no docking ports is of rather limited use, so this requirement is generally not too troublesome (though having enough docking ports on a station can be tricky at times).

- Secondly, Metal can be flown to the station and locally processed into RocketParts. Along with an available docking port, this method requires a means of converting Metal to RocketParts. This can be done using either the workshop or the rocket workbench, but since even the rocket workbench can give a significant boost to production speed, the only real disadvantages are the relevant part needs to be either flown up to the station and docked or built at the station and maneuvered into a convenient location, and the increased power requirements. A minor disadvantage is the conversion of Metal into RocketParts produces ScrapMetal. Normally, the ScrapMetal is simply thrown away, but the next solution takes care of that.

  The rocket workbench can be manhandled into position by two kerbals using KIS.

- Thirdly, if Metal to RocketParts production is already available on the station[20], a smelter can be added to the station allowing MetalOre to be flown in and processed into Metal. While this does significantly increase the fuel requirements of the station, the smelter is actually dual-use: not only can it convert MetalOre into Metal, but it can convert ScrapMetal into Metal, resulting in very low production losses. As an added bonus, the smelters have a small amount of storage for the three resources.

- Fourthly, and finally, a recycling bin can be added to the station. EL's recycling bins do not look like much, but they are ravenous maws that eat vessels of any size (except asteroids, but including unfortunate kerbals) and spit out resources. Most resources, except Ablator and SolidFuel, stored in tanks will be reclaimed as-is without loss, and the hull material will be recycled into ScrapMetal. For best results, the station needs large quantities of storage and a full production chain. The biggest advantage of using a recycling bin is unmanned[21] supply ships can be flown into the recycling bin and recycled. If there is sufficient storage on the station, then all remaining fuel and any stored resources will be automatically transfered to the station, and the hull of the supply ship will go towards the next construction project. Unfortunately, if there is insufficient storage for any resource, the the excess of that resource will be lost.

4.2. Grounded Construction

[19]The KSC counts as a production base as all EL resources can be loaded onto a vessel at launch time via tweakables.
[20]Or is included in the same module as the smelter.
[21]Or manned by very brave pilots.

While more difficult to get going due to the need to fly to the location and land safely, grounded construction can be much easier to maintain as the site of the construction base can be chosen for optimal resource extraction. Thus once production facilities (as described in the orbital construction section) are in place, the base can be self-sufficient for a very long time.

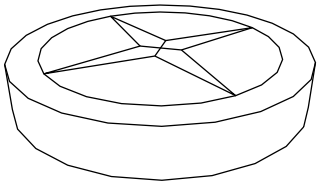


Figure 3: Micro-Pad

The main difference between orbital construction and grounded construction is that gravity can make getting built vessels off the pad rather awkward. Thus, instead of a launchpad, though launchpads can have their advantages (such as resource transfer), the base can be equipped with a survey station, a supply of survey stakes, a mallet to drive the stakes, and a KIS[**KIS**] container in which to store the stakes and mallet. As KIS is required for the survey parts to be available, the container should not be a problem[22].

4.3. Roughing It

NOTE: this section is out of date as it is relevant for when smelters required only ElectricCharge and not LiquidFuel and Oxidizer. However, it should still server as inspiration.

This is grounded construction the hard way[23]. The absolute minimum is flown to a site and the first things constructed are the modules needed to sustain construction.

Parts and resources to be flown in:

Suvery Station Needed to do the building on the marked site. This doubles as a workshop, though productivity is poor.

Survey Stakes Needed to mark where to build.

Mallet Needed to drive the stakes.

RocketParts Only a very small quantity is needed: $3.5328t$ is the minimum, though a little more is recommended.

KAS[KAS] Pipe Connectors Needed to attach the build modules to allow resource transfer.

KIS[KIS] Screwdriver While the wrench may be sufficient for attaching pipe connecters, the screwdriver is more versatile.

[22] Other mods might provide alternative "stakes" with a different planting mechanism thus removing the need for KIS.

[23] My preferred way.

Modules to be built first:

Tiny Auger $0.5328t$ This is the smallest auger supplied by EL. Needed to extract MetalOre.

Tiny Smelter $1.5t$ This is the smallest smelter suppllied by EL. Needed to convert MetalOre to Metal.

Rocket Workbench $1.5t$ While this is needed for converting Metal to RocketParts, it is also a fully equipped workshop with room for four kerbals. Once this is built, construction kerbals can be moved from the survey station to the workbench, significantly increasing productivity and thus allowing later builds to progress at a faster rate.

Once the required modules have been built, base production can go into full swing, even the construction of larger resource production chain, including recycling and the full-sized workshop.

Part II.
Modding Extraplanetary Launchpads

5. Configuration

5.1. Part Modules

For the most part, EL places no restrictions on the models used for parts using EL's module, so unless otherwise stated, models are completely free-form as far as EL is concerned.

5.1.1. ELControlReference

Allows the part to become a control reference ("Control From Here" in the part's PAW). Also provides a "Toggle Reference" action group that saves the previous control reference when becoming the reference and returns the vessel to the previous reference when toggled again.

Model Requirements The model's mesh needs an emissive material with `_EmissiveColor` set with appropriate RGB values. The alpha value is used for turning the emission on and off as a status indicator. The part's root game object is used as the reference transform consistent with command pods and docking ports, so the model should be shaped appropriately.

Part Requirements None

Module Fields None

5.1.2. ELConverter

Consumes a set of input resources to produce a set of output resources. An EL converter uses a converter recipe (see 5.2.3 on page 31) to specify the inputs and outputs while controlling the rate of flow of the resources specified by the active input recipe and distributing the recipes specified by the active output recipe. The active input and output recipes are selected by the current efficiency of the converter, and may be a blend between two specified recipes. The efficiency of the converter is either fixed, or determined by the temperature of the converter part.

`ELConverter` subtracts the total input heat from the part and adds the total output heat to the part (use negative heats to have heating associated with input ingredients and cooling with output ingredients).

If an efficiency curve[24] has been specified, then the converter's efficiency is determined by the part's internal temperature (otherwise the efficiency is constant). The converter's efficiency is then used to bake the converter recipe into a specific conversion recipe.

Once the specific conversion recipe has been obtained, `ELConverter` then bakes that recipe using the current mass flow rate. The current mass flow rate is the specified rate of the converter scaled by the relative mass of the active input recipe compared to the smallest mass input recipe in the converter recipe. This keeps the mass flow of certain resources constant while varying the flow of others appropriately[25] when suitable ratios are specified. The mass flow rate of the output recipe is always the same as that of the input recipe[26].

Note that KSP's thermal data in the PAW does not directly show the effects of resource transfer on the part's temperature, so there may be significant thermal flux despite the temperature being fairly stable.

Model Requirements None

Part Requirements None. However, it is strongly recommended to use appropriate thermal settings for converters that use heat. In particular, `skinInternalConductionMult` and `heatConductivity` should be adjusted to provided sufficient insulation when keeping the part hot is necessary.

Module Fields

EVARange Adjust the range (in meters) from which an EVA kerbal can activate or deactivate the converter. Default is $1.5m$.

ConverterRecipe The name of the converter recipe to use. Required.

[24] For certain values of "curve": it's piece-wise linear.

[25] For example, in a smelter, LiquidFuel and Oxidizer flow at a constant rate, while MetalOre is not consumed at all at 0% efficiency, but is consumed at the maximum rate at 100%. Thus the actual mass flow rate might vary from 864.49048g/s to 2461.37248g/s.

[26] Conservation of mass.

Rate The base mass flow rate of the converter, in kg/s. Defaults to 0. The base mass flow rate is that of the smallest mass input recipe in the converter recipe.

efficiency Specify the efficiency/temperature curve of the converter. Defaults to constant 1.0 (100%). If a single `efficiency` entry is given, then it specifies a constant efficiency and is of the form `efficiency = 0.9` (90%). If multiple EFFICIENCY entries are given, then they form keys in a piece-wise linear "curve". Each entry is of the form `efficiency = temp, eff` where `temp` is the part's internal temperature in Kelvin and `eff` is the converter's efficiency at that temperature. Note that the keys need to be specified in ascending order of temperature and `eff` should be between 0.0 and 1.0 (inclusive)[27]. For temperatures outside the range of the curve, the nearest key is extrapolated with a constant efficiency.

5.1.3. ELDisposablePad

Special one-shot launchpad that replaces itself with the built vessel, attaching the built vessel to the vessel owning the pad.

Model Requirements The only requirement is the model provides a transform with the up axis (positive Y-axis in KSP/Unity, Z-axis in Blender) pointing ***away*** from where the spawned vessel will be as it will be used to align the selected attach node of the vessel's root part. This is the opposite direction of the transform used by `ELLaunchpad`.

Part Requirements None.

Module Fields

SpawnTransform Specifies the model transform to be used as the launch transform. Optional, but using a spawn transform makes the virtual attach node independent of the part's transform.

PadName Specifies the name of the launchpad. Note that this is editable by the user both in the editor (VAB/SPH) or in flight.

Operational Persistent backing for `ELControlInterface.canOperate`. Defaults to `true`.

5.1.4. ELExtractor

Extracts resources from the environment.

5.1.5. ELLaunchpad

Builds complete vessels attached (pseudo-docked) to the current vessel. Allows post-build resource transfer without any extra fuss. Supports building both landed or in orbit.

[27] No checking is performed, so expect rhinodaemons should this rule be broken.

Model Requirements No requirements, but it is highly recommended that the part has plenty of free space "above" (positive Y-axis in KSP/Unity, Z-axis in Blender) the launch transform.

Part Requirements None.

Module Fields

SpawnHeightOffset Specifies the distance in meters above the launch transform of the lowest point of the spawned vessel. This is most useful when the model does not have a specific spawn transform. Defaults to $0.0m$.

SpawnTransform Specifies the model transform to be used as the launch transform. Optional, but using a spawn transform allows finer control over the launch position than that afforded by `SpawnHeightOffset`, and also allows the orientation to be specified. If not specified, the model's root transform will be used as the launch transform (setting `SpawnHeightOffset` is highly recommended, but not as highly as having a spawn transform).

PadName Specifies the name of the launchpad. Note that this is editable by the user both in the editor (VAB/SPH) or in flight.

Operational Persistent backing for `ELControlInterface.canOperate`. Defaults to `true`.

5.1.6. ELNoControlSwitch

Resets the vessel's control reference to what it was before a kerbal boards the external command seat. Used by the rocket workbench to prevent the vessel getting weird control references[28].

Model Requirements Somewhere for a kerbal to sit.

Part Requirements At least one KerbalSeat module.

Module Fields None.

5.1.7. ELRecycler

Destroys anything it touches (including unfortunate kerbals), reclaiming what resources it can. Implements `ELControlInterface`.

Model Requirements The only requirement is the recycle field. The recycle field must be a trigger collider and should (must?) not touch any other collider.

[28] Imagine how annoying this can be when using SAS hold option on a bendy station. Now imagine the motivation for creating this part module.

Part Requirements None.

Module Fields

RecycleField_name Specifies the name of the transform for the recycle field collider. Defaults to "ReycleField".

RecycleRate Specifies the recycling rate in tons/second. Defaults to $1.0t/s$.

Operational Persistent backing for `ELControlInterface.canOperate`. Defaults to `true`.

5.1.8. ELSurveyStake

Marks locations for survey station. In the current implementation, a stake must be the only part in the vessel for the survey station to recognize it.

Model Requirements None except any required by KIS[**KIS**] for ground attachment.

Part Requirements As the survey system will not look at vessels with more than one part to check for the `ELSurveyStake` module, the part should be configured to be ground attached using KIS[**KIS**]. However, parts designed to be dropped via staging or decoupling will work, too, so long as the resulting vessel consists of only the one part.

Module Fields None.

5.1.9. ELSurveyStation

Builds complete vessels at locations marked out using survey stakes (parts with the `ELSurveyStake` module). Does not allow post-build resource transfer (freedom is not free), but as KIS[**KIS**] is required to place the stakes, and KAS[**KAS**] is almost always installed with it, survey stations are probably the preferred tool for landed operations. Implements `ELControlInterface`.

Model Requirements None.

Part Requirements No requirements, but as kerbals improve its range, having crew capacity (`crewCapacity` > 0 or `KerbalSeat` modules) is recommended.

Module Fields

StationName Specifies the name of the survey station. Note that this is editable by the user both in the editor (VAB/SPH) or in flight.

Operational Persistent backing for `ELControlInterface.canOperate`. Defaults to `true`.

5.1.10. ELTarget

Allows a part to be targeted. Includes orientation so it works with any docking alignment mod (DPAI[**DPAI**], navball[**NBDOCK**], and navhud[**NAVHUD**] are known to work).

Model Requirements None.

Part Requirements None.

Module Fields

TargetTransform Specifies the model transform to be used as the target. If not specified (the default), the model's root transform will be used.

TargetName String to be added after the host vessel's name when set as target. Defaults to "Target".

5.1.11. ELWorkshop

Collect productivity from kerbals in the part. Works with either normal parts with crew capacity or command chairs.

Model Requirements None.

Part Requirements The part must have some crew capacity. This can be via either the part's `crewCapacity` field, or `KerbalSeat` (stock KSP) modules, or both. Note that parts may have multiple `KerbalSeat` modules on them (eg, EL's Rocket Workbench).

Module Fields

ProductivityFactor Specifies the multiplier for calculating kerbal productivity. Must be greater than 0. All workshops with `ProductivityFactor` greater than 1.0 are considered to be fully equipped (ie, even 0-star kerbals with the construction skill can contribute). Defaults to 1.0.

UnmannedProductivity Productivity of the workshop when unmanned. Not affected by `ProductivityFactor`. Defaults to 0.0.

FullyEquipped If true, then even workshops with productivity factors less than 1.0 are considered fully equipped allowing 0-star kerbals to contribute.

IgnoreCrewCapacity If true, the workshop will operate even if the part's `crewCapacity` is 0 (and not check for `KerbalSeat`). This is most useful on parts with dynamic crew capacities (eg, inflatables).

5.2. Recipes

Extraplanetary Launchpads provides, via recipes, a means of customizing the resource costs for building parts, their modules and resources, and thus whole vessels. The recipes are used also for recycling.

Essentially, recipes are config nodes with a list of ingredients with their ratios in the form of IngredientName = Ratio, although each recipe type will be a little more complex (details given below). The final ratio of each ingredient is calculated by dividing the specified ratio by the sum of the ratios of all ingredients in the recipe such that the final total is 1.0. This allows for flexibility in how the ratios are specified, so long as consistency is maintained throughout the individual recipe: they can be considered as "parts" as in when mixing drinks (one part this, two parts that...), as masses in any unit ($24g$ this, $16g$ that, $6g$ the other[29]), percentages (if they add up to 100), or raw ratios (if they add up to 1.0). Note, however, that EL always uses mass for its calculations.

For example, glucose ($C_6H_{12}O_6$) using approximate molar masses:

Using masses:	Using parts:	Using raw ratios:
`Carbon = 72`	`Carbon = 6`	`Carbon = 0.4`
`Hydrogen = 12`	`Hydrogen = 1`	`Hydrogen = 0.06667`
`Oxygen = 96`	`Oxygen = 8`	`Oxygen = 0.53333`

In general, the ingredients will be KSP resources. EL looks up the resource to find its density and calculates the amount of resource required based on the total mass of the recipe, the ingredient's ratio within the recipe as a fraction of the total mass, and the density of the resource to give the resource units required to achieve that mass.

Ingredients may be repeated within a recipe, in which case their ratios will simply be added together. This makes creating recipes from chemical formula a little easier.

Unknown ingredients contribute to the total ratio and thus affect the ratios of known ingredients. Their effects when building are undefined, but they operate as losses when recycling (i.e., unknown ingredients simply evaporate when a part is recycled).

Ingredients specifying KSP resources that have no mass do not contribute their ratios to the recipes total, but will be scaled by the same amount as the ingredients that do have mass. Unknown ingredients are assumed to have mass. A recipe consisting of only ingredients that have no mass will cause headaches.

5.2.1. Recipes for Building

Building parts (and thus vessels) requires resources. The total mass of the resources required for building a part is given by the part's mass, but the mix of resources needed by the part is given by the part's recipe. Also, certain resources stored in a part (e.g. SolidFuel, Ablator) cannot normally be created by other means (neither stock ISRU nor Kethane provide a means to produce them), nor can they be transfered, so recipes can be used for specifying how to build them.

[29] A popular compound.

Any resource listed in a recipe becomes a required resource (i.e. the build will not complete if any required resource runs out). Note, however, that normal resources that are simply stored in the part to be built remain optional. For example, to build a tank that holds RocketParts, A mass of RocketParts equivalent to the dry mass of the tank is required to build the tank itself, but the RocketParts used to fill the tank remain optional. Thus, if there is a supply of RocketParts sufficient to build the tank, but not enough to fill it, then the tank will be only partially full (or possibly even empty) when built.

In summary:

- Any resource mentioned in `EL_Recipe` or `EL_ModuleRecipe` is required for building (but not for filling tanks).
- Any stored resource that has an `EL_ResourceRecipe` becomes a required resource (eg, SolidFuel).
- Any resource stored in a part inside a KIS container becomes a required resource, regardless of recipes.

EL_Recipe Specify the resources needed to build a part. `EL_Recipe` is to be added to the part's config (either by hand or using Module Manager). `EL_Recipe` nodes are really two nested recipes: the outer recipe specifies the ratios between the part's structure (using `structure` as the ingredient name) and its part modules (the ingredient name is the name of the part module). The inner recipe is the `Resources` node, which specifies the resources needed to build the part's structure.

Any part that does not have an `EL_Recipe` node will be given the default shown below, but with additional `modulename = 1` lines for each part module on the part that has a corresponding `EL_ModuleRecipe`. Also, the `Resources` node will be taken from `EL_DefaultStructureRecipe` (both for parts that have no `EL_Recipe` node, or whose `EL_Recipe` node has no `Resources` node).

```
EL_Recipe {
    structure = 5
    Resources {
        RocketParts = 1
    }
}
```

EL_ModuleRecipe Specify the resources needed to build any part's module. Applies to all instances of that module. The module to which the recipe applies is specified by the `name =` line, and the actual recipe for the module is specified by the `Resources` node. The mass of the module is calculated from the part's mass using the ratio specified in the part's recipe. If the named module does not exist, no module is named (ie, no top-level `name =` line), or no recipe is given (no `Resources` node), the recipe will be dropped from the recipe database.

The example below gives EL's module recipe for `KerbalEVA`. The ratios are in kilograms, assuming a suited kerbal has a mass of $93.75kg$ ($10kg$ for the kerbal). It can be found in `Kerbal.cfg`[**elker**].

```
EL_ModuleRecipe {
    name = KerbalEVA
    Resources {
        Metal = 39
        loss = 44.75
    }
}
```

EL_DefaultStructureRecipe Specify the default recipe to be used for a part's structure when the part has no recipe or the recipe does not specify a recipe for its structure. There is no node in EL's configs: it is hard coded. However, providing an `EL_DefaultStructureRecipe` will override the hard-coded default.

```
EL_DefaultStructureRecipe {
    RocketParts = 1
}
```

EL_ResourceRecipe Specify the resources needed to "build" a resource. The resource to be "built" is specified by the `name =` line, and the recipe for the "built" resource is specified by the `Resources` node. If no resource is named (ie, no top-level `name =` line), or no recipe is given (no `Resources` node), the recipe will be dropped from the recipe database. Recipes for undefined resources are permitted, allowing for resource "macros"[30]. The example resource recipe shows Ablator being made from RocketParts. It and a similar recipe for SolidFuel can be found in `Recipes.cfg`[**elrec**].

```
EL_ResourceRecipe {
    name = Ablator
    Resources {
        RocketParts = 1
    }
}
```

5.2.2. Recipes for Recycling

First off, when recycling, part recipes (`EL_Recipe`) are used for breaking a part down into its constituent resources. Whether from breaking down the part or "drained" from the part's resource storage, any resource that has a resource recipe (`EL_ResourceRecipe`) will simply evaporate. However, if the resource has a recycle recipe (`EL_RecycleRecipe`),

[30]TODO: In theory: not tested.

then that recipe will instead be broken down to the resources specified by the recycle recipe. A transfer recipe (`EL_TransferRecipe`) is used to force a stored resource that would otherwise be lost or broken down by a recycle recipe to be transfered.

- Any resources stored in the part are drained. Those with an `EL_TransferRecipe` are transferred accordingly. Those with an `EL_ResourceRecipe` but no `EL_RecycleRecipe` are lost, otherwise they are broken down as dictated by the `EL_RecycleRecipe` and the resultant resources will be transfered.
- The part is then broken down into the resources specified by its `EL_Recipe` and `EL_ModuleRecipe`(s). Those resources with an `EL_ResourceRecipe` but no `EL_RecycleRecipe` are lost, otherwise they get broken down further in accordance with their `EL_RecycleRecipe`.
- Whether transferring (from a tank) or recycling (the part itself), resources with no recipe are reclaimed as-is at a 1:1 ratio.

EL_RecycleRecipe Specify the resources to which a resource will be broken down when recycling. Prevents evaporation of the resource when the resource has a resource recipe (`EL_ResourceRecipe`). The resource to be broken down is specified by the `name =` line, and the recipe for the broken down resource is specified by the `Resources` node. If no resource is named (ie, no top-level `name =` line), or no recipe is given (no `Resources` node), the recipe will be dropped from the recipe database. Ingredients specifying undefined resources are permitted, allowing for loss ratios to be specified. The example recycle recipe shows RocketParts being broken down to ScrapMetal with 10% loss (the exact name (`loss`) doesn't matter so long as it is not a defined resource). It can be found in `Recipes.cfg`[**elrec**].

```
EL_RecycleRecipe {
    name = RocketParts
    Resources {
        ScrapMetal = 9
        loss = 1
    }
}
```

EL_TransferRecipe Specifies how a resource that was stored in a part is to be transfered. Prevents the resource from being broken down by a recycle recipe when being drained from a part's storage. The resource to be transfered is specified by the `name =` line, and the recipe for the transfered resource is specified by the `Resources` node. If no resource is named (ie, no top-level `name =` line), or no recipe is given (no `Resources` node), the recipe will be dropped from the recipe database. Recipes for undefined resources are permitted, allowing for loss ratios to be specified. In general, the same resource should be specified in the recipe, but a transfer recipe can be used for converting a resource that does not have a recycle recipe, or for specifying a loss factor while transferring. The example transfer recipe shows

RocketParts being transfered without any loss or conversion. It can be found in `Recipes.cfg`[**elrec**].

```
EL_TransferRecipe {
    name = RocketParts
    Resources {
        RocketParts = 1
    }
}
```

EL_KerbalRecipe Specifies the resources making up a kerbal (whether on EVA or boarded[31]). This is a special part recipe that is not actually attached to any part[32], and is used only when the unfortunate kerbal gets recycled. The kerbal recipe shown below assumes a fully suited kerbal is $93.75kg$ (the default in KSP) with $10kg$ for the kerbal and $83.75kg$ for the suit, with a 30:1 *gain*[33] when converting the kerbal to Kethane. It can be found in `Kerbal.cfg`[**elker**].

```
EL_KerbalRecipe {
    structure = 10
    KerbalEVA = 83.75
    Resources {
        Kethane = 30
        loss = -29
    }
}
```

5.2.3. Recipes for Converting

EL_ConverterRecipe Conversion recipes make configuring chemically correct[34] resource converters fairly simple as they allow the conversion rate to be separated from the resource ratios. Each conversion recipe provides a name which can be referenced by the converter config, a set of input recipes, and a set of output recipes. At least one of each recipe set must be present for the converter recipe to be valid, but there is no need for the two sets to have the same number of recipes.

All ingredients in an input recipe must be either a defined KSP resource, or an EL_ResourceRecipe. Any ingredients in a referenced EL_ResourceRecipe also must be defined. In the example below, LFOMix is such. It is used because stock KSP LiquidFuel and Oxidizer densities are incorrect for RP-1 (0.81kg/L) and LOX (1.141kg/L) which makes getting the rates correct difficult. Thus LFOMix is a workaround.

[31] The kerbal is assumed to be suited or have a suit nearby in the same part

[32] The need for the node came from not having access to KerbalEVA part configs at the time, and keeping it after KSP 1.2 maintains flexibility.

[33] Highly unrealistic, but that is how the KE-WAITNONOSTOP-01 in Kethane is configured

[34] Or reasonable approximations thereof.

Any undefined ingredients in an output recipe simply evaporate (i.e., they are jettisoned).

Both input recipes and output recipes have a special ingredient: EFFICIENCY. The EFFICIENCY ingredient is used to calculate the actual recipe used by the converter based on the converter's current efficiency, acting as keys in an efficiency curve. The ratios of the other ingredients are interpolated between those specified by the recipes with efficiencies on either side of the converter's current recipe. Ingredients in converter recipes support an optional heat parameter which defaults to 0 and is scaled with the ingredient when the recipe is baked. Output heats are added to the part's internal flux while input heats are subtracted from the part's internal flux. If the ingredient ratios are in grams, then the heats are in kilo-joules. The mass flow of the conversion process is governed by the input recipe with the smallest mass.

```
EL_ConverterRecipe {
    name = LFOFiredSmelter
    Input {
        efficiency = 1
        LFOMix = 864.49048 -5186.94288
        MetalOre = 1596.882
    }
    Output {
        efficiency = 1
        CarbonDioxide = 1056.228
        Water = 288.24448
        Metal = 1116.9 -9627.678
    }
    Input {
        efficiency = 0
        LFOMix = 864.49048 -5186.94288
    }
    Output {
        efficiency = 0
        Carbon = 24.0214
        CarbonDioxide = 176.038
        CarbonMonoxide = 476.1717
        Formaldehyde = 30.02598
        Hydrogen = 14.11116
        Water = 144.12224
    }
}

EL_ResourceRecipe {
    name = LFOMix
```

```
        Resources {
            LiquidFuel = 9
            Oxidizer = 11
        }
    }
```

5.3. Resource Rates

The amount of work required to prepare resources may be configured using an EL_ResourceRates node. This node is very simple in that it is just a list of `resourcename = rate` lines. `resourcename` is the name of the resource (or `default` to specify the default rate), and `rate` is the amount of kerbal-hours / ton (or kerbal-seconds / unit for massless resources (such as ElectricCharge)). Note that this affects only those resources that are required to complete the build, not optional resources that will be transferred afterwards. If `default` is not specified, then it defaults to 5.

```
EL_ResourceRates {
    default = 5
    ElectricCharge = 1
    LiquidFuel = 0.36
    Oxidizer = 0.44
    MonoPropellant = 0.4
    XenonGas = 2
    Ore = 8
}
```

6. EL API

Calling it an API might be stretching things, but...

6.1. Interfaces

6.1.1. ELBuildControl.IBuilder

ELBuildControl is the actual workhorse for building vessels, the various pad modules all use ELBuildControl for the common functionality. The IBuilder interface is the methods and properties that ELBuildControl requires of the pad modules. There are several methods and properties, but main property of interest is control which provides access to the ELBuildControl object attached to the pad.

6.1.2. ELControlInterface

Its purpose is to allow other mods easy control and detection of the operational status of EL's modules. For example, mods with inflatable parts can disable the EL module when the part is deflated and block deflation when the EL part is busy.

Fields Note that these are actually properties, so access via reflection needs to be done using `GetProperty`, `PropertyInfo.GetValue` and `PropertyInfo.Setvalue`.

isBusy Read-only boolean that indicates whether the module is currently busy processing something. While not enforced, the module should not be disabled while it is busy (i.e. the part should not be deflated).

canOperate Boolean usable by other part modules to enable or disable (or detect) the operational status of the EL part module. Setting this may cause the implementing module to run other code (e.g. the `ELSurveyStation` module will recompute its range and possibly scan for sites), so mods must invoke the property setter to guarantee correct behavior.

6.1.3. ELWorkSink

Consumes vessel productivity to get something done.

6.1.4. ELWorkSource

Provides vessel productivity.

6.1.5. IResourceProvider

Interface used by ELExtractor for querying environmental resource availability and extracting a resource from the environment.

6.2. Part Modules

6.2.1. ELControlReference

6.2.2. ELConverter

6.2.3. ELDisposablePad

6.2.4. ELExtractor

6.2.5. ELLaunchpad

6.2.6. ELNoControlSwitch

6.2.7. ELRecycler

6.2.8. ELSurveyStake

6.2.9. ELSurveyStation

6.2.10. ELTarget

6.2.11. ELWorkshop

6.3. Vessel Modules

6.3.1. ELVesselWorkNet

Central control of productivity for the entire vessel. Keeps track of output from work sources (eg, workshops) and distributes the productivity over active work sinks (eg, pads). Operates on unloaded vessels as well as loaded vessels so that work is properly distributed when multiple work sinks are operating but they finish at different times while the vessel is unloaded. Also prevents erroneous productivity credit caused by leaving a vessel unattended for several game years[35].

[35] Background building was implemented by keeping track of how long the vessel was unloaded. When the vessel was loaded, the time was processed in chunks of (default) 21600 seconds every physics frame. This works out to fifty Kerbin days per second, so if a vessel had not been visited for ten Kerbin years, there would be an 85 second (game-time, worse for low FPS real-time) window after switching to the vessel in which new builds would finish instantly (assuming resource availability).